CMR
INSTITUTE OF
TECHNOLOGY

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

Internal Assesment Test – III

Sub:	ARM Microcontroller and Embedded System							Code:	15EC62
Date:	13/05/19	Duration:	90 mins	Max Marks:	50	Sem:	VI	Branch/Section	ECE A ,D section

Answer Any FIVE FULL Questions

		Marks	OBE	
			CO	RBT
1 .	Explain the term quality attributes in an embedded system development context. what are the different quality attributes to be considered in an embedded system	[10]	CO3	L2
2.	What is Hardware and Software co-design? Explain the fundamental design approaches in detail	[10]	C04	L2
3.	List and Explain the characteristics of embedded systems.	[10]	C03	L1
4.	Explain the different Embedded firmware approaches in detail	[10]	C04	L3
5.	Explain data flow graph and control flow graph model in embedded design	[10]	CO4	L3
6.	What are the basic functions of real time kernel ?Explain each	[10]	C05	L3
7.	Define process and explain process states and transition diagram	[10]	CO5	L1
8.a	Define process and explain process states and transition diagram	[5]	C05	L2
8.b	Differentiate between thread and process	[5]	C05	L2

1 .(a)	Explain the term quality attributes in an embedded system development context. what are the different quality attributes to be considered in an embedded system Quality attributes Explanation-2 M Operational Quality Attributes-4 M Non operational Quality Attributes-4 M	[05]

QUALITY ATTRIBUTES OF EMBEDDED SYSTEM

These are the attributes that together form the deciding factor about the quality of an embedded system.

There are two types of quality attributes are:-

1. **Operational Quality Attributes.**
   - These are attributes related to operation or functioning of an embedded system. The way an embedded system operates affects its overall quality.
2. **Non-Operational Quality Attributes.**
   - These are attributes **not** related to operation or functioning of an embedded system. The way an embedded system operates affects its overall quality.
   - These are the attributes that are associated with the embedded system before it can be put in operation.

Operational Attributes

a) Response

- Response is a measure of quickness of the system.
- It gives you an idea about how fast your system is tracking the input variables.
- Most of the embedded system demand fast response which should be real-time.

b) Throughput

- Throughput deals with the efficiency of system.
- It can be defined as rate of production or process of a defined process over a stated period of time.

c) Reliability

- Reliability is a measure of how much percentage you rely upon the proper

functioning of the system .

- Mean Time between failures and Mean Time To Repair are terms used in defining system reliability.
- Mean Time between failures can be defined as the average time the system is functioning before a failure occurs.
- Mean time to repair can be defined as the average time the system has spent in repairs.

d) Maintainability

- Maintainability deals with support and maintenance to the end user or a client in case of technical issues and product failures or on the basis of a routine system checkup
- It can be classified into two types :-

1. Scheduled or Periodic Maintenance

- This is the maintenance that is required regularly after a periodic time interval.
- Example : Periodic Cleaning of Air Conditioners Refilling of printer cartridges.

2. Maintenance to unexpected failure

- This involves the maintenance due to a sudden breakdown in the functioning of the system.
- Example:

1. Air conditioner not powering on
2. Printer not taking paper in spite of a full paper stack

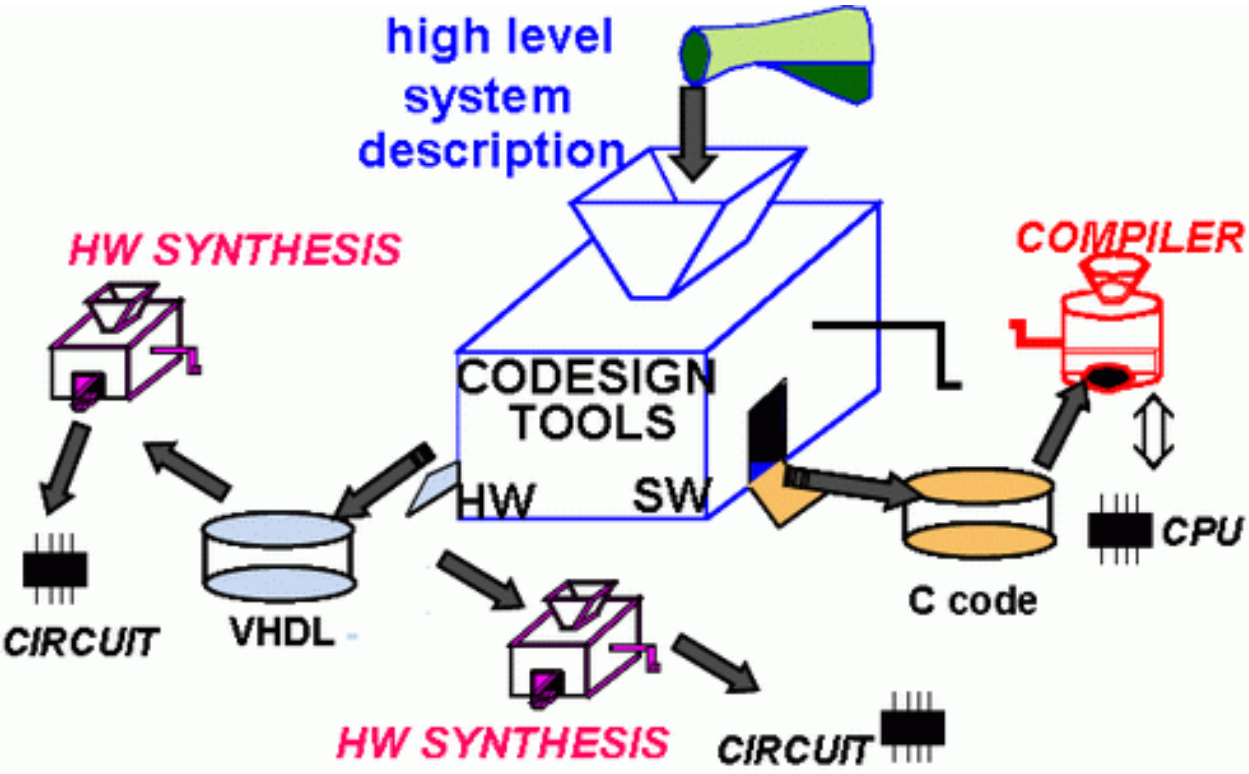
e) Security

- Confidentiality, Integrity and Availability are three corner stones of information security.
- Confidentiality deals with protection data from unauthorized disclosure.
- Integrity gives protection from unauthorized modification.
- Availability gives protection from unauthorized user
- Certain Embedded systems have to make sure they conform to the security measures.

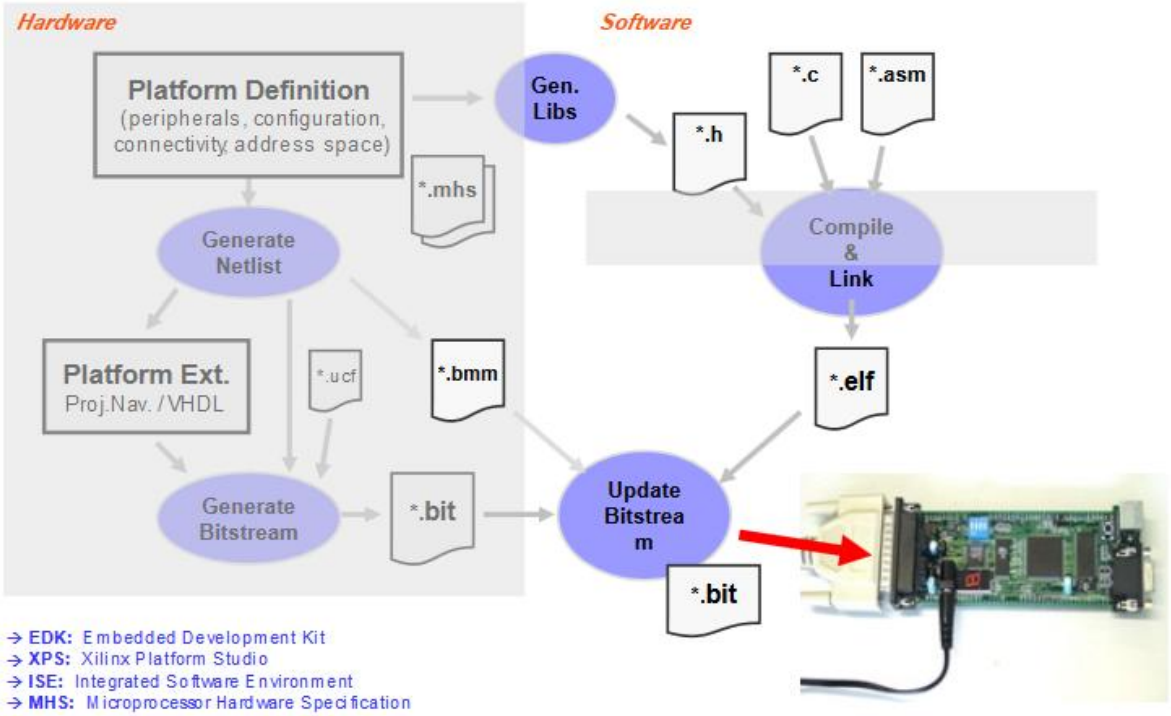
	Ex. An Electronic Safety Deposit Locker can be used only with a pin number like a password. **f) Safety** Safety deals with the possible damage that can happen to the operating person and environment due to the breakdown of an embedded system or due to the emission of hazardous materials from the embedded products.	

