UNIT-1

INTRODUCTION TO EMBEDDED SYSTEMS

Contents: Introduction to ESs, History of embedded systems, Classification of embedded systems based on generation and complexity, Purpose of embedded systems, The embedded system design process-requirements, specification, architecture design, designing hardware and software components, system integration, Applications of embedded systems, and characteristics of embedded systems.

INTRODUCTION TO EMBEDDED SYSTEMS

Definition:

An Embedded System is a system that has software embedded in computer hardware. It is designed to perform a dedicated function. Sometimes, it may be designed to perform a set of few tasks.

An embedded system mainly consists of hardware, OS, memory, peripherals (I/O devices) and interfaces for performing the set of dedicated tasks. The embedded software is called "firmware".

- A few embedded systems do not contain OS.
- A few embedded systems do not use any interfaces.

Embedded Systems Vs. General Computing Systems

A computer (PC or Laptop) is an example of general-purpose computing system. An optical mouse is an example of embedded system. Charles Babbage invented computer. Dr. Charles Stark Draper developed first modern embedded system.

Comparison:

S. No.	Category	General purpose com-putting system	Embedded System
1	Hardware and OS	It consists of general-purpose hardware & OS to perform large number of tasks.	It consists of application specific hardware and OS to perform one or a few tasks.
2	Memory	Needs larger memory.	Needs lesser memory.
3	Peripherals (I/O devices)	It uses many peripheral devices.	It uses few peripherals.
4	User Interface	It requires more user interfaces.	Lesser or no user interfaces.

5	Reprogramming	It can be reprogrammed for a new purpose.	It cannot be reprogramed by the end-user, except in a few cases.
6	Response requirements	No time constraints.	Stringent time constraints.
7	Size & Cost	Usually bigger in size and costly.	Smaller in size and less expensive.
8	Power Consumption	Requires more power.	Relatively less power.

HISTORY OF EMBEDDED SYSTEMS

- Embedded System (ESs) existed even before IT revolution. In the olden days, ESs were built using vacuum tubes and transistor technology. Embedded algorithms were developed in low level languages.
- With the advent of semiconductor technology, nano-technology and IT revolution, miniature ESs came into picture.
- The first modern, real-time ES was "**Apollo Guidance Computer (AGC)**". It was developed in 1960 by Dr. Charles Stark Draper at the MIT Instrumentation Labs.
- In 1965, **Autonetics developed the D-17B.** It was used in the computer used in the Minuteman missile guidance system. It is widely recognized as the first mass-produced ES.
- In 1968, **the first ES for a vehicle** was released.
- In 1971, Texas Instruments developed the **first microcontroller**.
- In 1987, **the first embedded OS**, VxWorks, was released by Wind River.
- In 1996, Microsoft developed its OS called "**Windows embedded Compact**".
- By the late 1990s, the **first embedded Linux system** appeared.

CLASSIFICATION OF EMBEDDED SYSTEMS

1. Classification based on Generation

First Generation ESs:

- They were built using 8-bit microprocessors like 8085 and Z80, and 4-bit microcontrollers.
- They used simple hardware.

- Firmware was developed in Assembly language.

Examples: Digital telephone keypads and stepper motor control units.

Second Generation ESs:

- ✓ They were built around 16-bit microprocessors and 8- or 16- bit microcontrollers.
- ✓ They used complex and powerful instruction sets.
- ✓ Some of them contained embedded OSs for their operation.

Examples: Data Acquisition Systems and SCADA (Supervisory Control and Data Acquisition systems) used in industrial plants.

Third Generation ESs:

- ➢ Built with 32-bit microprocessors and 16-bit microcontrollers supporting instruction pipelining.
- ➢ DSPs and ASICs were available to improve their performance.
- ➢ They used more complex and powerful instruction sets.
- ➢ General-purpose Oss and RTOS entered the market.

Examples: In robotics, industrial process control and networking.

Fourth Generation ESs:

- o System on Chips (SoCs), reconfigurable processors and multicore processors entered into the embedded market. They brought high performance and miniaturization.
- o Used RTOSs for their functioning.

Examples: Smart phones

Next Generation: The processor and embedded market are highly dynamic and demanding. So many sophisticated applications will come in next generation.

2. Classification based on Complexity and performance

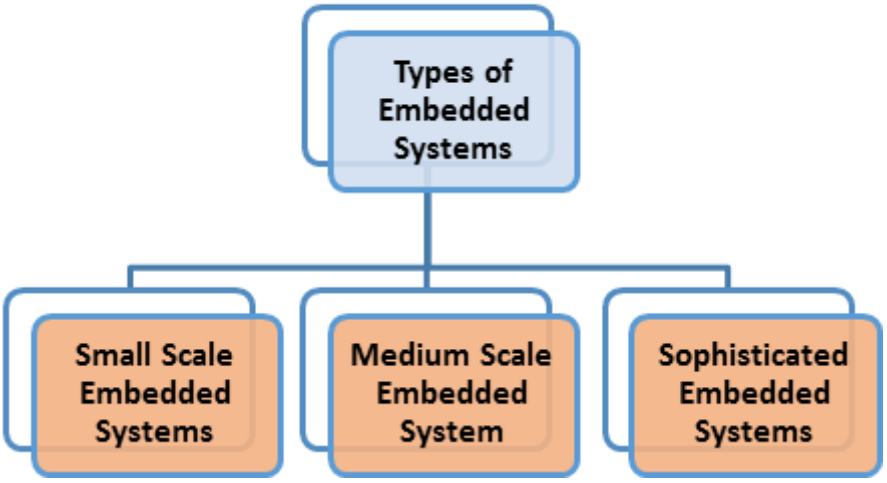


Fig. 1

(i) Small-Scale Embedded systems: They are simple systems.

- Built around a single 8- or 16- bit microcontroller or DSP.
- Performance requirements are not time critical
- Simple hardware and software and board-level design.
- May be battery operated. May or may not contain OS.
- May contain IDE (integrated development environment).
- Generally, use C or MATLAB.

Examples: Electronic toys, Automatic chocolate-vending machines, photocopier, etc.

(ii) Medium-Scale Embedded systems:

- Designed with one or few 16- or 32-bit microprocessors, DSP or RISC.
- Use RTOS for multiplexing.
- Use IDE with software tools and programming tools.
- Have both hardware and software complexities.
- Use C/C++/Visual C++/Java.

Examples: TV set-up box, SIM card in mobile phone, Video games

(iii) Sophisticated Embedded Systems:

- ✓ Designed with 32- or 64-bit RISC processors/controllers.
- ✓ May need new development tools.
- ✓ May be an SoC (system-on-chip).
- ✓ Use high-performance Real-Time Operating Systems (RTOS).

Examples: High speed LANs, multimedia processing, etc.

PURPOSE OF EMBEDDED SYSTEMS

Each ES is designed with one or more of the following purposes.

1. Data Collection/Storage/Representation
2. Data communication
3. Data (signal) processing
4. Monitoring
5. Control
6. Application Specific User Interface

1. Data Collection/Storage/Representation:

Here data means information like text, audio, video, electrical signals and other measurable quantities. Data is collected from the external world. Based on the purpose of the ES, the collected data may be used in one or more of the following ways: -

- Stored directly in the system.
- Transmitted to other systems.
- Processed by the system.
- Deleted instantly after giving a meaningful 'graphical representation' or 'Quantity value' (CRO without storage memory).

Example: **A digital camera (**Images can be captured, stored and presented to user through a graphic LCD unit).

2. Data communication:

- ✓ ESs are used in data communication applications ranging from simple home networking systems to complex satellite communication systems.
- ✓ The collected data can be transmitted to a remote system using wire-line medium (e.g., RS-232C, USB) or wireless medium (e.g., Bluetooth, ZigBee and Wi-Fi) by analog or digital means.

3. Data (Signal) processing:

The collected data may be used for various kinds of signal processing. Such ESs are used in applications like speech coding, synthesis, audio video codec.

Example: A digital hearing aid

4. Monitoring:

- All the embedded products used in medical domain are designed for monitoring purpose.
- They determine the status of some variables using sensors.
- They cannot control the variables.

Example1: ECG machine.

It monitors heartbeat of a patient. But it cannot control the heartbeat. Here the sensors used are electrodes connected to human body.

Other examples: Digital CRO, digital multimeter and logic analysers.

5. Control:

- ES can also control some output variables when input variables change.
- They use **sensors** at the input port and **actuators** at the output port to control the output variables and keep them in a specified range.

Example: ATM

6. Application specific user interface:

ESs use application-specific user interfaces like buttons, switches, keypad, lights, bells and display units.

Consider a mobile phone. In this, user interface is provided through the keyboard, graphic LCD module, system speaker, etc.

EMBEDDED SYSTEM DESIGN PROCESS

Design process: Design Process is a way of figuring out what you need to do in developing an embedded product. It describes design procedures, techniques, tools and design flow steps at various levels of abstraction.

Abstraction: Abstraction is the main concept in design process. Abstraction of design means presenting essential features of implementation and hiding the minor or background details.

Steps in Design Process (Levels of abstraction):

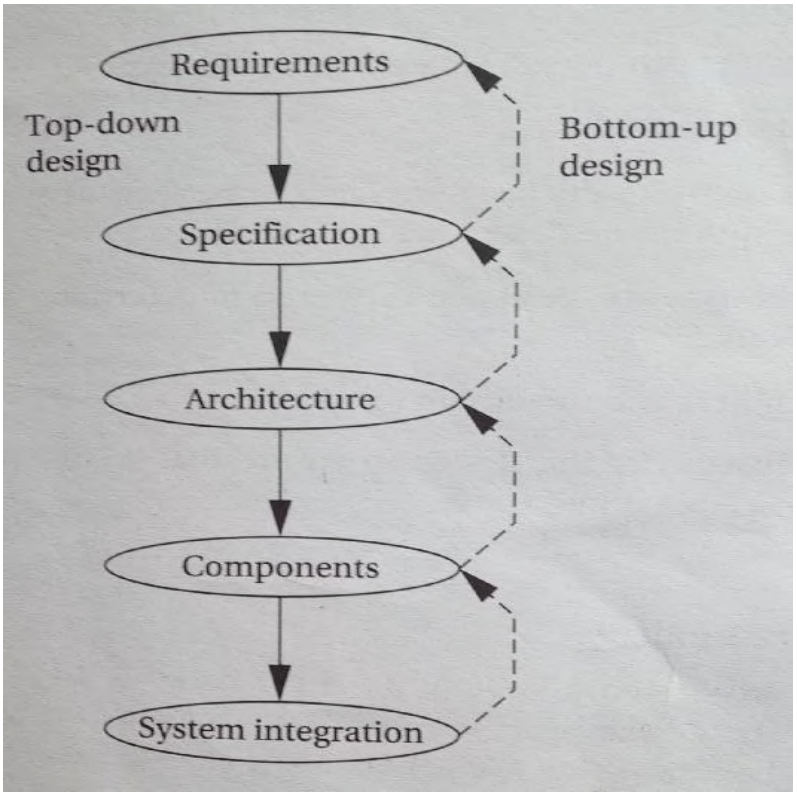


Fig. 2: Steps in Design Process

1. Requirements:

Requirements means the functions and quality attributes of the embedded system to be developed. Functions of the system are called **Functional requirements** (e.g**.,** soaking, washing, rinsing and drying functions of a washing machine). Quality attributes are called "**Non-functional requirements"** (e.g., performance, throughput, , security, reliability)

Requirements gathering: Customer initially gives all the requirements. But they may not be sufficient. They may also change with time due to various reasons. So, requirements are gathered by the conducting surveys, user meetings, questionnaires, usage scenarios and brain storming sessions with stakeholders.

Requirements analysis: Then the requirements are analysed by system analists to verify the following characteristics

- Correctness: Requirements should be correct and clear.
- Consistency: Requirements should not conflict with each other.
- Modifiability: They should allow changes when customer changes their needs.
- Traceability: Ability to trace a requirement forward and backward in the development life cycle.
- Verifiability: They should allow you to verify easily.

2. Requirements Specification: After gathering and analysing the requirements, requirements are specified in the form of a document called "Software Requirements Specification (SRS)" document Specification serves as the contract between the customer and the design team.

It gives guidance in the development process and represents overall expectations of the customer. It must be concise, understandable, unambiguous and verifiable.

Some important specifications for an embedded product:

- Functions of the embedded product (e.g., withdrawal, deposit and balance enquiry of ATM)
- Unit cost, size and power consumption
- Response time and throughput
- Product deadlines
- Reliability and safety measures

6. System Integration: Usually, the system is divided into various modules called units). Each unit is designed, developed and tested independently. Afters unit testing, all the modules are integrated to get the final product.

Components and modules may work fine independently. But when integrated, there will be many bugs. So, appropriate integration testing is performed to ensure good performance and quality.

3. Architecture Design: Architecture is a plan of the overall structure of the system that will be used later in design and development. The specification only says what the system does. Architecture describes how the system does the things. Architecture of an ES involves: -

- ✓ Data & Program models
- ✓ Data paths
- ✓ Hardware & software components
- ✓ Software structures, components and architecture layers.

4. Design of Hardware Components:

- ➢ Processor
- ➢ Memory and I/O units
- ➢ Sensors and Actuators
- ➢ ADC and DAC
- ➢ User Interfaces, etc.

5. Design of Software components such as:

- ✓ Operating System
- ✓ Control algorithms for various functions
- ✓ Programming language
- ✓ Software tools
- ✓ Device drivers, etc.

Prototype: A Prototype is a first version of the product. It is submitted as a part of design process. It need not have working parts. It may be a physical model of the product, built using thermocool or craft paper. It may also be a 2D or 3D diagram or video presentation. It gives a rough idea of how the system functions. It also helps in identifying software requirements.

APPLICATIONS OF EMBEDDED SYSTEMS

1. **Domain:** Consumer Electronics
   Applications: Digital cameras, digital wrist watches and DVD player
2. **Domain:** Household (Domestic) appliances
   Applications: Washing machine, microwave oven, TV and refrigerator
3. **Domain:** Home automation
   Applications: Automatic door locks (in home, buses and trains), automatic blinds,

4. **Domain:** Security systems

   Applications: CCTV cameras, and fire alarms in home and industry.

5. **Domain:** Automotive industry

   Applications: Airbag, anti-lock braking system (ABS) and other systems used for entertainment and safety.

6. **Domain:** Telecommunication

   Applications: Mobile phones and multimedia applications.

7. **Domain:** Computer Peripherals

   Applications: Printers, keyboards, keypads, LCD display, optical mouses, scanners and fax machines.

8. **Domain:** Computer networking systems

   Applications: Network switches, routers, hubs and firewalls.

9. **Domain:** Health care

   Applications: Different kinds of scanners, EEG and ECG machines.

10. **Domain:** Measurement & Instrumentation

    Applications: Digital multimeters, digital CROs and logic analysers.

11. **Domain:** Banking & Finance

    Applications: Automatic Teller Machines (ATMs), currency counters and points of sales (POS).

12. **Domain:** Card Readers

    Applications: Barcode readers and smart card readers

Smart Cards: ATM card, RFID card, SIM cards, access badges, etc.

CHARACTERISTICS OF EMBEDDED SYSTEMS

Each ES has a set of specific and unique characteristics. Some of the important characteristics of ESs are: -

1. Application and Domain specific
2. Reactive and Real Time
3. Operates in harsh environments
4. Distributed
5. Small size and weight
6. Power Concerns

1. Application and Domain specific:

Each ES is designed to perform one or few unique functions. It cannot be used for any other purpose.

For example: You cannot replace embedded control system

- ✓ For a microwave oven by that of a washing machine.
- ✓ For a Telecom domain with that of a consumer electronics domain.

2. Reactive and Real Time:

ESs continuously interact with the Real world through sensors, control algorithms and input devices at the input port of the system. So, they are also called reactive systems.

An ES produces changes in output response to the changes in the input variables. It is designed with proper response time for each task. It should not miss the deadline for any task.

Any change in the real world is called event.

An event may be a periodic event or unpredicted event. There is no problem with periodic events. They are well identified and handled. The important thing is an ES should not miss unpredicted events. The design of such systems should take worst-case situations into consideration.

Note: Some ESs may not be real time in operation. (e.g., electronic toys).

3. Operates in harsh environments:

Sometimes ESs may be installed in harsh environments. Such systems must be capable to withstand all the adverse conditions. The design should take care of operating conditions of the area where it is installed.

Example 1: ESs in high temperature zone

All the components used in the system should be of high temperature grade. Here we cannot go for compromise in cost.

Example 2: ESs in vibrations and shock environment

Proper shock absorption techniques should be provided.

Other examples for harsh environments: Power supply fluctuations, corrosion, component aging, etc.

4. Distributed

Sometimes many distributed ESs may form a single large ES. All of them are independent embedded units. But they work together to perform the specified function.

Example 1: Automatic chocolate vending machine (ACVM) contains a user keypad, coin counting module, and chocolate dispensing mechanism. Each of them is an individual ES. All of them work together to perform overall vending function.

Example 2: an Automatic Teller Machine (ATM) contains a card reader for reading and validating ATM card, cash dispensers, receipt printers and LCD Display units.
All the units are individual ESs. But they work together to achieve a common goal.

5. Small size and weight:

Generally, a user gives much importance to product aesthetics (size, weight, shape, style, etc.) in choosing a product. (e.g., Mobile phone)
Many users like small devices. It is not just their like. A small, less weighted and compact (occupying less space) device is always convenient to handle than a bulky one.
Same is the situation in embedded domain also.

6. Power Concern:

Power consumption: Power consumption is the amount of energy used per unit time. This implies that ESs should have low power consumption. This can be achieved by using the ultra-low power consuming components available in the market.
Also, battery life of portable systems (e.g., Cell phones and laptop computers) is limited by power consumption.

Power dissipation: An ES with large power dissipation needs cooling fans, which occupy additional space and make the system bulky. So ESs should have low power dissipation.
To achieve this, ESs are designed using regulators with low dropout and processors / controllers with power saving mode.

Note; Low dropout means small difference between supply voltage and load voltage

******* OVER*******

UNIT-2

TYPICAL EMBEDDED SYSTEM

Contents: Core of the embedded system-general purpose and domain specific processors, ASICs, PLDs, COTs; Memory-ROM, RAM, Memory according to the type of interface, Memory shadowing, Memory selection for embedded systems, Sensors, Actuators, I/O components: Seven Segment LED, Relay, Piezo buzzer, Push button switch, Other sub-systems: Reset circuit, Brownout Protection circuit, Oscillator Circuit, Real time clock, Watch dog timer.

CORE OF THE EMBEDDED SYTEM

Core means central part. Core of an ES contains any one of the following: -

1. General Purpose and Application Specific Processors
    - Microprocessors
    - Microcontrollers
    - Digital Signal Processors (DSPs)
2. Application Specific Integrated Circuits (ASICs)
3. Programmable Logic Devices (PLDs)
4. Commercial off-the-shelf components (COTS)

1. GENERAL PURPOSE AND DOMAIN SPECIFIC PROCESSORS

A Processor is a computer component which fetches, interprets and executes instructions. Based on the purpose they serve; processors are classified as General-Purpose Processors (GPPs) and Application-Specific Instruction Set Processors (ASIPs).

General Purpose Processor (GPP):

A GPP is a processor designed for general computational tasks (e.g., processors used in desktop and laptop computers). Due to mass-production their cost is low.

Application-Specific Instruction Set Processor (ASIP):

ASIP is a processor with architecture and instruction set optimized for a specific application.

Domain and application: Domain means the major area where a system is used and application refers to particular task in a domain. CCTV is an application in security domain. Digital camera is an application in consumer electronics domain.

Based on the domain and application, the processor may be a microprocessor, microcontroller or a digital signal processor (DSP). Most of the applications use microprocessors or microcontrollers. Some speech and video signal processing applications use DSPs.

(i) Microprocessors: A Microprocessor is a central processing unit (CPU) fabricated on a single silicon chip. It mainly contains Arithmetic Logic Unit (ALU), Control Unit (CU) and working registers. It needs other hardware like memory, timer unit, interrupt controller, etc., for proper functioning. An n-bit (n = 8, 16. 32 or 64) microprocessor is capable of handling n-bit data and program memory.

Examples: Zilog Z80 (8-bit), Intel 8086 (16-bit), Motorola 68020 (32-bit)

(ii) Microcontrollers:

A Microcontroller basically contains CPU, memory and I/O devices on a single silicon chip. Microcontrollers are more powerful and more preferred than microprocessors.

They may be either general-purpose microcontrollers (Intel 8051) or domain specific microcontrollers (Atmel's AVR). TMS1000 (from Texas instruments) is world's first microcontroller.

Units of a microcontroller:

- CPU.
- Scratchpad RAM (SPRAM), On-chip ROM/FLASH memory
- General-purpose and special registers
- Timer and interrupt control units
- Dedicated I/O ports

Note: Scratchpad RAM is a high-speed internal RAM directly connected to the CPU. It is used to hold very small items of data for rapid retrieval.

