CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test – II

Sub:	Microcontrollers & Embedded Systems							Code:	BCO601
Date:	24/ 05/ 2025	Duration:	90 mins	Max Marks:	50	Sem:	6th	Branch:	CSE(AIML)
Answer Any FIVE FULL Questions									

		Marks	OBE	
			CO	RBT
1.	a. Explain the various purposes of Embedded system in detail with examples. b. Explain the difference between ASIC and ASSP.	[06] [04]	CO3	L2
2.	Differentiate and detail the Operational and non-operational quality attributes to be considered in any embedded system design.	[10]	CO4	L3
3.	a. Explain the concept of Binary Semaphore (Mutex). b. List the various types of firmware embedding techniques for a non-OS based embedded system.	[05] [05]	CO5	L2
4	Explain the different step modes for stepper motor.	[10]	CO3	L2
5	a. Explain the different Characteristics of Embedded System inn detail. b. The availability of an embedded product is 90%. The MTBF of the product is 30 days. What is the MTTR in days/hours for the product?	[05] [05]	CO4	L3
6	a. What are the differences between user level and kernel level threads b. What is the difference between Hard and Soft - RealTime systems? Give suitable examples..	[05] [05]	CO5	L2

1.	a. Explain the various purposes of Embedded system in detail with examples. b. Explain the difference between ASIC and ASSP.	[06] [04]	CO3	L2

Purpose of Embedded Systems:

Each Embedded System is designed to serve the purpose of any one or a combination of the following tasks:

1. Data collection/Storage/Representation
2. Data communication
3. Data (signal) processing
4. Monitoring
5. Control
6. Application specific user interface

Data Collection/Storage /Representation:

Embedded systems designed for the purpose of data collection performs acquisition of data from the external world. Data collection is usually done for storage, analysis, manipulation and transmission. The term "data" refers all kinds of information, viz. text, voice, image, video, electrical signals and any other measurable quantities. Data can be either analog (continuous) or digital (discrete). Embedded systems with analog data capturing techniques collect data directly in the form of analog signals whereas embedded systems with digital data collection mechanism converts the analog signal to corresponding digital signal using analog to digital (A/D) converters and then collects the binary equivalent of the analog data. If the data is digital, it can be directly captured without any additional interface by digital embedded systems.

The collected data may be stored directly in the system or may be transmitted to some other systems or it may be processed by the system or it may be deleted instantly after giving a meaningful representation. These actions are purely dependent on the purpose for which the embedded system is designed. Embedded systems designed for pure measurement applications without storage, used in control and instrumentation domain, collects data and gives a meaningful representation of the collected data by means of graphical representation or quantity value and deletes the collected data when new data arrives at the data collection terminal. Analog and digital CROs without storage memory are typical examples of this. Any measuring equipment used in the medical domain for monitoring without storage functionality also comes under this category.

Data Communication:

The data collecting embedded system can incorporate data communication units like wireless modules (Bluetooth, ZigBee, Wi-Fi, EDGE, GPRS, etc.) or wire-line modules (RS-232C, USB, TCP/IP, PS2, etc.). Certain embedded systems act as a dedicated transmission unit between the sending and receiving terminals, offering sophisticated functionalities like data packetizing, encrypting and decrypting. Network hubs, routers, switches, etc. are typical examples of dedicated data transmission embedded systems. They act as mediators in data communication and provide various features like data security, monitoring etc.

Data (Signal) Processing:

The data collected by embedded systems may be used for various kinds of data processing requirements like speech coding, synthesis, audeo-video codec, transmission applications, etc. A digital hearing aid is an example of an embedded system employing data processing, which improves the hearing capacity of impaired persons.

Monitoring:

Embedded system products falling under the medical domain are with monitoring functions only. They are used for determing the state of some variables using input sensors. They donot impose control over variables.

Eg: ECG machine: The sensors used in ECG are the different electrodes connected to the patient's body. The system is only used for monitoring the various parameters sensed by thesew electrodes.

Control:

Some embedded systems can have controlling unctionalities over certain variables according to the changes in the input variables. Such embedded systems contain both sensors and actuators.

Sensors are connected to the input ports for capturing the environmental parameters while actuators Which are connected to the output ports can be used to control the changes in input vaeiables so as to get the desired impact. Eg: Air Conditioner

Application Specific User Interface:

Eg: Smart Running Shoes can be assumed to be an application specific embedded system wherein t he application ccan tend to adaptive cushioning, adaptive shock absorbing characteristics as the speed changes or running mode changes.

