Q.1 What are the 5 differences between MCU and MPU? Answer with some examples.

(Marking scheme: Any 5 points (0.8 × 5=4))

Aspect	MCU (Microcontroller Unit)	MPU (Microprocessor Unit)
Architecture Complexity	Integrates CPU, memory, and peripherals on a single chip.	Contains only the CPU core, requiring external components.
Integration Level	Highly integrated with on-chip resources.	Requires external components for functionality.
Resource Usage	Optimized for resource-constrained environments.	Suitable for resource-rich environments.
Processing Power	Typically offers lower processing power.	Provides higher processing power.
Cost	Generally lower cost due to integration.	Higher cost due to separate components.
Power Consumption	Designed for low-power applications.	Consumes more power.
Real-time Performance	Used in real-time systems with deterministic behavior.	May not be suitable for real-time applications.
Development Complexity	Simplifies development with integrated peripherals.	Requires additional effort in hardware and software.
Flexibility	Less flexible in terms of upgradability.	Offers more flexibility and upgradability.
Example Applications	Smart home devices, wearables, automotive control systems.	Desktop computers, laptops, servers, gaming consoles.

Q.2 How do you define Embedded Systems and their Differences with a computer? List 5 differences.
(Marking scheme: Any 5 points (0.8 × 5=4))

Ans:

1. Definition:

- Embedded Systems: Embedded systems are specialised computing systems designed to perform specific functions within a larger system or product. They are typically tightly integrated into hardware and often have real-time constraints.

- **Computer:** A computer is a general-purpose device that can execute various tasks and applications. It consists of hardware components like a CPU, memory, storage, and input/output devices and runs an operating system to manage resources and provide a user interface.

2. Purpose:

- **Embedded Systems:** Designed for specific tasks or functions within a larger system, such as controlling machinery, processing sensor data, or managing communication protocols.

- **Computer:** Intended for general-purpose computing tasks, such as web browsing, word processing, gaming, and multimedia playback.

3. Size and Form Factor

- **Embedded Systems:** Often compact and integrated directly into the device or product they serve, with custom form factors to fit space constraints.

- **Computer:** Generally larger, typically housed in a standalone case or enclosure, and designed for desktop, laptop, or server use.

4. Operating System:

- **Embedded Systems:** We may run a real-time operating system (RTOS) or a lightweight embedded OS tailored for specific applications, with minimal user interaction.

- **Computer:** Runs a full-featured operating system such as Windows, macOS, or Linux, offering various functionalities and user interactions.

5. Resource Constraints:

- **Embedded Systems**: Often have limited resources (CPU power, memory, storage) compared to computers, optimised for efficiency and cost-effectiveness.

- **Computer:** Typically equipped with ample resources, allowing for multitasking, running resource-intensive applications, and storing large amounts of data.

6. User Interface:

- E**mbedded Systems:** These systems may have minimal or no user interface, relying on sensors, indicators, or other interaction devices, and are often controlled remotely or through a simple interface.
- **Computer:** Provides a graphical user interface (GUI) or command-line interface (CLI) for user interaction, supporting input devices like keyboards, mice, and touchscreens.

7. Connectivity:

- Embedded Systems: Often equipped with specific communication interfaces tailored to their application, such as UART, SPI, I2C, Ethernet, Wi-Fi, or Bluetooth.
- **Computer:** Offers a wide range of networking connectivity options, including Ethernet, Wi-Fi, Bluetooth, USB, HDMI, and other standard interfaces.

8. Power Consumption:

- Embedded Systems: Designed for low power consumption, often powered by batteries or energy-efficient power sources to enable long-term operation in remote or resource-constrained environments.
- **Computer:** Typically consumes more power due to higher processing capabilities and larger components, often requiring a stable power source like mains electricity.