(iii) Digital Signal Processors (DSPs):

- They are powerful, special-purpose, 8- or 16- or 32-bit microprocessors. They are used in signal processing applications. (e.g., Speech processing, image processing, video processing, video games, digital cameras. HDTV, radar, sonar, etc.).
- They are 2 or 3 times faster than the general-purpose microprocessors.
- They involve large number of real-time calculations.
- They implement algorithms in hardware which speeds up the execution.

Note: General purpose processors implement the algorithms in firmware while DSPs implement algorithms in hardware.

Examples for DSP: Motorola 56001

Units of a DSP:

- Program memory
- Data memory
- Computational Engine: It performs the signal processing as per the program stored in memory.

Classification of Processors/Controllers:

Processors/controllers are classified according to the following criteria:

A. Instruction set
B. Architecture
C. Order of storing bits

A. Based on instruction set:

(i) RISC processors/Controllers:

RISC stands for Reduced instruction Set Computing. RISC processors/controllers have a smaller number of instructions.

Example: ARM, AVR microprocessors

Important applications: Security systems, home automation, etc.

(ii) CISC processors / Controllers:

CISC stands for Complex Instruction Set Computing. CISC processor / controller has large and complex instruction set.

Example: VAX, AMD microprocessors

Important applications: Smartphones, PDAs.

Comparison of RISC and CISC:

S. No.	RISC	CISC
1	Few, fixed length, simple instructions.	Large number, variable length instructions appearing like macros in C.
2	Instruction-decoding is simple.	Complex.
3	Instruction pipelining is used. So, execution speed is high.	No instruction pipelining feature. So, less speed.
4	Harvard architecture	Harvard or Von-Neuman architecture.

5.	**Example:** ARM, AVR microprocessors	Examples: VAX, AMD microprocessors
6.	Important applications: Security systems, home automation, etc.	Important applications: Smartphones, PDAs

B. Based on architecture:

(i) Von-Neumann architecture microprocessors/microcontrollers

- It shares a single bus for program memory and data memory.
- Processor first fetches instruction and then fetches data needed. Two separate fetches slow down processor operation.
- Program memory and data memory are stored on same chip. So, there is a chance of accidental corruption of program memory.
- Cheaper and no memory alignment problems.

(ii) Harvard architecture microprocessors/microcontrollers:

- ✓ It contains separate buses for program memory and data memory. So, it has more speed.
- ✓ Program memory and data memory are stored at different locations. So, no chances for accidental corruption of program memory.
- ✓ It is costly and has memory alignment problems.

C. Based on order of storing bytes of data: -

Suppose the word length is two bytes. Then the following is the difference between two types.

(i) Little-Endian Processors/Controllers: Lower order byte is stored in memory at the lowest address. Higher order byte is stored at highest address.

(ii) Big-Endian Processors/Controllers: It is just opposite to above case.

Note: Endianness means 'sequence of bytes' of a word in digital data.

2. APPLICATION SPECIFIC INTEGRATED CIRCUITS (ASICs)

- ASIC is a microchip designed for a specific application.
- ASICs are pre-fabricated or customized for a special purpose.
- They are not reprogrammable.
- They consume small area and so reduce the system size.
- On mass-production, ASICs are profitable.

- Most of the ASICs are proprietary products. So, their developers keep the internal details confidential.
- Non-Recurring Engineering (NRE) Cost: It is a onetime cost to research, design, develop a new product. Once invested, you need not spend again each time while producing additional number of products.

  If the NRE cost is spent by a third party, the ASIC is made openly available in the market. Then it is called **Application Specific Standard Product (ASSP)**.

Applications of ASICs: Chips used for toys, hand held computers, cell phones and voice recorders.

3. PROGRAMMABLE LOGIC DEVICES (PLDS)

There are two types of logic devices-fixed logic devices and programmable logic devices. Fixed logic devices contain permanent digital circuits. They perform one or a few functions. Once manufactured, they cannot be changed.

Programmable logic devices (PLDs) have reprogrammable digital circuits like PALs, PLAs and PROMs. Thus, they allow frequent changes in design. This reduces development time.

Major Classification of PLDs: FPGAs and CPLDs.

(i) Field Programmable Gate Arrays (FPGAs):

An FPGA is an IC configured by a customer after manufacturing. FPGAs are quickly programmed using Hardware Description Language (Hardware Description Language (HDL)) like VHDL or Verilog. This reduces development time. Basic building block of FPGA is called "Configurable Logic Block (CLB)".

FPFAs have CLBs, programmable interconnects and other components. They offer high logic density and performance with many features (e.g., on-chip RAM, DSP, etc.)

Examples: Xilinx Vertex^{TM} series. It provides **8 million** gates.

Applications:

- Video surveillance cameras
- Medical instruments
- Safety systems in vehicles
- Missile guidance systems

(ii) Complex Programmable Logic Devices (CLPDs):

- CPLD is a combination of a re-programmable AND/OR gate-array and a bank of macrocells.

- Macrocell is the basic building block of the CPLD, which can perform sophisticated logic functions.
- CPLDs contain architectural features of both PALs and FPGAs.
- Complexity of CPLDs lies between that of PALs and FPGAs.
- Even though CPLDs have low logic density, they offer very predictable characteristics. So, they are ideal for critical control applications.

Examples: Xilinx CoolRunner™ series.

Applications of CPLDs:

- Used in small design applications like address decoding.
- Because of their small size and low power consumption, they are suitable for cost sensitive, battery operated, portable devices (e.g., like mobile phones and handheld computers).

Comparison of CPLD and FPGA:

S. No.	Feature	CPLD	FPGA
1	Block Density	Less. It can store a few thousand logic blocks	*More. It can store up to 100,000 tiny logic blocks*
2	Power Consumption	Larger	Lower
3	Cost	Cheap	Costly
4	Security	More secured	Less secured
5	Volatility	Non-volatile.	Volatile.
6	Applications Areas	Simple applications.	Complex applications

4. COMMERCIAL OFF-THE-SHELF COMPONENTS (COTS)

COTS refer to hardware or software products that are ready-made and available for sale, lease, or license in the commercial market. They need not be designed and developed. They are available like Over-the-Counter (OTC) medicines.

OTC medicines: Generic medicines for cold, cough, headache, etc., available without prescription.

Advantages of COTS:

- Reduce development time.
- Readily available and up-to-date.

- You can access reviews and advice from existing users.

Disadvantages of COTS:

- Have some unwanted extra features.
- Additional licensing cost.
- Manufacturer of the product may withdraw the product or discontinue the production of COTS at any time.

Examples of COTS: MS-Office, Anti-virus software, Adobe Photoshop, etc.

MEMORY

Memory is a device used to store instructions and data required by a computing system. If the memory is built in processor/controller chip, it is called **on-chip memory**. If the memory is external to the processor/controller chip, it is called **off-chip memory**.

On-chip memory is smaller and faster than off-chip memory. Frequently used data is stored in on-chip memory. The types of on-chip memory are cache and on-chip SRAM.

Basically, memory is classified as RAM and ROM

RANDOM ACCESS MEMORY (RAM)

- A RAM is a memory which allows both read and write operations.
- It is a direct access memory. i.e., we can directly access any location in the memory in same time (like in MP3 player).
- It is used as a data memory (also called working memory). It is a volatile memory. i.e., its contents are lost when power supply is turned OFF or during power failure. It uses MOS technology.

Types of RAMs: RAMs are of 3 types - Static RAM, Dynamic RAM and Non-Volatile RAM.

Static RAM (SRAM):

SRAM is a type of RAM that retains data bits as long as power supply is there. Basic cells for storing information in SRAM are flip-flops. Conventionally each flipflop has 6 transistors as shown in Fig. 1(a). Once a flip-flop stores a bit, it keeps that value until the opposite value is stored in it.

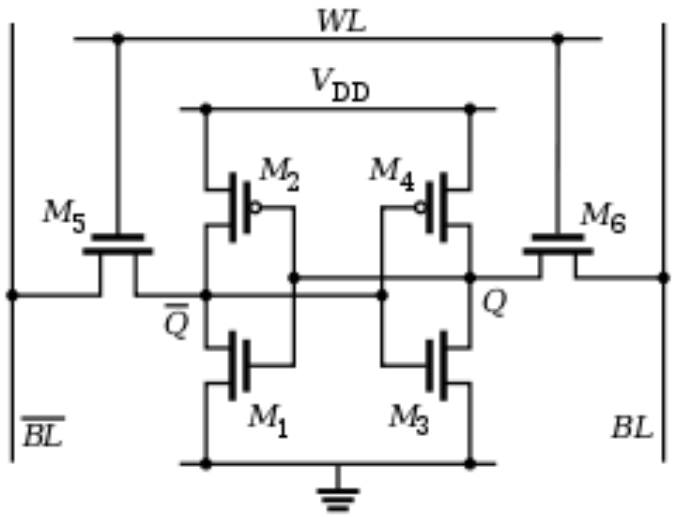


Fig. 1 (a): SRAM Cell

(BL means bit line and L means Write Linde)

Applications of SRAMs:

- ✓ As cache memory.
- ✓ In digital Cameras.
- ✓ In cell Phones.
- ✓ In LCD screens and printers.

Dynamic RAM (DRAM):

DRAM is a type of RAM that stores each bit of data in a memory cell, consisting of a small capacitor and a transistor as shown in Fig. 1 (b). Here information is stored as charge on the capacitors. If capacitor is charged, it represents a 1 state. Otherwise, 0 state.

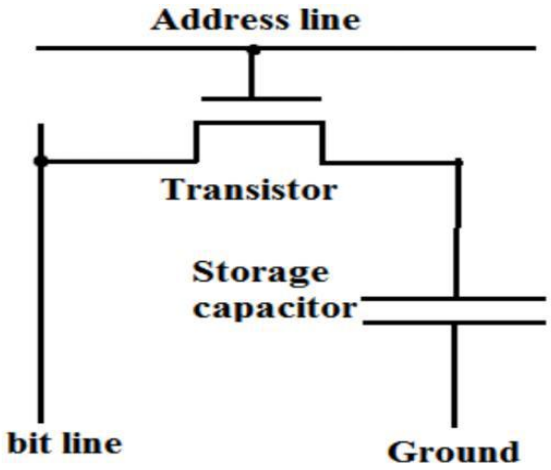


Fig. 1 (b): DRAM Cell

As the capacitor has leakage problem, DRAM is to be refreshed for every 64 milliseconds.

Applications of DRAM: DRAM is widely used in low-cost and high-capacity memory applications. Some of its applications are:

- ✓ Main memory in computers.
- ✓ In PCs, Laptops and handheld computers (PDAs).
- ✓ Digital electronics equipment.
- ✓ Workstations and servers.

3. Non-Volatile RAM (NVRAM):

NVRAM is a RAM with a battery backup. It contains static RAM based memory and a small battery. Battery is used for providing supply to the memory during power supply failure. The memory and battery are packed together in a single package. The life span of NVRAM is expected to be around 10 years.

Example: DS1744

Applications: In computer monitors, printers, etc.

Comparison of SRAM and DRAM:

S. No.	SRAM	DRAM
1	A conventional SRAM uses 6-transistor flipflops. Each flipflop stores one bit of information.	DRAM uses capacitors and very few transistors. Each bit is stored as a charge on a separate capacitor.
2	It has lower access time, i.e., it is faster than DRAM.	It has higher access time, i.e., is slower than SRAM.
3	No need for refreshing.	Needs refreshing for every 64 mSec.
4	Less power consumption.	More Power consumption.
5	SRAM offers low package density. (1 MB to 16 MB)	High Package density. (1 GB to 16 GB)
6	Costly.	Less expensive.
7	Applications of SRAMs: ✓ As cache memory. ✓ In digital Cameras. ✓ In cell Phones. ✓ In LCD screens and printers.	**Applications of DRAM:** DRAM is widely used in low-cost and high-capacity memory applications. Some of its applications are: ✓ Main memory in computers.

		✓ In PCs, Laptops and handheld computers (PDAs). ✓ Digital electronics equipment. ✓ Workstations and servers. .

READ ONLY MEMORY

Read Only Memory (ROM) is a memory in which the stored contents are permanent. A ROM permits only read operation but not write operation. ROM is used to store programs. It is a non-volatile memory. i.e., its contents are not lost when power is turned off or during power failure. Depending on the fabrication, erasing and programming techniques, ROMs are classified into following categories.

(i) Masked Read Only Memory (MROM):

Manufacturer writes the information, based on the users' needs. Once written, the information can't be changed. Thus, frequently used programs, look-up tables, disk drives, etc. are written into MROM. As MROMs are mass-produced, their cost is low.

Applications: Storing fonts for laser printers, Storing sound data in electronic musical instruments, etc.

(ii) Programmable Read Only Memory (PROM): PROM is not pre-programmed by manufacturer. End-user programs this memory using an instrument called PROM Programmer. Programming means writing the contents. PROM is programmed by burning (or blowing) some fuse links. Originally all basic cells in PROM contain 1s. wherever 0s are to be written, fuses are blown at those locations. Once programmed, the contents can't be changed. So, a PROM is also called **One-Time Programmable (OTP) device**.

Applications: Cell phones, digital cameras, PCs, etc.

(iii) Erasable Programmable Read Only Memory (EPROM):

EPROM is also called 'windowed ROM'. Here, contents can be erased and rewritten. EPROM chip has a small quartz window. The window is exposed to UV (ultra violet) light for 20 t0 30 minutes. UV light enters the chip through a quartz window on the chip and erases all the contents of the chip. Then it is rewritten. Even sunlight or fluorescent light can erase the contents. But it takes so many hours.

Drawbacks:

- Erasing process is tedious and time-consuming.

- Here, all the contents of the chip are erased. Partial erasing is not possible.
- For erasing, chip is to be removed from circuit board.

Because of the above drawbacks, EPROM chips are now replaced by EEPROMS and FLASH.

Applications: EPROMS were earlier used to store computer BIOS, electronic gadgets, etc.

(iv) Electrically Erasable Programmable Read Only Memory (EEPROM):

EEPROM can be erased electrically in a few milliseconds and then reprogrammed. Erasing can be possible at byte level. Erasing and reprogramming can be done at board-level. i.e., we need not remove the chip from the circuit board.

EEPROMs are faster and convenient to use. But they are more expensive and have low capacity (a few kilobytes). They are expensive compared to PROMs and EPROMs.

Applications: Used as BIOS and in embedded systems.

(v) FLASH memory:

It is the latest and most popular ROM technology used in present ESs. It combines the re-programmability feature of the EEPROM and the high capacity of standard ROMs. Flash memory is organized as blocks. The erasing of information can be done at block level without affecting the other blocks.

Example: W27C512

Applications of FLASH: Web applications, digital cameras, mobile phones, etc.

MEMORY ACCORDING TO THE TYPE OF INTERFACE

Interfaces connecting the memory and processor/controller may be serial, parallel or serial peripheral interfaces. Parallel interfaces were used in earlier computers for connecting peripherals. Serial interface is commonly used for data memory like EEPROM. These interfaces may be onboard interfaces or external interfaces.

The memory density of a serial memory interface is expressed kilobits, whereas that of a parallel memory interface is expressed in kilobytes.

Note: Communication Interfaces are discussed separately in Unitr-3.

MEMORY SHADOWING

A RAM access is 3 times as fast as ROM access. So, if program is stored in RAM, it gives high execution speed. But RAM is volatile. i.e., its contents are lost if power supply is OFF.

On the other hand, a ROM is non-volatile memory. Its stored contents are permanent. But it gives low execution speed. Memory shadowing technique resolves this execution speed problem.

In computer systems (e.g., PCs) and video systems, a configuration called BIOS (Basic Input Output System) is used to hold ROM. This information is required during boot up. Since it is stored in ROM, it is time-consuming.

Now, manufacturers use Memory shadowing. They put a RAM behind the logical layer of BIOS. This RAM acts as a shadow to the BIOS. While booting, BIOS is copied to shadow RAM, write-protected and the BIOS reading operation is disabled.

Thus, information is accessed from RAM instead of from ROM.

MEMORY SELECTION FOR EMBEDDED SYSTEM

Selection of RAM and ROM depends on the type of ES and the application. Important factors while selecting the RAM and ROM are: -

(i) Need for external memory: We need to check whether the on-chip memory is sufficient or external memory is required. Then we should estimate the memory required.

Examples:

(i) A Windows mobile device needs 64 MB RAM and 128 MB ROM

(ii) In small applications (toys), a microcontroller with less data memory, i.e., a few bytes of internal RAM, Flash and EEPROM (if necessary), are needed. We don't need external memory.

(ii) Memory size: Memory chips come in standard sizes like 512 bytes, 1024 bytes (1 kilobyte), 2048 bytes (2 kilobytes), 4 Kb, 8 Kb, 16 Kb, 32 Kb, 64 Kb, 128 Kb, 256 Kb, 1024 Kb (1 megabyte), etc. If an ES needs 20 Kb, we have to go for 32 Kb, but not 16 Kb.

(iii) Address range supported by the processor: A processor with 16-bit address bus can address a maximum of 2^{16} (= 64 Kb) memory locations. Hence it is meaning less to select a 128 Kb memory chip.

(iv) Sharing of memory: The memory may also be shared by I/O devices and other IC chips.

(v) Word size of the memory: Word size is the number of bits that can be read/written together at a time. 4, 8, 12, 16, 24, 32, etc., are the word sizes supported by memory chips. Word size supported by the memory chip should match with the width of the data bus of the processor/controller.

SENSORS AND ACTUATORS

An ES is in constant interaction with real world. It monitors the changes in the input variables and controls the output variables. For this purpose, it uses sensors and actuators.

Sensors:

A Sensor is a device that detects (or measures) a physical variable and responds to, in its environment. For example, a thermometer is a temperature sensor. In ESs, sensors are used at the input port.

Other examples for sensors:

- Pressure sensors
- Humidity sensors
- Smoke and gas sensors
- Vibration sensors
- IR sensor sensors (They detect people or objects in motion)

Actuators:

To" actuate" means to put something into mechanical action or motion (movement). An actuator is a component that is responsible for moving and controlling a mechanism or system. For example, a windmill actuates a pump. In ESs, actuators are used at the output port.

Other examples for actuators:

- ✓ Electric motor
- ✓ Stepper motor
- ✓ Solenoid
- ✓ Piezoelectric actuator
- ✓ Hydraulic cylinder

I/O COMPONENTS

1. Seven Segment LED
2. Relay
3. Piezo Buzzer
4. Push Button Switch

1. SEVEN SEGMENT LED

It is an output device used for displaying alphanumeric characters. It contains 8 light emitting diode (LED) segments arranged in a special pattern. Out of the them, seven are used for displaying 'alphanumeric characters' and one is used for displaying 'decimal point'. Fig. 2 (a) shows a 7-segment LED display. The 7 LED segments are named A to G and the decimal point LED segment is named as DP.

These segments are lit (ON) to display the numbers and **some** characters.

Examples:

For displaying '**4**', the segments B, C, F and G are lit.

For displaying '**3.**', the segments A, B, C, D, G and DP are lit.

For displaying the letter '**d**', the segments B, C, D, E and G are lit.

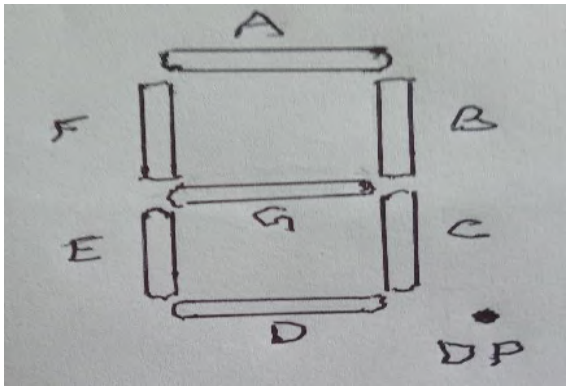


Fig. 2 (a): 7-segment LED Display

These displays are available in two configurations: 'Common anode' and 'Common cathode' configurations as shown in Figs. 2 (b) and 2 (c).

In common anode configuration, all the anodes share a common line. Cathode of each LED segment is connected to the respective port pin interfaced to display.

In common cathode configuration, all the cathodes share a common line. Anode of each LED segment is connected to the port pin interfaced to display.