ASIC and ASSP:

Application Specific Integrated Circuit (ASIC) is a microchip designed to perform a specific or unique application. It is used as replacement to conventional general purpose logic chips. It integrates several functions into a single chip and there by reduces the system development cost. Most of the ASICs are proprietary products. As a single chip, ASIC consumes a very small area in the total system and thereby helps in the design of smaller systems with high capabilities/functionalities.

ASICs can be pre-fabricated for a special application or it can be custom fabricated by using the components from a re-usable '*building block*' library of components for a particular customer application. ASIC based systems are profitable only for large volume commercial productions. Fabrication of ASICs requires a non-refundable initial investment for the process technology and configuration expenses. This investment is known as Non-Recurring Engineering Charge (NRÈ) and it is a one time investment.

If the Non-Recurring Engineering Charges (NRE) is borne by a third party and the Application Specific Integrated Circuit (ASIC) is made openly available in the market, the ASIC is referred as Application Specific Standard Product (ASSP). The ASSP is marketed to multiple customers just as a general-purpose product is, but to a smaller number of customers since it is for a specific application. "The ADE7760 Energy Metre ASIC developed by Analog Devices for Energy metreing applications is a typical example for ASSP".

Since Application Specific Integrated Circuits (ASICs) are proprietary products, the developers of such chips may not be interested in revealing the internal details of it and hence it is very difficult to point out an example of it. Moreover it will create legal disputes if an illustration of such an ASIC product is given without getting prior permission from the manufacturer of the ASIC. For the time being, let us forget about it. We will come back to it in another part of this book series (Namely, Designing Advanced Embedded Systems).

2.	Differentiate and detail the Operational and non-operational quality attributes to be considered in any embedded system design.	[10]	CO4	L3

Operational Quality Attributes

These relate to **how the system performs during runtime** — impacting its **functionality, performance, and behavior** in real-world operation.

Attribute	Description
Performance	How efficiently the system processes data (e.g., speed, response time, throughput). Critical in real-time systems.
Safety	Ensures the system does not cause harm — especially important in medical, automotive, or industrial applications.
Security	Protection against unauthorized access, data theft, or tampering (hardware/software).
Availability	System's readiness for correct service at any time (e.g., uptime percentage).
Reliability	Probability of failure-free operation over a specified time.
Maintainability	Ease with which the system can be diagnosed, repaired, or updated.
Robustness	Ability to function correctly under abnormal conditions (e.g., noise, power fluctuations).
Real-Time Behavior	Ensuring deadlines are met (hard or soft real-time constraints).

2. Non-Operational Quality Attributes

These attributes concern **how the system is built, supported, and evolves** — not its behavior during operation but in **lesign, development, and deployment**.

Attribute	Description
Testability	Ease of validating that the system functions correctly (e.g., using simulators or TAG).
Modifiability	Ease of updating the system to accommodate future changes (hardware/software).
Scalability	Ability to scale up in terms of functionality or number of devices (e.g., IoT ıetworks).
Portability	Ability to run on different hardware or platforms with minimal changes.
Reusability	Ability to reuse software modules or hardware IP blocks in other designs.
Configurability	Ability to change parameters without altering source code (e.g., through firmware ettings).
Design Complexity	Lower complexity improves debugging, integration, and future maintenance.
Documentation	Good technical documentation supports future development, certification, and esting.

3.	a. Explain the concept of Binary Semaphore (Mutex). b. List the various types of firmware embedding techniques for a non-OS based embedded system.	[05] [05]	CO5	L2

a. Concept of Binary Semaphore (Mutex)

A **Binary Semaphore** — commonly used as a **Mutex (Mutual Exclusion lock)** — is a synchronization mechanism that ensures **only one task or thread** can access a **shared resource** at a time. It is fundamental in **embedded systems**, **operating systems**, and **multithreaded applications** to prevent **race conditions** and ensure **data consistency**.

1. Binary Semaphore Basics

- A **binary semaphore** has only **two states**:
  - 1 → **Unlocked** (resource is free)
  - 0 → **Locked** (resource is in use)
- It is used to **signal availability** of a resource.
- Can be used for **mutual exclusion** or **event signaling** depending on context.

