

GOVT. POLYTECHNIC BALASORE



[LECTURE NOTE]

(c) Embedded System TH 4

DIPLOMA

4TH SEMESTER, E & TC ENGINEERING

**PREPARED BY:-ER. YOGASAKTI YOGAMAYA
(Lect. In E&TC)**

**DEPT. OF ELECTRONICS & TELECOMMUNICATION
ENGINEERING**

Session: 2025-26

Yogasakti Yogamaya

Embedded Systems

TH:4(c)- EMBEDDED SYSTEMS

L			Total Marks: 100	Course Code: ETCPE202(c)		
T						
P						
3						
0						
0						
Total Contact Hours					Theory Assessment	
Theory			: 45Hrs		End Term Exam	70
					Progressive Assessment	30
Pre Requisite			: Nil			
Credit			3		Category of Course : PE	

RATIONALE:

In the rapidly growing digital world, the role of embedded systems is increased in various domains such as industrial and home automation, entertainment systems, medical equipment and many more. The core of all embedded systems is powered by electronic hardware and associated software. It is therefore evident to impart the knowledge of the related technology and hands on skills to develop and maintain electronics hardware based embedded systems. Through this course, students can develop their own embedded system which is application specific to solve real time problem by using open source platform. This course is very important for students who want to work in the automation sector of electronic industry.

LEARNING OUTCOMES:

After completion of the course, the students will be able to

- Describe the advanced concepts of Embedded System Architecture
- Design systems for various embedded applications.
- Design software systems such as RTOS using embedded controllers.
- Analyze Hardware Software co-design trade-offs
- Use embedded C programming language to maintain embedded systems

DETAILED COURSE CONTENTS

Unit No.	Topic/Sub-Topic	Allotted Time (Hours)
I	Introduction to Embedded System 1.1 Block diagram of embedded system with hardware components 1.2 Hardware and Von-Neumann architecture 1.3 RISC and CISC processors 1.4 Characteristics of Embedded System 1.4.1 Processor ✓ 1.4.2 Power 1.4.3 Memory 1.4.4 Operating system ✓ 1.4.4 Reliability ✓ 1.4.5 Performance ✓ 1.4.6 Power consumption ✓ 1.4.7 NRE cost ✓ 1.4.8 Unit cost, size, flexibility ✓ 1.4.9 Time to prototype and time to market ✓ 1.4.10 Maintainability ✓ 1.4.11 Correctness and safety 1.5 Classification of Embedded System 1.5.1 Small scale 1.5.2 Medium scale 1.5.3 Sophisticated 1.5.4 Standalone 1.5.5 Real time	15
II	Embedded System Design 2.1 The concept of embedded systems design 2.2 Embedded microcontroller cores 2.3 Embedded memories.	5
III	Examples of embedded systems 3.1 Technological aspects of embedded systems 3.1.1 Interfacing between analog and digital blocks 3.1.2 Signal conditioning 3.1.3 Digital signal processing 3.1.4 Sub-system interfacing 3.1.5 Interfacing with external systems 3.1.6 User interfacing 3.2 Design tradeoffs due to process compatibility and thermal considerations	10
IV	Software aspects of embedded systems 4.1 Real time programming languages 4.2 Operating systems for embedded systems.	5
V	Programming Embedded System in C 5.1 Data transfer 5.2 Arithmetic and Logical operations 5.3 Decision Control & Looping 5.4 Timer/Counter Programming with Embedded C for Microcontroller 5.5 Interrupt Control programming with Embedded C for Microcontroller	10

Unit - I

Introduction to Embedded System.

System: System is a set of interrelated parts/components which are designed/developed to perform common task or to do some specific work for which it has been created.

Embedded: Embedded means something which is integrated or attached to another thing.

Embedded System: Embedded System is a computational system that has embedded software in a computer hardware in order to perform a given task.

→ It can be said as a dedicated computer system has been developed for some particular task.

or

* An embedded system is a custom-built special purpose computer used for a specific purpose. It is the system that combines both hardware and software to perform a certain task.

→ It can be used as an individual system or a part of another large system.

Examples of Embedded System:

1. Digital Watch
 2. ~~Watch~~ Washing Machine
 3. Toys
 4. Television
 5. Digital phones
 6. Camera
- etc.

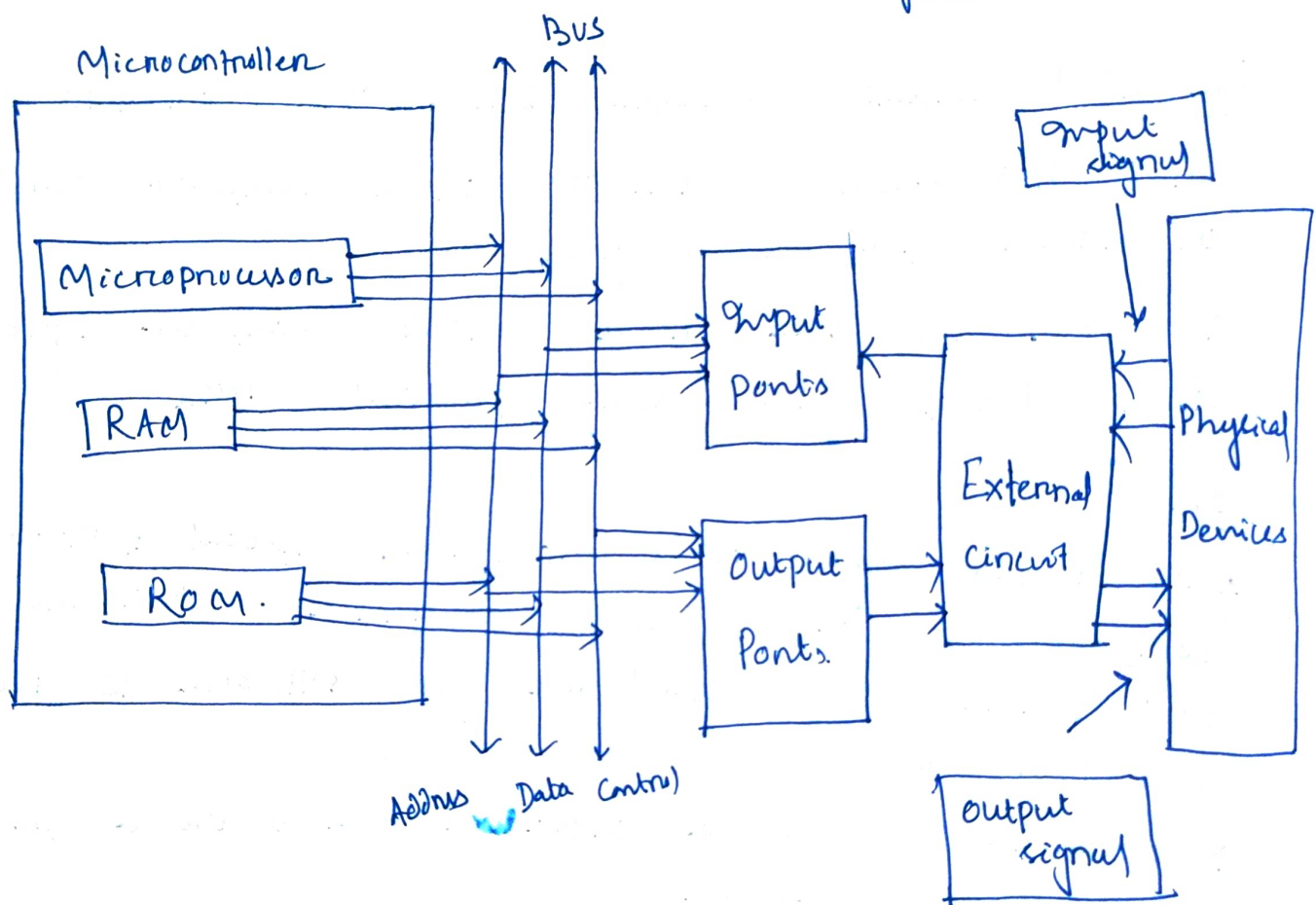
Applications:

1. Consumer Electronics (cell phone, pager, digital camera, camcorder, VCR, calculator, video game etc.)
2. Home appliances (lighting system, TV, washing machine, microwave, answering machines etc.)
3. Office automation (fax, copier, printer, scanner)
4. Business Equipment (cash register, credit card check-in, alarm system)

card readers, product scanners, ATM).

5. Automobiles (transmission control, cruise control, fuel injection, antilock brakes & active suspensions).

Block Structure of Embedded System:



Advantages of Embedded System:

- i) Small size
- ii) Enhanced real-time performance.
- iii) Easily customizable for specific application.
- iv) Low power consumption
- v) Reliability & stability.