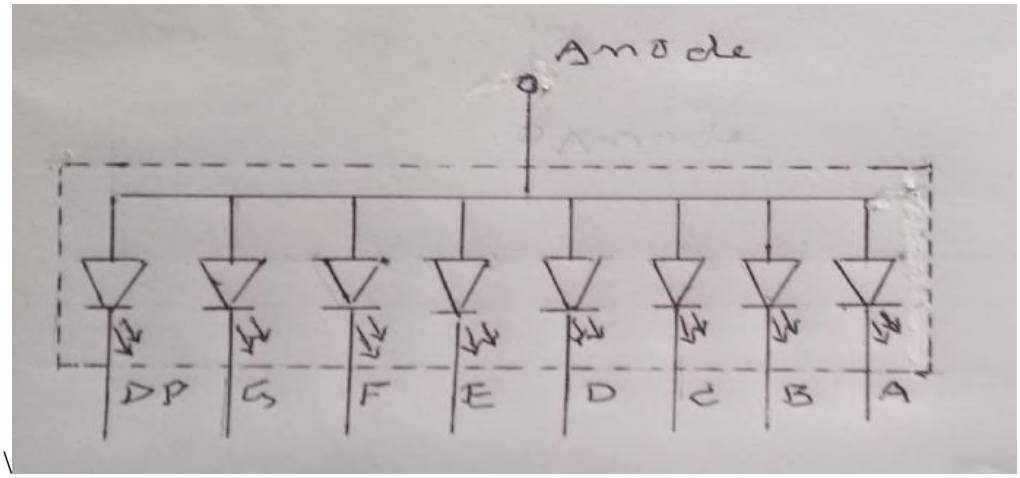
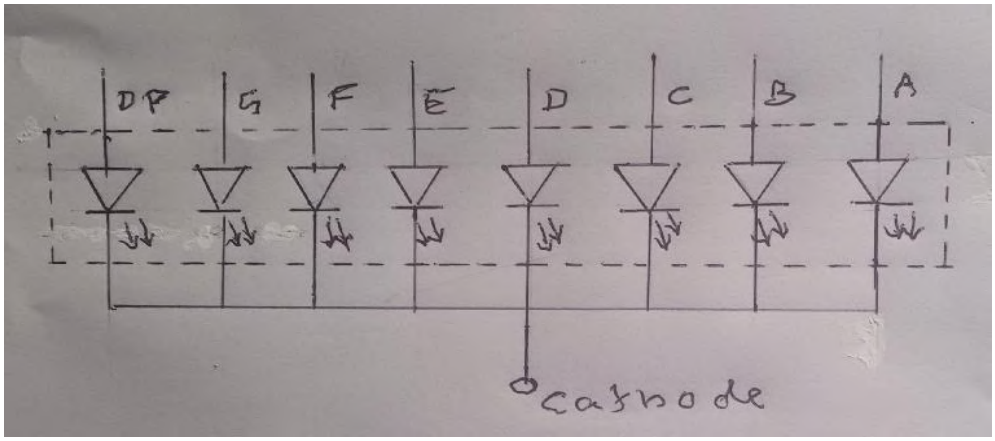


Fig. 2 (b) Common Anode Display



2(c) Common Cathode Display

Applications: It is a popular choice for low-cost embedded applications like -

- Public telephone call monitoring devices.
- Petrol/diesel bunks.
- Point of sales terminals, etc.

2. RELAY

Relay is an actuator. In embedded applications, it is used to select a signal/power path dynamically. The relay unit contains a **relay coil** made up of insulated wire wound on a metal core, and a **metal armature** with one or more contacts.

Relay works on electromagnetic principle. When a voltage is applied to the relay coil, current flows through the coil and generates a magnetic field. This magnetic field attracts the armature core and moves the contact point. The movement of the contact point changes the power/signal path.

Relay Configurations:

Fig. 3 illustrates the widely used relay configurations for embedded applications.

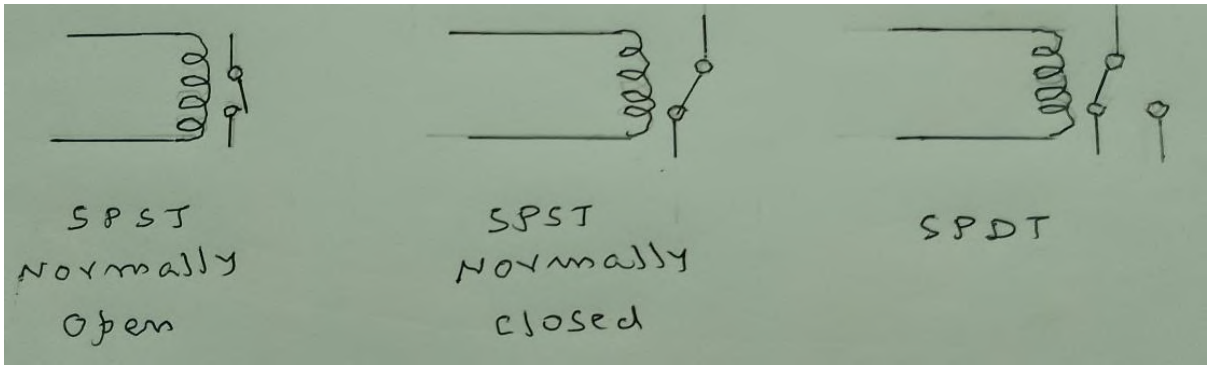


Fig. 3: Relay Configurations

SPST (Single pole single throw) Configuration: It has only one path for information flow. The path is either open or closed in normal condition.

- Normally open SPST relay: The circuit is normally open and it becomes closed when the relay is energised.
- Normally closed SPST relay: The circuit is normally closed and it becomes open when the relay is energised.

SPDT (Single pole Double throw) configuration: There are two paths for information flow and they are selected by energizing or de-energizing the relay.

The relay is normally controlled using a relay driver circuit connected to the port pin of the processor/controller.

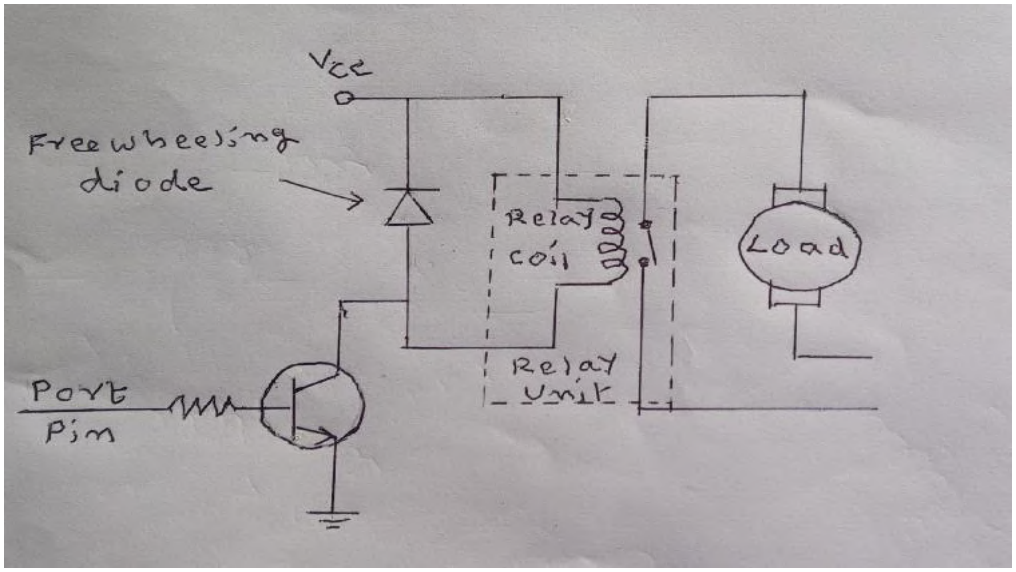


Fig. 4: NPN relay switch circuit

Fig. 4 shows a transistor-based relay driving circuit. *The freewheeling diode protects the relay and transistor from voltage spikes* when the relay coil is de-energized.

Most of the industrial relays are bulky and require high voltage to operate. Special relays called **'Reed relays'** are available for embedded applications which require switching of low voltage DC signals.

Advantages of relay:

- Less power consumption.
- Fast operation
- Used for both ac and dc systems
- Simple, compact and reliable.

Applications:

- Control motor, heaters and lamps.

- Industrial process controllers.
- Traffic control.
- Home appliances.

3. PIEZO BUZZER

A 'piezoelectric buzzer' or '**piezo buzzer**' is a type of electronic device that is used to produce a tone, alarm or sound. It is a low-cost and light weight device with simple construction,

A piezo buzzer contains a piezoelectric diaphragm. When you apply voltage, the diaphragm vibrates and produces sound. A piezo buzzer can be directly interfaced to the port pin of the processor /controller.

Active and Passive Piezo Buzzers: Active piezo buzzers operate on DC voltage. They are often called transducers. Passive piezo buzzers operate on AC voltage.

Self-driving and External-driving piezo buzzers:

- The Self-driving type generates a fixed single tone, when a voltage is applied.
- External driving type can generate different tones. The tone can be varied by applying a variable pulse train to the piezoelectric buzzer.

Applications:

- Fire alarms
- Microwave oven
- Automobile alerts
- In pest repelling devices (Pest is a destructive insect or animal that attacks crops and food).

4. PUSH BUTTON SWITCH

Push button switch is basically a mechanical device. It comes in 2 configurations, namely 'Push to Make' and 'Push to Break'. Break means open and make mean cloe.

Push to Make Configuration: The switch is normally in the open state and it makes a circuit contact when it is pushed or pressed. When push button is released it breaks the circuit connection.

Push to Break Configuration: The switch is normally in the closed state and breaks the circuit contact when it is pushed or pressed. When push button is released it makes the circuit or open.

Examples for push button switch:

- Doorbell
- Calculator buttons
- Keys on a keyboard

In embedded applications, push button is generally used as a 'reset and start switch' and pulse generator. The push button is normally connected to the port pin of the host processor/ controller.

Depending on the way in which the push button is interfaced to the controller, it can generate either a "LOW' pulse or a "HIGH" pulse as shown in Figs. 5 (a) and 5 (b) respectively.

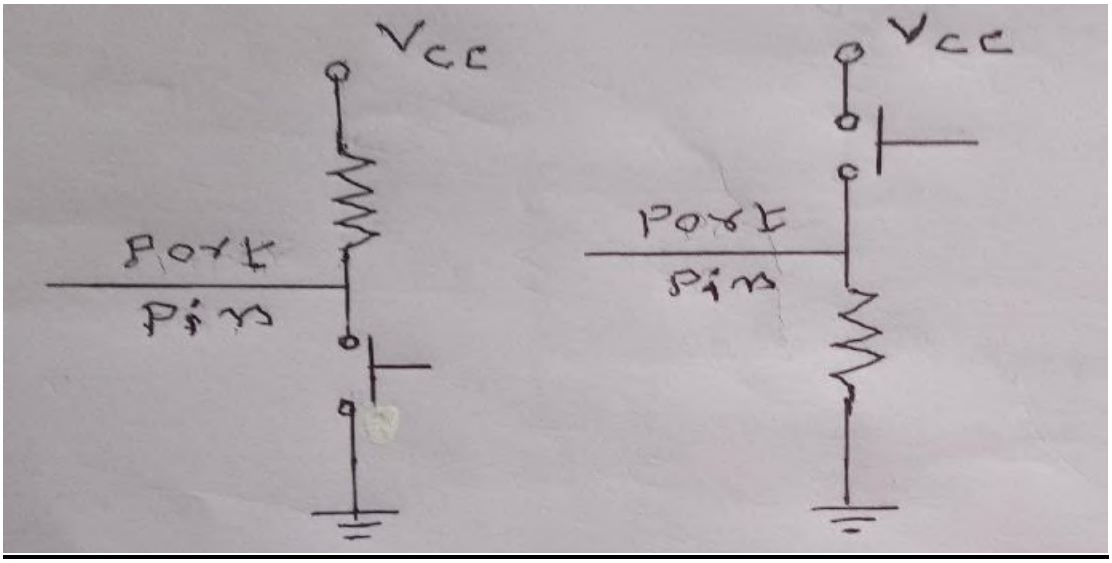


Fig. 5 (a): 'LOW" Pulse Generator **Fig. 5 (b): 'HIGH' Pulse Generator**

OTHER SUB-SYSTEMS

The 'other subsystems' refer to the components/circuits/ICs necessary for proper functioning of the ES.

1. Reset Circuit
2. Brown-out Protection Circuit
3. Oscillator Unit
4. Real-Time Clock
5. Watchdog Timer

1. RESET CIRCUIT

'To reset' means 'to restart' a device or system in a good condition. A Reset circuit helps a microprocessor re-initialize itself and resume its normal operation, whenever an undesirable error occurs. It ensures:

- Proper power supply levels.
- Accurately settled processor-clocks.
- Properly loaded internal registers.

There are 2 types of reset circuits.

(i) An RC based circuit

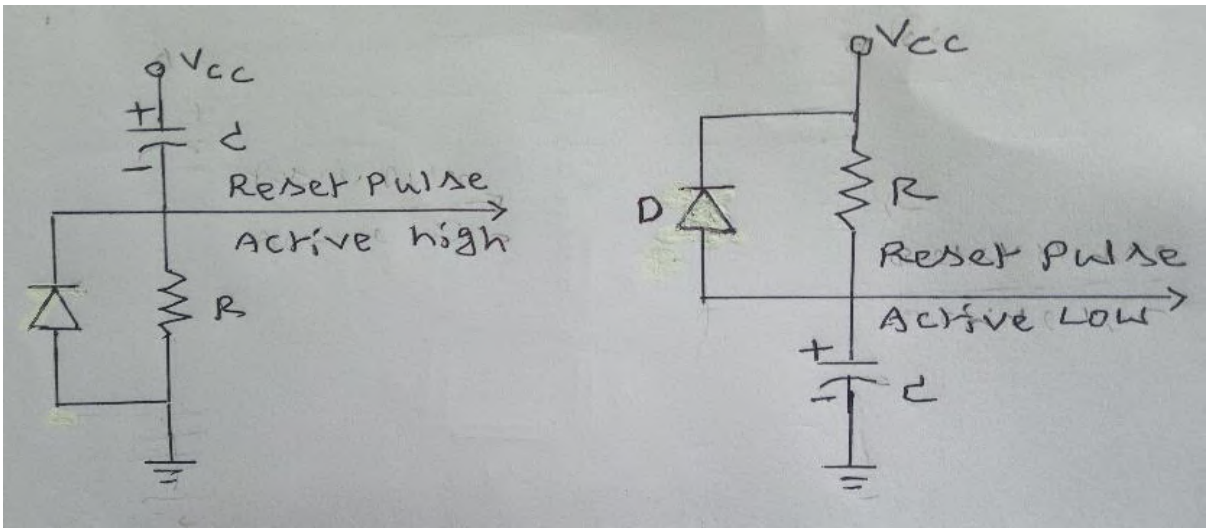


Fig. 6: RC based Reset Circuits

Fig. 6 shows RC based reset circuits. Diode D is **freewheeling diode** used for protection. The reset pulse width can be adjusted by changing the values of R and C.

(ii) A Reset IC: A Reset IC is selected based on the type of reset signal and logic level (e.g., CMOS/TTL) supported by the processor/controller in use.

Some microprocessors/controllers contain built-in internal reset IC. So, they don't require external reset circuit.

2. BROWN-OUT PROTECTION CIRCUIT

A "brown-out" of a microcontroller is a partial and temporary reduction in the power supply voltage below the normal operating level. i.e., the supply voltage drops from its normal level to a lower voltage and then returns to original value.

Many microcontrollers have a protection circuit which detects such situation. When the supply voltage goes below a threshold value the circuit resets the device. This ensures proper start-up after the normal condition is resumed.

It is essential for battery powered devices since there are greater chances for the battery voltage to drop below the required limit.

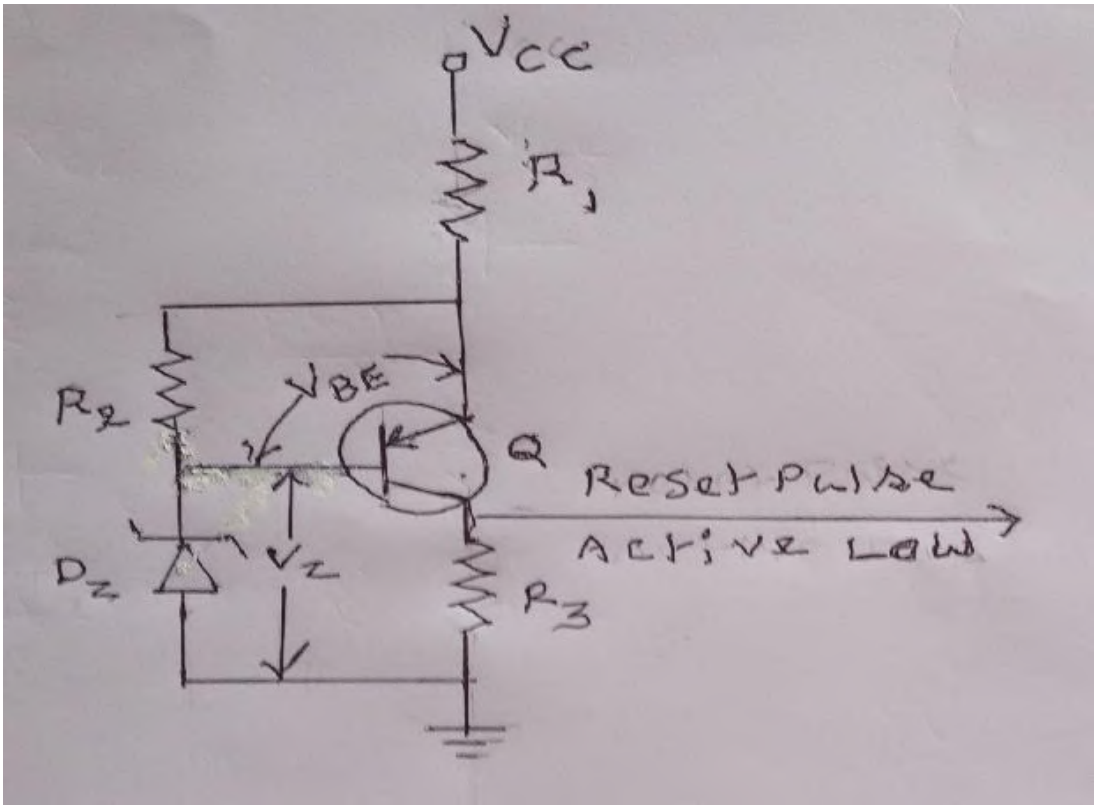


Fig. 7: Brown-out protection circuit with active low output

Fig. 7 shows a Brown-out protection circuit. Here D_Z is Zener diode and Q is a pnp transistor. Let V_{BE} be the base-emitter voltage of Q and V_Z be the Zener voltage.

When supply voltage $> (V_{BE} + V_Z)$, the transistor conducts.

When supply voltage $< (V_{BE} + V_Z)$, the transistor stops conducting.

For setting the lower threshold of V_{CC}, Zener diode with proper voltage is selected.

R_1, R_2 and R_3 are selected, based on the absolute maximum voltage and current ratings.

Example: DS1232

3. OSCILLATORS

Oscillator circuit of ES generates the clock pulses for synchronizing the internal and external operations during program execution. There are two types of oscillators.

On-chip oscillators: Some microcontrollers (8051) have on chip oscillator circuit and require only a quartz crystal or ceramic resonator for producing necessary clock signals.

External oscillators: Quartz crystal oscillators are available as IC chips. If built-in oscillator unit is not present, they can be used for generating clock pulses.

Fig. 8 (a) and Fig. 8 (b) show on-chip oscillator and external oscillators respectively.

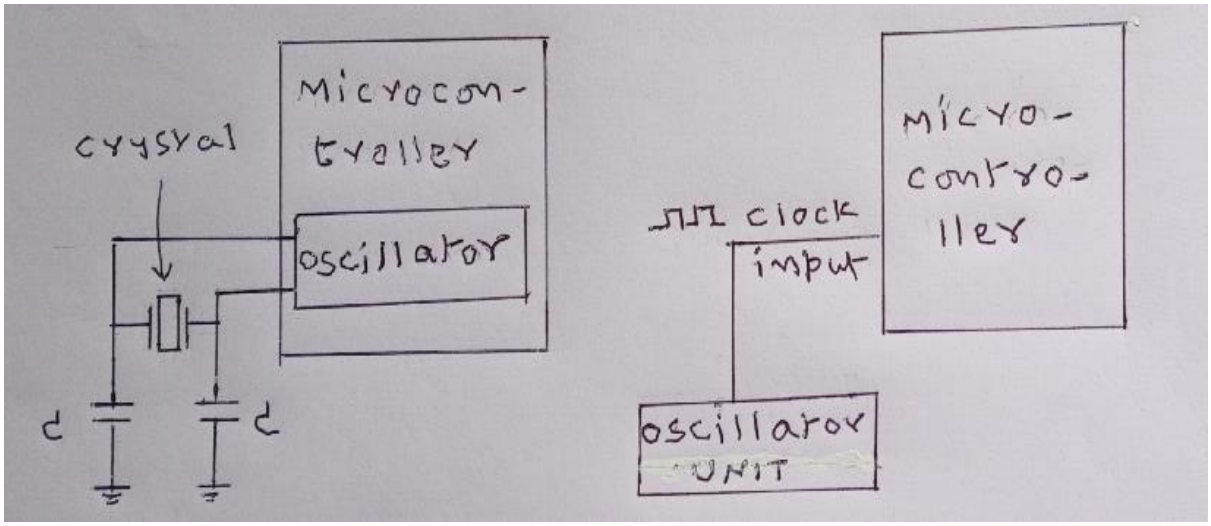


Fig. 8 (a): On-Chip Oscillator **Fig. 8 (b) External Oscillator**

Important parameters of oscillators:

- Clock frequency: The execution speed is directly proportional to clock frequency. But we should not blindly increase the clock frequency to increase execution speed. There is an upper threshold of clock frequency for logical devices on the chip. Beyond that, the system becomes unstable and non-functional.
- Power consumption: Power consumption of system is directly proportional to clock frequency. If we increase the clock frequency, power consumption will also increase.
- Accuracy: The accuracy of program execution depends on the accuracy of clock signal. The accuracy of quartz oscillator or ceramic resonator is expressed in +/- ppm (parts per million).