9. Customization and Scalability:

- Embedded Systems: Often highly customised for specific applications, with limited scalability and flexibility beyond their intended use case.
- **Computer:** Offers greater flexibility and scalability, allowing for hardware upgrades, software customisation, and the installation of additional peripherals or expansion cards.

10. Deployment Environment:

- **Embedded Systems:** Deployed in diverse environments ranging from consumer electronics (e.g., smartphones, smart appliances) to industrial automation, automotive, healthcare, and aerospace applications.
- Computer: Primarily deployed in office, home, educational, and enterprise settings for general-purpose computing tasks.

Aspect	Embedded Systems	Computers
Definition	Specialized computing systems for specific functions.	General-purpose devices for a wide range of tasks.
Purpose	Designed for specific tasks within larger systems.	Intended for general computing tasks.
Size and Form Factor	Compact, custom form factors to fit space constraints.	Larger, often housed in standalone cases or enclosures.
Operating System	May run a lightweight OS or real-time OS.	Runs full-featured OS like Windows, macOS, or Linux.
Resource Constraints	Limited resources optimized for efficiency and cost.	Ample resources for multitasking and resource-intensive tasks.
User Interface	Minimal or no UI, often controlled remotely.	GUI or CLI for user interaction with input devices.
Connectivity	Specific communication interfaces tailored to applications.	Offers diverse connectivity options for networking.
Power Consumption	Designed for low power consumption.	Consumes more power due to higher capabilities.
Customization/Scalability	Highly customized, limited scalability.	Offers scalability and customization options.
Deployment Environment	Used in diverse environments from consumer to industrial.	Primarily deployed in office, home, and enterprise settings.

Q.3 How do you define Von Neumann Architecture and Harvard architecture and Harvard Architecture. Specify the differences.
(Marking scheme: Definition 2 marks and Differences 2 marks)

Ans:
Von Neumann Architecture:
- **Definition:** The Von Neumann Architecture, named after mathematician and physicist John von Neumann, is a computer architecture model that describes a design with a single memory space for both program instructions and data. It comprises a CPU, memory, input/output devices, and a control unit.

- **Characteristics:**
  - **Single Memory:** Both instructions and data are stored in the same memory space.
  - **Sequential Execution:** Instructions are fetched from memory one at a time and executed sequentially.
  - **Stored Program Concept:** Programs are stored in memory and treated as data.
  - **Prone to Bottlenecks:** Since the CPU and memory share the same bus, they may contend for access, potentially leading to performance bottlenecks.

Harvard Architecture:

- **Definition:** Harvard Architecture is a computer architecture model that separates the memory space for instructions and data. It features distinct memory units and buses for program instructions and data, enabling simultaneous access to both types of memory.