Disadvantage:

- i) High development cost
- ii) Time-consuming design process
- iii) As it is application specific, less market analysis

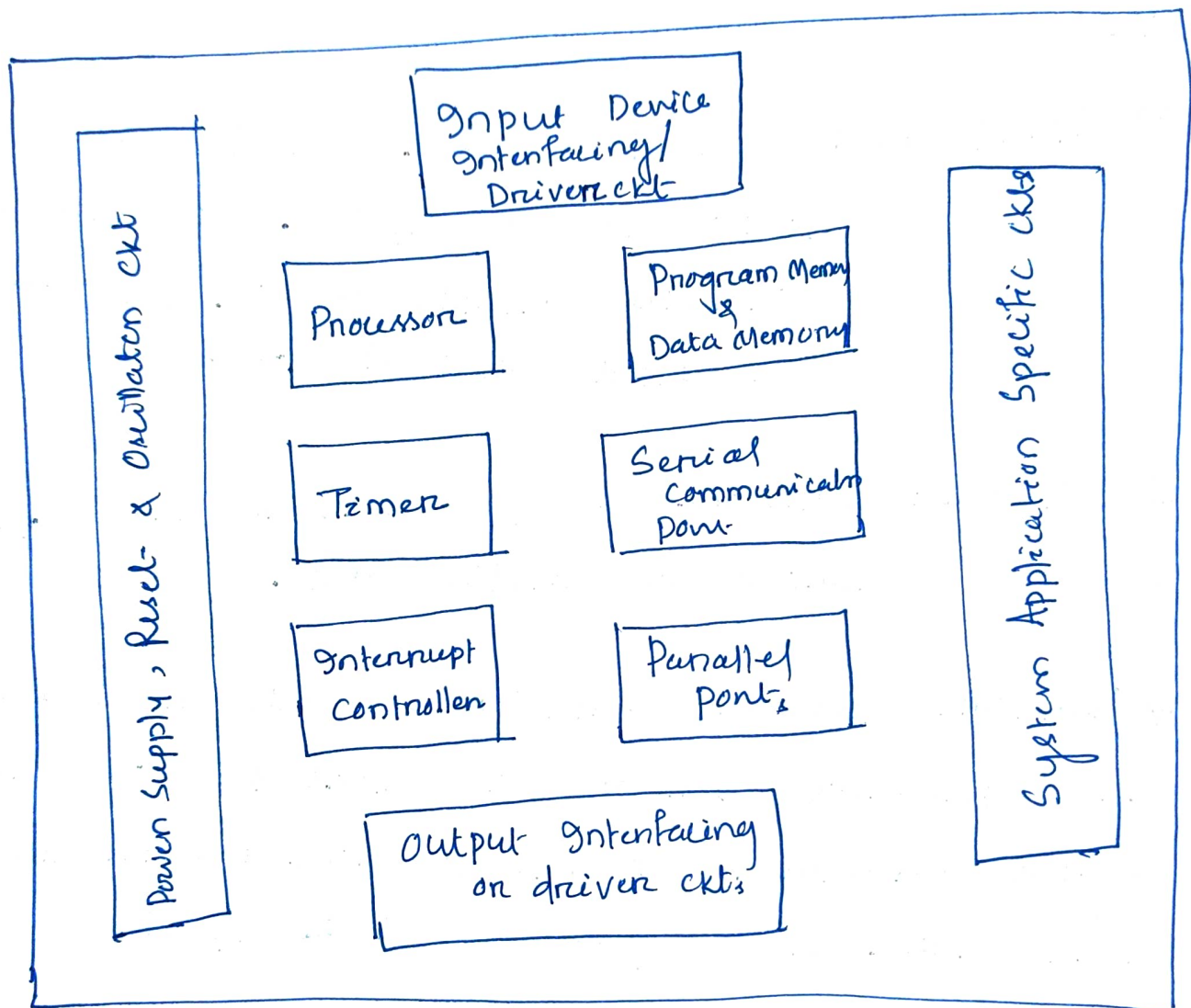
Embedded System Components :

An embedded system has 3 main components embedded into it.

- i) Hardware : Embedded system embeds hardware similar to a computer. The hardware includes embedded memory, peripherals & input-output devices.
- ii) Software : Software usually embeds in the ROM, Flash memory or media card.
- iii) Real time Operating System (RTOS) : The RTOS supervises the application software & controls the access to the system resources. It enables finishing the execution of the tasks of a program within specified time intervals.

Block diagram

Block diagram of Embedded Systems with hardware component



- 1) Processor: It is the heart of the embedded system
- It has two essential unit: Control unit & Execution unit.
 - Control unit fetches instructions from memory & execution unit includes ALU & circuits to perform execution of the instructions for a program control task.

→ Processor chip or core in an embedded system can be a

i) Microprocessor

ii) Application specific instruction - set Processor - (Microcontroller) ^{DSP}

iii) Single purpose processor as an additional processor.

ii) Power supply: Most of the system have their own power supply. It may be powered by external power supply e.g. USB based embedded system.

Reset: Reset means that the processor begins processing of instructions from starting address set by default in program counter on ~~power~~ ^{power} up.

CLK oscillator: ~~CLK~~ The ~~CLK~~ CLK ckt controls execution time of instructions, CPU machine cycles. The frequency depends on the ~~CLK~~ processor ckt execution rate.

Interrupt Controller: It is an interrupt handling mechanism which must exist in embedded system to handle interrupts from various processes and handling multiple ~~processor~~ interrupt simultaneously.

Timers: Timers ckt is suitably configured as system clk or RTC (Real time clock). To ~~do~~ schedule various tasks & for real time programming on RTC on system clock is needed.

Program & data memory:

In Embedded Systems various forms of memories are used.

- i) Internal RAM
- ii) Internal ROM | PROM | EPROM.
- iii) External RAM.
- iv) Internal Caches
- v) Internal EEPROM | Flash etc.

→ The ROM is called code memory that stores program code. It is a non volatile memory which means it stays stored program when power supply is removed.

→ RAM is called the data memory & is a volatile memory which means it is used for temporarily storing of information & removed when power supply turned off.

Microprocessor:

* A microprocessor is a single VLSI chip that has a CPU & additional unit.

→ By using VLSI technology the entire CPU built into a single chip & called microprocessor.

→ CPU is a unit that centrally fetches and ~~execute cycle is~~ processes a set of general purpose instructions using the processor.

→ The fetch and execute cycle of microprocessor is guided by a clock.

→ A new generation processor operates at 800 MHz to 3.2 GHz clock frequency.

* A microprocessor is used as general purpose processor where software is located in the external memory chip on secondary memory.

* A microprocessor on general purpose processor chip on core can be used for the embedded systems design due to following characteristics:

i) Processing by the known instructions available at predefined general purpose instruction set results in fast system development.

ii) Once the ckt board & I/O interfaces are for general purpose processor (GPP), these can be used for a new system by just changing embedded software in ROM or Flash.

iii) Ready availability of a compiler or framework facilitates embedded software development in high level language.

iv) Ready availability of OS & processor specific Application program interfaces. (APIs)

Microcontroller:

- A microcontroller is an integrated chip or core in VLSI or SoC (System on chip).
- A microcontroller has the processor, memory & several other hardware units interfaced together. It therefore, functions as a microcomputer.
- A microcontroller is the essential component of a control or communication circuit.
- Microcontrollers are particularly suited for use in embedded systems for real-time control applications.
- Ex: small scale embedded system: 8051 family, PIC 16F8X, AVR 32
- Medium scale Embedded system: 8051 MX, PIC 16F876, PIC18 etc.
- Large scale Embedded system: ARM family Cortex M3, Texas Instruments
- Choosing a specific microcontroller as a processing unit is as per the application specific features in a system.
- A microcontroller is used when a small or part of the embedded software has to be located in internal memory & when the on chip functional units like interrupt-handler, port, timer, ADC, PWM, ~~CAN controller~~ CAN controller, ZigBee or USB interfaces are needed.
- Important Microcontroller chips for embedded system are Intel 8051 & 80251 Verum, ATMEL mega series 8/168/324/1284/2560, ~~ARM 9~~ ARM 9, ARM10 etc.

Microprocessor

1) Microprocessors are multitasking in nature. Can perform multiple tasks at a time. For example, on computer we can play music while writing text in text editor.

2) RAM, ROM, I/O ports & Timers can be added externally & can vary in numbers.

3) Designers can decide the number of memory or I/O ports needed.

4) External support of external memory & I/O ports makes a microprocessor-based system heavier & costlier.

5) External devices require more space & their power consumption is higher.

Microcontroller

1) Single task oriented. For example, a washing machine is designed for washing clothes only.

2) RAM, ROM, I/O ports & Timers cannot be added externally. These components are to be embedded together on a chip & are fixed in numbers.

3) Fixed numbers for ~~numbers~~ memory on I/O makes a microcontroller ideal for a limited but specific task.

4) Microcontrollers are lightweight & cheaper than a microprocessor.

5) A microcontroller based system consumes less power & takes less space.

Von Neumann Architecture:

→ The von Neumann architecture was first proposed by a computer scientist John von Neumann.

→ Von Neumann architecture is a computer design where instructions & data are stored in same memory space.

→ In this architecture, one data path or bus exists for both instruction & data.

→ As a result the CPU does one operation at a time.