4. REAL-TIME CLOCK

Real-Time Clock is an IC used in PCs, ESs, servers and other electronic systems, to keep track of time and date. It counts hours, minutes, seconds. It also counts days, months and years. Thus, it acts like a clock as well as calendar.

Example: *DS12885 (From Maxim)*

Important features of (RTC):

- It mainly consists of a controller, oscillator, and an embedded quartz crystal.
- It ensures that all the processes occurring in the system are properly synchronized.
- During power failure or low power state, it works on battery backup.
- A system clock has same purpose as RTC. The main differences between an RTC and the system clock are: -

- RTC runs even when the system is off or in a low power state
- RTC is more accurate than system clock.

Advantages of RTC:

- More accurate than other methods (Setting a timer, etc.)
- Frees the main system from time-critical tasks.
- Low power consumption.
- Good frequency stability.

5. WATCHDOG TIMER (WDT)

Watchdog timer (WDT) is a timing device set for a preset time interval. If a task is not completed successfully in that time interval, the device will generate a timeout signal to reset the processor. It may be on the CPU chip or external to the processor. A WDT may be an up counting or down-counting timer.

(a) Up counting WDT:

Let the WDT is in enabled state. Before starting executing a piece of code, the firmware writes a 0 in its time register and starts counting. It increments a free running counter for each clock pulse until the count reaches highest preset value. If the firmware execution does not complete due to malfunctioning, it generates a reset signal to reset the processor. If the firmware execution completes before the pre-set time, its count is reset to 0.

(b) Down counting WDT:

Its operation is similar to up counting timer. But here highest value is written in beginning instead of 0. Each clock pulse decrements the counter until the count reaches 0.

External WDT circuit:

Instead of the firmware-based writing, External WDT uses hardware logic for enabling/disabling, resetting the watchdog count, etc. The microprocessor supervisor IC (e.g., DS1232) integrates a hardware watchdog timer in it.

WDTs in modern systems: In modern systems running on embedded OSs, the watchdog timer can be implemented in a different way. When a watchdog timeout occurs, an interrupt is generated instead of resetting the processor. The interrupt handler for this interrupt handles the situation in an appropriate way.

UNIT-3

COMMUNICATION INTERFACE

Contents: Onboard Communication Interfaces-I2C, SPI, CAN, Parallel interface; External communication interfaces-RS232 and RS485, USB, Infrared, Bluetooth, Wi-Fi, ZigBee, GPRS, GSM.

INTRODUCTION

A communication interface (CI) is essential for an ES to communicate with its subsystems or with external world. There are two types of CIs for an ES: Onboard CIs and External CIs.

Onboard Communication Interfaces:

Communication channels or buses used to connect various ICs and peripherals within the ES are called Onboard Communication Interfaces. They are also called "Device/Board Level Communication Interfaces".

Examples:

1. Inter-Integrated Circuit (I2C or I^2C) Bus
2. Serial Peripheral Interface (SPI) Bus
3. Controlled Area Network (CAN)
4. Parallel Interface

External Communication Interfaces:

They can be either wired media or wireless media and they can be a serial or parallel interfaces. They are also called "Product Level Communication Interfaces".

Examples:

1. RS-232C & RS-485
2. Universal Serial Bus (USB)
3. Infrared
4. Bluetooth
5. Wi-Fi
6. ZigBee
7. GSM
8. General Packet Radio Service (GPRS)

ONBOARD COMMUNICATION INTERFACES

1. INTER-INTEGRATED CIRCUIT (I2C)

I2C is a synchronous, bi-directional, half duplex, two-wire interface bus used for serial communication.

An I2C is also called **IIC** or **I^2C**. it was designed by Philips Semiconductors in 1982. Originally it was used for connecting a μP/μC system and the peripheral ICs in **TV sets**. Now it is a **widely used protocol** for short-distance communication.

I2C is used to connect microcontrollers, EEPROMs, I/O interfaces, and other peripheral devices in an embedded system. **A microcontroller is often used as the 'master device', and peripheral devices are used as 'slave devices'.**

At the physical level it consists of two wires: SCL (clock line) and SDA (data line). SCL is responsible generates clock pulses and SDA transmits the serial data across the devices.

Device connected to I2C bus acts either as 'Master' device or as 'Slave' device. I2C supports multi masters on the same bus. A master/slave can act either as a transmitter or a receiver. Master generates the clock pulses and also controls the communication. It initiates/terminates data transfer, sends data and generates the necessary synchronizing clock pulses. Slave devices wait for the commands from the master and respond upon receiving the commands.

Interface diagram in Fig. 1 illustrates the connection of master and slave devices on I2C bus.

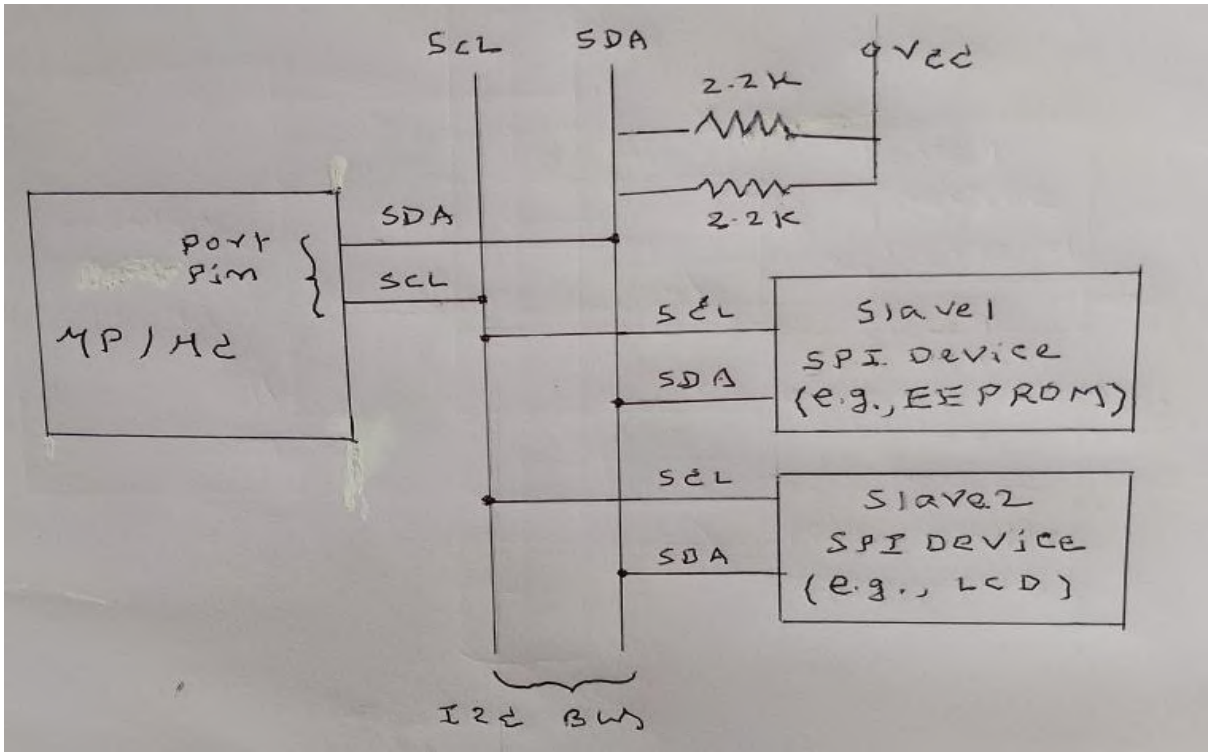


Fig. 1: I2C Bus Interfacing

Sequence of operations for communicating with an I2C slave device:

1. First, master makes SCL = high and SDA = low. It is the 'START' condition for data transfer.
2. Master sends 7-bit or 10-bit address of the slave on SDA line, and clock pulses on SCL line for synchronization. The
3. Master sends data on the bus. It sends MSB first. This data is valid when clock pulse is 'HIGH'.
4. Master sends Read bit '1' or Write bit '0' as per the need.
5. Slave receives it and sends acknowledge bit '1' on SDA line.
6. Master receives the acknowledge bit.
7. If write operation is requested, master sends 8-bit data to slave on SDA line. slave sends acknowledgement bit to master.
8. If read operation is requested, slave sends data to master on SDA line. Master sends acknowledgement.
9. Master terminates the transfer by making **SDA line high when SCL is high**. It is STOP operation.

Data rates supported by I2C:

- Standard mode: Data rate up to 100 Kbps
- Fast Mode: up to 400 Kbps
- High speed mode: up to 3.4 Mbps

Applications of I2C

- Reading some memory ICs.
- Accessing DACs and ADCs.
- Transmitting and controlling user-directed actions.
- Reading hardware sensors.
- Communicating with multiple microcontrollers.

2. SERIAL PERIPHERAL INTERFACE (SPI) BUS

SPI bus is a synchronous, bi-directional, full duplex, four-wire used for serial communication.

The concept of SPI was introduced by Motorola. SPI is a single master multi-slave system. It is possible to have more than one master. But only one master device should be active at any given point of time. SPI requires 4 signal lines for communication. They are:

- **Master Out Slave In (MOSI):** It carries the data from master to slave device.

  It is also known as Slave Input/Slave Data In (SI/SDI).

- **Serial Clock (SCLK):** It carries the clock signals.
- **Master In Slave Out (MISO):** It carries the data from slave to master device.

  It is also known as Slave Output/Slave Data Out (SO/SDO).

- **Slave Select (SS):** It is an active low signal used to select the slave device.

The bus interface diagram shown in Fig. 2 illustrates the connection of master and slave devices on the SPI bus.

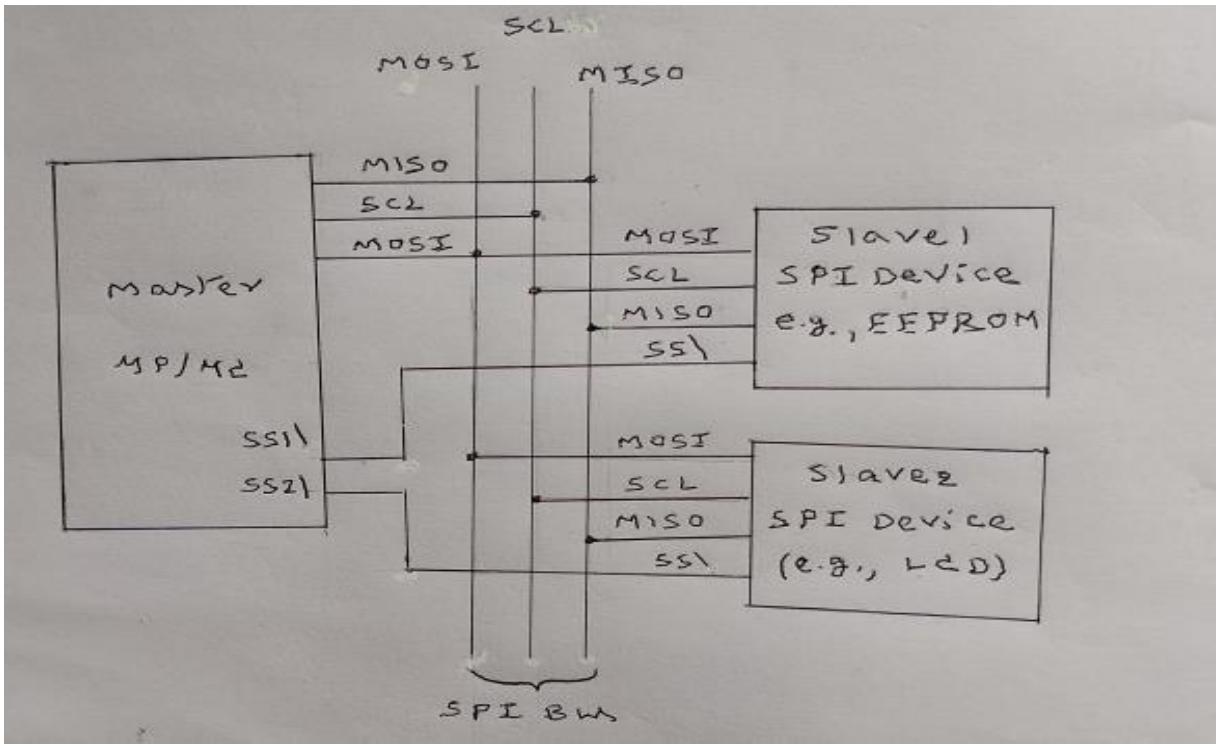


Fig. 2: SPI Bus Interfacing

The master device generates clock signals and selects the required slave device by making its select signal 'LOW'. The data out line MISO of all the slave devices floats at high impedance state, when not selected.

SPI devices contain the following registers for holding these configurations.

- Serial peripheral control register' to hold configuration parameters like master/slave selection, baudrate, clock signal control, etc.

- Status register to holds the status of various conditions for transmission and reception.

Working Principle:

- SPI works on the principle of 'Shift Register' (SR). The master and slave devices contain a special SR for transmitting and receiving data. The size of the shift register is device dependent. Normally it is a multiple of 8-bits.
- During transmission from the master to slave, the data in SR of master is shifted out to the MOSI pin of the slave device. At the same time the shifted-out data bit from the SR of slave enters the SR of the master device through MISO pin.
- In summary the SRs of master and slave devices form a circular buffer. Some devices are configurable to select LSB/MSB as first bit to send.

When compared to I2C, SPI bus is most suitable for applications requiring transfer of data in streams. The only limitation is SPI doesn't support an acknowledgement mechanism.

3. CONTROLLED AREA NETWORK (CAN)

CAN is a serial communication protocol used over a pair of wires. It is used in many real-time applications. It was developed by Robert Bosch to provide communication among various electronic components of vehicles.

Later it was extended to automation and industrial applications. CAN specification doesn't specify the layout and structure of the physical bus. But a device connected to CAN bus is able to transmit on the physical bus.

CAN protocol defines data packet format and transmission rules ---

- ✓ To prioritize messages
- ✓ Guarantee latency times
- ✓ Handle transmission errors
- ✓ Retransmit corrupted messages
- ✓ Distinguish between a permanent failure of a node versus temporary errors

Important characteristics of CAN Protocol

- Used in cars, buses, trucks and aircrafts in (i) Safety systems like airbag control (ii) Antilock Brake System (ABS) (iii) Navigational systems like GPS.
- Real-time support.
- Offers medium speed (up to 125 Kbps) and high speed (up to 1 Mbps) data transfer.

- Has 11-bit addresses
- Error handling mechanism

Error correction in CAN:

Error correction is typically done by a 'retransmission and acknowledgement' protocol.

Here transmitter sends a data packet and expects to receive an acknowledgement from the receiver indicating that transmission is correct. If an acknowledgement is not received in a specified time, the transmitter retransmits the data packet and waits for a second acknowledgement.

Applications of CAN:

- Used in cars, buses, trucks and aircrafts in (i) Safety systems like airbag control (ii) Antilock Brake System (ABS) (iii) Navigational systems like GPS.
- Elevator Controllers
- Photo Copiers
- Medical instruments
- Production line control systems

4. PARALLEL INTERFACE

Parallel Interface is used for communicating with peripheral devices which are memory mapped to the host processor. Fig. 3 illustrates the interfacing of devices using parallel interface.

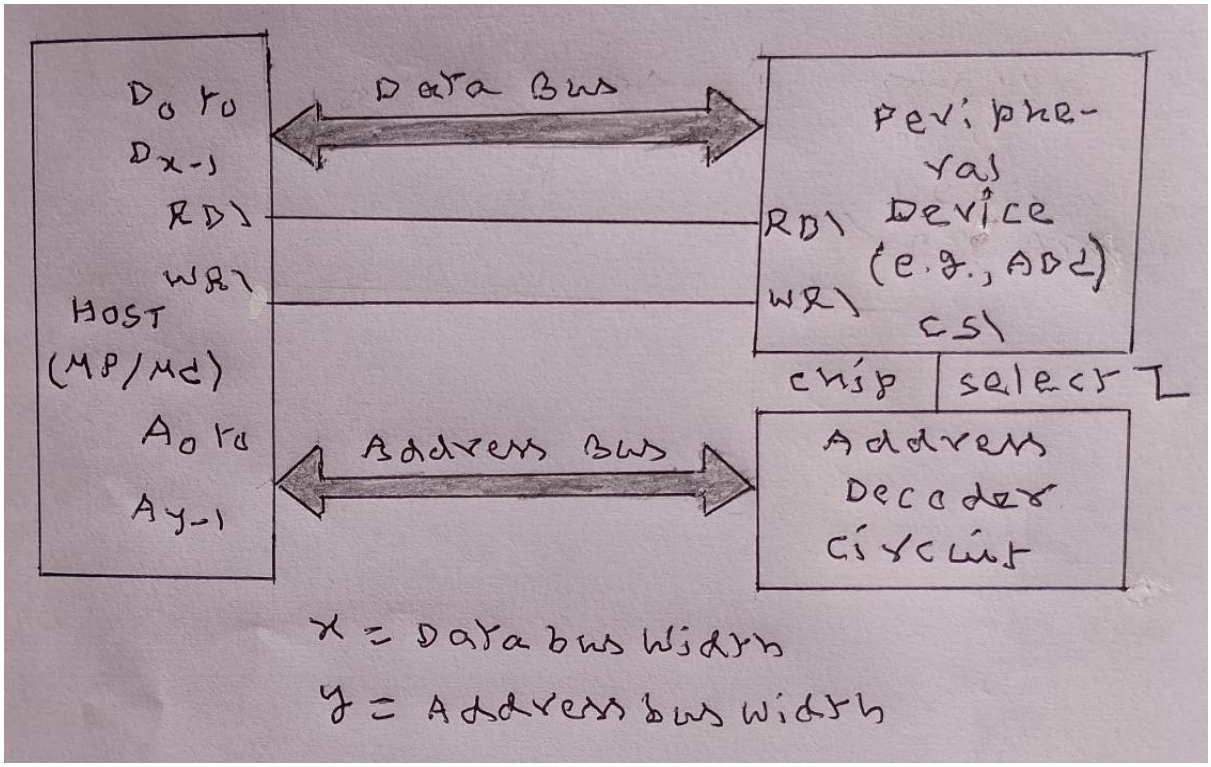


Fig. 3: Parallel Interface bus

Important Features of Parallel Interface bus:

- Host processor has a parallel bus and control over read/write signals.
- Communication is controlled by ‘control signal interface’ between host and device.
- Read’ and ‘Write’ signals control the direction of data transfer It also provides control signals.
- Each device connected to the processor is assigned a range of addresses.
- When the address selected is in this range, a decoder circuit activates the chip select line. Thus, the device becomes active.
- Here, processor initiates the parallel communication. If device wants to initiate communication, it sends an interrupt signal to the processor.
- Parallel data communication offers highest speed for data transfer.
- Width of the parallel interface can be 4-bit, 8-bit, 16-bit, 32-bit or 64-bits. It must match with the width of the data bus of the processor

EXTERNAL INTERFACES

1. RS-232 C & RS-485

RS-232 C stands for Recommended Standard number 232, revision C. RS-232 and RS-232C are almost same. The names are used interchangeably.

Important features of RS-232:

- RS-232 is a legacy, full duplex, wired and communication interface.
- It extends the UART communication signals for external data communication.
- It was developed by the Electronic Industries Alliance (EIA) in early 1960s.
- In addition to 'Transmit' and Receiver' signal lines, RS-232 interface defines various handshaking signals (e.g., RTS – Request to Send, CTS – Clear to Send) and control signals for communication.
- RS-232 supports 2 different types of connectors, namely, DB-9 (9-Pin connector) and DB-25 (25-Pin connector), as shown in Fig. 4 (a) and 4 (b).

Fig. 4 (a): DB-9Connector Interface

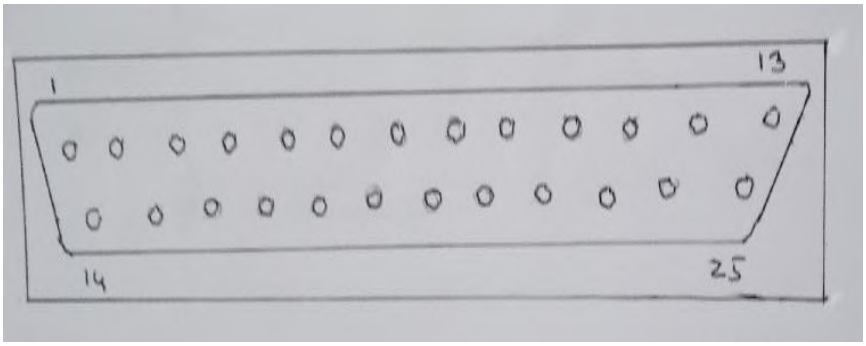


Fig.4 (b): DB-25 Connector Interface

- It supports data rates up to 1 Mbps and distance up to 50 ft.
- It supports only point-to-point communication. It is not suitable for multipoint communication.

