ESIOT
Unit-1
Practice Questions

1. Which architecture is followed by general purpose microprocessors?
   a) Harvard architecture
   b) Von Neumann architecture
   c) None of the mentioned
   d) All of the mentioned
   Answer: b
2. Which architecture involves both the volatile and the non volatile memory?
   a) Harvard architecture
   b) Von Neumann architecture
   c) None of the mentioned
   d) All of the mentioned
   Answer: a
3. Which architecture provides separate buses for program and data memory?
   a) Harvard architecture
   b) Von Neumann architecture
   c) None of the mentioned
   d) All of the mentioned
   Answer: a
4. Harvard architecture has ______________
   a) dedicated buses for data and program memory
   b) pipeline technique
   c) complex architecture
   d) all of the mentioned
   Answer: d
5. Which of the two architecture saves memory?
   a) Harvard
   b) Von Neumann
   c) Harvard & Von Neumann
   d) None of the mentioned
   Answer: b
6. Which architectural scheme has a provision of two sets for address & data buses between CPU and memory?
   a. Harvard architecture
   b. Von-Neumann architecture
   c. Princeton architecture
   d. All of the above
   Answer: a.
7. Which type of memory is suitable for low volume production of embedded systems?
   a) Non-volatile
   b) RAM
   c) Volatile
   d) ROM

Answer: a

8. How an embedded system communicate with the outside world?
   a) Memory
   b) Output
   c) Peripherals
   d) Input
   Answer: c
9. Which of the following helps in reducing the energy consumption of the embedded system?
   a) emulator
   b) debugger
   c) simulator
   d) compilers
   Answer: d

10. The RS232 is also known as
    a) UART
    b) SPI
    c) Physical interface
    d) Electrical interface

    Ans: D

11. Which of the following have an asynchronous data transmission?
    a) SPI
    b) RS232
    c) Parallel port
    d) I2C

    Ans: B

12. Which of the following can be used for long distance communication?
    a) I2C
    b) Parallel port
    c) SPI
    d) RS232

    Ans: D

13. Two wire interface is also called as __________
    a) UART
    b) SPI
    c) I2C
    d) USART

    Ans: C

14. Inter Integrated Circuit is a ______________
    a) Single master, single slave

b) Multi master, single slave
c) Single master, multi slave
d) Multi master, multi slave

Ans: D

15. SPI device communicates in __________
    a) Simplex
    b) Half duplex
    c) Full duplex
    d) Both half and full duplex
    Ans: C
16. SPI uses how many lines?
    a) 4 lines
    b) 1 line
    c) 3 lines
    d) 2 lines

    Ans: D

17. An interconnected collection of piconet is called ____________
    a) scatternet
    b) micronet
    c) mininet
    d) multinet

    Ans: A

18. Bluetooth is the wireless technology for ___________
    a) local area network
    b) personal area network
    c) metropolitan area network
    d) wide area network
    Ans: B
19. Bluetooth uses ___________
    a) frequency hopping spread spectrum
    b) orthogonal frequency division multiplexing
    c) time division multiplexing
    d) channel division multiplexing

    Ans: A

20. ZigBee is a _____________ task group.
    A. Body Area Network
    B. Personal Area Network
    C. Mesh Area Network
    D. Coexistence Area Network
    Ans: B

21. Which is not a type of ZigBee Devices?
    A. Zigbee Coordinator Device
    B. Zigbee Router
    C. Zigbee Start Device
    D. Zigbee End Device
    Ans: C

1. What is the main characteristic that distinguishes real-time embedded systems from general embedded systems?

Answer: The main characteristic that distinguishes real time embedded systems from general embedded systems is the presence of strict timing constraints. Real-time embedded systems must perform their tasks and respond to events within specified time intervals, often referred to as deadlines. Failing to meet these deadlines can have severe consequences, ranging from degraded performance to catastrophic failures, depending on the system's criticality.