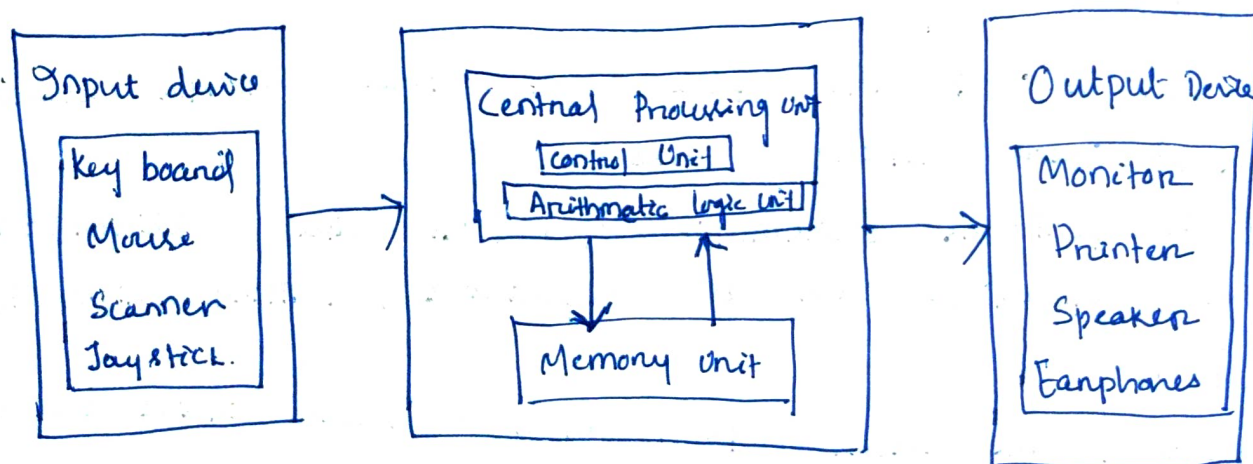
→ It either fetches an instruction from memory or performs read/write operation on data.

→ So an instruction fetch and a data operation cannot occur simultaneously, sharing a common bus.

→ Historically there have been 2 types of computers.

i) Fixed Program computers: Their function is very specific, and couldn't be reprogrammed. e.g → calculator.

ii) Stored program computers: These can be programmed to carry out many different tasks, applications are stored on them.



→ Von-Neumann architecture supports a simple hardware. It allows use of single, sequential memory.

Components of Von Neumann Architecture:

It is made up with three main components.

i) CPU

ii) Memory

iii) I/O Devices

i) CPU (Central Processing Unit):

→ The CPU is the main part of the computer that controls all operations.

→ It is made up of control unit, main memory & Arithmetic logic unit (ALU).

→ CPU handles instructions from programs, processes data, store information & produce results.

Control unit (CU):

→ The control unit manages how the processor works by sending control signals.

→ It controls data flow, input & output operations & fetches the instructions from memory for execution.

ALU:

→ The ALU is the part of CPU that handles calculation & decision making tasks.

→ It performs arithmetic operations like addition, subtraction, logical operations such as comparisons, shifting, AND, OR etc.

Registers:

→ Registers are fastest type of memory located inside the CPU.

→ They temporarily store information that the processor is currently working on, making program execution & operation faster & more efficient.

[I/O device with a Hand drehen]

Key characteristics:

1. Single memory for data & instructions.
2. Shared Bus for data & instructions.
3. Sequential ~~exec~~ execution of instructions.

Applications:

- 1) Personal Computer & Laptops
- 2) Smartphones & Tablets
- 3) Embedded Systems
- 4) Server & cloud computing
- 5) Gaming consoles

Advantages:

- 1) Simplicity
- 2) Cost effective
- 3) Flexibility

Disadvantage:

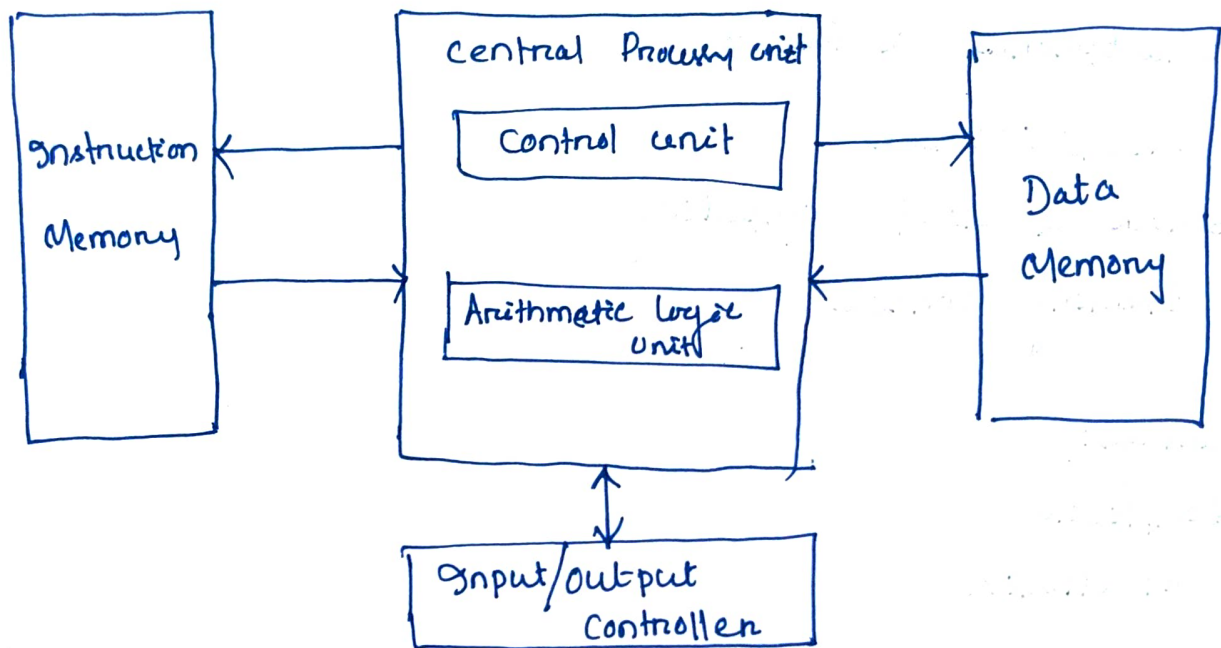
- 1) Slow as ^{we} shared bus.
- 2) Memory corruption

Von Neumann bottleneck:

- In this architecture instructions can only be done one at a time & can only be carried out sequentially.
- Both these factors hold back the competence of the CPU.
- This is ~~ref~~ referred as 'Von Neumann bottleneck'.
- (Bottleneck is any situation where work is delayed).
- To overcome this we can ~~can~~ add more ~~ca~~ caches to CPU, Prefetch instructions/data to cache etc.

Harvard Architecture:

- Harvard Architecture is a computer design model program instructions & data are stored in separate memory units, that are accessed through independent buses.
- This separation allows the processor to fetch instructions & access data simultaneously which makes faster performance.



Working Principle:

- In Harvard architecture, fetching an instruction from ^{instruction} memory & read/write data from/to data memory happens at the same time without waiting for one to finish.
- Ex: While an instruction is being executed, the next instruction can be fetched simultaneously by speed up process.

Bus: Buses are used as signal pathways.

In this architecture there are separate buses for both instruction & data. Types of Buses:

- i) Data Bus
- ii) Address data Bus
- iii) Instruction Bus
- iv) Instruction address Bus.

Components of Harvard Architecture:

- i) ALU ii) CU iii) Input/output system.

[ALU & CU same as von-Neumann]

Input/output System:

- Input devices are used to read data into main memory with help of CPU input instructions.
- The information from a computer as output is given through output devices.
- The computer gives the results of computation with the help of output devices.

Application:

- 1) Digital signal processor - Audio, video processing, radar system, Telecommunication.
- 2) Microcontrollers : Embedded systems in consumer electronics, IoT devices, industrial automation.
- 3) Network Processors : Routers, Switches
- 4) Automotive systems : Engine control units, Advance driver assistance system (ADAS) etc.

Advantages:

- 1) Faster Processing
- 2) Improved Security
- 3) Efficient use of Resources

Disadvantages:

- 1) Complexity
- 2) Higher cost
- 3) Less flexibility competition

Features:

- 1) Separate memory spaces
- 2) Fixed instruction length
- 3) Parallel instruction & data access
- 4) More efficient memory ^{usage} ~~usage~~
- 5) Suitable for embedded system
- 6) Limited flexibility.

Von Neumann Architecture

- i) It is ancient computer architecture.
- ii) Same physical memory address is used for instructions & data.
- 3) There is common bus for data & instruction transfer.
- 4) Two clock cycles are required to execute single instruction.
- 5) It is cheaper in cost.
- 6) It is used in personal computers & small computers.
- 7) Simple in design.
- 8) Higher Speed, thus less time consuming.

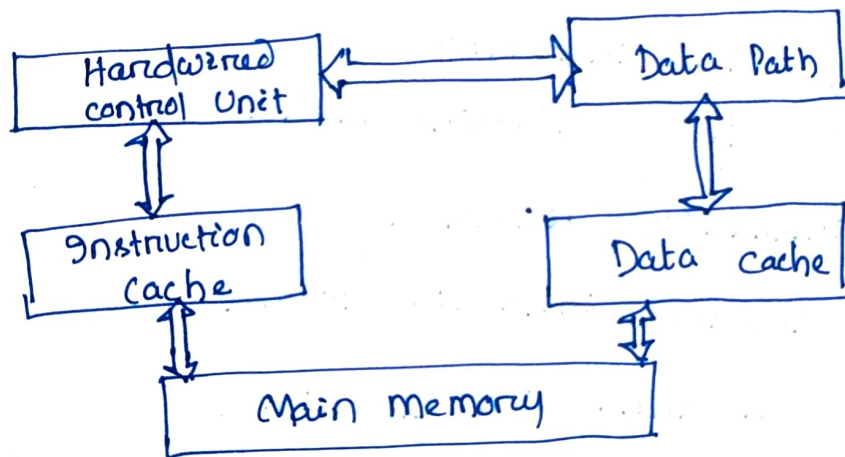
Harvard Architecture

- i) It is modern computer architecture.
- 2) Separate physical memory address is used for instructions and data.
- 3) Separate buses are used for transferring data & instructions.
- 4) An instruction is executed in a single cycle.
- 5) It is costlier than Von Neumann Architecture.
- 6) It is used in microcontrollers & signal processing.
- 7) Complex in design.
- 8) Slower in speed, thus more time-consuming.