Applications of RS-232C:

- Most PCs had RS232 compatible serial ports. They were used to connect peripherals such as keyboards, mice, and printers to the computers.
- It was most popular during the olden days before the advent of Bluetooth, USB, etc. It is still popular in some legacy applications.

RS-485:

- ✓ It is also called 'TIA-485' or 'EIA-485'.
- ✓ RS-485 standard allows for long cabling distances in electrically noisy environments.
- ✓ Uses balanced signalling.
- ✓ One transmitter can be connected up to 32 receiving devices on the bus.

Applications of RS-485:

- ✓ Defining the electrical characteristics of drivers and receivers in serial communications systems.
- ✓ Building automation.
- ✓ Interconnecting security control panels and devices.
- ✓ Video surveillance systems.

Comparison of RS-232 and RS-485:

Feature	**RS-232 C**	**RS-485**
1. Signalling	Single ended signalling	Balanced signalling
2. Type of communication	Point to point	Multipoint
3. Mode of operation	Simplex or full duplex	Simplex or half duplex
4. Max. Cable length	50 feet	4000 feet
5, Max. data rate	1 Mbps	10 Mbps

2. USB

USB stands for Universal Serial Bus. It is a wired high-speed bus used for serial data communication. It was released in 1995 by the USB core group members organizations. "USB.ORG" is the standards body for defining and controlling the standards for USB communication.

Important features of USB:

(i) Topology and hubs:

- It follows a tired-star topology with a USB host at the center and one or more peripheral devices connected to it. In place of any peripheral device, we can connect another USB which supports some more peripherals. This type of connection is called a hub.
- Host and peripherals are connected by using cables.
- USB cable supports a distance up to 5 meters.
- USB transmits data in packets with standard data format.

(ii) USB Host Controller (Interface):

The USB host controller is an interface which:

- Manages and controls driver software
- Manages bandwidth required by peripherals connected to the bus.
- Allocates electrical power to the USB devices. So, we don't need AC power box for many devices.
- Packetizes and formats data.

(iii) PID and VID:

Each USB device contains a Product ID (PID) and a Vendor ID (VID), which are used for loading the device drivers. These IDs are embedded into the USB chip by the USB device manufacturer.

(iv) Types of data transfer supported by USB:

- **Control transfer:** Used to query, configure and issue commands to the USB device.
- **Bulk transfer:** Used to send a block of data to a device (e.g., transferring data to a printer). It supports error checking and correction.
- **Isochronous data transfer:** Used for real-time data communication, where data is transmitted as streams (e.g., frames in TV). It doesn't support error checking and retransmission of data.
- **Interrupt transfer:** Used for transferring small amounts of data (e.g.: data from mouse and keyboard).

(v) Data rates supported by USB:

- USB 1.0/Low-Speed: 1.5 Megabits per second (Mbps)
- USB 1.1/Full-Speed: 12 Mbps.
- USB 2.0/Hi-Speed: 480 Mbps.
- USB 3.0/SuperSpeed: 5 Gbps.

- USB 3.1/SuperSpeed: 10 Gbps.

3. INFRARED (IrDA)

- IrDA stands for Infrared Data Association. It provides specifications for a set of protocols for wireless infrared (IR) communications.
- Infrared technology is a serial, half duplex, wireless technology for data communication between devices.
- Infrared waves are used for transmitting the data.
- IrDA standards have been used to install several low-cost, short-range communication systems in laptops, printers, handheld PCs, and PDAs.
- The IrDA protocol contains Physical Layer, Media Access Control (MAC) and Logical Link Control (LLC). The physical layer defines the physical characteristics of communication like range, data rate, power, etc.
- It supports point-to-point and multipoint communications using line of sight propagation.
- Its typical communication range is 10 cm to 1m. The range can be increased by increasing the transmitting power of the IR device.
- It supports data rates ranging from 9600 bit/seconds to 16 Mbps.
- An Infrared LED is used as transmitting source, while a photodiode acts as receiver.
- Sometimes same device can be used as transmitter and receiver (e.g.: Mobile phone). Then such device is called a **transceiver**.
- Certain devices always require unidirectional communication. So, they have separate transmitter and receiver. For example, in TV remote control device, the remote-control device contains the transmitter unit and TV contains the receiver unit.

Advantages of IrDA:

- ➢ Inexpensive, secure and fast.
- ➢ Easy to use the devices.
- ➢ Less power consumption.
- ➢ Good bandwidth.

Disadvantages:

- ✓ Uses line of sight propagation.
- ✓ Devices cannot be move around while transmission is in progress.

- ✓ Not suitable for long distance applications.

Applications of IrDA:

- IrDA is in use from the olden days of communication. The remote control of your TV or VCD player use IrDA.
- It was the transmission channel in mobile phones before Bluetooth's existence. Even now, most of the mobile phone devices support IrDA.
- Popularly used in low-cost, short-range communication systems in laptops, printers, handheld PCs, and PDAs.

4. BLUETOOTH

- It is a low-cost wireless technology used for data and voice communications.
- It operates at 2.4 GB and supports a data rate up to 1Mbps.
- It supports only a range of around 10 meters (30 feet).
- A Bluetooth device can function either as a master or as a slave.
- Each Bluetooth device will have a 48-bit unique address.
- Bluetooth supports point-to-point and multipoint communication.
- Bluetooth communication follows packet-based data transfer and uses Frequency Hopping Spread Spectrum (FHSS) technique for communication.
- **Piconet:** When a network is formed with one master and many slaves, it is called Piconet. A Piconet consists of one master and a maximum of 7 slave devices.
- **Scatternet:** Piconets can be combined to form a network called a scatternet. A slave in one piconet may act as the master in another piconet.

Advantages of Bluetooth:

- ➢ Used for voice and data transfer.
- ➢ No interference from other wireless devices.
- ➢ Low power consumption.
- ➢ Easily upgradeable.
- ➢ Relatively low cost.

Disadvantages:

- ✓ It can sometimes lose its connection.

- ✓ It has low bandwidth as compared to Wi-Fi.
- ✓ Lower bandwidth.
- ✓ Due to security issues, it can be hacked.

Applications of Bluetooth: Bluetooth is popularly used in:

- Laptops, notebooks and PDAs
- Printers
- Wireless mice and keyboards
- Mobile phones
- Wireless headsets
- Hearing aids
- Fitness devices (Apple watches)
- Video games and toys
- MP3 players

5. WI-FI

Wi-Fi stands for wireless fidelity. It is a wireless technology used to connect computers, laptops, smartphones and other devices to the internet. Wi-Fi follows the **IEEE 802.11 standard.**

Important Features of Wi-Fi:

- It is intended for network communication and it supports Internet Protocol (IP) based communication, where each device is identified by a unique network address called 'IP address'.
- Wi-Fi based communications require an intermediate agent called Wi-Fi router/Wireless access point (WAP) to manage the communications.
- Wi-Fi router is responsible for
    - ➢ Restricting the access to a network
    - ➢ Assigning IP address to a device on the network
    - ➢ Routing data packets to the intended devices on the network.

- Wi-Fi enabled devices contain a wireless adaptor for transmitting and receiving data in the form of radio signals through an antenna. The hardware part of it is known as Wi-Fi Radio.
- Wi-Fi operates at 2.4 GHz or 5 GHz of radio spectrum and co-exists with other ISM band devices like Bluetooth.
- Wi-Fi supports data rates ranging from 1 Mbps to 150 Mbps and access/modulation method.
- Depending on the type of antenna and usage location (indoor/outdoor), Wi-Fi offers a range of 100 to 300 feet.
- When its Wi-Fi radio is turned ON, Wi-Fi device searches Wi-Fi network in its vicinity and lists out the Service Set Identifier (SSID) of the available networks. If the network is security enabled, a password may be required to connect to a particular SSID.
- Wi-Fi employs different security mechanisms like Wired Equivalency Privacy (WEP), Wireless Protection Access (WPA), etc., for securing the data communication.

Applications of Wi-Fi:

- Business applications
- Mobile applications
- Computer applications
- Automotive applications
- Video conference

6. ZIGBEE

ZigBee is a low-cost and low-power wireless communication technology based on a simple protocol. It has been developed for Wireless Personal Area Networks (WPANs).

It uses mesh network topology, where every device in the network is connected to every other device. Each device acts as a router and it increases the reliability of the entire network.

ZigBee device falls under anyone of the following device categories.

i. A ZigBee Coordinator (ZC) Which acts as the root of the ZigBee Network.
ii. A ZigBee Router (ZR) for passing information from a device to another device or another ‘ZC’.
iii. ZigBee End Device (ZED) that contains ZigBee functionality for data communication. It can talk only with a ZR or ZC.

Fig. 5 shows a ZigBee network model.

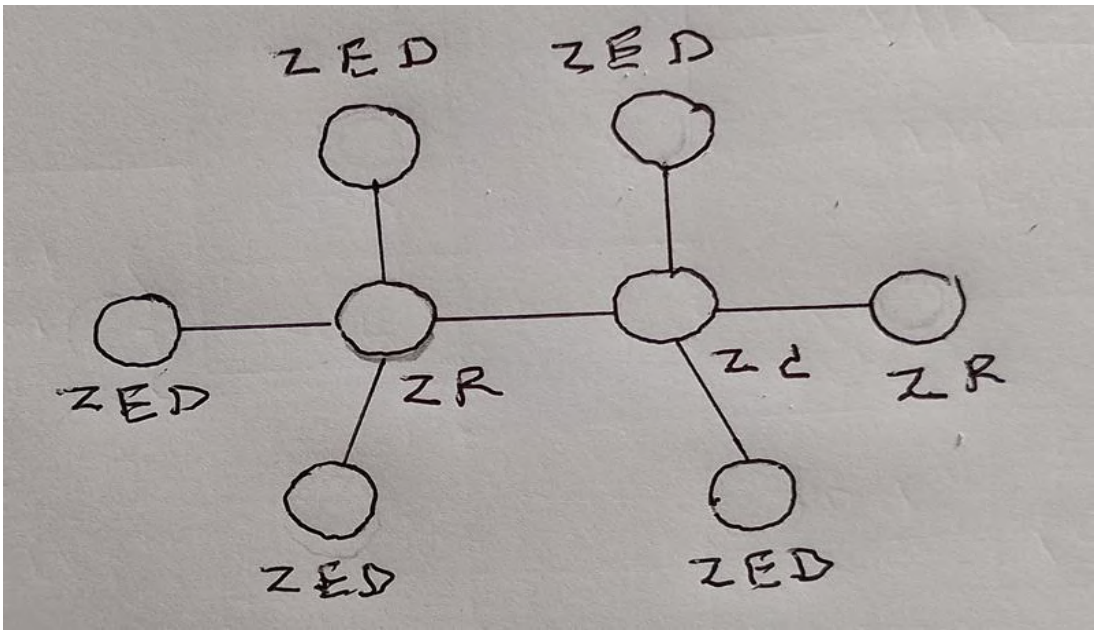


Fig. 5: ZigBee network model

ZigBee alliance (www.zigbee.org) is the monitoring organization for development, standardization and certification of ZigBee standard.

Specifications of ZigBee:

- Standard: IEEE 802.15.4
- Frequency: It operates worldwide in the unlicensed bands-868 to 869.6 MHz, 902 to 928 MHz, and 2.4 to 2.484 GHz. For home automation, 2.4 GHz is used.
- Operating Range: Up to 100 meters
- Data Rate; 20 kbps to 250 kbps

Advantages of ZigBee

- ✓ Low-cost.
- ✓ Low power consumption.
- ✓ High efficiency (due to long battery life).
- ✓ Simple installation, monitoring and control

Disadvantages of ZigBee:

- ✓ Prone to network interferences and so less security
- ✓ Short range communication.
- ✓ Low data rate compared to Bluetooth and Wi-Fi.
- ✓ Implementation is costly.

Applications of ZigBee:

- **Home automation:** Controlling lighting, heating, ventilation, air conditioning and electronic appliances
- **Security and surveillance:** Smoke detectors, water and gas leakage detectors, intruder alarms, fire alarms, etc. in home/office.
- **Industrial automation and control:** Automatic meter reading, energy management, security over power theft, remote temperature monitoring
- **Medicine:** Remote patient monitoring, medical data collection, healthcare devices
- **Other applications:** Fitness trackers, electronic shopping tags, RFIDs and asset tracking.

7. GSM

The concept of GSM was conceived in Bell Laboratories in early 1970s. GSM stands for **Global System for Mobile Communication**. GSM is a digital cellular 2G technology used for transmitting voice and data over mobile network. It also provides roaming service, i.e., your GSM phone number can be used in another GSM network. GSM is also the name of a standardization group to create a common European mobile telephone standard.

Important features of GSM:

- **Type of switching:** GSN uses a circuit-switched network.
- **Frequencies:** It operates in the mobile communication bands 900 MHz and 1800 MHz.
- **Access methods:** Combines TDMA and FDMA.
- **Modulation:** Gaussian Minimum Shift Keying (GMSK) modulation
- **Data rate:** Up to 9.6 Kbps
- **Channel spacing:** 200 KHz
- **Compatibility:** It is compatible with ISDN and other telephone company services.
- **GSM devices:** Mobile telephones, PCMCIA cards, embedded radio modules, and external radio modems.

GSM Services:

- Teleservices: Telephony, Videotext, Facsimile and SMS
- Data services: Send and receive data through a GSM phone.
- Supplementary Services: Conferencing, Call waiting, Call Holding, Call Forwarding, Number Identification.

Advantages of GSM:

- Worldwide connectivity and extensive coverage.
- GSM signals don't have any deterioration.
- No roaming charges on international calls.
- The mobile phone works based on the SIM card. Thus, user can easily change his phone.
- It can be easily integrated with other wireless technologies such as CDMA and LTE.
- Ability to use repeaters.
- More talk time due to the pulse nature of transmission.

Disadvantages of GSM:

- It has the problem of interference (e.g., Interference in hearing aids). As a result, many locations, such as hospitals, airports and petrol pumps require cell phones to be turned off.
- Needs repeaters to increase coverage.
- Low data rate (9.6 Kbps).
- No end-to-end encryption of user data.
- Several incompatibilities within the GSM standards.

8. GENERAL PACKET RADIO SERVICE (GPRS)

GPRS is a packet-based wireless communication technology used in 2G and 3G cellular networks. In this, data is divided into packets at the transmitting end and transmitted across the mobile network like GSM. At the receiver end, the packets are received and assembled to recover the original information.

GSM vs GPRS: GSM is a digital cellular **2G technology** used for transmitting mobile voice and data services globally. GPRS is basically an up-gradation of the GSM. It uses Time Division Multiple Access (TDMA) to share the channel.

Important Features of GPRS:

- GPRS technology is faster than GSM (Data rate of GSM is about 9.6 Kbps, while that of GPRS is about 14.4 to 115.2 Kbps).
- GPRS provides 124 channels with 200 KHz spacing.
- It provides an uninterrupted connectivity to the internet for mobile phones and computers. Moreover, services will be easy and quick to access.
- It provides voice and data services globally.

- Eight different users could share the same uplink and downlink channels. Each is divided into eight time slots. These time slots are dynamically allocated to various users as and when needed.
- GPRS is also known as GSM-IP (Global-System Mobile Communications-Internet Protocol). It supports Internet Protocol (IP), Point to Point Protocol (PPP) and X.25 protocols for communication.
- It uses one or more frequency-bands the radio supports (850, 900, 1800, 1900 MHz).
- The GPRS specifications are written by the European Telecommunications Standard Institute (ETSI).
- GPRS is an old technology and it is being replaced by new techniques like EDGE (Enhanced data rates for GSM evolution), HSDPA (High Speed Downlink Packet Access), etc. which offer higher bandwidths for communication.

GPRS services:

- SMS-messaging & broadcasting services
- Multimedia messaging service (MMS)
- Mobile services: Provides constant voice and data communications, while on the move.
- Push-to-talk over cellular: A push button is used to switch from voice reception mode to transmit mode like in walkie talkie.

Advantages of GPRS:

- ✓ Provides high data rate than GSM.
- ✓ Un-interrupted internet access
- ✓ Fast connection set up.
- ✓ Supports bursty applications like email, broadcasting, web browsing, etc.
- ✓ High bandwidth.
- ✓ Used for point to point and multipoint communications.
- ✓ Users send or receive voice calls while browsing the internet or downloading data. Thus, users can have both voice call and data call together.

Disadvantages of GPRS:

- ➢ Speed is much lower in reality.
- ➢ The data rates are slower when compared to the latest technologies.

- Networks can be affected when more users use GPRS from same location at the same. time. It leads to traffic congestion and slows down data connection.
- If issues occur, it is difficult to troubleshoot.

UNIT-4

EMBEDDED FIRMWARE DESIGN AND DEVELOPMENT

Contents: Embedded firmware design approaches-Super loop-based approach, Operating system-based approach; Embedded firmware development languages-Assembly language-based development, High level language-based development.

EMBEDDED FIRMWARE DESIGN APPROACHES

1. Super loop-based approach
2. OS-based approach

1. SUPER LOOP BASED APPROACH

It is also called "**Infinite Loop Based Approach"**. It is very similar to conventional procedural programming, where code is executed task by task. The task listed at the top of the program is executed first, then the second task is executed second, and so on. Thus, tasks are executed sequentially. After last task is executed, control goes to the first task and the sequence repeats.

Almost all tasks in embedded applications are non-ending. Thus, they are repeated forever, throughout the operation of embedded product. This repetition is achieved by using an infinite loop called super loop, illustrated below.

Firmware execution flow:

1. Configure the common parameters.
2. Initialize hardware components like memory, registers, etc.
3. Execute first task.
4. Execute second task.
5. :
6. :
7. Execute the last task.
8. Go To step 3.

The only way to come out of the loop is either by using a hardware reset or issuing an interrupt.

A **Hardware reset** *re-initializes the hardware components and restarts the program.*

*A*n I**nterrupt** suspends the current task execution temporarily and executes a routine called interrupt service routine (ISR). Then, control goes back to the interrupted program.

Advantages and Applications of Super loop-based design:

(i) It is simple and straight forward. It has no OS related overheads because: -

- The priorities are fixed. So, no need for assigning priorities.
- Order of task-execution is fixed. So, no need for scheduling.

(ii) It is more suitable for applications which are not time critical and the response time is not important.

Example: An electronic video game toy with keypad and display unit.

Here, the program running inside this product is designed such that: -

- ✓ If any key is pressed, it is detected and the graphic display is updated.
- ✓ 'The keyboard scanning' and 'display updating' happens at reasonably high rate.
- ✓ Even if the application misses a key press, it won't create any critical issues. It will be treated as a bug in the firmware.

Drawbacks:

(i) If one task fails, total system is affected:

Suppose, while executing a task, the program hangs up at a point. Then, it will remain there forever and the product stops functioning.

To overcome this, watchdog timers (WDTs) are to be used. If a preset time is exceeded due to hanging up of processor or unexpected failure, the WDT will time out and reset the system. But this results in additional hardware cost and firmware overheads.

(ii) Lack of real-timeliness:

Let there be more tasks in an application, and each task is repeated many times

Then, some events are likely to be missed.

2. OPERATING SYSTEM-BASED APPROACH

This is also called **"Task scheduling"** approach. Here, functions to be executed are split into tasks. Then, the tasks are run using a scheduler which is a part of OS kernel. In this approach, a General-purpose operating system (GPOS) or a Real-time operating system (RTOS) is used to host the user-written application firmware.

(i) GPOS based design:

- It contains an OS like Windows/Unix/Linux, etc. for desktop PCs and user applications are created and run on top of it using application program interfaces (APIs). For example,
- The GPOS Microsoft ® windows XP used in developing Personal Digital Assistants (PDAs), Point of Sale (PoS) terminals, etc.

(ii) RTOS based design:

- ✓ It provides real-time response in timely and predictable manner. An RTOS contains a real time kernel which performs pre-emptive multitasking, multithreading, task scheduling, etc. It also provides communication among tasks using resources like CPU and memory.
- ✓ RTOS like VxWorks, Mbed OS, FreeRTOS, QNX and Symbian are used to produce embedded products like mobile phones, PDAs etc. Most of the mobile phones are built around the popular RTOS 'Symbian'.
- ✓ Any OS based application also requires 'Driver software' for different hardware devices.

EMBEDDED FIRMWARE DEVELOPMENT LANGUAGES

You can use

- ✓ Assembly Language (AL),
- ✓ High Level Language (HLL) like C, C++, JAVA, etc., or
- ✓ Mixing of AL and HLL

to write firmware.

AL is processor/controller specific, whereas HLL is processor/controller independent.

ASSEMBLY LANGUAGE BASED DEVELOPMENT

Important features:

- AL is processor/controller dependent. Assembly language programming (ALP) involves processor specific machine code in mnemonic form. ALP written for one processor/controller family will not work with others.
- In the beginning, ALP was used to write firmware. In 1960s, the majority of console videogames were written in AL.
- Even today, all low-level system related programming is carried out using AL.

- Some OS dependent tasks require low-level languages. For instance, ALP is used for device driver programming.
- The general format of an AL instruction is an opcode followed by one or more operands. The opcode tells the processor/controller what to do. Operands provide the required data. In some instructions, opcode implicitly contains the operand and no operand is required.
- A software called **"assembler"** is used to convert AL to ML.