2.	What is Hardware and Software co-design? Explain the fundamental design approaches in detail.	[5]
	Hardware,Software co design –Explanation-2 M Fundamental Approches-8 M	

- ✓ The product requirements captured from the customer are *converted into system level needs* or processing requirements **rather than partitioning them to either h/w or s/w**
- ✓ The **system level processing** requirements are then transferred into **functions which can be simulated and verified against performance and functionality**
- ✓ The Architecture design follows the system design. The partition of system level processing requirements into hardware and software takes place during the this phase
- ✓ Each system level processing requirement is mapped as either hardware and/or software requirement
- ✓ The partitioning is performed based on the hardware-software trade-offs
- ✓



Design Flow: Combine HW + SW



→ **EDK:** Embedded Development Kit
→ **XPS:** Xilinx Platform Studio
→ **ISE:** Integrated Software Environment
→ **MHS:** Microprocessor Hardware Specification

Fundamental issues in H/w S/w Co-design

1. **Model Selection**

- **A Model captures and describes the system characteristics.**
- A model is a formal system consisting of **objects and composition rules.**
- The objectives vary with each phase.
- **Computational Models in Embedded Design :Data Flow Graph/Diagram (DFG) Model ,Control Data Flow Graph/Diagram (CDFG) Model ,State Machine Model, Sequential Program Model,** Concurrent/Communicating Process Model and **and Object Oriented Model**

Architecture Selection

- ✓ **A model only captures the system characteristics and does not provide information on 'how the system can be manufactured?'**
- ✓ The architecture specifies how a system is going to implement in terms of the number and types of different components and the interconnection among them.
- ✓ commonly used architectures in system design
- ✓ Complex Instruction Set Computing (CISC)
- Reduced Instruction Set Computing (RISC),
- Very long Instruction Word Computing (VLIW)
- Single Instruction Multiple Data (SIMD)
- Multiple Instruction Multiple Data (MIMD) etc
- ✓ Controller architecture
- ✓ Datapath Architecture

3.	List and Explain the charecteristics of embedded systems	[5]
	List the charecteristics of Embedded Systems-2 M Explain the charecteristics of Embedded Systems-8 M **CHARACTERISTICS OF EMBEDDED SYSTEM** Following are some of the characteristics of an embedded system that make it different from a general purpose computer: **1. Application and Domain specific** An embedded system is designed for a specific purpose only. It will not do any other task. Ex. A washing machine can only wash, it cannot cook Certain embedded systems are specific to a domain: ex. A hearing aid is an application that belongs to the domain of signal processing. **2. Reactive and Real time** Certain Embedded systems are designed to react to the events that occur in the nearby	

environment. These events also occur real-time.

- Ex. An air conditioner adjusts its mechanical parts as soon as it gets a signal from its sensors to increase or decrease the temperature when the user operates it using a remote control.
- An embedded system uses Sensors to take inputs and has actuators to bring out the required functionality.

3. Operation in harsh environment

- Certain embedded systems are designed to operate in harsh environments like very high temperature of the deserts or very low temperature of the mountains or extreme rains.
- These embedded systems have to be capable of sustaining the environmental conditions it is designed to operate in.

4. Distributed systems

- Certain embedded systems are part of a larger system and thus form components of a distributed system.
- These components are independent of each other but have to work together for the larger system to function properly.
- Ex. A car has many embedded systems controlled to its dash board. Each one is an independent embedded system yet the entire car can be said to function properly only if all the systems work together.

5. Small size and weight

- An embedded system that is compact in size and has light weight will be desirable or more popular than one that is bulky and heavy.
- Ex. Currently available cell phones. The cell phones that have the maximum features are popular but also their size and weight is an important characteristic