RISC Processor:

- A reduced instruction set computer (RISC) is a computer architecture designed to simplify the individual instructions given to the computer to accomplish tasks.
- Ex: ARM, SPARC
- It is designed for high performance, efficiency & speed by using a small, highly optimized set of instructions.
- It ~~process~~ ~~each~~ ~~instruction~~ ~~in~~ ~~single~~ ~~clock~~ ~~cycle~~.
- RISCs are used when the system needs to perform ~~intensive~~ intensive computation, Ex: in speech processing system.

RISC architecture:



- RISC has a small set of instructions which generally include register to register operations.
- Thus data is stored in processor registers for computations & results of computations are transferred to the memory using store instructions.
- All operations are performed within the register of the CPU.
- All instructions have simple register addressing ~~mode~~ addressing & hence use less number of addressing modes.

→ RISC uses relatively simple & easy to decode instruction format.

→ Hence ~~instruction~~ it has fixed instruction length.

→ The RISC processor can execute one instruction per clock cycle.

→ This is done by Pipelining, which involves overlapping the the fetch, decode & execute phase of two or three instructions.

→ It takes less time to execute its program.

→ It is used in smart phones, IoT, smart watch, tablet, Routers & switches, video/image processing, home automation etc due to low power consumption.

Features:

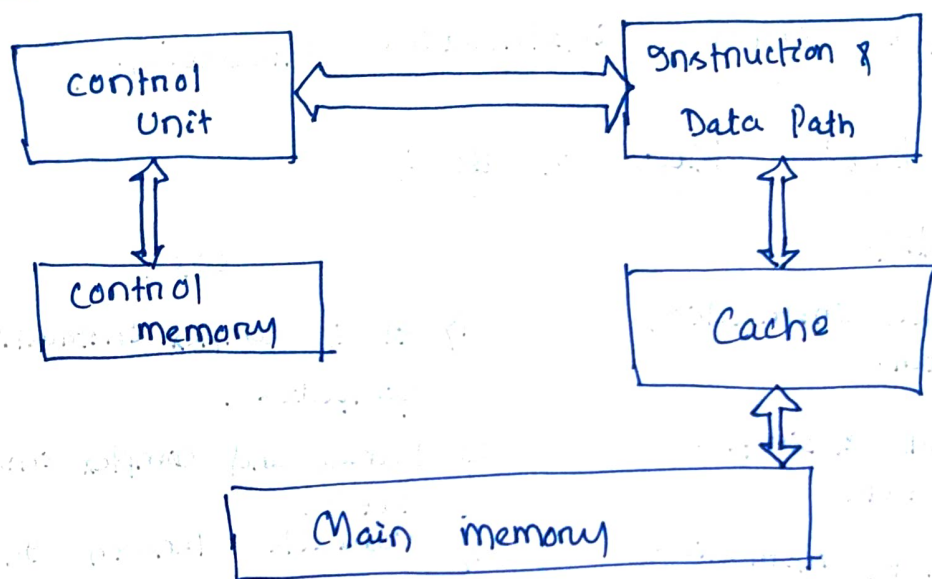
1. Relatively few instructions
2. Relatively few addressing mode.
3. Fixed instruction length.
4. Easily decoded instruction format.
5. Single cycle instruction execution.
6. It can be hardware rather than microprogrammed control.
7. Limited to access memory, operate directly on registers.
8. ~~large~~ None general purpose registers.
9. Optimized for pipelining.

CISC Processor:

→ A complex instruction set computer (CISC) is a computer architecture which includes a large set of instructions many of which are complex & perform multiple operations (e.g. memory access & computation).

→ Ex: 8086 microprocessor

→ Architecture:



→ It has many addressing modes & instructions.

→ It has a simpler hardware.

→ Execution unit consists of a control memory.

→ The control memory saves the microinstructions & nano instructions for each instruction which is processed.

→ The complex instructions take greater numbers of cycles.

→ Thus CISC has a variable-length encoding system of instructions.

→ It provides direct manipulation of operands that are in memory.

→ It is used in Laptop, Desktop computers.

Features:

- i) complex instructions
- ii) variable instruction length
- iii) Memory to memory operation
- iv) Hardware complexity
- v) Large instruction set
- vi) More addressing mode
- vii) Multiple cycle instruction execution

Difference betⁿ RISC & CISC:

RISC

- 1) It is Reduce Instruction Set computer.
- 2) It has small & simple instruction set.
- 3) Fixed length instructions
- 4) One cycle per instruction
- 5) More general purpose Registers
- 6) Simple & limited addressing mode
- 7) Larger code size.
- 8) Simple hardware
- 9) Ex: ARM, RISC-V, MIPS
- 10) Register to Register ~~code~~ LOAD & STORE
- 11) Emphasizing software efficiency
- 12) Consume less power

CISC

- 1) It is Complex instruction Set computer.
- 2) Large and complex instruction set
- 3) Variable length instructions
- 4) Multiple cycle per instructions
- 5) Fewer Registers
- 6) Complex & many addressing modes
- 7) Smaller code size
- 8) Complex hardware
- 9) Ex: Intel 8086, Intel 80386
- 10) Memory to Memory LOAD & STORE
- 11) ~~Emph~~ Focusing on hardware capability.
- 12) Consume higher power.

Characteristics of Embedded System:

21

① Single Functioned: An embedded system usually executes a specific program repeatedly.

Ex: A pager is always a pager. But a desktop system execute a ~~variety~~ variety of programs like spreadsheet, word processor, video games etc.

→ Embedded system program can also be updated with a newer program version. Ex: Cell phones are updated such manner.

→ In ~~some~~ Embedded system several programs swapped in and out of a system due to size limitation.

→ Ex: Some missiles run one program while in cruise mode, then load 2nd program for locking on to target.

2) Tightly constrained: All computing system have constraints on design metrics.

→ A design ~~metric~~ metric is a measure of an implementation's features such as cost, size, performance & power.

→ Embedded system often must cost just a few dollars, must be sized to fit on a single chip, must perform fast enough to process data in real time, & must consume minimum power to extend battery life or prevent necessity of a cooling fan.

3) Reactive & Real time:

→ Many embedded systems must continually react to changes in the system's environment & must compute certain results in real time without delay.

→ Ex: A car's cruise controller continuously monitors and reacts to speed & brake sensors.

22 { It must compute acceleration & deceleration amounts repeatedly within a limited time

→ It is the ability of embedded system to operate in real time.

→ This means they can respond to external events or inputs immediately or within a strict time frame.

→ So timing is ~~very~~ as important as functional requirements.

→ Ex: In car's airbag system, the airbag must deploy within milliseconds after a collision is detected. Any delay could result in serious injury or loss of life.

4) Efficiency & Performance:

→ Embedded systems are designed for efficiency & performance.

→ Since they are application specific, they can be optimized for the specific task they are design to perform.

→ This includes optimal use of memory, efficient code execution & minimal power consumption.

→ The performance of Embedded system depends of execution time of system.

5) Reliability & Stability:

→ Reliability & stability are key characteristics of embedded system.

→ They often operate in environments where failure is not an option.

→ For instance, the embedded systems in a ~~spacecraft~~ ^{spacecraft} must operate flawlessly in the harsh environment of space.

This use of can

→ This requires careful design, rigorous testing & use of fault-tolerant techniques to ensure the system can handle unexpected situations.

6) NRE cost: (Nonrecurring Engineering Cost): The one-time monetary cost of designing the system.

→ Once the system is designed, any number of units can be manufactured without incurring any additional design cost; hence the term nonrecurring.

7) Unit cost: The monetary cost of manufacturing each copy of the system, excluding NRE cost.

8. Size: The physical space required by the system, often measured in bytes for software & gates or ~~the~~ transistors for hardware.

9. Power: The amount of power consumed by the system, which may determine life time of a battery, or the cooling requirements of the IC, since more power means more heat.

10. Flexibility: The ability to change the functionality of the system without incurring heavy NRE cost.

→ Software is typically considered very flexible.

11. Time to Prototype: The time needed to build a working version of the system, which may be bigger or more expensive than the final system implementation, but it can be used to verify the system's usefulness and correctness & refine the system's functionality.

24

12) Time to Market: The time required to develop a system to the point that it can be released and sold to the customer. The main contributors are design time, manufacturing time & testing time.

13) Maintainability: The ability to modify system after its initial release, especially by designers who did not originally design the system.

14) Correctness: We can check the ^{system} functionality throughout the process of designing the system, and we can insert test test circuitary to check that manufacturing was correct.

15) Safety: The probability that the system will not cause harm.

Classification of Embedded System:

By considering degree of design complexity & System performance, embedded system can be classified into following types.

① Small Scale Embedded Systems:

- These systems are designed using an 8 bit or 16 bit microcontroller or DSP.
- They can be powered by a battery.
- The processor uses very less/limited resources of memory & processing speed.
- These systems does not act as an independent system. they act as any component of computer system but did not compute ~~to~~ a dedicated for a specific task.
- Ex: Electronic toys, digital watches, simple calculator.
- They also have relatively lower hardware & software complexity & involve board level design.
- The main programming tools used for these systems include an editor, cross assembler & integrated development environment (IDE).
- May or may not be multitasking & may or may not use an RTOS.