2. What are the four key characteristics of real-time embedded systems?

Answer: The four key characteristics of real-time embedded systems are

1. Constant Response: A real-time embedded system always responds in the same manner to a certain situation and is not allowed to deviate from its normal designated output.
2. Deadline: A deadline is crucial to the working of a real-time embedded system, and a missed deadline can cost lives and finances.
3. Accuracy: Accuracy is an important characteristic for real-time embedded systems, as inaccuracy can have severe consequences (e.g., a pacemaker failing to maintain the correct heartbeat).
4. Quick Response: The real-time embedded system must be swift enough to respond to the changing external environment with immediate effect.

3. Give an example of a hard real-time embedded system and explain the consequences of missing deadlines in such a system.

Answer: An example of a hard real-time embedded system is an airbag deployment system in a car. If the microcontroller doesn't detect a collision or electronically trigger the airbag within a fraction of a second, the consequence can be tragic, potentially leading to loss of life or severe injury. In hard real-time embedded systems, missing deadlines can have catastrophic consequences, as they are often used in safety-critical applications where failure to meet timing constraints can result in unacceptable outcomes.

4. Make a Program in Arduino IDE to implement temperature monitoring system

Answer:
```
// Define the pin for the LM35 temperature sensor
const int TEMP_SENSOR_PIN = A0;
void setup() {
  // Initialize the serial communication
  Serial.begin(9600);
}

void loop() {
  // Read the temperature from the LM35 sensor
  int temperature = analogRead(TEMP_SENSOR_PIN);
  // Print the temperature to the serial monitor
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" C");
  delay(1000); // Wait for 1 second before taking the next reading
}
```

5. **Make a Program in Arduino IDE to turn on/off heater by sensing temperature**

```
// Define the pin for the LM35 temperature sensor
const int TEMP_SENSOR_PIN = A0;
// Define the pin for the heater control
const int HEATER_CONTROL_PIN = 9;
// Set the desired temperature range
const int TARGET_TEMP = 25; // Target temperature in Celsius
const int TEMP_TOLERANCE = 2; // Temperature tolerance in Celsius
void setup() {
  // Initialize the serial communication
  Serial.begin(9600);
  // Set the heater control pin as an output
  pinMode(HEATER_CONTROL_PIN, OUTPUT);
}
void loop() {
  // Read the temperature from the sensor
  int currentTemp = analogRead(TEMP_SENSOR_PIN);
  // Check if the temperature is too high
  if (currentTemp > TARGET_TEMP + TEMP_TOLERANCE) {
    turnOffHeater();
    Serial.println("Temperature too high, turning off heater.");
  }
  // Check if the temperature is too low
  else if (currentTemp < TARGET_TEMP - TEMP_TOLERANCE) {
    turnOnHeater();
    Serial.println("Temperature too low, turning on heater.");
  }
  // Temperature is within the desired range
  else {
    Serial.println("Temperature is within the desired range.");
  }

  delay(1000); // Wait for 1 second before taking the next reading
}
// Function to turn on the heater
```

```
void turnOnHeater() {
  digitalWrite(HEATER_CONTROL_PIN, HIGH);
}
// Function to turn off the heater
void turnOffHeater() {
  digitalWrite(HEATER_CONTROL_PIN, LOW);
}
```

6. Consider a real-time embedded system that receives sensor data from a device at regular intervals of 10 milliseconds (ms). The system needs to process this data and send a control signal within a maximum latency of 5 ms to ensure proper system operation. Calculate the average latency, maximum latency, and jitter for this system.

Suppose the measured latency values (in ms) for the last 10 samples are: 4.8, 5.2, 4.9, 5.1, 5.3, 4.7, 5.0, 5.4, 4.6, 5.2.

Answer: Average latency = (4.8 + 5.2 + 4.9 + 5.1 + 5.3 + 4.7 + 5.0 + 5.4 + 4.6 + 5.2) / 10 = 5.02 ms

Maximum latency = 5.4 ms

Jitter = Maximum deviation from the ideal time (10 ms)

= Max(|10 - 4.8|, |10 - 5.2|, |10 - 4.9|, |10 - 5.1|, |10 - 5.3|, |10 - 4.7|, |10 - 5.0|, |10 - 5.4|, |10 - 4.6|, |10 - 5.2|)

= Max(5.2, 4.8, 5.1, 4.9, 4.7, 5.3, 5.0, 5.4, 5.4, 4.8) = 5.4 ms

7. Consider a set of three periodic tasks with the following characteristics, Determine if the task can be scheduled with Rate Monotonic Scheduling (RMS)

Task	**Period (T)**	**Execution Time (C)**
A	10	3
B	15	4
C	20	2

Answer: To determine if this task set is schedulable using RMS, we need to calculate the processor utilization factor (U) and compare it with the schedulability bound for RMS.