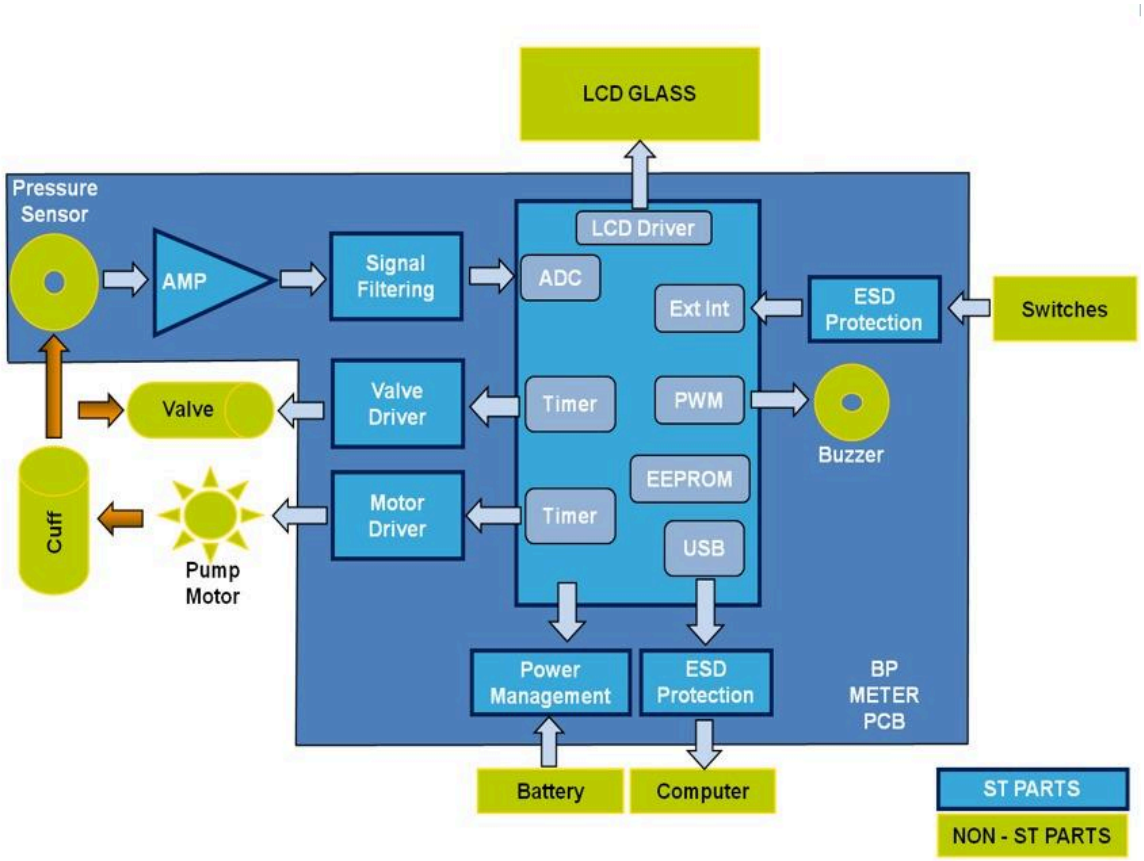
- **Characteristics:**
  - **Separate Memory Spaces:** Instructions and data are stored in separate memory units, each with its bus.
  - **Simultaneous Access:** Allows simultaneous fetching of instructions and data, improving performance.
  - **Increased Throughput:** Avoids contention for memory access, reducing potential bottlenecks.
  - **Complexity:** Typically requires more complex hardware due to separate memory units and buses.

Aspect	Von Neumann Architecture	Harvard Architecture
Memory Structure	Utilizes a single memory space for instructions and data.	Features separate memory spaces for instructions and data.
Access Mechanism	Fetches instructions and data sequentially.	Allows simultaneous access to instruction and data memory.
Performance	May suffer from bottlenecks due to contention for memory access.	Offers improved performance by enabling simultaneous access to instruction and data memory.
Complexity	Generally simpler in design.	Typically more complex due to separate memory units and buses.
Examples	Common in most modern computers, laptops, smartphones.	Found in microcontrollers, DSPs, some embedded systems.

Q.4 Design the block diagram for IR LED based remote control?

Marking scheme: *For the IR remote, we should have* ***switches as inputs, Low power MCU, Battery power, an IR Transmitter, and an optional local display (One mark each)***

Q5. Design the block diagram for Smart-BP Machine.



(Marking scheme: *If the Input, output, battery, and connectivity are shown with Low-Power MCU, One mark each.****)***

Q.6 How would you define an embedded system ecosystem offered by semiconductor companies? Provide any examples to justify the answer.
(Marking scheme: Definition 2 marks and Examples 2 marks)

Ans: An embedded system ecosystem offered by semiconductor companies refers to a comprehensive suite of hardware, software, tools, and support services provided to developers and manufacturers to facilitate the design, development, and deployment of embedded systems. This ecosystem aims to streamline the process of creating embedded solutions by offering integrated solutions encompassing all aspects of the embedded system development lifecycle.