[RTOS: Real time operating system: RTOS is a special kind of operating system design to handle tasks that need to be completed quickly & on time. Ex: Airline traffic control system,

(2) Medium Scale Embedded System:

- Medium scale embedded systems are designed using an 16 bit or 32 bit microcontroller or DSP or ~~Reduced~~ RISC.
- It is faster than that of small scale Embedded systems.
- It can also be designed with available single purpose processors.
- Using real-time operating system (RTOS) for multitasking.
- Uses complex software design using programming tools: C/C++/Visual C++/Java, RTOS.
- Uses software tools for providing the solutions to the hardware complexities.
- ~~these comp~~ The VLSI design form of these systems use the readily available IP (Intellectual Property) for various functions. examples: bus controller & USB interface.
- [IP → refers to reusable, pre-designed & verified unit or logic that is integrated into system-on-chip (SOC) or ASIC.
- These systems have both hardware & software complexities.

3. Sophisticated Embedded System:

- It is also known as Large scale Embedded System or complex Embedded System.
 - Sophisticated embedded systems have enormous hardware & software complexities.
 - They are used in high performance ~~32 bit or 64 bit~~ ^{applications}
 - These are designed on 32 or 64 bit RISC.
 - Needs hardware & software co-design.
 - Needs component integration in final system.
 - High performance Real-time Operating System (RTOS) are typically used in complex Embedded systems for managing, scheduling & prioritizing tasks.
 - For these system programming tools may not be readily available at reasonable cost or may not be available at all. There may be a need to develop a compiler or retargetable compiler.
 - These systems are used for cutting edge applications that need hardware & software co-design & components integration in the final system.
- Ex: Mobile computing system.

Hardware Complexity:

① Resource constraints } limited memory, power, cost
 } limited speed, efficient hardware
 } Selection.

2) Physical Integration - Designing PCB that incorporate micro controllers, sensors, actuators & communication module with-in strict size & power limitation.

3) Reliability & Performance - High level tasks requires precise timing, necessity high speed, often multi-core, processors.

4. Analog/Digital Interface: Managing signal processing betⁿ physical world & digital processor.

Software Complexity:

- (1) Real time constraints: Software must adhere to rigid deadlines to prevent software failure.
- (2) Multitasking: Managing multiple task simultaneously.
- (3) Firmware & Drivers: Writing low level code to control hardware directly.
- (4) Optimization - Optimizing code for memory footprint & power efficiency.

Stand-alone Embedded System:

- It can produce output without a host computer.
- It works independently.
- It does not depend on external ~~ex~~ sources or networks to perform their functions.
- All the necessary resources, data & instructions are typically stored or provided within the system itself.
- These systems are typically designed for a specific purpose or task. They perform that ~~task~~ task efficiently & do not hold the flexibility to adopt new functions or update their firmware through network connections.
- These systems have all the embedded system components they need to function enclosed within a single unit.
- As it operates independently & are not connected to external networks, it is often considered as more secure & ~~less~~ & less susceptible to external vulnerabilities like hacking & or ~~unauthorized~~ unauthorized access.
- Their isolation from the external world can enhance their reliability for specific tasks.
- Home appliances device like microwave ovens, washing machines, and refrigerators often contain these systems to manage their operation.

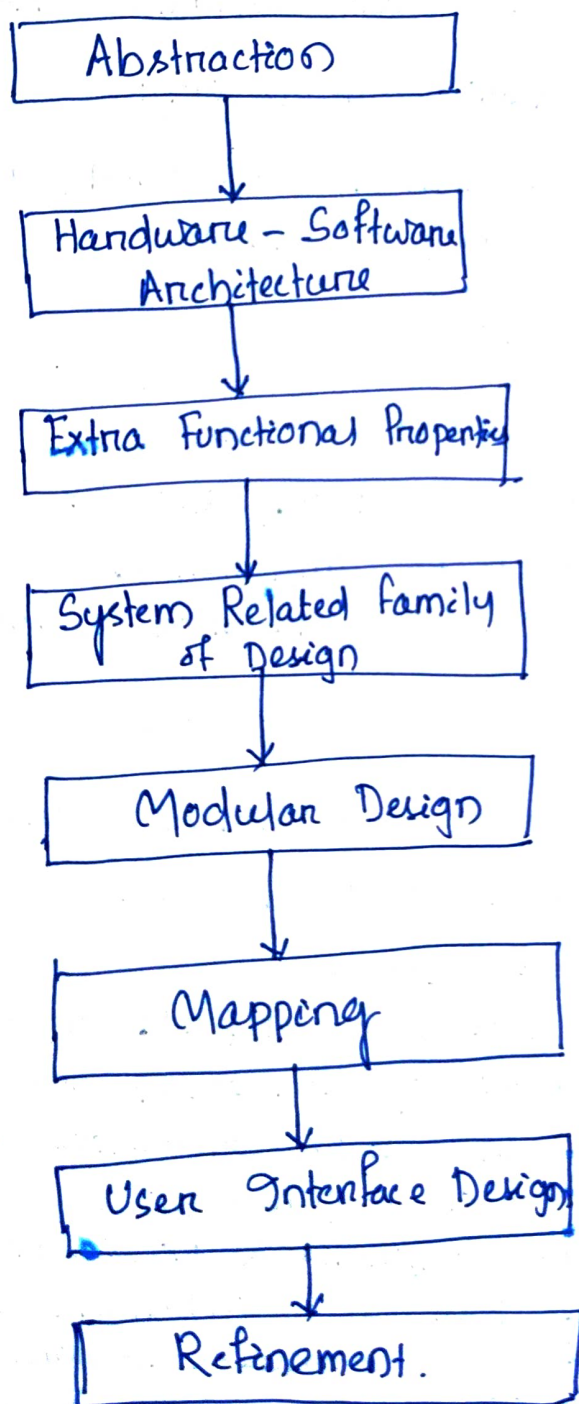
Real Time Embedded System:

- Real time embedded systems are carefully designed to meet strict timing requirements.
- These type of embedded system designed to respond within set ~~at~~ deadlines, ensuring the precise execution of crucial tasks.
- These systems can be categorized in hard real time where missing a deadline is disastrous & "soft real time" where occasional delays are acceptable within defined limits.
- Deterministic behavior ensures that tasks & operations within the system executed with a known & predictable timing.
- To achieve this, real time embedded systems employ scheduling algorithms like priority-driven scheduling and strict resource allocation.
- These are used in defense sector, medical & health care centre, etc.
- Ex: Traffic light control system.

Embedded System Design :

The Concept of embedded system design.

- * The Embedded System design is the process of developing specialized computing system that perform dedicated functions within larger mechanical or electrical systems.
- * The different steps in the embedded system design flow



- * There are two design approaches.
 - i) Top down design: Starting with high level specification & decomposing into detailed components.
 - ii) Bottom up Design: Building individual components first & integrating them into complete system.

i) Abstraction:

- Each problem component is first abstracted.
- Ex: abstraction of a robotic system problem can be in terms of control of arms & motions.

ii) Hardware - software Architectures:

- Proper knowledge of hardware & software to be known before starting any design process.

iii) Extra Functional Properties:

- Extra functionalities required in the system being developed should be well understood from the design.

iv) System related family of Design:

- When designing a system, one should refer to previous system-related family of design.

v) Modular Design:

- Modular design concept is used.
- System designing is decomposition of software into modules that are to be implemented.
- Modules should be clearly understood & should maintain continuity.

vi) Mapping:

- Based on software requirements mapping is done.
- Ex: Data flow & program flow are mapped into one.

vii) User Interface Design:

- User Interface Design depends on user requirements, environment analysis & function of the system.
- Ex: If we want to reduce power consumption of mobile phones we take care of other parameters.

ix) Refinements: Every component & module must be refined appropriately so that the software team can understand .. Architectural description language is used to describe software design.

- a) Control Hierarchy b) Partition & structure c) Data structure & Hierarchy d) software procedure.

Design challenges in Embedded System Design:

Following are the challenges faced during an embedded system design.

1. Type & amount of hardware needed.
2. Optimising the power dissipation & energy consumption.
3. Process Deadlines
4. Flexibility & ability to upgrade.
5. Reliability.

i) Type & amount of hardware:

* Hardware based on SoC (system on chip) or VLSI design has very high NRE cost and hardware design on a ckt board depends on available chips.

* Processor, memory & peripheral chips will have varying performance, cost & design metrics.

→ Design challenges to select most appropriate type and amount hardware.

ii) Power dissipation: An embedded system may need to run continuously, without being switch off. Ex: Phone it dissipates stand by power & power when talking & receiving or transmitting data.

iii) Energy consumption: Energy consumption need to be controlled. challenge is to optimize the energy consumption by appropriate hardware & software design.

Ex: Phone battery charging.

111
34

→ (Specific time by which a task must be complete its execution to avoid system failure or performance degradation)

iv) **Process Deadline:** Meeting the deadline of all process in the system while keeping memory, power dissipation, processor clock rate & cost at minimum is a challenge.

Ex: Displaying video frame with synchronise pictures & audio.

v) **Flexibility:** flexibility in design at a little cost is a challenge. A product needs to process flexible feature.

Ex: Digital camera provide USB to connect computer.
Flexibility of product will provide Options of USB, wireless USB, Bluetooth or wifi connectivity with a computer.

vi) **Ability to Upgrade:** Ability to upgrade design while keeping ~~the~~ the cost minimum & ~~without~~

Ex: Upgrade mobile camera 2M pixel to 5M pixel requires significant engineering cost.