U = (C1/T1) + (C2/T2) + (C3/T3)
U = (3/10) + (4/15) + (2/20)
U = 0.3 + 0.2667 + 0.1
U = 0.6667
The schedulability bound for RMS with 3 tasks is approximately 0.78 (using the formula n(2^(1/n) - 1), where n is the number of tasks).
Since U (0.6667) is less than the schedulability bound (0.78), the task set is schedulable using RMS.

8. Why is it important for real-time embedded systems to have a constant response to a given situation, and what could be the consequences of deviating from the expected output?

Answer: It is important for real-time embedded systems to have a constant response to a given situation because these systems are often used in critical applications where consistency and predictability are essential. Deviating from the expected output or behavior can have severe consequences, depending on the application domain. For example, in an industrial control system, an unexpected response or behavior could lead to equipment malfunctions, production line shutdowns, or even safety hazards for workers. In medical devices like pacemakers or automated drug delivery systems, deviations from the expected output could put patients' lives at risk. Therefore, real-time embedded systems must be designed to always respond in the same manner to a certain situation, as any deviation from the designated output can have potentially catastrophic consequences.

9. Differentiate between Hard, Soft and Firm Real time Embedded Systems

Feature	**Hard Real-Time Embedded Systems**	**Soft Real-Time Embedded Systems**	**Firm Real-Time Embedded Systems**
Definition	Systems where missing a deadline can lead to catastrophic failures.	Systems where missing a deadline can degrade performance but is not catastrophic.	Systems where missing a deadline occasionally is tolerable but repeated misses are problematic.
Examples	Airbag systems, pacemakers, industrial control systems	Multimedia systems, online transaction processing	Automated teller machines (ATM), navigation systems
Deadline Importance	Strict and non-negotiable	Flexible, negotiable within limits	Generally strict, but some deadlines can be missed occasionally
Consequences of Missing Deadlines	Catastrophic failures, possible loss of life or major system failures	Reduced quality of service, performance degradation	Degradation in quality, potential for system errors if repeated misses occur
System Predictability	Highly predictable and deterministic	Less predictable, can tolerate some variability	Predictable with some tolerance for infrequent deadline misses
Scheduling Approach	Priority-based or time-triggered	Best-effort or dynamic priority-based	Mixture of hard and soft real-time scheduling
System Design Complexity	High, due to need for strict timing guarantees and predictability	Moderate, balancing performance and deadline adherence	Moderate to high, depending on tolerance for deadline misses

Typical Use Cases	Critical systems where safety and reliability are paramount	Consumer electronics, telecommunication systems	Financial systems, some mission-critical applications with tolerance for occasional misses

10. What are some common scheduling algorithms used in Real-Time Operating Systems (RTOS), and how do they differ in their approach to task prioritization?

Answer: Some common scheduling algorithms used in Real-Time Operating Systems (RTOS) include:

1. Rate Monotonic Scheduling (RMS): This algorithm assigns static priorities to tasks based on their periods, with shorter periods receiving higher priorities. It is suitable for periodic task sets and provides a simple and efficient scheduling mechanism.
2. Earliest Deadline First (EDF): In this algorithm, tasks are scheduled based on their absolute deadlines, with the task having the earliest deadline receiving the highest priority. EDF is an optimal scheduling algorithm for preemptive task sets and can achieve higher processor utilization than RMS.
3. Least Laxity First (LLF): This algorithm schedules tasks based on their laxity, which is the amount of time remaining before a task's deadline minus its remaining execution time. Tasks with lower laxity values are given higher priorities, as they have less flexibility in meeting their deadlines.
4. Priority-based Scheduling: In this approach, tasks are assigned static or dynamic priorities based on various factors, such as importance, criticality, or deadline requirements. Higher-priority tasks are scheduled before lower-priority tasks.