Components of an Embedded System Ecosystem:

1. Hardware Platforms: Semiconductor companies offer a range of hardware platforms, including microcontrollers, microprocessors, system-on-chips (SoCs), development boards, and reference designs tailored for specific applications. These platforms often feature integrated peripherals, connectivity options, and optimised architectures to meet the requirements of diverse embedded applications.

Example: STMicroelectronics offers the STM32 family of microcontrollers and development boards, providing a wide range of scalable options for embedded developers across various industries, from IoT devices to automotive systems.

2. Software Development Tools: Embedded system ecosystems include software development tools such as integrated development environments (IDEs), compilers, debuggers, and software libraries. These tools are designed to streamline the software development process and enable efficient code writing, debugging, and optimisation for target hardware platforms.

Example: NXP Semiconductors provides the MCUXpresso integrated development environment, along with software development kits (SDKs) and middleware components, to support the development of applications on their microcontroller and processor platforms.

3. Operating Systems and Middleware: Semiconductor companies often offer operating systems (OS) and middleware components tailored for embedded systems, including real-time operating systems (RTOS), communication protocols, file systems, and device drivers. These software components help manage system resources, facilitate communication, and accelerate application development.

Example: Texas Instruments offers TI-RTOS, a real-time operating system optimised for their microcontroller and processor platforms, and a range of middleware components such as USB and TCP/IP stacks.

4. Support and Documentation: Embedded system ecosystems provide comprehensive technical documentation, application notes, tutorials, and online forums to assist developers throughout the design and development process. Additionally, semiconductor companies offer technical support services, training programs, and collaboration opportunities to help developers overcome challenges and accelerate time-to-market.

Example: Renesas Electronics offers extensive technical documentation, software libraries, and online communities through their Renesas Synergy Platform to support developers in building innovative embedded solutions.

Benefits of an Embedded System Ecosystem:

- **Integration:** A cohesive ecosystem integrates hardware, software, and support services, providing developers with a unified environment to develop and deploy embedded solutions efficiently.

- **Scalability:** With a range of hardware platforms, software tools, and support services, developers can scale their projects from prototyping to production, leveraging the resources offered by the ecosystem.

- **Time-to-Market:** By offering pre-validated hardware platforms, software components, and development tools, embedded system ecosystems help reduce development time and accelerate time-to-market for embedded products.

- Reliability and Quality: Semiconductor companies invest in rigorous testing, certification, and quality assurance processes for their hardware and software components, ensuring reliability, performance, and interoperability in embedded systems.

- Innovation: By providing access to cutting-edge technologies, development resources, and collaborative opportunities, embedded system ecosystems empower developers to innovate and create next-generation embedded solutions.

Overall, an embedded system ecosystem offered by semiconductor companies serves as a foundation for developers to build reliable, scalable, and innovative embedded products across a wide range of applications.

Q.8 Write down five features of the STM32CubeMx tool. How is it useful to design an embedded system?
(Marking scheme: Any 5 points (0.8 × 5=4))

Ans:
STM32CubeMX is a graphical configuration tool STMicroelectronics provides for STM32 microcontroller-based embedded systems. Here are ten features of STM32CubeMX and how they help design an embedded system:

1. Graphical Pinout Configuration:
- Users can visually configure pin assignments for various peripherals and functions on the STM32 microcontroller, simplifying the hardware design process.

2. Peripheral Configuration:
- The tool provides an intuitive interface to configure and initialise various peripherals such as GPIOs, timers, USART, SPI, I2C, ADC, and DAC, allowing users to customise their functionality according to project requirements.

3. Automatic Code Generation:
- STM32CubeMX generates initialisation code (e.g., HAL, LL, or CMSIS) based on the user's configuration, reducing the time and effort required for software development and ensuring consistency and correctness in initialisation routines.

4. Clock Configuration:
- Users can configure the clock tree settings, including system clock source, frequency, and distribution, to optimise power consumption, meet timing requirements, and ensure reliable operation of the microcontroller.