Steps of conversion of AL to ML:

1. Source File to Object File Translation
2. Library File Creation and Usage
3. Linker and Locater
4. Object File to Hex File Conversion

1. Source File to Object File Translation

Assembler translates assembly code to machine code. Different assemblers are available in the market for the same processor. Some assemblers are available as 'open source'. Other assemblers are proprietary and require license from the vendor.

Example: A51 Macro Assembler from Keil software is popular assembler for the 8051-family of microcontrollers.

Each source module is written in AL and is stored as ".src" file or ".asm" file. After checking for syntax errors and incorrect assembly instructions, each .src/.asm file is converted to an object file with extension ".obj." These object files are re-locatable. i.e., they can be placed at any location in code memory. It is the responsibility of the program called 'linker' to allocate absolute addresses for these modules.

PUBLIC and EXTERN declarations for compiler: Modules can share variables and functions among them using the key words "PUBLIC" and "EXTERN".

The 'PUBLIC' keyword informs the assembler that variables or functions declared as 'PUBLIC' need to be exported. If more than one module in project declares same variable/function 'PUBLIC', it will generate 'linker' errors.

The 'EXTRN' Keyword tells the assembler that the variables or functions declared as 'EXTRN' need to be imported from some other modules. One or more modules use these variables/functions using 'EXTRN' keyword.

2. Library File Creation and Usage

A library is 'a collection of programs of important functions' used by application programs. These programs are used by a software called linker. You use a file called header file to access library files. You need to include all the appropriate header files in your project. 'Library file' is some kind of source code hiding technique. Suppose you don't want to reveal the source code of functions but like to allow developers to use them in their applications. Then put them as library files and provide the details of the public functions like function name, function input/output, etc.

- A Library **allows you to reuse code of important functions without compiling each time**. (e.g., functions/programs for performing multiplication, floating point arithmetic, etc.)
- When the linker processes a library, it uses the object modules in library, which are necessary to create program.
- Library files are generated with extension ".lib".
- If you are using a commercial version of the assembler/compiler, its vendor provides the required library files.

3. Linker and Locater:

Linker/Locater is a utility program that links various relocatable object modules of a program and assigns absolute addresses to each module. It has the following activities:

- Checks for object modules (.obj files) of the program in the library.
- Extracts them.
- Assigns absolute address to each module.
- Combines them into a single object file.

It is the responsibility of a linker to link external dependent variables/functions declared in various modules and resolve external dependencies among the modules.
"BL51" from Keil Software is an example for a Linker/Locater for A51 Assembler and C51 Compiler for 8051-specific controllers.

4. Object to Hex File Conversion:

This is the final stage in the conversion from AL to ML. Hex File is representation of the machine code in hexadecimal format. All code and data reside at fixed memory locations. It is dumped into the code memory of the processor/controller. It doesn't contain any relocatable code or data

Important features:

- Hex file is created from the final 'Absolute Object File' using the utility program called "Object to Hex File Converter".
- The hex file representation is processor/controller specific.
- For Intel processors/controllers, the target hex file format will be 'Intel Hex' and for Motorola, it is 'Motorola Hex' format.
- 'OH51' from Keil software is an example for an Object to Hex File Convert utility for A51 Assembler/C51 Compiler for 8051 specific controllers.

Advantages of AL Based Firmware Development:

(i) Optimized code:

If the developer is well versed with the processor architecture and memory organization, optimized code can be written for performing operations. This leads to less utilization of code memory and efficient utilization of data memory.

(ii) High Performance:

Optimized code not only improves the code memory usage but also improves the total system performance.

(iii) Low Level Device-Specific Operation Support:

Assembly coding is more convenient than HLL coding for accessing device-specific registers, device drivers, and low-level interrupt routines, etc.

(iv) Code Reverse Engineering:

- ✓ Reverse Engineering is a process of recovering the design, requirement specifications and functions of a product from a finished product.
- ✓ Its purpose is to improve the performance of product and correct deviations in the expected functionality.
- ✓ Hackers use reverse engineering to reveal the technology behind proprietary products.
- ✓ A program called **"disassembler"** is used to translate machine language into assembly language.
- ✓ Assembly Language is one of the programming languages used in Reverse Engineering.

Drawbacks of Assembly Language Based Firmware Development:

(i) High Development Time:

- AL Programming involves more lines of coding than HLL programming.
- The developer has to carry with him, an instruction set reference manual for the processor used.
- Developer requires thorough knowledge of architecture and hardware details of the target processor/controller. Learning the inner details of the processor and its assembly instructions is highly time consuming.

Hence, firmware development in AL is tedious and time consuming.

(ii) Developer Dependency:

- There are no written rules for developing AL-based applications. Developers have the freedom to choose the different memory locations and registers.
- The programming approach varies from developer to developer depending on his taste.

  Example: Different approaches are used to move data from a memory location to accumulator.
- If the coding done by a developer is not documented properly, a new developer may not be able to understand what is done and why it is done.
- Program written by one programmer is very difficult to understand by another, even if it is well documented.
- If the code is too large and complex, documenting all lines of code may not be productive.

(iii) Non-Portable:

AL instructions are processor specific. Applications written for one family of processors (e.g.: Intel x86 family of processors) cannot be reused with other family of processors (e.g.: ARM11 family of processors).

Hence, we have to completely rewrite the instructions. This is the major drawback of AL programming and it makes the AL application non-portable.

HIGH LEVEL LANGUAGE BASED DEVELOPMENT

Assembly Language-based programming is highly tedious, time consuming and requires skilled programmers with sound knowledge of the target processor architecture. Also, applications developed in AL are not portable.

HLLs like C, C++ or Java can be used for embedded firmware development. C language was very popular in this area because it has excellent cross-compiler support. Now a days, cross compilers for C++ are also available. So, C++ is also convenient.

Consider the program written in Embedded C/C++ using an Integrated Development Environment (IDE). The IDE contains programming tools like editor, compiler, linker, loader, debugger, simulator, etc. IDE varies from one family of processors/controllers to other.

The various steps involved in HLL based firmware development are similar to the steps in AL based development, with the following differences.

- Any text editor like 'Notepad' or 'WordPad' from Microsoft or text editor provided by the IDE can be used for writing the program. Most of the HLLs support and use modular programming approach. Here program is divided into multiple modules. i.e., multiple source files.
- The program written in any HLL is saved with the corresponding language extension (.c for C, .cpp for C++, etc.)
- The cross compiler in the IDE environment converts the HLL program into a hex file. Which contains the machine code in hexadecimal format.
  Example for cross-compiler: The IDE "Keil µvision3" is used for 8051 family microcontrollers. It contains the Keil C51 C Compiler. It is the most popular cross-compiler available for 'C' language for the 8051 family of microcontroller.

Advantages of HLL Based Development:

(i) Reduced Development Time:

Developer requires little knowledge of architecture and hardware details of the processor / controller. HLL coding involves lesser number of coding lines compared to AL. Thus, development time is reduced.

(ii) Developer Independency:

The syntax used by most of the HLLs are universal. So, a HLL program written by one programmer can easily be understood by another programmer knowing the syntax of that language.

(iii) Portability: The code in HLL is highly portable. It works with any processor/controller with little modification. Only thing we have to do is to recompile the program in new processor's IDE, after placing the required "include files" for that processor.

(v) Ease of debugging: If the source code contains necessary comments and document lines, it is very easy to understand and debug.

(vi) Best choice for beginners: For a beginner best choice is writing source code in HLL.

(vii) It provides Scope of refinement.

Drawbacks of HLL Based Development:

(i) Some cross-compilers available for HLLs may not be so efficient in optimization of code size and performance. If number of instructions increases, the time required to execute a task also increases. Modern cross-compilers can avoid this problem to some extent.

(ii) The investment required for HLL based firmware development tools (IDE containing cross-compiler) are higher than the AL counterpart.

MIXING ASSEMBLY AND HIGH-LEVEL LANGUAGES

Certain situations may demand mixing of AL and HLL (C, C++, JAVA, etc.) Then, this mixing can be done in 3 ways.

1. Mixing AL to HLL
2. Mixing HLL to AL
3. In-line Assembly Programming

Let 'C' language is the HLL used. Then we discuss above types of mixing.

1. Mixing AL to HLL:

Program is written in HLL 'C'. Some essential assembly routines are added to it.
Consider the following possible situations: -

- Cross-compiler cannot support Interrupt Service Routine (ISR) functions.
- Programmer wants to add machine code to optimize the code or improve speed. Then he uses machine code generated by hand written assembly code instead of using machine code generated through 'C' using cross-compiler.
- When accessing certain low-level hardware, a cross compiler cannot offer the required time critical specifications accurately.

Then required routine is written in AL and it is invoked from C. Here, the programmer must be aware of the following: -

- How parameters are passed from C routine to assembly routine.
- How the values are returned from assembly routine to C routine.
- How assembly routine is invoked/activated from C.

These functions are cross-compiler dependent. Different cross compilers implement these functions in different ways depending on the general-purpose registers and the memory used. There is no universal rule for this. So, you must get the information from documentation of the cross compiler in use.

2. Mixing HLL with AL:

Source code is written in AL. C routines are included in that assembly code.

The entire source code is planned in AL for various reasons like

- Optimizing code and performance.
- Efficient code memory utilization.
- Skilled AL programmers are available.

The problems we face here are:

- Some portions of the code may be very difficult and tedious to code in AL.
  Example: 16-bit multiplication and division in 8051 AL.
- We have to include built in C library functions provided by the cross compiler.
  Example: Built in Graphics library functions and String operations supported by C.

The programmer should be aware of the following: -

- How the parameters are passed to the functions
- How values are returned from the functions
- How the function is invoked from the AL environment.

3. Inline Assembly:

It is a technique used to insert processor-specific assembly instructions at any location of a source code written in C. This avoids the delay in calling assembly routine from C code'. Special keywords are used to indicate that the starting and ending of Assembly instructions. The keywords are cross compiler specific. C51 uses #progma asm and #pragma endasm to indicate a block of code written in assembly language.

e.g.:

```
#pragma asm
MOV A, #13H
#progma endasm
```

UNIT-5

RTOS BASED EMBEDDED SYSTEM DESIGN

Contents: Operating system basics, Types of operating systems, Tasks, Process and Threads, Multiprocessing and Multitasking, Task scheduling: Non-pre-emptive and Pre-emptive scheduling; Task communication-Shared memory, Message passing, Remote Procedure Call and Sockets, Task Synchronization: Task Communication/Synchronization Issues, Task Synchronization Techniques.

OPERATING SYSTEM BASICS

INTRODUCTION:

Operating System (OS):

An OS acts as a bridge between the user applications (or tasks) and the system resources. It is a system software responsible for managing system memory, resources and application programs. This program is loaded into the computer by a boot program.

Application Program Interface (API):

API acts as an interface between application program and OS. In addition, users can interact directly with the OS through a user interface (UI).

User Interfaces (UIs):

A UI is the point of human-computer interaction and communication. (e.g.: display screen, keyboard, mouse, etc.).

Types of UIs: CLI, GUI, VUI and GBI

- **Command Line Interface (CLI)**: It is a text-based user interface used to run program. Users use traditional keyboard to enter specific commands and arguments related to specific tasks.
- **Graphical User Interface (GUI):** It is a visual interface that uses icons, menus and a mouse. Mouse is used to click on the icon or pull down the menus to manage interaction with the system. GUI is more user-friendly.
- **Voice Controlled Interfaces (VUI):** The users interact through their voices (e.g.: Amazon's Alexa)
- **Gesture-Based Interfaces:** Users interact through bodily movements. (e.g., VR Games)

Kernel:

Kernel is the core of the OS. It is responsible for managing the hardware and software resources and communication among them. Kernel acts as an abstraction layer between system resources and Kernel mainly contains a set of system libraries and services.

OS Scheduler:

OS Scheduler is a system software. It is a part of the kernel process. In multiprocessing environment, it chooses the order of execution of processes. i.e., it decides which process is to be executed first and which one is next. It also takes an active role in interrupt handling and exception handling.

KERNEL SERVICES OF OS

1. Process Management
2. Memory Management
3. File Management
4. I/O Device Management
5. Inter-Process Communication (IPC)
6. Protection
7. Error Handling
8. Job accounting
9. Interrupt Handling
10. Synchronization
11. Managing time-critical requirements (RTOS)

(Last two are the most important services provided by an RTOS)

1. Process Management:

A process/task is a program under execution. Process Management deals with managing the processes. Kernel provides the following process Management services:

- It allocates system memory and resources.
- Loads process-code into the memory.
- Schedules the execution of processes.
- Creates Process control block (PCB).
- Provides synchronization for process using IPC (Inter Process Communication)
- Executes the process.
- Terminates/deletes the process.

2. Memory Management:

(a) Primary (Main) Memory Management:

Before a process is executed, process and required data are stored in main memory. It is basically a RAM. It has high speed and can be accessed directly by the CPU. Kernel provides th following primary Memory management services:

- ✓ Allocating memory space when required and deallocating it later,
- ✓ Keeping track of memory used.
- ✓ Protection of important in formation (like stack contents) against unauthorized access.

(b) Secondary Memory Management:

Secondary storage devices like magnetic tapes, magnetic disks and optical disks are used for long-term storage purpose. Each of these devices has its own properties like speed, capacity, data transfer rate and data access methods. These devices are non-volatile and so used for backup of programs and data. In most of the systems hard disk is used for secondary storage.

Secondary memory services of kernel:

- ➢ Disk storage allocation
- ➢ Disk scheduling.
- ➢ Free disk space management.

3. File System Management:

File is a collection of related information. It may be a program file, text file, word document, audio/video file, etc. In computers, files are stored on secondary storage, for long-term storage purpose. OS uses a method or data structure called 'file system' to control how data is stored and retrieved. Kernel provides the following file system management service of kernel:

- The creation, deletion and modification of files and directories
- Allocation of space for file.
- Providing flexible naming convention for the files.
- Saving of files in secondary memory (e.g.: Hard disk)

These operations differ from OS to OS.

4. I/O Device Management: Kernel

- ✓ Provides access to the required I/O devices through appropriate interfaces.
- ✓ Maintains a list of all I/O devices

- ✓ It provides a service called 'Device manager' to handle I/O operations using device drivers. Device Manager loads and unloads device drivers and sends data and control signals to and from the I/O devices.

5. Inter-Process Communication (IPC): IPC means communication between processes. Kernel provides the following IPC mechanisms.

- Signals
- Message queues & Mailboxes
- Semaphores
- Pipes & Sockets
- Remote Procedure Calls (RPCs)

6. Protection:

'Protection' means protecting system resources from unauthorized access and manipulation. The system must also be protected from external attacks like viruses and worms. Protection services of kernel provide different levels of permissions (e.g.: no access, read only access, etc.). It provides authentication mechanism for each user by means of passwords.

7. Error/Exception Handling: It deals with registering and handling the errors occurred and exceptions raised during the execution of processes/tasks. It constantly checks for possible errors and takes appropriate action. It ensures correct and consistent processing. It also ensures reliable data transmission across vulnerable networks.

Examples for errors/exceptions: Programming errors (syntax errors, logical errors and errors at compilation time and run time), Resource errors (e.g.: Bus error), Interface errors, Communication error, divided by zero.

8. Job Accounting: OS keeps track of 'time and resources' used by various tasks and users. This information can be used to track resource usage for a particular user or group of users.

9. Interrupt Handling Service:

An interrupt is an event or signal that alters the sequence in which the processor executes instructions. It causes processor to stop executing current program temporarily and executes an Interrupt Service Routine (ISR) to handle that interrupt**.** Then control is passed to the main program to resume its execution. Handling of interrupts is based on priorities.

10. Other important services provided by RTOS:

(i) Task Synchronization: Task/Process synchronization means efficient sharing of system resources by concurrent processes without any conflicts.

(ii) Meeting time-critical requirements

REAL-TIME OPERATING SYSTEMS (RTOS)

Real time implies deterministic (predictable) behavior. i.e., execution produces same output in all runs for the same input. It is a special-purpose OS. An RTOS service consumes known and expected amount of time. i.e., RTOS has predictable performance. RTOS decides which applications should run in which order and how much time needs to be allocated for each application.

An RTOS is used for real-time applications which have critically defined time constraints.

Examples for Real-time applications:

- Air-traffic control systems
- ATM
- Industrial control systems
- Missile launching systems
- Networked Multimedia Systems
- Anti-lock Brake Systems (ABSs) in vehicles
- Heart Pacemaker

Examples of RTOSs: VxWorks, Mbed OS, FreeRTOS, QNX, etc.

RTOS vs GPOS:

S. No.	Real-Time Operating System (RTOS)	General-purpose Operating System (GPOS)
1.	Uses block-based memory allocation.	Uses dynamic memory allocation.
2.	Optimizes memory resources.	Does not optimize the memory resources
3	It doesn't have large memory.	It has large memory.
4.	Requirements are time-critical.	Not time-critical.
5.	Time Response is predictable.	Not predictable.
6.	It has a task deadline.	No task deadline.

*Note: Left column in above table indicates important features of RTOS.

Types of RTOSs:

There are 3 types of RTOSs. They are OSs for Hard real-time, Soft real-time and Firm real-time systems.

Hard real-time system: It has a set of strict deadlines. Missing even single deadline is considered a system failure. (e.g.: Flight control system, missile guidance system)

Soft real-time systems: Here, one or more failures to meet the deadlines are not considered complete system failure, but performance is considered to be degraded (e.g.: ATM, audio/video systems).

Firm real-time system: In this, a few missed deadlines will not lead to total failure, but missing more than a few may lead to complete or catastrophic system failure. (e.g.: Satellite-based surveillance applications, financial forecast systems),

TASKS, PROCESSES & THREADS

A 'Task' is a unit of work to be done. In computing systems, a task consists of sequentially executable code that runs on CPU. In OS context, task is defined as the program in execution. It is also called a 'Process' or 'Job'.

PROCESS

A process (or task)/is an executable program. It runs on CPU under the control of OS Scheduler. A process requires various system resources like CPU, memory for storing the process code and data, files and I/O devices. OS allocates and manages all these resources. Each process mainly consists of Process states, Process structure and Process Control Block (PCB).

Process states:

A process state indicates the current status of the process. In a 'Process life cycle', there are 5 states as shown in Fig. 1. The process traverses though these states during its transition from creation to termination.

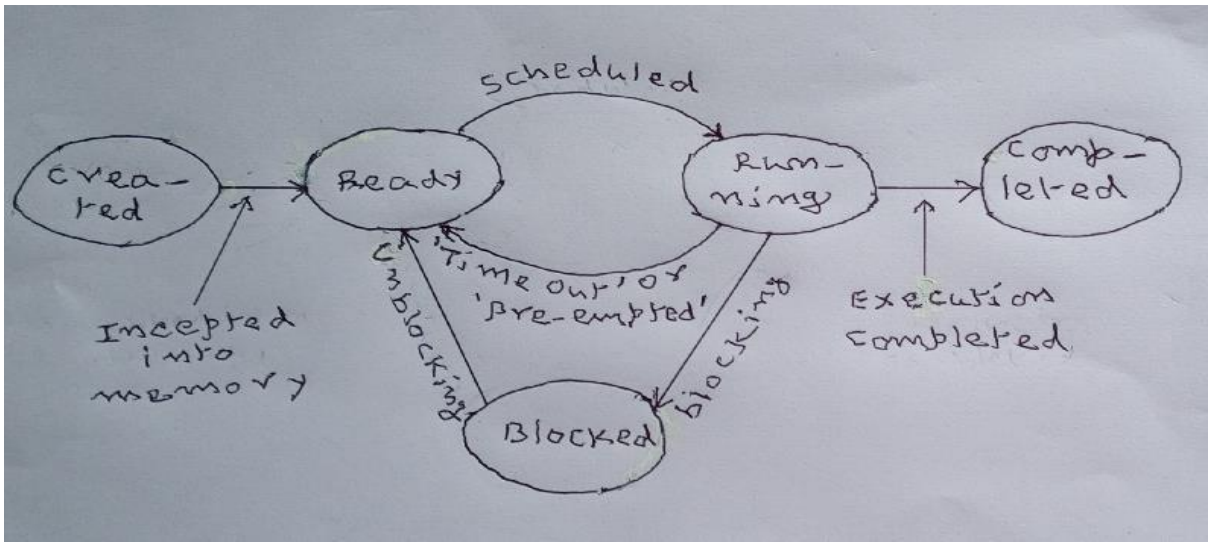


Fig. 1: Process States

'Create' state: In this state, the process is about to be created but not yet created. OS recognises it but no resources are allocated to process. Create state is also called 'New' state. or 'Dormant' state.

'Ready' state: It is the state where process in loaded into the memory and placed in the ready queue maintained by the OS. A ready queue is a queue of processes which are waiting for CPU allocation. After creation, a process enters in the ready state.

'Running' state: It is the state at which source code instructions of process are executed.

'Blocked' state: It is also called 'waiting' state or 'interrupted' state. It is the state where a process is waiting for

- Some event to occur
- For a resource or
- For the completion of an I/O operation.