~~Ex: So ability to upgrade with short period & little cost overhead is a design challenge.~~
→ So ability to upgrade with short period & little cost overhead is a design challenge.

vii) **Reliability:** Designing a reliable product needs appropriate design, thorough thorough testing, verification & validation.

Verification: It refers to activity to ensure that specific functions are correctly implemented.

Validation: It refers to an activity that ensure that the system that has been created is as per requirement agreed upon the analysis phase & ensure ensure its quality.

Testing: Goal of testing is to get reliable product by finding errors by simulation & debugging & validating the implemented software.

Embedded Microcontroller Core:

* An embedded microcontroller core is the central processing unit (CPU) inside a microcontroller that actually performs computations & controls operations in embedded system.

→ The core executes instructions, process data & control peripherals.

→ Examples of core used in embedded system are: 8051, ARM, PIC.

→ The main Roles of core are.

i) Instruction Execution

ii) Arithmetic & Logical operation / ~~Data processing~~

iii) Control of registers & instruction flow

iv) Interrupt handling

v) Data processing

vi) control peripherals

vii) Real time control

viii) Memory management

ix) Power management

x) Device automation

xi) System integration

→ The core of the embedded system fall into following categories.

i) General purpose & domain specific processors

a) Microprocessor

b) Microcontroller

c) Digital signal processor

ii) Application specific circuit (ASIC)

iii) Programmable logic device (PLD)

iv) Commercial off the shelf components (COTS)

Microcontroller cores are classified based on data width, architecture & application needs. The choice of core depends on system ~~core~~ complexity, speed & power requirements.

→ 8 bit Microcontroller cores processes 8 bit of data at a time. Simple & low-cost suitable for small scale embedded system. Ex: ~~ATmega 88P~~, 8051

→ 16 bit Microcontroller core process 16 bit data at a time. It is more powerful than 8 bit MC & supports moderate memory & faster processing. Ex - ~~ATmega 16P~~ ^{PIC2X}

→ 32 bit Microcontroller core process 32 bit data at a time. It has high performance, supports complex operation & larger memory. Ex: ARM7

→ ARM cortex cores are widely used 32 bit cores with high performance & low power.

→ Special purpose MC cores are designed for specific application like DSP, automotive control etc.

Different uses of Microcontroller core are:

(1) Instruction Execution: The core fetches, decodes & execute instruction in memory.

(2) Data ~~operation~~ ^{processing}: Performs arithmetic & logical operations using ALU.

(3) Control Peripherals: It controls input/output & other devices such as ADC, DAC, Timers, counter & USART etc.

4. Real time Control : Used in real time embedded system to respond quickly to events like sensor input.
5. Interrupt handling - Manage interrupts for immediate response to external or internal events.
6. Memory management: Control access to RAM, ROM, flash & registers.
7. Power management: Supports low-power modes for battery-operated devices.
8. Communication Handling: Enables data transfer ~~transfer~~ between devices using serial communication protocols.
9. Device automation: Automates operations in systems such as washing machines, traffic lights and medical devices.
10. System Integration: Integrates processor, memory & ~~per~~ peripherals into single compact chip, reducing cost & size.

Examples of Embedded Systems

Technological aspect of embedded system:

→ The technological aspect of an embedded system refers to the key technologies and components used to design, build and operate embedded systems.

→ The technological aspects:

1. Hardware Technology: This includes the physical components used in embedded system.

i) Microcontrollers / Microprocessors: The core processing unit that executes instructions.

ii) Memory - RAM, ROM, Flash memory for storing programs & data.

iii) Sensors & Actuators: Sensors collect data from the environment (temperature, motion, light).

Actuators perform actions (Motors, LEDs, relays).

iv) Input/Output Interface: Buttons, displays, communication ports (GPIO, UART, SPI, I2C).

Ex: A smart thermostat uses temperature sensors & microcontrollers to control heating.

2. Software Technology: Software controls how the Embedded system behaves.

i) Embedded Programming Language → commonly C, C++, Assembly.

ii) Firmware - Permanent software stored in ROM / Flash that controls hardware.

iii) Real time operating system (RTOS) - Used for time critical applications.

iv) Device drivers - Programs that allow the OS to interact with hardware components.

Ex: Firmware in microwave controls cooking time & power levels.

3. Communication Technology: Embedded systems often communicate with other devices.

42

Common protocols include

- UART
- SPI (Serial peripheral interface)
- I²C (Inter-Integrated circuit)
- CAN (Controller Area network) - widely used in automobiles.

• Wi-Fi, Bluetooth, Zigbee for wireless comm.

Ex: Smart home devices communicate through Wi-Fi or Bluetooth.

4. Power Management Technology: Efficient power usage is important, especially for portable device.

Key features: Low-power processors

• Sleep / standby mode.

• Battery management system.

• Energy-efficient circuit.

Example: Smartwatches use low-power chips to extend battery life.

5. Real time Processing: Many embedded systems must respond within strict timing constraints.

• Hard Real-Time System - Missing a deadline can cause failure (e.g. aircraft control).

• Soft Real-Time Systems - Delays are acceptable but undesirable (e.g. multimedia systems).

6. Security Technology: Embedded systems require protection from cyber threats.

Security measures include

- Encryption
- Secure boot
- Authentication protocol
- Hardware security modules

Ex: Payment terminal use encryption to secure transaction.

7. Integration with IoT: Modern embedded systems are often part of the Internet of Things (IoT)

- Features include -
- cloud connectivity
 - Remote monitoring
 - Data analytics
 - Smart automation

Ex: Smart refrigerator that monitor food & connect to mobile app.

3.1.2 Signal Conditioning:

* Signal conditioning is the process of making a sensor signal suitable for an embedded system to read and process accurately.

(on) → Signal conditioning is a critical process in an embedded system that involves modifying or processing raw sensor signal to prepare them for further processing or analysis.

Need of Signal Conditioning / Importance of Signal Conditioning

Sensors often produce signals that are not directly compatible with embedded system.

Common issues:

- i) Very low voltage signal / weak signal.
- ii) Noise interference
- iii) Non linear output
- iv) Analog signal needs to be converted digital signal.
- v) Voltage levels outside ADC range.

Signal conditioning ensures:

- i) Accurate measurement
- ii) Noise Reduction
- iii) Proper voltage / current levels
- iv) Safe interface with microcontroller.
- v) Improve signal quality.

Basic Signal Conditioning Operations / Techniques:

i) Amplification: Adjust the amplitude of signal to bring it within dynamic range of analog to digital converter or other processing components.

ii) Filtering: Removes unwanted noise, high frequency interference or low frequency drift from sensor signal.

iii) Linearization: Converts non-linear sensor outputs into a linear relationship with the physical parameter being measured signal that microcontroller can easily calculate.

iv) Isolation : Provides electrical isolation betⁿ sensor & rest of the system from high voltage surges & ground loops .

v) Impedance Matching : Matches the output impedance of the sensor to the input impedance of ADC or other processing components .

vi) Power Regulation : Ensure that the sensor is supplied with stable & regulated power, preventing variations in the power supply from affecting sensor performance .

vii) Analog to Digital conversion : Ensuring the signal is perfectly scaled & prevent aliasing & optimising the resolution of the ADC .

Example : In temperature monitoring system, small sensor signal must be amplified before processing .

3.1.4 Sub-System Interfacing :

* Sub-System Interfacing is the process of connecting hardware or software modules with the microcontroller or processor so that they can communicate

& work together to perform specific function .
 → A subsystem is a smaller functional unit inside a larger embedded system,
 → In Embedded System, the subsystem can be a

i) Sensor

ii) Memory

iii) Display unit

iv) Communication unit

v) Input & output device

vi) Actuators

vii) Data conversion .

- i) Sensor - Provide input data. Ex: Temperature, Motion, Pressure.
- ii) Actuation: It performs physical actions. Ex: Motor, Relays, Buzzer.
- iii) Memory: Embedded system often interface with memory.
Ex: RAM, ROM, flash → store data & firmware stores
- iv) Communication: UART, SPI (Serial Peripheral Interface) I2C (Inter Integrated Circuit)

Types of Subsystem Interfacing:

- i) Hardware Interfacing: Physical connection between Subsystem.
 - connect MC to memory
 - connect I/O devices like Keypad, LCD, ADC, etc
 - signal conditioning & ADC conversion
- 2) Communication Bus Interfacing: Subsystems communicate using standard protocols.

Protocol	Use
I2C	Multiple device with two wires
SPI	High-speed communication
UART	Serial communication
CAN bus	Automotive Systems

- 3. Software Interfacing: communication managed through software layer.

Ex: Microcontroller firmware communicates with Wi-Fi module using UART driver.

4. Analog & Digital interfacing:

- Analog interfacing
- Digital interfacing.

Communication Protocol :

* Communication protocol in Embedded systems are set of rules governing data exchange between microcontroller & sensors & peripherals.

→ Core protocols include I²C, SPI, UART, CAN & USB which are selected based on speed, distance & pin count requirements.

1. UART (Universal Asynchronous Receiver & Transmitter)

→ It is an asynchronous serial communication protocol used for short distance communication.

→ Developers often use it for debugging because it is ~~easy~~ easy to monitor with basic hardware or software tools.

Features:

- i) Use Tx & Rx lines
- ii) No clock signal required
- iii) Simple & low cost
- iv) Point to point communication.