4	Explain the different firmware approaches in detail.	[5]

	Embedded Firmware Approches 5 Types-2 marks each	
	✓ The embedded firmware is responsible for controlling the various peripherals of the embedded hardware and generating response in accordance with the **functional requirements of the product.** ✓ The embedded firmware is usually stored in **a permanent memory (ROM)** and it is non alterable by end users ✓ **Designing Embedded firmware** requires **understanding of the particular embedded product hardware, like various component interfacing, memory map details, I/O port details, configuration and register details of various hardware chips** used and some programming language (either **low level Assembly Language or High level language like C/C++ or a combination of the two)** ✓ There exist two basic approaches for the design and implementation of embedded firmware, namely; ✓ **The *Super loop* based approach** ✓ **The *Embedded Operating System* based approach** ✓ The decision on which approach needs to be adopted for firmware development is purely dependent on the complexity and system requirements **Embedded firmware Design Approaches – The Super loop** ✓ Suitable for applications **that are not time critical and where the response time is not so important** (Embedded systems where missing deadlines are acceptable) ✓ Very similar to a **conventional procedural programming where the code is executed task by task** ✓ The tasks are executed in a never ending loop**. The task listed on top on the program code is executed first and the tasks just below the top are executed after completing the first task** ✓ **A typical super loop implementation will look like:** 1 **Configure the common parameters and perform initialization for various hardware components memory, registers etc.** 1 **Start the first task and execute it** 1 **Execute the second task** 1 **Execute the next task** 1 **:** 1 **:** 1 **Execute the last defined task** **Jump back to the first task and follow the same flow** **Pros:** ✓ Doesn't **require an Operating System for task scheduling** and monitoring and free from OS related overheads	

	✓ **Simple and straight forward design** ✓ **Reduced memory footprint** **Cons:** ✓ **Non Real time in execution behavior** (As the number of tasks increases the frequency at which a task gets CPU time for execution also increases) ✓ **Any issues in any task execution may affect the functioning of the product** **Enhancements:** ✓ Combine **Super loop based technique with interrupts** ✓ **Execute the tasks** (like keyboard handling) **which require Real time attention as Interrupt Service routines** **Embedded firmware Design Approaches – Embedded OS based Approach** ✓ The embedded device contains an **Embedded Operating System which can be one of:** ✓ **A General Purpose Operating System (GPOS)** ✓ **A Real Time Operating System (RTOS)** ✓ **A General Purpose Operating System (GPOS)** ✓ **The processor is generic , The OS can be used as a generalized purpose** Microsoft® Windows XP Embedded is an example of GPOS for embedded devices ✓ **A Real Time Operating System (RTOS)** ✓ **Its specific to timing constrains** ✓ **Its deterministic based on the time response** ✓ *Windows CE*', '*Windows Mobile*','*QNX*', '*VxWorks*',	

5.	Explain data flow graph and control flow graph model in embedded design	[10]

Data flow graph model-5 M

Control flow graph model-5 M

- **Data Flow Graph/Diagram (DFG) Model**

✓ Translates the **data processing requirements into a data flow graph**

✓ A data driven model in which the **program execution is determined by data.**

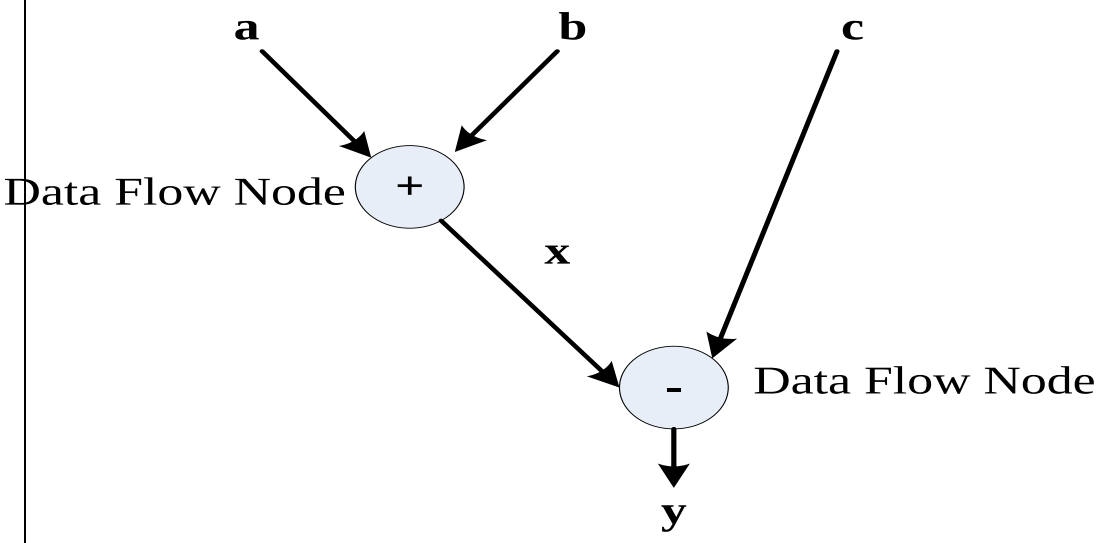
✓ Emphasizes on the data and operations on the data which transforms the input data to output data.