2. Mutex (Mutual Exclusion)

A **Mutex** is a special case of a binary semaphore used **exclusively for mutual exclusion** — i.e., ensuring that **only one thread/task enters the critical section** (shared code or resource) at a time.

b. In **non-OS based embedded systems** (also known as **bare-metal systems**), firmware is written and embedded directly to run on the hardware **without the support of an operating system**. This requires efficient techniques to manage code, memory, timing, and peripheral control.

9.1 EMBEDDED FIRMWARE DESIGN APPROACHES

The firmware design approaches for embedded product is purely dependent on the complexity of the functions to be performed, the speed of operation required, etc. Two basic approaches are used for Embedded firmware design. They are '*Conventional Procedural Based Firmware Design*' and '*Embedded Operating System (OS) Based Design*'. The conventional procedural based design is also known as '*Super Loop Model*'. We will discuss each of them in detail in the following sections.

9.1.1 The Super Loop Based Approach

The Super Loop based firmware development approach is adopted for applications that are not time critical and where the response time is not so important (embedded systems where missing deadlines are acceptable). It is very similar to a conventional procedural programming where the code is executed task by task. The task listed at the top of the program code is executed first and the tasks just below the

top are executed after completing the first task. This is a true procedural one. In a multiple task based system, each task is executed in serial in this approach. The firmware execution flow for this will be

1. Configure the common parameters and perform initialisation for various hardware components memory, registers, etc.
2. Start the first task and execute it
3. Execute the second task
4. Execute the next task
5. :
6. :
7. Execute the last defined task
8. Jump back to the first task and follow the same flow

From the firmware execution sequence, it is obvious that the order in which the tasks to be executed are fixed and they are hard coded in the code itself. Also the operation is an infinite loop based approach.

Super Loop based Approach:

Since the tasks are running inside an infinite loop, the only way to come out of the loop is either a hardware reset or an interrupt assertion. A hardware reset brings the program execution back to the main loop. Whereas an interrupt request suspends the task execution temporarily and performs the corresponding interrupt routine and on completion of the interrupt routine it restarts the task execution from the point where it got interrupted.

The '***Super loop based design***' doesn't require an operating system, since there is no need for scheduling which task is to be executed and assigning priority to each task. In a super loop based design, the priorities are fixed and the order in which the tasks to be executed are also fixed. Hence the code for performing these tasks will be residing in the code memory without an operating system image.

This type of design is deployed in low-cost embedded products and products where response time is not time critical. Some embedded products demands this type of approach if some tasks itself are sequential. For example, reading/writing data to and from a card using a card reader requires a sequence of operations like checking the presence of card, authenticating the operation, reading/writing, etc. it Should strictly follow a specified sequence.

Interrupt-Driven Technique

- Core logic still in a loop, but **time-critical tasks are handled via ISRs** (Interrupt Service Routines).
- Useful when certain events (e.g., UART Rx, Timer, GPIO) must be serviced immediately.

Key components:

- ISRs for critical tasks.
- Main loop for background jobs.

Ideal for real-time responses without an RTOS.

State Machine-Based Firmware

- System is broken into **states** with defined **transitions and actions**.
- Especially useful in control systems, communication protocols, or menu-driven devices.

Example States:

- INIT → IDLE → PROCESSING → ERROR → IDLE

Improves code readability and modularity for complex decision-making logic.

Polled Loop with Flag Checking

- Similar to super loop, but uses **flags set by ISRs or hardware**.
- Main loop checks and clears flags to perform corresponding tasks.

Used when ISR must be kept short and logic handled in main loop.

Cooperative Multitasking (Function Scheduling)

- Tasks are written as **non-blocking functions**.
- Functions are manually scheduled in the main loop based on timing, flags, or priorities.

Timing control via:

- Software timers
- Delay counters
- RTC modules

Useful for managing multiple periodic tasks without preemption.

4	Explain the different step modes for stepper motor.	[10]	CO3	L2

2.3.3.4 Stepper Motor A stepper motor is an electro-mechanical device which generates discrete displacement (motion) in response to dc electrical signals. It differs from the normal dc motor in its operation. The dc motor produces continuous rotation on applying dc voltage whereas a stepper motor produces discrete rotation in response to the dc voltage applied to it. Stepper motors are widely used in industrial embedded applications, consumer electronic products and robotics control systems. The paper feed mechanism of a printer/fax makes use of stepper motors for its functioning.