Advantages:

- Easy implementation
- Low hardware requirements.

Application:

- i) Debugging system
- ii) GPS module
- iii) Bluetooth module.

2. SPI (Serial peripheral Interface):

→ It is a full duplex - high speed protocol uses four wires.

Features: Uses four wires ~~to & from master~~ MOSI, MISO, SCLK, SS

- Full duplex commⁿ
- High speed.

Advantages:

- i) Faster than I²C
- ii) Simple protocol

Application:

- ADC / DAC
- EEPROM
- SD cards.

3. I²C (Inter-Integrated Circuit) :

- Uses two wires (SDA & SCL) to connect multiple slave device to master.
- It is ideal for low speed communication with sensors & LCDs.
- It is a Synchronous Serial communication protocol used for short distance communication between multiple devices.

Advantages: Requires fewer wires
Supports multiple devices.

Application: Temperature Sensors
RTC modules
EEPROM.

4. CAN (Controller Area Network)

- It is designed for real time & high reliability applications.
- It is highly robust, error checking protocol common in automotive & industrial applications for interdevice communication.

Feature - Error detection & correction

- High noise immunity

Advantages: - Reliable in harsh environment
- Suitable for long distance communication.

Application - Automobile Systems
- Industrial automation
- Robotics

~~5. USB: Used for high speed data transmission~~

5. USB (Universal Serial Bus):

- Used for high speed data transmission & in many cases power delivery.

Hardware-Software Co-design in Embedded System:

- * Hardware-software co-design means both hardware & software are designed simultaneously for better performance, low cost, reduce power consumption & faster system development.
- Designer consider which part of communication computation & when that part will be executed by hardware & by the software respectively.

Hardware-Software Trade-off:

- Both hardware & software are needed in embedded system.
- Trade-off means balancing betⁿ the use of software running on GPP & implementation of special functions on hardware ^(General Purpose Processor) cent & optimizing the design metrics.

Interfacing between Analog & digital blocks:

- * Interfacing betⁿ analog blocks & digital blocks in an embedded system is all about enabling smooth commⁿ betⁿ real world signals (analog) to digital processing unit (MC, DSP etc).

- Since analog signals are continuous & digital signals are discrete, specific components & design techniques are used to bridge the gap.

Key building blocks are:

1. Analog to digital converter (ADC).

- Convert analog signals into digital values.

Ex: Temperature sensor → ADC → MC

- Important parameters are

i) Resolution ii) Sampling Rate

iii) Accuracy

2. Digital to Analog Converter (DAC)

- Convert digital values into analog signals.
- Ex: $\mu\text{c} \rightarrow \text{DAC} \rightarrow \text{speaker or actuator}$
- Used in audio output waveform generation, motor control.

3. Signal conditioning ckt:

- Before analog signals reach the ADC, they often need conditioning.
- Amplifier: Boost weak signals.
- Filters: LPF removes high frequency noise.
- Level shifting: Match voltage level to ADC input range.

4. Multiplexer: Allow multiple analog inputs to share a single ADC channel.

- ## 5. Sample & Hold ckt: capture & hold a signal value steadily during ADC conversion.
- Important for fast changing signals.

Digital Signal Processing:

- A field ^{technique} that uses algorithms, techniques & mathematical methods to process digital signals to extract useful information or enhance specific features is called DSP.
- It involves the processing, analysis & transformation of digital signal.
- Uses algorithm & computational techniques to refine information or modify signal characteristics.
- These signals are represented as sequence of numbers sampled at regular intervals.

Key concept of DSP:

1. Sampling: converting a continuous signal into discrete samples. The sampling rate must be at least twice the highest frequency in the signal.
2. Quantization: Approximating sampled values to a finite set of levels.
3. Discrete time signal: signal defined only at specific time intervals.
4. Systems: A DSP system processes input signal to produce output signals.

Common DSP Operations:

- Filtering (Remove noise, Enhance signal)
- Compression (Reduce data size)
- Fourier Transform (analyze frequency components)
- Convolution (used filtering & system response)

Advantages:

- High accuracy
- High speed
- Noise immunity / Noise reduction
- Easy to ~~store~~ storage & transmission
- Programmable & flexible.

Disadvantages:

- Increase design complexity, high power consumption

Application:

- Audio processing (noise reduction, echo cancellation)
- Image processing (compression like JPEG)

- Communication System (mobile networks, Wi-Fi)
- Medical (ECG, MRI analysis)
- Speech recognition (used app like Google Assistant, Alexa)

Basic DSP System flow:

Analog signal $\rightarrow$ Sampling $\rightarrow$ Quantization $\rightarrow$ DSP $\rightarrow$ Output.

Interfacing with External System in Embedded System.

- $\rightarrow$ Interfacing with external system in embedded system involves connecting microcontrollers to peripherals (Sensors, displays, actuators) using digital I/O or communication protocols like SPI, I2C, UART & CAN bus.
- $\rightarrow$ It requires matching electrical characteristics, proper timing & protocol implementation to enable data exchange such as reading sensor data or controlling motors.

Types of Interfacing:

- Input Interfacing:** Devices that send data to MC.
 - Signals may be analog or digital.
 - Ex: Temp. sensors, switches & buttons.
- Output Interfacing:** Devices that receive signals from MC.
 - Ex: LEDs, Motors, Buzzers
 - MC outputs High - LED turns ON
- Communication Interfacing:** Used to communicate with other systems or modules.
 - Communication Protocols are UART, I2C, SPI, CAN.

Real-world Applications:

- Smart home systems
- Industrial automation.

User Interfacing :

→ User Interface allows interaction between user and embedded system.

Types of User Interface :

1) Input Interface: These allow users to send commands

→ common input devices:

- Push buttons / switches
- Keypads (numeric or matrix)
- Touchscreens
- Rotary encoders
- Voice input.

2) Output Interfaces: These provide feedback to the user.

→ common output devices

- LEDs
- LCD / OLED displays
- 7 segment displays
- Buzzers / speakers

3) Graphical User Interface (GUI): More advance systems use graphical displays.

- Uses icons, menus, touchscreens.
- Ex: smart appliances, medical devices.

4) Audio Interface: Uses voice commands.

- voice prompts, Beeps/alerts.

Application :

i) House hold appliances: Washing machines, Microwave.

ii) Industrial: Control panels (touchscreens)

iii) IoT devices: Remote monitoring.

Design tradeoff due to process compatibility and thermal considerations:

Process compatibility Trade off:

- Process compatibility refers to ability to fabricate different types of components (digital, analog, RF, power) using a single semiconductor manufacturing process.
- Different components have different fabrication needs, but embedded system try to integrate them on one chip.

Trade off:

1. Digital Vs. Analog Design

Digital ckt: High speed, high density
- optimizing for scaling.

Analog ckt: Require precision, low noise.

Tradeoff: A digital optimized process may degrade analog accuracy & noise performance.

2. CMOS Vs BiCMOS:

CMOS → Low power, high density

BiCMOS - High speed, better analog

Tradeoff - BiCMOS improves performance but increase cost & process complexity.

3. Single chip integration Vs. Performance

- Integrating all components (SoC):
Reduce size & cost

- Separate components: Better individual optimization

Trade off: Integration reduces cost but may compromise performance of some modules.

Thermal Considerations Trade off :

* Thermal considerations deal with heat generation and dissipation in embedded system.

* Excess heat can

- Reduce performance
- cause system failure
- Shorten device lifespan

* The sources of heat are

- Switching activity
- Leakage current
- High frequency operation.

Key Trade off :

1. Performance Vs Power Vs Heat :

Higher speed $\rightarrow$ More power $\rightarrow$ More heat

Trade off: High performance leads to thermal issues.

2. Miniaturization Vs. Heat Dissipation:

Smaller device: compact & portable but poor heat dissipation.

Trade off: compact design Vs overheating risk.

3. Integration Vs Thermal Density:

More components on chip: higher heat concentration.

Trade off: High integration increases thermal management difficulty.

4. ~~Power~~ Cost Vs cooling Mechanism:

Cooling solutions: Heat sink, fans, thermal pads

Trade off: Better cooling increases system cost & size.

Thermal Management Techniques:

- Heat sinks & cooling fans
- Thermal-aware PCB design
- Dynamic voltage & frequency scaling (DVFS)
- Power gating.

* Design tradeoff in embedded system arise because a single fabrication process cannot optimally support all components & increasing performance leads to higher heat generation.

→ Designers must balance integration, performance, cost and thermal management to achieve reliable operation.

Software Aspects:

The software aspects of embedded system controls how the hardware behaves & interacts with users & other systems.

* The software aspect of embedded system refers to the specialized programming or embedded software or firmware that controls the hardware to perform a dedicated function.

→ The software aspects are

- 1) Embedded software (Firmware)
- 2) Real time Operating system (RTOS)
- 3) Device drivers
- 4) Application software
- 5) Middleware
- 6) Memory management
- 7) Interrupt handling
- 8) Scheduling and Task Management
- 9) Communication protocol
- 10) Power management software
- 11) Security
- 12) Testing & debugging.

i) Firmware: It is a low level software stored in ROM/Flash memory. It directly controls hardware. ~~It controls hardware components such as sensors, actuators etc.~~ It is written in C, C++ or assembly language.

ii) Real time operating system: Embedded system uses RTOS to manage task scheduling, memory management, inter task communication etc. Ex: FreeRTOS, VxWorks, RTLinux etc.