After the event occurs or resource is available, the process again goes to ready state.

'Completed' state: A state where process completes its execution. It is also called 'Terminated' state, 'finished' state, 'exit' state or 'deleted' state. Here process is removed from main memory. OS deletes PCB and releases the associated resources.

Process structure:

Process structure means data, objects and resources related to process. Process structure mainly refers Stack, Program Counter, Process status and CPU registers used by the process. A process structure is available in PCB.

Process Control Block (PCB):

PCB is a data structure used by OS to store process structure and all information about a process. Each process has a PCB. It is created when process is created and deleted when process is terminated.

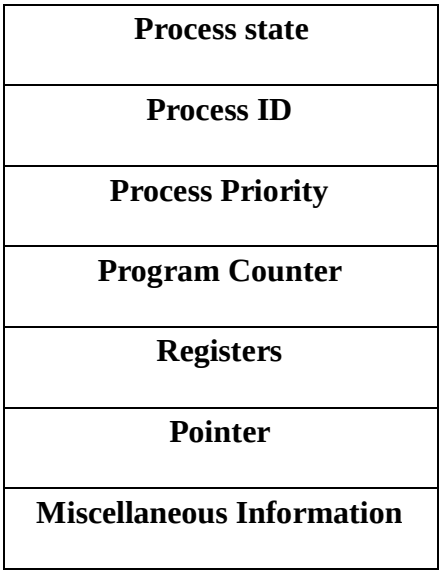


Fig. 2: Process Control Block

Fig. 2 shows the main contents of a PCB. They are described below.

Process State: It is the current state of the process i.e., create, ready, running, blocked or completed state.

Process ID: Identification number of the particular process.

Process Priority: Priority assigned to the process.

Program Counter: Register which contains the address of the next instruction to be executed in the process.

Registers: This specifies the other registers used by the process. (Accumulator, index registers, stack pointer, general purpose registers, etc.)

Pointer: It points to the address of the next PCB in ready state.

Miscellaneous Information:

- Memory Management Information: Page tables or the segment tables, etc.in the memory.
- I/O Status Information: List of files and I/O devices used by the process.
- Accounting information: Amount of CPU used, time constraints, etc.

Process Memory:

It is the memory occupied by a process. It contains three regions, namely, stack memory, data memory and code memory. Stack memory holds all temp data such as variables local to the process. Data memory holds all global data for the process. Code memory contains the program code (instructions) corresponding to the process.

Steps in Process Management:

- Allocating system memory and resources
- Loading of the process-code into the memory.
- Scheduling the execution process.
- Creation of PCB for the process.
- Running the process code.
- Syncrobnizing the process using IPC.
- Terminating/deleting the process.

THREADS

A thread is a segment of a process. It is the smallest code sequence that can be managed independently by scheduler. It is also called a lightweight process. A process can have one or more threads. Different threads in a process share the data memory code memory and heap memory area in the address space. Threads maintain their own thread status (Stack and CPU register values) and program counter.

MULTIPROCESSING AND MULTITASKING

Multiprocessing:

It is the ability of an OS to execute multiple processes simultaneously. Multiprocessor systems have multiple CPUs and execute multiple processes simultaneously.
In single processor systems, multiprocessing/**multitasking** involves switching of CPU from executing one task to another. It creates the illusion of multiple tasks executing in parallel.

A process is considered as a **virtual processor**, because it has its own CPU registers, stack and program counter like a **physical processor**. When CPU switches from one process to another, the properties of process (virtual processor) are converted to those of physical processor. OS scheduler controls this switching.

When CPU is interrupted, it temporarily stops the execution of the current process, and switches to interrupting process. This situation is called **context switching**. Before doing so, it saves the context of current process. Context means the state of a process that includes the details of CPU registers, memory, system resource usage, execution details, etc.
After executing the new process, control comes to the first process. Then CPU retrieves the saved context, uses it and runs the process.

Example for multiple tasks in Automatic chocolate vending machine (ACVM):

- Getting user input from keypad.
- Reading the inserted coins' amount.

- Delivering the chocolate.
- Display messages (e.g.: Displaying the message “Thank you-Visit again”)
- GUI-task for graphical user interface.
- Communication service (messaging machine status and information to owner).

Multithreading:

A process/task in an application consists of many suboperations like:

- Getting i/p from I/O devices.
- Performing some internal calculations.
- Updating some I/O devices, etc.

If all suboperations are executed in sequence, it takes more time and CPU utilization is not efficient. For example, if the proces is waiting for a user input, CPU has to stay idle.
So the process is split into different threads.

Each thread occupies a portion of the process. It can be independently managed by OS scheduler. Each thread is meant for one suboperation.

Advantages of multithreading:

- Better memory utilization: because multiple threads within a process share address space for data memory.
- Since variables can be shared across threads, IPC becomes easier.
- Better CPU utilization: When one thread enters wait state, other threads can utilize CPU. This speeds up the execution of process.

Example for multiple threads in a process: Display-Process in mobile phone has the following threads:

- ✓ A thread to display clock time and date.
- ✓ A thread to display battery power.
- ✓ A thread to display silent or active mode
- ✓ A thread to display unread messages in the in box
- ✓ A thread to display call status: whether dialing or call waiting
- ✓ A thread to display menu.

Process vs Thread:

S.No.	Process	Thread
1	A process is a program under execution i.e., it is an active program. A process is also called heavy weight process.	A thread refers to the segment of the process. It is a lightweight process.
2	Processes don't share memory with other processes.	Threads share memory with other threads of the same process.
3	Individual processes are independent of each other.	Threads of a process are dependent on each other.
4	Inter process communication takes more time.	Communication between threads takes relatively less time.
5	Process requires more time for creation, termination and context switching and needs more resources.	Thread requires more time for creation, termination and context switching and needs few resources.
6	If a process gets blocked, remaining processes can continue execution.	If a thread gets blocked, all of its peer threads also get blocked.

TASK SCHEDULING

Scheduling and Kernel:

- Determining the order of running the processes is known as scheduling.
- Kernel is the core of the system software OS.
- OS scheduler is a part of the kernel, which is responsible for scheduling.
- OS Scheduler decides which process is executed first and which is next.

Scheduling Policies/Algorithms:

Scheduling policies form the guidelines for scheduling. A scheduling policy is implemented as an algorithm, which is run by the kernel as a service.

The selection of scheduling algorithm depends on the following factors:

- CPU utilization: How effectively the CPU is utilized.
- Throughput: Number of processes executed per unit time.

- Turn Around Time (TAT): Total amount of time spent by a process in the system.
  (TAT = Waiting time + Bust time)
- Waiting Time: Time for which a process waits in the ready queue for getting a CPU.
- Bust time: Execution time for a process.
- Response Time: Amount of time after which a process gets the CPU for the first time after entering the ready queue.

A good scheduling algorithm has high CPU utilization, minimum turnaround time, maximum throughput and least response time.

Queues maintained by OS in CPU Scheduling:

- **Job Queue**: This queue contains all the processes in the system.
- **Ready Queue:** Contains all the processes residing in main memory that are ready to run.
- **Device Queue:** Contains the processes which are waiting for an I/O device.

A process migrates through theses queues during its life cycle. In OS context, queue acts as a buffer.

SCHEDULING MECHANISMS:

Based on the scheduling algorithm used, the scheduling mechanisms are classified as Non-pre-emptive and Pre-emptive scheduling mechanisms.

In **non-pre-emptive scheduling**, currently executing process is allowed to run until it terminates or enters to 'wait' state. In other words, once CPU is allocated to a process, it holds the CPU till it terminates or enters 'wait' state.

A **Pre-emptive scheduling** mechanism is a priority-based scheduling. Processes with higher priorities are honored first. If a higher priority task becomes ready to run, OS pre-empts a lower priority task that is already running. Lower priority task is suspended and higher dpriority task runs on CPU.

NON-PREEMPTIVE SCHEDULING ALGORITHMS

1. First-Come-First-Served (FCFS) Scheduling policy:

It is also called First-in-First-Out (FIFO) algorithm. It executes the processes in ready queue in the order of their arrival into the queue. i.e., it allocates first job arrived in the queue to the CPU first, then allocates the second one, and so on.

2. Shortest Job First (SJF) Scheduling policy:

It is also called "Shortest Job Next (SJN) Algorithm"**.** It executes the process with the shortest estimated run time first, followed by the next shortest process, and so on. Here, the OS needs to know (or guess) the execution time of each process on CPU. It is not suited for interactive jobs, where execution time is not known.

3. Last-Come-First-Served (LCFS) Scheduling:

It is also called Last-Come-First-Out (LIFO) Scheduling. Here the last job entered the queue is served first, followed by the last but one, and so on.

4. Priority Based Non-preemptive Scheduling:

It ensures that a process in the ready queue with high priority is serviced first. The priority of a process can be indicated through various mechanisms. One way is while creating a process a priority can be assigned to the process. For example, 0 to 255 is assigned as priority in Windows CE. Here '0' indicates highest priority and '255' indicates the lowest priority.

Example: In SJF Algorithm, the lower the time required to complete a process the higher is its priority.

PRE-EMPTIVE SCHEDULING ALGORITHMS

1. Shortest Remaining Time (SRT) Scheduling:

- It is Pre-emptive version of SJF scheduling algorithm.
- Non-pre-emptive SJF scheduling algorithm sorts the ready queue only when the current process is completely executed or enters the wait state.
- Pre-emptive SJF Scheduling algorithm sorts the ready queue when a job enters the queue .and checks whether the execution time of new process is shorter than remaining time of the currently executing process. If the execution time of new process is less, the currently executing process is pre-empted and the new process is scheduled for execution.

2. Round Robin (RR) Scheduling Algorithm:

Each process in the ready queue is executed for a pre-defined time slot (time slice). The scheduler picks the first job in the ready queue first. It is executed for in its time slot. When the alloted time elapses, or the process completes, the next process in the ready queue is selected for execution. This is repeated for all the process in the ready queue

After executing all the processes once, the scheduler picks the first process in the ready queue for execution. Thus, ready queue acts as a circular queue. The sequence is repeated until all the processes in the queue are completely executed.

Normally, OS assigns the time slice. The time slice varies in the order of a few microseconds to milli seconds. Some OS kernels permit user to assign time slice.

Note: Round-robin algorithm is a pre-emptive algorithm as the scheduler forces the process out of the CPU once the time quota expires.

Comparison of two scheduling mechanisms:

Preemptive Scheduling	Non-Preemptive Scheduling
CPU is allocated to a process for a specific time period**.**	CPU is allocated util the current process terminates or enters waiting state.
More CPU utilization.	Less CPU utilization
Waiting time and response time are less.	More waiting time and response time.
More overhead due to context switching.	No context switching. Less overhead.
It is used in real-time systems where time critical tasks are given higher priorities.	It can make real-time and priority scheduling difficult. It can also lead to starvation of real-time tasks
Examples: Shortest Remaining Time First & Round Robin algorithms.	Examples: First-Comes-First-Serve (FCFS) & Shortest Job First (SJF) algorithms.

TASK COMMUNICATION-

In a multitasking system, multiple tasks/processes run concurrently. There may be interaction/communication between tasks. Based on the degree of interaction, the processes running on OS are classified as:

(i) Co-operating Processes: One process requires inputs from other process to run. These processes exchange data through some shared resources. They also communicate for synchronization.

(ii) Competing Processes: The competing processes do not share anything among themselves but they share the system resources like files, display devices, etc.

Processes communicate with each other through Inter-Process Communication (IPC). IPC is essential for process coordination. Different IPC mechanisms are described below.

SHARED MEMORY:

Processes share a part of the memory to communicate with each other. The implementation of shared memory concept is Kernel dependent. Important IPC mechanisms used in this context are pipes and memory mapped objects.

PIPES

.

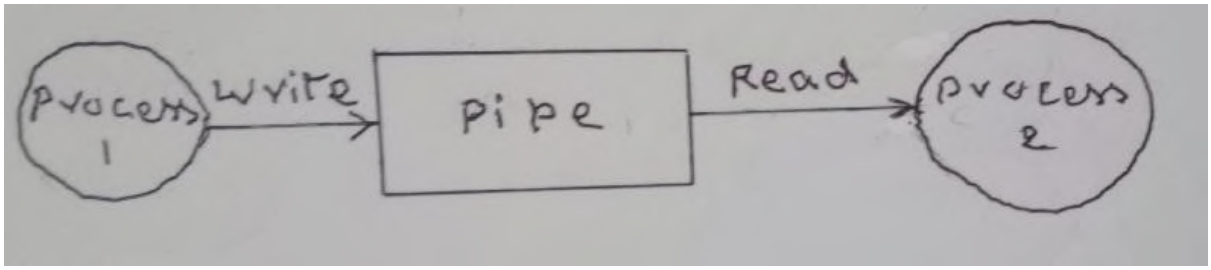


Fig 3.: A pipe used for IPC

A pipe is a section of shared memory used by processes for communication. It has two ends as shown in Fig. 3.

A pipe may be unidirectional or bidirectional. In unidirectional pipe, one process writes information, while other process reads. In bidirectional pipe, both processes can read as well as write.

Pipes follow the client-server architecture. A process which creates a pipe is called a pipe server and a process which is connected to the pipe is called a pipe client. The implementation of pipes is kernel dependent.

There are two types of pipes- 'anonymous pipes' and 'named pipes'.

(i) Anonymous Pipes: They are unnamed and unidirectional pipes used for data exchange between two processes.

(ii) Named pipes: They are named and 'uni- or bidirectional' pipes. With named pipes any process can act as both client and server allowing point-to-point communication. The two processes may run on same machine or different machines connected to the network.

In the context of pipes, there is parent-child relationship, as described below.

Parent Process and Child Process:

- Kernel creates a parent process. Parent process creates the child process using a system call "fork ( )".
- A parent process can have one or more child processes.
- A child process is like a copy of parent and it inherits many attributes from parent
- A child process and its parent process run independently and asynchronously.
- Each child process has a unique 'Process ID (PID)'. But all the child processes of a parent have the same 'Parent ID (PPID)'.
- A parent process will not terminate until all its child processes are terminated.

System calls used by pipe:

- ✓ **A fork ( ) system call:** It is used by a parent process to create a child process.
- ✓ **A Kill ( ) system call:** It is used by the parent to send messages to child using the PID.
- ✓ **Signal ( ) function call:** It is used by child process and call appropriate functions.

OS provides functions for creating, opening and closing a pipe device, connecting a thread or task to a pipe, reading and writing. Those functions are described below.

OS Functions for a pipe:

1. pipeDevCreate ( ): It creates a pipe device.
2. open ( ): It opens a pipe device.
3. connect ( ): It connects a thread or task to a pipe.
4. write ( ): It writes into the pipe from the bottom.
5. read ( ): It reads from the bottom.
6. close ( ): It closes the pipe device.

MEMORY MAPPED OBJECTS

It is a shared memory technique adopted by some RTOSs like 'Windows CE RTOS' It allocates a shared block of memory which can be concurrently accessed by many processes. Some synchronization techniques should be applied to prevent inconsistent results.

In this approach, a mapping object is created and physical storage is reserved for it. Any process which wants to share data with other processes can map the physical memory area or a block of the mapped object to its virtual memory space and use it for sharing the data.

Note: Virtual address is a logical address that acts as a pointer to the physical address. It appears like physical address, but doesn't exist physically in the memory

MESSAGE PASSING

Message passing is an asynchronous IPC mechanism.

Shared memory vs Message passing: (i) Through shared memory lots of data can be shared. But through message passing, only limited amount of data is passed. (ii) Message passing is relatively fast and free from the synchronization overheads compared to shared memory.

Message passing can be done by 3 IPC mechanisms- Message Queue, Mailbox and Signaling.

(I) MESSAGE QUEUES

If Process1 wants to send a message to Process2, Process-1 first sends the message to a First-In-First-Out queue called Message queue. The queue stores the message temporarily in a system defined memory object, to pass it to the desired process. The messages are exchanged through a message queue. This is shown in Fig. 4.

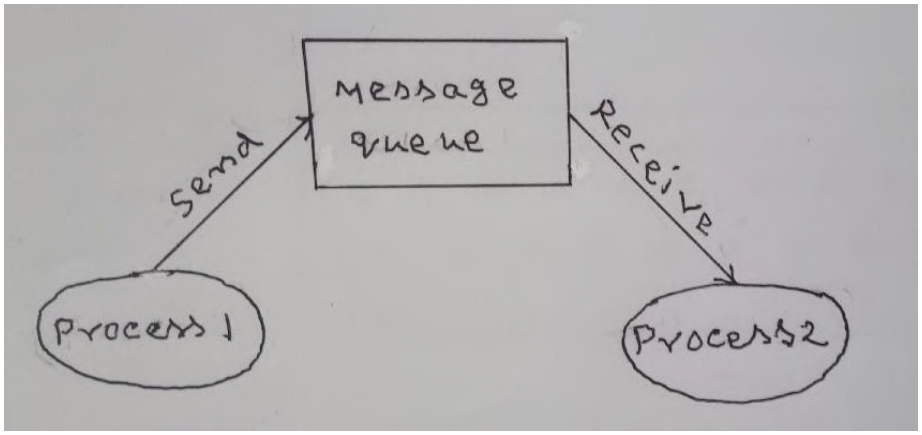


Fig. 4: Message Queue

Messages are sent and received using 'send' and 'receive' IPC functions offered by OS. The implementation of queue, send and receive methods are kernel dependent

Important features of a Message Queue:

- Message queue provides asynchronous communication.
- It operates on FIFO principle.
- Each queue must be initialized before using the message queue functions in the scheduler.
- Each message queue has an ID.
- There may be provision for multiple queues.
- Each queue has either a user definable size (upper limit for number of bytes) or a fixed size assigned by the scheduler.
- When queue is full, there may be need for error handling and need for user codes for blocking the tasks.

IPC functions for serving message queues:

1. OSQCreate: This queue function creates a queue and initializes the Q message blocks with front and back pointers called *QFRONT and *QBACK respectively.
2. OSQPost: It posts (sends) a message to the message block.
3. OSQPend: To wait for a queue message at the queue. It reads the message and deletes it when received.

4. OSQAccept: Reads the present *QFRONT after checking its presence (Yes or No). After the read *QFRONT increments.
5. OSQFlush: It reads the queue from front to back and deletes all the messages in the queue. After the flush, the *QFRONT and *QBACK point to the beginning of queue.
6. OSQQuery: It queries the queue message block when read, but the messages are not deleted. This function returns a pointer to the *QFRONT, number of queued messages, size of the queue and table of tasks waiting for the messages from the queue.
7. OSQPostFront: Sends a message as per the *QFRONT. It is used when a message is urgent or has higher priority than all the previous posted messages into the queue.

Note: These functions are offered by OS and used by ISRs and processes/tasks.

Mailbox:

Mailbox is an alternate form of 'Message Queue' and it is used as an IPC mechanism in certain RTOSs. It is used for one way messaging.

Example: Mailbox in a mobile phone.

Important Features:

- The thread which creates the mailbox is known as 'mailbox server' and the threads which subscribe to the mailbox are known as 'mailbox clients.
- The mailbox server posts the messages to the mailbox and notifies it to the clients which are subscribed to the mailbox.
- The clients read the message from the mailbox on receiving the notification.
- The implementation of mailbox is OS kernel dependent.
- Each mailbox has an ID. OS provides the IPC functions such as create, post (send message), pend (wait for a mailbox-message), and query (queries the mailbox and obtains the information about a mailbox).
- Each mailbox for a message is initialized with a NULL pointer before posting any message into the box.

IPC functions of a mailbox: (OS provides these functions)

1. OSMboxCreate: Creates a mailbox and initializes the mailbox with NULL pointer.
2. OSMboxPost: Sends (writes) a message to the box
3. OSMboxWait (Pend): Waits for a mailbox-message and reads the message, which is read when received.

4. OSMboxAccept: Checks the mailbox to see if a message is available. This function does not block (suspend) the calling task if a message is not available. If available, it returns the pointer.
5. OSMboxQuery: Queries the mailbox and obtains information about a mailbox.

SIGNALLING

Signalling is an IPC mechanism used by OS. A signal is a notification to a 'process' or 'thread within the same process' to indicate the occurrence of an event for which the other process/thread is waiting. Since we cannot predict its occurrence, it is called an asynchronous notification.

Important Features:

- Signals are not queued and they do not carry any data.
- Signals are used for interrupt and exception-handling processes. They are also used for OS or user-defined processes.
- A signal handler is associated with each signal. Whenever a specific signal occurs, its signal handler handles it. Signal handler is a function defined in the program code and registered with the kernel.
- A signal can be specified with a number or name. Usually, a signal name starts with SIG.
- A signal is generated by a system, or through a program. When a signal is raised, there are three situations:
    - ✓ Signal performs a default action.
    - ✓ Signal is handled to perform other actions.
    - ✓ Signal is ignored.

Examples for signal calls/functions:

- signal () function: It sets a signal handler based on the argument in the brackets
- os_send_signal kernel call in 'RTX51 Tiny' OS sends a signal from one task to another specified task.