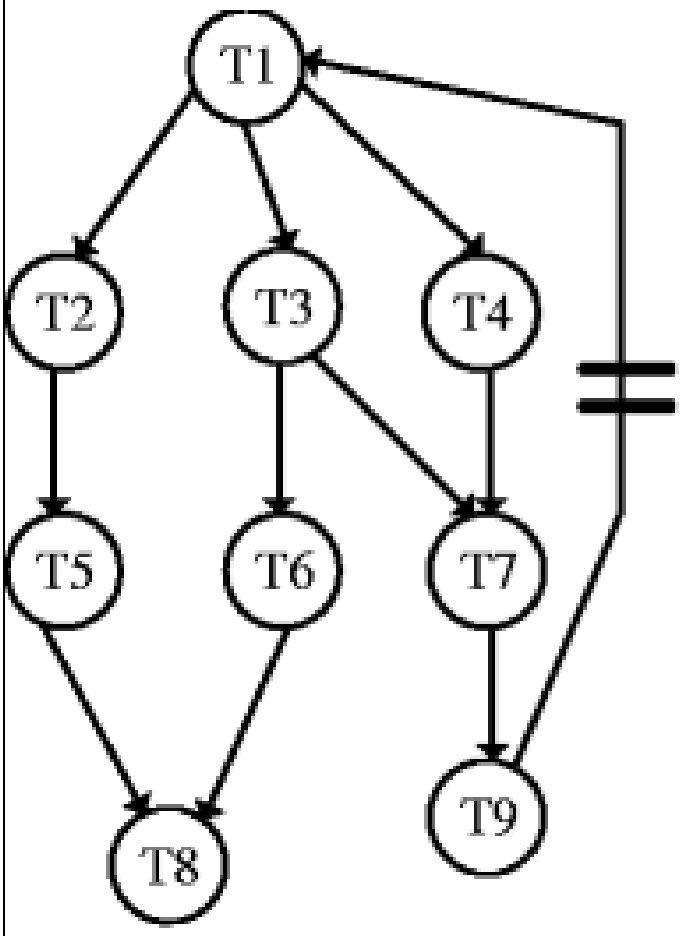
✓ A visual model in which the operation on the **data (process)** is represented using a **block (circle)** and data flow is represented using arrows. An **inward arrow to the process (circle)** represents **input data** and an **outward arrow** from the **process (circle) represents output data** in DFG notation

✓ Best suited for modeling Embedded systems which are **computational intensive (like DSP applications)**

✓ A DFG model is said to be acyclic DFG (ADFG) if it doesn't contain multiple values for the input variable and multiple output values for a given set of input(s).

✓ E.g. Model the requirement x = a + b; and y = x – c




T1
T2
T3
T4
T5
T6
T7
T8
T9

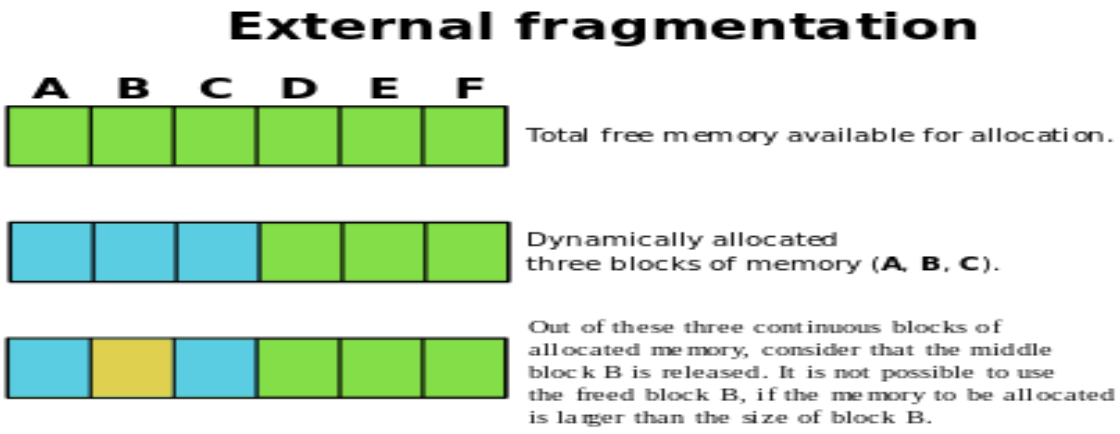
6.	What are the basic functions of real time kernel ?Explain each
	List the basic functions of Kernel-5 M Explanation-5 M **The Real Time Kernel** The **kernel of a Real Time Operating System is referred as Real Time kernel**. In complement to the conventional OS kernel, the Real Time kernel is highly **specialized and it contains only the minimal set of services required for running** the user applications/tasks. The basic functions of a Real Time kernel are – Task/Process management – Task/Process scheduling – Task/Process synchronization – Error/Exception handling – Memory Management – Interrupt handling – Time management **Real Time Kernel – Task/Process Management** Deals with setting up the **memory space for the tasks**, loading the task's code into the memory space, allocating system resources, setting up a Task Control Block (TCB) for the task and task/process termination/deletion. A Task Control Block (TCB) is used for holding the information corresponding to a task. TCB usually contains the following set of information • ***Task ID:*** Task Identification Number • ***Task State:*** The current state of the task. (E.g. State= 'Ready' for a task which is ready to execute) • ***Task Type***: Task type. Indicates what is the type for this task. The task can be a hard real time or soft real time or background task. • ***Task Priority***: Task priority (E.g. **Task priority =1 for task with priority = 1**) • ***Task Context Pointer***: Context pointer. **Pointer for context saving** • ***Task Memory Pointers***: **Pointers to the code memory, data memory and stack memory for the task** • ***Task System Resource Pointers*: Pointers to system resources** (semaphores, mutex etc) used by the task • ***Task Pointers:*** Pointers to other TCBs (TCBs for preceding, next and waiting tasks) • *Other Parameters* Other relevant task parameters • **Task/Process Scheduling:** Deals with sharing the CPU among various tasks/processes**. A kernel application called '*Scheduler*' handles the task scheduling.** Scheduler is nothing but