Based on the coil winding arrangements, a two-phase stepper motor is classified into two. They are:

1. Unipolar
2. Bipolar

1. Unipolar A unipolar stepper motor contains two windings per phase. The direction of rotation (clockwise or anticlockwise) of a stepper motor is controlled by changing the direction of current flow. Current in one direction flows through one coil and in the opposite direction flows through the other coil. It is easy to shift the direction of rotation by just switching the terminals to which the coils are connected. Figure 2.18 illustrates the working of a two-phase unipolar stepper motor.

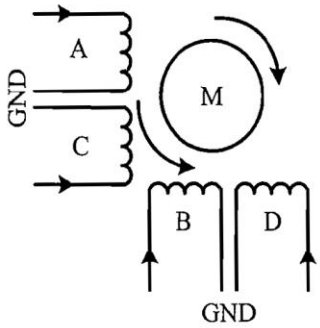


Fig. 2.18 **2-Phase unipolar stepper motor**

The coils are represented as A, B, C and D. Coils A and C carry current in opposite directions for phase 1 (only one of them will be carrying current at a time). Similarly, B and D carry current in opposite directions for phase 2 (only one of them will be carrying current at a time).

2. Bipolar A bipolar stepper motor contains single winding per phase. For reversing the motor rotation the current flow through the windings is reversed dynamically. It requires complex circuitry for current flow reversal. The stator winding details for a two phase unipolar stepper motor is shown in Fig. 2.19.

The stepping of stepper motor can be implemented in different ways by changing the sequence of activation of the stator windings. The different stepping modes supported by stepper motor are explained below.

Full Step In the full step mode both the phases are energised simultaneously. The coils A, B, C and D are energised in the following order:

Step	Coil A	Coil B	Coil C	Coil D
1	H	H	L	L
2	L	H	H	L
3	L	L	H	H
4	H	L	L	H

It should be noted that out of the two windings, only one winding of a phase is energised at a time.

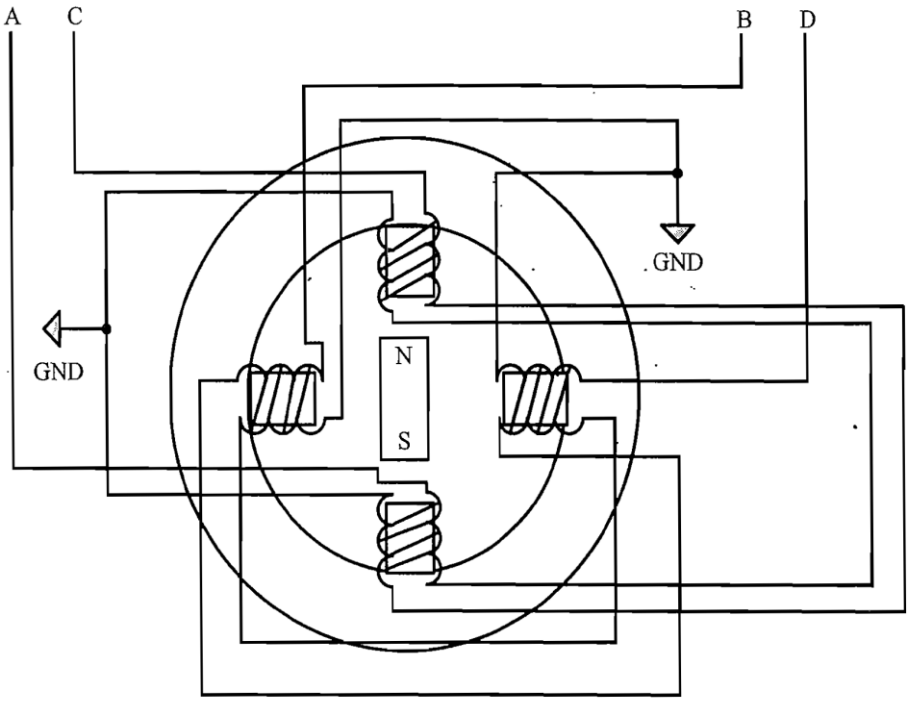


Fig. 2.19 **Stator Winding details for a 2 Phase unipolar stepper motor**

Wave Step In the wave step mode only one phase is energised at a time and each coils of the phase is energised alternatively. The coils A, B, C and D are energised in the following order:

Step	Coil A	Coil B	Coil C	Coil D
1	H	L	L	L
2	L	H	L	L
3	L	L	H	L
4	L	L	L	H

Half Step It uses the combination of wave and full step. It has the highest torque and stability. The coil energising sequence for half step is given below.