- 3) Device Driven: It provides communication between hardware & higher level software. It controls peripherals like sensors, motors, displays & communication interfaces, & handle interrupts.
- 4) Application software: It performs the specific task of the system. Ex: temperature control
- 5) Middleware - It provides common services. It provides communication protocols, libraries for graphics, audio or networking.
- 6) Scheduling & Task Management: It determines how the tasks are executed. It ensures high priority tasks run on time.
- 7) Communication Protocols: It enables interaction with other system/devices.
- 8) Power management software: It optimizes energy usage. The different techniques are sleep mode, peripheral shutdown etc.
- 9) Security: It includes secure boot, Encryption, Authentication, protection against firmware tampering.
- 10) Testing & debugging: Specialized tools & techniques are
 - i) on-ckt debugging (JTAG)
 - ii) Simulation
 - iii) Unit & integration testing.

Characteristics of embedded Software:

- 1) Real time operation:
- 2) Deterministic Behaviour
- 3) Hardware dependent
- 4) High efficiency
- 5) Low power consumption
- 6) Reliability
- 7) Resources constrained
- 8) Interrupt driven
- 9) Multitasking

Real time Programming Languages:

* Real time programming languages are used in embedded systems to ensure that tasks are executed within strict timing constraints (deadlines).

→ Real time programming languages are design to

- i) handle critical operations
- ii) support predictable execution
- iii) Work efficiently with limited resources
- iv) interface closely to hardware

→ Common Real-time Programming languages are C, C++, Assembly language, Ada, Rust, Python, Java

i/c: C is the most widely used language in embedded system. It offers direct access to hardware, minimal runtime & excellent performance. It is used in MC, device drivers & RTOS kernels.

ii) C++ → C++ used in more complex system. It supports object oriented programming. It provides features like improve safety & ^{high} performance. It is used in automotive systems & Robotics. It is popular due to better code reuse & maintainability.

iii) Assembly language: It is a low level language specific to processor architecture. It provides full control over hardware.

iv) Ada: It is designed for high-reliability, ^{safety &} real time systems. It provides strong compile-time checks. It is used in Aerospace & defence system.

v) Rust: It is a modern system programming language. It provides the low level control while enforcing strong memory safety &. It eliminates common bugs like null pointer dereferencing & buffer overflows, making it ideal for safety-critical embedded system. It is used in IoT devices.

vi) Python: It is used in Education & IoT. It is not ideal for strict real time constraints. It is easy to learn & use.

vii) Java: It is commonly used in embedded Linux & Android based system. It provides portability & reliability for higher-level embedded applications. It is good for complex applications. It is used in smart devices, embedded ~~linux~~ Linux devices.

Key Features of Real-Time Languages:

1. Determinism (predictable timing)
2. Efficient memory usage
3. Interrupt handling support
4. Concurrency (multi tasking)

Operating Systems for embedded system :

* An operating system (OS) in embedded system is a software that manages hardware resources & provide services for application programs.

→ Embedded system requires operating system to support their basic functionality.

→ Example of embedded operating system is the Android OS which is used in many smart phones & tablets. Other Examples are QNX RTO & VxWorks.

→ An embedded operating systems is designed to

- i) Work with limited memory & processing power.
- ii) Provide fast & predictable response
- iii) Control specific hardware functions.

Characteristics of Embedded OS:

- i) Small size & light weight.
- ii) High reliability
- iii) Deterministic behaviour
- iv) Low power consumption
- v) Fast boot time.
- vi) designed for single specific function

62 Functions of Embedded OS:

1. Task Management → Handles multiple task/process
→ Uses scheduling (Priority based in RTOS)
2. Memory Management → Efficient use of limited RAM & ROM
3. Device Management → Controls hardware through device drivers.
→ Manage sensors, actuators & peripherals.
4. Interrupt handling → Responds to external/internal events instantly.
→ Ensures real time response.
5. Scheduling → Determines execution orders of task.
6. Timing & Synchronization → Provides Timers & delay
→ Ensure task runs at correct intervals.
7. Processor Management → Allocate CPU time to different tasks.
→ Ensures priority based execution

Types of Operating System in Embedded System:

1. Real time operating system (RTOS) : Ex: FreeRTOS, VxWorks, RTLinux
2. General purpose OS (GPOS) → used in powerful embedded system.
→ No strictly real time.
→ Ex: Embedded Linux, Android.
→ used in smart phones, smart TV etc.
3. NO OS (Bare-Metal) → No operating system used.
→ Software runs directly on hardware.
→ It is ~~used~~ used in low cost devices & MC based system.

Real Time Operating System (RTOS) :

63

- A RTOS is a special kind of operating system design to handle tasks that need to ~~be~~ be completed quickly & on time.
- GPOS are good at multitasking & user interface where as RTOS focuses on doing things in time.
- It is designed to respond to task quickly & predictably.
- ^{It} Ensure tasks are completed within a fixed time limit. (deadline)
- Used in systems where timing is critical. (Ex. robots and medical devices).
- Support multitasking with precise scheduling.
- Ex of RTOS are - Airline traffic control system, Heart pacemakers, robots, Airbags in car, Command Control system etc.

Types of RTOS :

- 1) Hard Real time ~~Operatng~~ Operating System
- 2) Soft Real time ~~Operatng~~ Operating System
- 3) Firm Real time ~~Operatng~~ Operating System.

i) Hard RTOS : These ~~system~~ OS guarantee that critical tasks are completed within range of time. So deadline must never be missed. Failure to meet a ~~dead~~ deadline is considered system failure. Ex: airbag, pacemakers, medical devices, weapon systems.

ii) Soft RTOS : This OS provides some relaxation in the time limit. It provides high reliability but occasional deadline misses are tolerable. Performance may be degraded but not system failure. Ex: video streaming, Audio processing.

14) From RTOS: This os is also follow deadlines. Rare deadline missed tolerated but late results are useless.

Features / Importance of RTOS:

1. Deterministic behaviour - Predict exactly how long a task will take to execute.
2. Task Scheduling → Manage execution of multiple tasks
3. Multitasking → Execute multiple task concurrently.
 → Ability to switch from low priority task to high priority task.
4. Interrupt handling - Quickly responds to external events.
5. Intertask Communication - Allows tasks to share data using message queue, Mutexes
6. Low Latency - Minimal delay in responding to events.
7. Reliability - High stability in critical applications.

Components of RTOS:

- 1) Scheduler
- 2) Task Manager
- 3) Time Manager
- 4) Memory manager
- 5) Device drivers

Examples of RTOS:

- i) FreeRTOS
- ii) VxWorks
- iii) QNX
- iv) RTLinux

Application of RTOS:

- i) Automatic System → ABS, engine control
- ii) Medical device - Heart monitor
- iii) Industrial automation → Robotics, PLCs
- iv) Consumer Electronics → Smart appliances
 (TV, microwave, watches, phones)
- v) Aerospace systems - Flight controls
- vi) Defense System - Radar

Programming Embedded System in C

* Embedded in C is an extension of C programming language to write programs for embedded systems.

→ Embedded C programs does not require traditional OS.

~~They run on~~ They run on embedded systems ~~hardware~~ hardware with minimal kernels or RTOS.

Characteristics of Embedded C:

- 1) Efficiency
 - 2) ~~Direct~~ Direct hardware interaction
 - 3) Low level programming
- Real time operation

Basic Embedded C Programming Steps:

- i) Requirement Analysis - Understanding the requirements of the embedded system to be ~~developed~~ developed according to the problem.
- ii) Selecting environment set up - Selection of proper tools such as ~~choosing~~ choosing integrated Development Environment (IDE), compiler, debugger & other necessary tools for embedded C programming.
- iii) code development: Writing the embedded C code based on the system requirements & design specification, ~~must be done in~~

1. Give examples of initializing a timer in Embedded C.

Ex: 8051 MC.

```
TMOD = 0x01 ; // set timer 0 in Mode 1
```

```
TH0 = 0xFC ; // Load high byte.
```

```
TLO = 0x66 ; // Load low byte.
```

```
TR0 = 1 ; // start timer 0.
```

2. Give an example of if-else statement in embedded C.

```
if (temp > 30)
```

```
{ LED = 1 ; // Turn on LED.
```

```
}
```

```
else
```

```
{ LED = 0 ;
```

```
LED = 0 ; // Turn off LED.
```

```
}
```

3. Example of enabling an interrupt in Embedded C.

Ex: 8051 MC.

```
IE = 0x82 ; // Enable Enable global interrupt & timer 0  
interrupt on individually.
```

```
EA = 1 ; // Enable global interrupt.
```

```
TR0 = 1 ; // start timer 0.
```

```
ET0 = 1 ; // Enable timer 0 interrupt.
```

4. Addition of two numbers using C in embedded sys.

```
#include <reg51.h>
```

```
void main()
```

```
unsigned char a, b, c;
```

```
P1 = 0xFF;
```

```
P2 = 0xFF;
```

```
a = P1;
```

```
b = P2;
```

```
c = a + b;
```

```
P3 = c;
```

```
while(1);
```

```
}
```

5. Write a program in embedded C to check whether a number is even or odd using if-else.

```
#include <reg51.h>
```

```
void main()
```

```
unsigned int num = 25;
```

```
if (num % 2 == 0)
```

```
{ P1 = 0x01; // Even number identification
```

```
}
```

```
else
```

```
{ P1 = 0x02; // odd number identification
```

```
while(1);
```

```
}
```

// if number divided by 2 is 0 then even otherwise odd.