an **algorithm implementation**, which **performs the efficient and optimal scheduling** of tasks to provide a deterministic behavior.

- **Task/Process Synchronization**: Deals with **synchronizing the concurrent access of a resource**, which is shared across multiple tasks and the **communication between various tasks.**
- **Error/Exception handling:** Deals with registering and **handling the errors occurred/exceptions raised during the execution of tasks.** Insufficient **memory, timeouts, deadlocks, deadline missing, bus error, divide by zero, unknown instruction execution etc**, are examples of errors/exceptions. Errors/Exceptions can happen at the kernel level services or at task level. **Deadlock is an example for kernel level exception**, whereas **timeout is an example for a task level exception.** The OS kernel gives the information about the error in the form of a **system call** (API).

Memory Management

- ✓ The memory management function of an **RTOS kernel is slightly different compared to the General Purpose Operating Systems**
- ✓ **In general,** the memory allocation time increases depending on the size of the block of memory needs to be allocated and the state of the allocated memory block (**initialized memory block consumes more allocation time than un-initialized memory block)**
- ✓ Since **predictable timing and deterministic behavior are the primary focus for an RTOS**, RTOS achieves this by compromising the effectiveness of memory allocation
- ✓ **RTOS generally uses '*block*' based memory allocation technique**, instead of the usual dynamic memory allocation techniques used by the GPOS.
- ✓ RTOS kernel uses **blocks of fixed size of dynamic memory and the block is allocated for a task on a need basis**. The **blocks are stored in a '*Free buffer Queue*'.**



- ✓ Most of the RTOS kernels allow tasks to access any of the memory blocks without any memory protection to achieve predictable timing and avoid the timing overheads
- ✓ RTOS kernels assume that the whole design is proven correct and protection is unnecessary. Some commercial RTOS kernels allow memory protection as optional and the kernel enters a *fail-safe* mode when an illegal memory access occurs

Memory Management

- ✓ The memory management function of an RTOS kernel is slightly different compared to the General Purpose Operating Systems
- ✓ A few RTOS kernels implement ***Virtual Memory* concept for memory allocation** if the system supports **secondary memory storage (like HDD and FLASH memory).**

- ✓ In the '*block*' based memory allocation, a block of fixed memory is always allocated for tasks on need basis and it is taken as a unit. Hence, there will not be any memory fragmentation issues.
- ✓ The memory allocation can be implemented as constant functions and thereby it consumes fixed amount of time for memory allocation. This leaves the deterministic behavior of the RTOS kernel untouched

Interrupt Handling

- ✓ Interrupts inform the processor that an external device or an associated task requires immediate attention of the CPU.
- ✓ Interrupts can be either ***Synchronous* or *Asynchronous***.
- ✓ Interrupts which occurs in sync with the currently executing task is known as ***Synchronous* interrupts**. Usually the software interrupts fall under the Synchronous Interrupt category. Divide by zero, memory segmentation error etc are examples of **Synchronous interrupts.**
- ✓ For synchronous interrupts, the interrupt handler runs in the same context of the interrupting task.