Step	Coil A	Coil B	Coil C	Coil D
1	H	L	L	L
2	H	H	L	L
3	L	H	L	L
4	L	H	H	L
5	L	L	H	L
6	L	L	H	H
7	L	L	L	H
8	H	L	L	H

5	a. Explain the different Characteristics of Embedded System in detail. b. The availability of an embedded product is 90%. The MTBF of the product is 30 days. What is the MTTR in days/hours for the product?	[05] [05]	CO4	L3

3.1 CHARACTERISTICS OF AN EMBEDDED SYSTEM

Unlike general purpose computing systems, embedded systems possess certain specific characteristics and these characteristics are unique to each embedded system. Some of the important characteristics of an embedded system 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

3.1.1 Application and Domain Specific

If you closely observe any embedded system, you will find that each embedded system is having certain functions to perform and they are developed in such a manner to do the intended functions only. They cannot be used for any other purpose. It is the major criterion which distinguishes an embedded system from a general purpose system. For example, you cannot replace the embedded control unit of your microwave oven with your air conditioner's embedded control unit, because the embedded control units of microwave oven and airconditioner are specifically designed to perform certain specific tasks. Also you cannot replace an embedded control unit developed for a particular domain say telecom with another control unit designed to serve another domain like consumer electronics.

3.1.2 Reactive and Real Time

As mentioned earlier, embedded systems are in constant interaction with the Real world through sensors and user-defined input devices which are connected to the input port of the system. Any changes happening in the Real world (which is called an Event) are captured by the sensors or input devices in Real Time and the control algorithm running inside the unit reacts in a designed manner to bring the controlled output variables to the desired level. The event may be a periodic one or an unpredicted one. If the event is an unpredicted one then such systems should be designed in such a way that it should be scheduled to capture the events without missing them. Embedded systems produce changes in output in response to the changes in the input. So they are generally referred as Reactive Systems.

Real Time System operation means the timing behaviour of the system should be deterministic; meaning the system should respond to requests or tasks in a known amount of time. A Real Time system should not miss any deadlines for tasks or operations. It is not necessary that all embedded systems should be Real Time in operations. Embedded applications or systems which are mission critical, like flight control systems, Antilock Brake Systems (ABS), etc. are examples of Real Time systems. The design of an embedded Real time system should take the worst case scenario into consideration.

3.1.3 Operates in Harsh Environment

It is not necessary that all embedded systems should be deployed in controlled environments. The environment in which the embedded system deployed may be a dusty one or a high temperature zone or an area subject to vibrations and shock. Systems placed in such areas should be capable to withstand all these adverse operating conditions. The design should take care of the operating conditions of the area where the system is going to implement. For example, if the system needs to be deployed in a high temperature zone, then all the components used in the system should be of high temperature grade. Here we cannot go for a compromise in cost. Also proper shock absorption techniques should be provided to systems which are going to be commissioned in places subject to high shock. Power supply fluctuations, corrosion and component aging, etc. are the other factors that need to be taken into consideration for embedded systems to work in harsh environments.

3.1.4 Distributed

The term distributed means that embedded systems may be a part of larger systems. Many numbers of such distributed embedded systems form a single large embedded control unit. An automatic vending machine is a typical example for this. The vending machine contains a card reader (for pre-paid vending systems), a vending unit, etc. Each of them are independent embedded units but they work together

to perform the overall vending function. Another example is the Automatic Teller Machine (ATM). An ATM contains a card reader embedded unit, responsible for reading and validating the user's ATM card, transaction unit for performing transactions, a currency counter for dispatching/vending currency to the authorised person and a printer unit for printing the transaction details. We can visualise these as independent embedded systems. But they work together to achieve a common goal.

Another typical example of a distributed embedded system is the Supervisory Control And Data Acquisition (SCADA) system used in Control & Instrumentation applications, which contains physically distributed individual embedded control units connected to a supervisory module.