REMOTE PROCEDURAL CALL (RPC)

RPC is an IPC mechanism used for connecting two remotely placed processes. It is used by one process to call a procedure of another process running on the same CPU or on a different CPU that is interconnected in a network.

Important Features:

- OS provides RPCs for distributed environment like client-server systems. Here client and server are often on remote systems connected by network. But they may also be on same machine.
- Interface Definition Language (IDL) defines the interface for RPC.
- In order to make the communication platform-independent, certain standard formats are essential.
- The RPC communication can be either Synchronous (Blocking) or Asynchronous (Non-Blocking).

  In the former case, the calling process is blocked until it receives a response from the other process. In later case, the calling process continues its execution while the remote process executes the procedure. Result from the remote procedure is received through call-back functions.
- Mechanism like IDs, public key cryptography (like DES, 3DES), etc. are used by the client for authentication.
- Sockets are also used for RPC communication.

Advantages of RPC:

- RPC hides the message passing mechanism from the user.
- It is very useful in distributed environment. It works equally well in local environment.
- Many of the protocol layers are omitted by RPC to improve performance.

Disadvantages of RPCs:

- ✓ RPC is a concept that can be implemented in different ways. It is not a standard.
- ✓ There is no flexibility in RPC for hardware architecture. It is only interaction based.
- ✓ It involves more cost.

SOCKETS

Socket is an IPC mechanism Which allows user to exchange information between processes on the same machine or two different machines. Each socket may have the source address and destination address.

Important Features of Sockets:

- A socket is as an end point for communication. Two processes communicating over a network employ a pair of sockets.
- A Socket is a bidirectional device between client and server.

- Sockets are available on every OS. They are designed to run over a standard Network layer like TCP or UDP.
- Sockets provide point-to-point, two-way communication.
- Each socket on internet has a port number, source address and destination address. IP addresses of source and destination may be on the same computer or different computers on the network. A protocol is used for communication between source and destination.
- Functions are available for clients and server that run on same CPU or distinct CPUs on internet.
  (Server is a machine or system that requires services from client. Client is a machine or system that serves those requests).
- Client and server may use different domain (e.g.: One socket domain may be TCP (Transport Control Protocol) and another socket domain may be UDP (User Datagram Protocol).
- Advantage of socket design is that client and server can be loaded on same machine.

Applications of Sockets:

- Interconnecting two tasks in two different ESs
- Task communications in ESs in distributed environment
- A TCP/IP socket for Internet.
- When a task writes information into a computer file using NFS (Network File System) protocol

OS Functions for socket in Unix:

Socket ( ): It is like open ( ) function of a pipe. It gives socket function descriptor **sfd**.

Unlink ( ): It is used before the bind ( ):

bind ( ): It is used for binding 'a thread or task inserting bytes into the socket' to 'the thread/task deleting bytes from the socket'.

listen ( ): It is used for listening 16 queued connections from client socket.

accept ( ): It accepts the client connection and gives a second socket descriptor.

recv( ); Used for deleting and receiving from the socket from the bottom of the memory space in the buffer filled after writing into the socket.

send ( ): Used for inserting (writing) and sending from the socket from the bottom of the memory space in the buffer filled after writing into the socket.

close (): It is used for closing the device, to enable its use from beginning of its allocated buffer, only after opening it again.

TASK SYNCHRONIZATION

'Task Synchronization' means efficient sharing of system resources by concurrent tasks without any conflicts. It is most essential in multitasking environment.

Example for conflict: Suppose 2 processes try to access a shared memory area and perform updates on a shared memory location.

To address such conflicts, we should see that each process is aware of other processes accessing the shared resources.

TASK COMMUNICATION/SYNCHRONIZATION ISSUES

1. Racing
2. Deadlock
3. The Dining Philosopher's Problem
4. Producer-Consumer/Bounded Buffer Problem
5. Readers-Writers Problem
6. Priority Inversion

1. Racing:

Racing occurs in an OS when two or more concurrent processes or threads try to access and change same shared resource (such as a variable or file). Then the outcome of their execution depends on the order in which they are executed. If the processes/threads are not correctly synchronized, one thread can overwrite other's changes. This is an undesirable situation. This can lead to incorrect results, system crashes, deadlocks or security problems.

A real-world example: Two employees performing updates on a shared document at the same time.

Detection of racing: Debugging tools (e.g.: race condition detectors, code analysis tools, etc) can be used to detect and debug race conditions. Thorough testing and code reviews also help to predict race conditions.

Ways to prevent racing:

(i) Using atomic operations: Atomic means indivisible. An atomic operation involves set of instructions that guarantee any updates to a shared variable, by allowing only one thread to access it at a time.

(ii) Using synchronization mechanisms: Use locks or semaphores to ensure that only one process or thread can access a shared resource at any given time.

(iii) Avoid global variables.

(iv) Using message passing: Use message passing instead of shared memory to communicate between processes or threads.

(v) Proper design process: When designing your application, make sure that your design can handle multiple processes or threads accessing shared resources.

2. Deadlock:

We know that a race condition produces incorrect results.

A deadlock is a situation where none of the processes can progress in their execution. i.e., all processes stop running. It **occurs when a group of processes are blocked in a state where each process is waiting for a resource from some other process.**

Example: Consider two processes A and B. Suppose A is holding a resource 'x' and it wants a resource 'y' held by Process B. Process B is currently holding 'y'. let B wants resource 'x' held by A. So, both A and B compete for the resource held by other process. The result of this competition is deadlock, where both processors stop execution.

Possible conditions for deadlock: E.G. Coffman described 4 possible conditions for dead lock situation.

(i) Mutual exclusion: Only one process can hold a shared resource at a time.

Example: A printer can be accessed by only one process.

(ii) Hold and Wait: A process holds a shared resource by locking and waits for additional resources held by other processes.

(iii) No resource pre-emption: A resource can be released only voluntarily by the process holding it. i.e., resource cannot be released until the process holding it is executed.

(iv) Circular Wait: Let there be n processes P_1, P_2, P_3, …, P_N. Let P_1 is waiting for a resource held by P_2, P_2 is waiting for a resource held by P_3, ……, P_N is waiting for a resource held by P_1. This forms a circular wait queue, which results in deadlock.

Deadlock Handling Methods:

- Using a protocol which ensures that the system will never enter a deadlock state.
- Detecting and removing deadlocks, when they occur.
- Ignoring the problem altogether, pretending that deadlocks never occur in the system. This solution is used by most OSs. (Linux, Windows, etc.)

Deadlock handling schemes:

(i) Deadlock prevention:

We need to ensure that the any of the four possible conditions for dead lock (Coffman's conditions) will never occur.

(ii) Deadlock avoidance:

It can be done by careful resource allocation techniques of OS. It is like traffic light mechanism used to avoid traffic jam.

OSs keep a resource graph in their memory. It is updated on each resource request and release. A deadlock is detected by analysing this graph using graph analyser algorithms. Once deadlock is detected, the system can terminate a process or pre-empt the resource to break deadlock.

Recovery from Deadlock:

(i) Abort all deadlocked processes, or

(ii) Abort one process at a time until the deadlock cycle is eliminated

Livelock: In dead lock, a process enters in wait state for a resource and continues in that state forever without making any progress in the execution. In a livelock condition, a process always does something but is unable to make any progress in completing execution.

Starvation: In multitasking system, starvation is the condition in which a process doesn't get the required resources for a long time. As the time progresses, the process staves on resources. Starvation may arise due to

- ✓ The deadlock preventing mechanism, or
- ✓ Scheduling policy which always honours higher priority tasks or tasks with shortest execution time.

3. The Dining Philosopher's Problem:

Suppose five philosophers P1, P2, P3, P4, P5 are sharing a circular table and they eat and think alternately. There are 5 chop sticks as shown in Fig. 5.

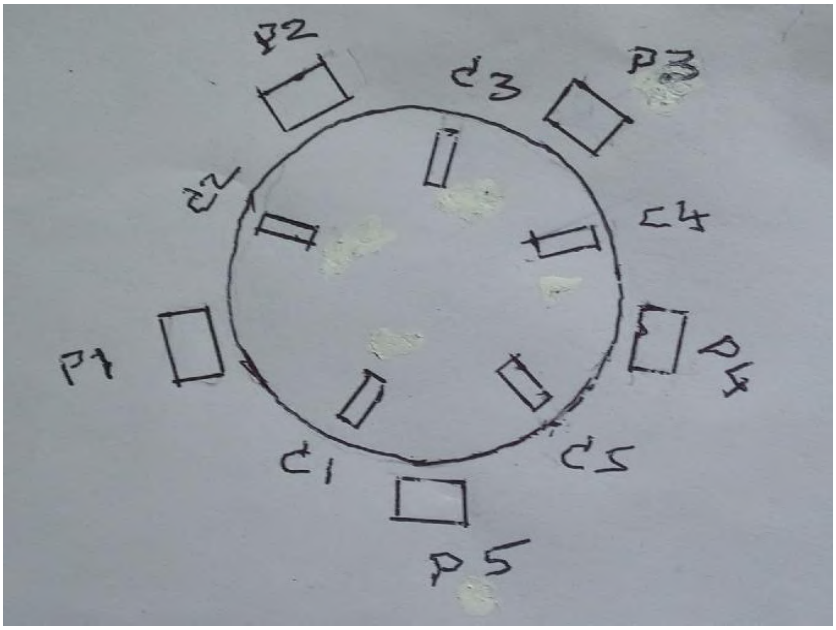


Fig. 5: Philosophers and chop sticks

A philosopher can eat only if he can pick up two chop sticks one on his left side and one on the right. Suppose P1 wants to eat. He needs chop sticks C1 and C2. C1 may be in the hands of P5 or on the table. Similarly, C2 may be with P2 or on the table. If any stick (C1 or C2) is not on the table, P1 cannot eat.

Various possibilities:

- No two adjacent philosophers can eat at a time.
- More than two philosophers cannot eat at a time.
- If 3 persons put the sticks on table, other two may get benefit.
- If a philosopher gets a left stick, he can pick up right stick if it is on the table, If his right person is using it, he has to wait until right person puts it on table.Then
    - He puts left stick on the table. (Then his left person gets it).
    - He waits without putting left stick on table.
- **Deadlock:** Sometimes 5 persons get single stick and wait for the other stick. It is a **deadlock** situation.
- **Racing:** After certain time all the 5 may put the sticks on table. Then also **racing** may occur for stick.
- **Another Racing situation:** When P1 has C5 and P2 has C2. Both compete and try to pick C1. It's like race condition
- **Livelock:** if all the philosophers try to lift the forks the same time, it results in a **livelock.**

- Starvation: Any philosopher may have to wait for a long time and cannot get two sticks. This is called starvation.

Assume that philosophers are processes and sticks are shared resources. Then OS faces the same situation.

Solutions:

(i) Using Round Robin allocation

(ii) Using FIFO allocation

(iii) Imposing rules in accessing the forks: The philosophers should put down the fork in his hand (say left fork) after waiting for a fixed duration for the second fork (right fork). Also, he should wait for fixed time before making next attempt. This solution works to some extent. But,

(iv) Using Semaphores: Each philosopher acquires a semaphore (mutex) before picking up any fork. When a philosopher wants to eat, he checks whether his left and right philosophers are using the fork. It is done by checking the state of associated semaphores. If forks are in use, he waits until the forks are available. When a philosopher finished eating, he puts the fork down and informs his left and right philosophers. It is done by signalling the semaphores associated with forks.

Note: This solution gives maximum concurrency.

4. Product-consumer problem (or Bounded Buffer Problem):

Suppose two processes concurrently access a shared buffer, with fixed size. A thread/process which produces data is called 'Producer thread/process'. A thread/process which consumes data is called 'Consumer thread/process'. Suppose both of them are continuously working using the buffer.

If producer produces data at a faster rate than the rate at which it is consumed by consumer. It causes 'buffer overrun'. i.e., producer tries to put data in a full buffer.

If the consumer works at a faster rate than produce, it leads to 'buffer under run'. i.e., after sometime consumer tries to read data from empty buffer.

Both of these conditions will lead to inaccurate data and data loss. This problem can be rectified in many ways. One simple solution here is the mutual exclusion through 'sleep and wake up' technique.

5. Readers-Writers Problem:

If many processes try to read a shared data concurrently, there is no problem. But, when many processes try to write and read concurrently, it will definitely create inconsistent results.

Example: Suppose one process in a bank system tries to read available balance in an account and other process tries to update the available balance in that account. This leads to inconsistent results.

Proper synchronization techniques are applied to avoid the Readers-Writers problem.

6. Priority Inversion Problem:

'Priority inversion' means inverting the priority of a high priority task with that of a low priority task.

It results in a situation, where a high priority task needs to wait for a low priority task to release a resource which is shared among the high priority task, low priority task, and a medium priority task. It is illustrated by the following example.

- Let A, B and C be 3 processes with high, medium and low priorities respectively.
- Let A and C share a variable 'x' whose access is synchronized by a binary semaphore.
- Let OS Scheduler picked up C to execute. Suppose C needs 'x'. Then C acquires semaphore over 'x'.
- Suppose B started executing, and A enters ready state at his stage.
- Since A has higher priority than B, it pre-empts B. So, A is scheduled for execution.
- A needs 'x'. Since C acquired semaphore, A is put into blocked state.
- C continues its execution.
- Now A has to wait until C executes and releases the semaphore.

This produces unwanted delay in the execution of higher priority task A. This may also lead to missing of deadlines for A.

Problems due to Priority Inversion:

- A system malfunction may occur if a high priority process is delayed and if deadlines are not met. It is quite undesirable when requirements are **time-critical**.
- Reduces the performance of the system.

Work around mechanisms used to handle the priority inversion problem:

(i) Priority Inheritance:

Suppose 2 tasks P and Q are sharing a resource, where P has low priority. If P accesses the shared resource and holds a lock, it inherits the priority of Q. i.e., priority of P is boosted to

priority of high priority task Q. In other words, P inherits the priority of Q. P continues to execute and holds the shared resource, till it completes the execution.

When P releases the shared resource, its priority is brought to its original value. Q gets a lock over it and starts execution.

This method checks the priorities of all tasks which tries to access shared resources and adjust the priorities dynamically. It is a run-time overhead. It is only a work around. It will not eliminate the waiting delay of Q.

(ii) Priority Ceiling:

Here, a priority is associated with each shared resource. The priority associated with each resource is the priority of highest priority task which uses that particular resource. This priority level is called 'Ceiling priority'. Whenever a task accesses a shared resource, the scheduler elevates its priority to ceiling priority of the resource.

If the low priority task accesses the shared resource, its priority is temporarily boosted to priority of highest task that may share that resource.

This eliminates the pre-emption of the task by other medium priority tasks. Once the task completes its execution, its priority is brought back to its original level.

Advantages:

- Concurrent access of shared resources is automatically handled by boosting the priorities. There is no need for synchronization techniques like locks.
- All the overheads are at compile time instead of run-time.

Drawback:

It may produce hidden priority inversion because the priority of a task is always elevated irrespective of other higher priority tasks that want the shared resources. This always gives the low priority task the luxury of running at high priority when it accesses shared resources.

TASK SYNCHRONIZATION TECHNIQUES

Let us define 'Critical section of code', before going through the synchronization techniques.

Critical section of code:

It is the segment of code where processes access shared resources, such as common variables and files, and modify them by write operations.

Synchronization is essential while accessing critical section of code. This can be achieved through mutual exclusion mechanism. i.e., only one process can act upon critical section at a time.

MUTUAL EXCLUSION MECHANISM

Mutual exclusion ensures that no two processes can exist in the critical section at any given point of time.

Consider two processes A and B. Let A is running and it entered its critical section. Let B has higher priority than A and it also has access to critical section. If process B continues and enters critical section, it results in racing condition. In this situation, Then, there are 2 mutual exclusion techniques for syncroniation. They are discussed below.

1. MUTUAL EXCLUSION THROUGH BUSY WAITING (SPIN LOCK)

This technique uses a 'lock variable' or 'lock'. If a process/thread is already in critical section it sets lock to '1'. Otherwise, lock is set to '0'. Each process/thread checks this lock before entering critical section. This lock is also called 'Spin lock'.

Spin Lock and busy waiting:

Suppose a low-priority task is running and a little time is only needed for its completion. Then it is better to see that a higher priority task will not pre-empt it during that little time. A spin lock is used to do this. If the high-priority task wants to access a shared resource, it spins or waits in a loop until the lock becomes available. This enforces mutual exclusion. Thus, a spin lock results in busy waiting.

Advantage of spin lock is that it doesn't have any overhead due to context switching. It is only held for a short time, and it is useful for multiprocessor systems.

This method basically needs: - (i) Reading (ii) Testing and (iii) Setting the lock variable.. There is no atomic (single) instruction to combine these operations. These are implemented using multiple low-level instructions, which depend on the instruction set of the processor. This can be achieved with the combined support of hardware and software. Most of the processors support an instruction 'TSL' (Test and Set Lock) for this purpose. This instruction is processor specific.

Drawbacks:

'Busy waiting' technique makes the CPU always busy. It is because it involves in continuous checking for a lock. This results in the wastage of CPU time and leads to high power consumption. This is not suited for an ES powered on battery. An alternative to this technique is 'Sleep & Wakeup' mechanism.

2. MUTUAL EXCLUSION THROUGH SLEEP & WAKEUP

Suppose a process is holding lock on critical section and another process tries to access the section. Then the second process is made to sleep, i.e., enters wait state. When first process leaves the critical section, it sends a wakeup message to the sleeping process and wakes it up. The implementation of this policy is OS dependent.

SEMAPHORE

Semaphore is a technique used for mutual exclusion through Sleep & Wake up mechanism

Important features of semaphore:

- Semaphore is a system object.
- Basically, a semaphore is an integer variable **shared among multiple processes.**
- **It is used for process synchronization, and access control for a common resource in a concurrent system such** as a multitasking OS.
- If a process wants to access a shared resource, it first acquires semaphore and indicates the other processes the same.
- The initial value of a semaphore depends on the problem. Usually, the number of resources available is taken as the initial value.

Advantages of semaphores:

- ✓ Used in process synchronization.
- ✓ Used to avoid deadlocks.

Disadvantage: It may lead to a priority inversion.

Types of semaphores:

The shared resources can be either exclusively used by one process (e.g.: The display device of an ES needs exclusive access by a process). or shared by a few processes (e.g.: a hard disk is shared by a few processes.) Based on this situation, semaphores are classified as Binary semaphores and Counting Semaphore, respectively.

Binary Semaphore (or Mutex): When a process owns a binary semaphore, it allows only one process to access a shared resource. Its implementation is OS kernel dependent.

Only one process/thread can own the mutex object at a time. The state of mutex object is set to signalled when it is **not owned** by any process/thread, and set to non-signalled when it is **owned** by any process/thread.

When a thread is holding Mutex lock in its critical section, other threads must wait till the lock is released.

A binary semaphore can have only two integer values: 0 or 1. It's simpler to implement and provides mutual exclusion. We can use a binary semaphore to solve the critical section problem.

Example applications of binary semaphores:

Two trains are in the nearby stations. There is only one track. They have to move in the opposite direction. Then, at a time, one train is given signal and other has to wait.

Counting Semaphore:

Binary semaphore limits one process to use a shared resource. A Counting semaphore allows a fixed number of processes to access a shared resource.

Counting semaphore maintains a count between zero and a value and limits the resource to this count.

We can use Counting semaphores to resolve synchronization problems like resource allocation. Here, semaphore count is the number of available resources. If new resources are added, semaphore count automatically incremented and if some existing resources are removed, the count is decremented.

States of Counting semaphore:

- State of the counting semaphore is set to be 'signalled' when the count of semaphore is > 0. Then, the count associated with a semaphore object is decremented by one, when a process/thread acquires it and the count is incremented by one when a process/thread releases the semaphore object.
- The state of the Counting semaphore is set to be 'non-signalled' when the semaphore is acquired by the maximum number of processes/ threads that the semaphore can support (i.e., when the count becomes zero)

Example:

We use them in the dining philosophers' problem.

Critical section Objects:

In Windows CE, a critical section object is same as the mutex object except that critical section object can only be used by the threads of a single process.

Illustrate binary and counting semaphores by real world examples:

Real word example for Binary Semaphore: Hotel rooms

- Any user who pays and follows the norms of the hotel can avail the rooms for accommodation.

- A person wants to avail the hotel room facility can contact the receptionist. If room is available, the receptionist will hand over the room key to the user. If room is not available currently, the user can register his name in advance-booking register.
- When a person gets a room, he/she is granted the exclusive access to room facilities like TV, telephone, toilet, etc. (Not sharing like in dormitory)
- When a user vacates the room, he gives the keys back to receptionist. The receptionist informs the next user who booked room in advance.

Real World Example for Counting Semaphore: A dormitory

- ✓ Let a dormitory contains fixed number of beds.
- ✓ Let at any point of time, a maximum of 5 users can share it.
- ✓ If a person wants a bed, he can contact the dormitory caretaker.
- ✓ If beds are available, caretaker will give keys to the user. If not available, the user can register his name in advance-booking register.
- ✓ Those who are availing the dormitory share the facilities like TV, telephone, toilet, etc.
- ✓ When a user vacates, caretaker gets keys from him and informs the next person in the register

*****