Asynchronous interrupts are interrupts, which occurs at any point of execution of any task, and are not in sync with the currently executing task

- ✓ The interrupts generated by **external devices** (by asserting the Interrupt line of the processor/controller to which the interrupt line of the device is connected) connected to the processor/controller, timer overflow interrupts, serial data reception/ transmission interrupts etc are examples for asynchronous interrupts**.**
- ✓ For asynchronous interrupts, the interrupt handler is usually written as separate task (Depends on OS Kernel implementation) and it runs in a different context. Hence, a context switch happens while handling the asynchronous interrupts.
- ✓ Priority levels can be assigned to the interrupts and each interrupts can be enabled or disabled individually.
- ✓ Most of the RTOS kernel implements **'*Nested Interrupts*' architecture. Interrupt nesting allows the pre-emption (interruption) of an Interrupt Service Routine (ISR), servicing an interrupt, by a higher priority interrupt.**

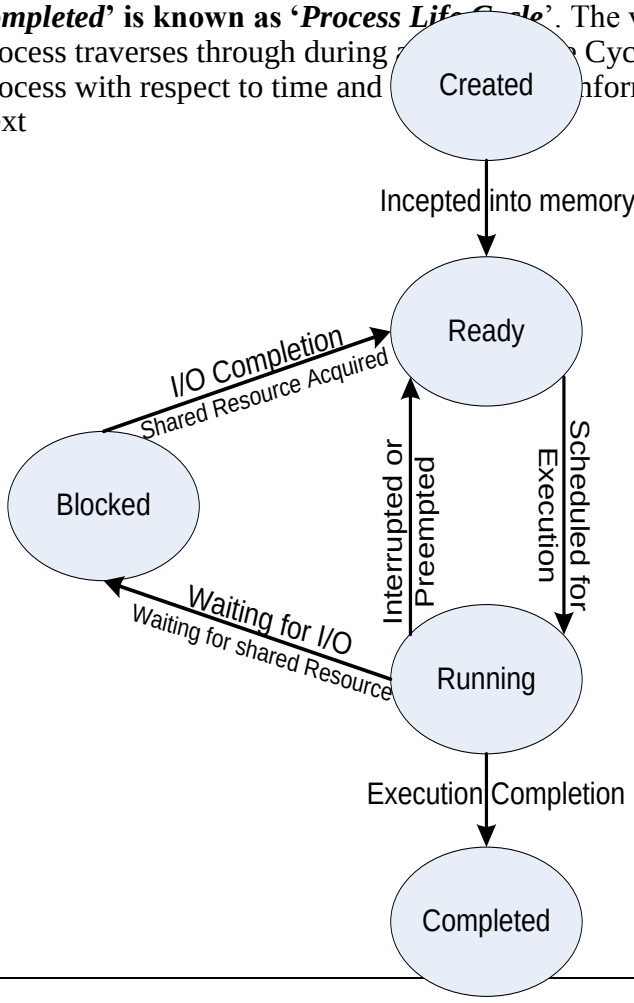
Time Management

- ✓ Interrupts inform the processor that an external device or an associated task requires immediate attention of the CPU.
- ✓ Accurate time management is essential for providing **precise time reference for all applications**
- ✓ The time reference to kernel is provided by ***a high-resolution Real Time Clock (RTC) hardware chip (hardware timer)***
- ✓ The hardware timer is programmed to interrupt the processor/controller at a **fixed rate. This timer interrupt is referred as '*Timer tick*'**
- ✓ The '*Timer tick*' is taken as the timing reference by the kernel. The '*Timer tick*' interval may vary depending on the hardware timer. Usually the '*Timer tick*' varies in the **microseconds range**
- ✓ The time parameters for tasks are expressed as the multiples of the '*Timer tick*'
- ✓ The System time is updated based on the '*Timer tick*'
- ✓ If the System time register is 32 bits wide and the '*Timer tick*' interval is 1 microsecond, the System time register will reset in
  $2^{32} * 10^{-6} / (24 * 60 * 60) = \sim 0.0497$ Days = 1.19 Hours

	If the '*Timer tick*' interval is 1 millisecond, the System time register will reset in $2^{32} * 10^{-3} / (24 * 60 * 60) = 497$ Days = 49.7 Days =~ 50 Days **Time Management** The '***Timer tick*'** interrupt is handled by the 'Timer Interrupt' handler of kernel. The '*Timer tick*' interrupt can be utilized for implementing the following actions. ✓ Save the current context (Context of the currently executing task) ✓ Increment the System time register by one. Generate timing error and reset the System time register if the timer tick count is greater than the maximum range available for System time register ✓ Update the timers implemented in kernel (Increment or decrement the timer registers for each timer depending on the count direction setting for each register. Increment registers with count direction setting = '*count up*' and decrement registers with count direction setting = '*count down*') ✓ Activate the periodic tasks, which are in the idle state ✓ Invoke the scheduler and schedule the tasks again based on the scheduling algorithm ✓ Delete all the terminated tasks and their associated data structures (TCBs) ✓ Load the context for the first task in the ready queue. Due to the re-scheduling, the ready task might be changed to a new one from the task, which was pre-empted by the 'Timer Interrupt' task