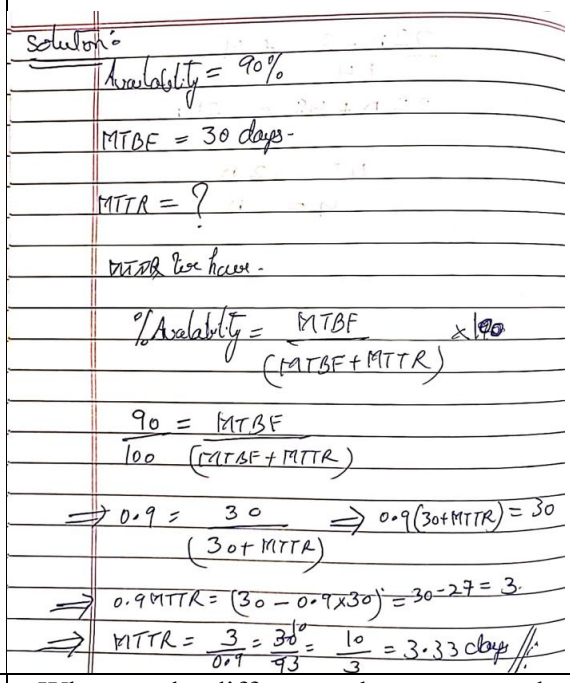
3.1.5 Small Size and Weight

Product aesthetics is an important factor in choosing a product. For example, when you plan to buy a new mobile phone, you may make a comparative study on the pros and corns of the products available in the market. Definitely the product aesthetics (size, weight, shape, style, etc.) will be one of the deciding factors to choose a product. People believe in the phrase "Small is beautiful". Moreover it is convenient to handle a compact device than a bulky product. In embedded domain also compactness is a significant deciding factor. Most of the application demands small sized and low weight products.

3.1.6 Power Concerns

Power management is another important factor that needs to be considered in designing embedded systems. Embedded systems should be designed in such a way as to minimise the heat dissipation by the system. The production of high amount of heat demands cooling requirements like cooling fans which in turn occupies additional space and make the system bulky. Nowadays ultra low power components are available in the market. Select the design according to the low power components like low dropout regulators, and controllers/processors with power saving modes. Also power management is a critical constraint in battery operated application. The more the power consumption the less is the battery life.

b.



6	a. What are the differences between user level and kernel level threads b. What is the difference between Hard and Soft - RealTime systems? Give suitable examples..	[05] [05]	CO5	L2

Feature	User-Level Threads (ULTs)	Kernel-Level Threads (KLTs)
Managed By	User-level thread libraries (e.g., POSIX `pthreads`)	The operating system kernel
Visibility to OS Kernel	Invisible to the kernel; OS sees only the main process	Fully visible; each thread is individually managed by the kernel

Context Switching	Fast (no kernel involvement)	Slower (requires kernel mode switch)
Scheduling	Done in user space by the thread library	Done by the kernel scheduler
Overhead	Low	Higher due to system calls and kernel scheduling
Blocking Behavior	If one thread blocks, the entire process blocks	Blocking one thread does not block others
Portability	Highly portable across OSes	Less portable; tied to OS thread APIs
Resource Management	Shared stack, registers are maintained manually	Managed automatically by the kernel
Example Use Cases	Lightweight embedded applications, simulations, green threads	Multicore systems, real-time OS, Linux threads

b.

In **real-time systems**, correctness depends **not only on logical results**, but also on the **time at which the results are produced**. Based on **timing constraints**, real-time systems are classified into:

1. Hard Real-Time Systems

Definition:

A **hard real-time system** is one in which **missing a deadline is considered a total system failure**. The system **must respond within a strictly defined time limit**, or it will cause severe consequences.

Characteristics:

- **Strict deadlines**
- **Predictable and deterministic**
- Safety-critical
- Failure to meet timing = **catastrophic results**

Examples:

Application Area	**Example Description**
Medical Systems	**Pacemakers, defibrillators – late response can be fatal**
Automotive Systems	**Airbag deployment in cars – must trigger within milliseconds**

2. Soft Real-Time Systems

Definition:

In a soft real-time system, deadlines are important but not absolute. Occasional deadline misses are tolerated, although system performance may degrade.

Characteristics:

- **Flexible deadlines**
- **Best-effort scheduling**
- **Used in applications where timeliness enhances performance, but is not life-critical**

Examples:

Application Area	**Example Description**
Multimedia	**Video streaming or playback – occasional frame drop is acceptable**
Gaming	**Real-time gaming systems – lag may reduce experience but not failure**
Telecommunications	**VoIP, where occasional delays or jitter are tolerable**

Conclusion:

Hard real-time systems demand **absolute timing guarantees** and are used in **critical applications**.

	Soft real-time systems aim for **timeliness**, but allow for **some flexibility**, ideal for user-interactive or media systems.