8a)	Differentiate between real time system and soft real time system Hard Real Time System & Soft Real Time System-Differentiation-5 points – 1 mark each	[10]
	Design:5 marks Diagram:5 marks **Hard Real-time System** ✓ A Real Time Operating Systems which strictly adheres to the timing constraints for a task ✓ A Hard Real Time system **must meet the deadlines for a task without any slippage** ✓ **Missing any deadline may produce catastrophic results** for Hard Real Time Systems, including permanent data lose and irrecoverable damages to the system/users ✓ Emphasize on the principle '***A late answer is a wrong answer*'** ✓ **Air bag control systems and Anti-lock Brake Systems (ABS) of vehicles** are typical examples of Hard Real Time Systems ✓ As a rule of thumb, Hard Real Time Systems does not implement the virtual memory model for handling the memory. This eliminates the delay in swapping in and out the code corresponding to the task to and from the primary memory ✓ The presence of ***Human in the loop* (*HITL*)** for tasks introduces un-expected delays in the task execution. Most of the Hard Real Time Systems are automatic and does not contain a 'human in the loop' **Soft Real-time System** ✓ Real Time Operating Systems that **does not guarantee meeting deadlines**, but, offer the best effort to meet the deadline ✓ **Missing deadlines for tasks are acceptable** if the frequency of deadline missing is within the compliance **limit of the Quality of Service (QoS)**	

	✓ A **Soft Real Time system emphasizes on the principle '*A late answer is an acceptable answer, but it could have done bit faster*'** ✓ Soft Real Time systems most often have a '***human in the loop (HITL)***' ✓ Automatic Teller Machine (ATM) is a typical example of Soft Real Time System. If the ATM takes a few seconds more than the ideal operation time, nothing fatal happens. ✓ An **audio video play back system** is another example of Soft Real Time system. No potential damage arises if a sample comes late by fraction of a second, for play back	

7.)	Define process and explain process states and transition diagram Process Definition-5 M Process State Diagram-Explanation-5 M	[5]
	Process States & State Transition ✓ The creation of a process to its termination is not a single step operation ✓ The process traverses through a series of states during its transition from the newly created state to the terminated state ✓ The cycle through which a process changes its state from '***newly created***' to '***execution completed***' **is known as '*Process Life Cycle*'**. The various states through which a process traverses through during a ... Cycle indicates the current status of the process with respect to time and ... information on what it is allowed to do next  	

- **Created State:** The state at which a process is being created is referred as 'Created State'. The Operating System recognizes a process in the '*Created State*' but no resources are allocated to the process
- **Ready State:** The state, where a process is incepted into the memory and awaiting the processor time for execution, is known as '*Ready State*'. At this stage, the process is placed in the '*Ready list*' queue maintained by the OS
- **Running State:** The state where in the source code instructions corresponding to the process is being executed is called '*Running State*'. Running state is the state at which the process execution happens.
- **Blocked State/Wait State:** Refers to a state where a running process is temporarily suspended from execution and does not have immediate access to resources. The blocked state might have invoked by various conditions like- the process enters a wait state for an event to occur (E.g. Waiting for user inputs such as keyboard input) or waiting for getting access to a shared resource like semaphore, mutex etc
- **Completed State:** A state where the process completes its execution

✓ The transition of a process from one state to another is known as '*State transition*'

✓ When a process changes its state from Ready to running or from running to blocked or terminated or from blocked to running, the CPU allocation for the process may also change

8b). Differentiate between thread and process [10]

Thread & Process –Differentiation-5 points-1 mark each

Thread	**Process**
Thread is a single unit of execution and is part of process.	Process is a program in execut
A thread does not have its own data memory and heap memory. It shares the data memory and heap memory with other threads of the same process.	Process has its own code mem

	A thread cannot live independently; it lives within the process.	A process contains at least one thread.
	There can be multiple threads in a process. The first thread (main thread) calls the main function and occupies the start of the stack memory of the process.	Threads within a process share the code, data ar thread holds separate memory area for stack (sh memory of the process).
	Threads are very inexpensive to create	Processes are very expensive to create. Involves
	Context switching is inexpensive and fast	Context switching is complex and involves lot o comparatively slower.
	If a thread expires, its stack is reclaimed by the process.	If a process dies, the resources allocated to it ar and all the associated threads of the process also