5. Power Consumption Optimization:
- The tool provides options to configure low-power modes, peripheral clock gating, and voltage scaling settings, helping designers optimise power consumption for battery-powered or energy-efficient embedded systems.

6. Peripheral Interconnection:
- STM32CubeMX allows users to easily configure and manage interconnections between different peripherals and modules, facilitating communication and data exchange within the system.

7. Middleware Integration:
- The tool supports the integration of middleware components such as USB, TCP/IP, FAT filesystem, and FreeRTOS, enabling rapid development of advanced embedded applications with standardised protocols and functionalities.

8. Hardware Abstraction Layer (HAL) Support:
- Users can generate code based on the HAL abstraction layer, providing a high-level API for peripheral configuration and operation, simplifying code development and portability across different STM32 microcontrollers.

9. Real-Time Peripheral Configuration Validation:
- STM32CubeMX includes real-time validation checks to ensure that the configured peripherals and settings are compatible and valid, helping prevent configuration errors and ensuring system reliability.

10. Integration with STM32CubeIDE and Other Toolchains:
- The tool seamlessly integrates with STM32CubeIDE and other popular development toolchains, enabling a streamlined development workflow from initial configuration to code generation, debugging, and deployment.

Overall, STM32CubeMX simplifies the embedded system design process by providing a user-friendly interface for configuring peripherals, generating initialisation code, optimising power consumption, and ensuring compatibility with STM32 microcontrollers and development environments. It accelerates the development cycle, reduces time-to-market, and improves system reliability and performance.

Q.9 You are asked to design a smartwatch to measure the steps taken by the user. Which communication technology would you see to download steps to count? Justify with reasons.
***(Marking scheme:* (2+1+1=4))**

Ans: Bluetooth Low Energy (BLE) technology would be an ideal choice for downloading steps taken by the user from a smartwatch. Here are the reasons for selecting BLE:

1. Low Power Consumption:
- BLE is designed for low power consumption, making it suitable for battery-operated devices like smartwatches. It allows the smartwatch to communicate with a smartphone or other compatible devices while minimising energy consumption, thereby extending its battery life.

2. Compatibility with Smartphones:
- smartphones and tablets, including iOS and Android, widely support BLE. This compatibility ensures seamless communication between the smartwatch and the user's smartphone, enabling easy data transfer.

3. Short-Range Communication:
- BLE operates over short distances, typically up to 10 meters, and is suitable for personal devices like smartwatches. It ensures that the communication remains localised and secure without the risk of interference from nearby devices.

4. High Data Transfer Rate:
- Despite low energy, BLE offers a reasonable data transfer rate, which is sufficient for transmitting step count data from the smartwatch to the smartphone. This ensures that the step data can be downloaded quickly and efficiently without causing significant delays.

5. Established Standard:
- BLE is an established standard for short-range wireless communication, with extensive support from both hardware and software manufacturers. This ensures interoperability between different devices and simplifies the development process for smartwatch manufacturers.

6. Easy Implementation:
- BLE technology is relatively easy to implement in smartwatches, with many microcontrollers and system-on-chip (SoC) solutions offering built-in BLE capabilities.

This simplifies the hardware design and firmware development process for smartwatch manufacturers.

7. Low Cost:

- BLE chips and modules are cost-effective, making them suitable for mass-produced consumer electronics like smartwatches. This helps keep the overall manufacturing cost of the smartwatch competitive while offering advanced connectivity features.

8. Secure Communication:

- BLE supports encryption and authentication features to ensure secure communication between devices. This is important for protecting the privacy and integrity of the step count data transmitted from the smartwatch to the smartphone.

Overall, Bluetooth Low Energy (BLE) technology offers a compelling combination of low power consumption, compatibility, data transfer speed, and security, making it an ideal choice for downloading steps taken by the user from a smartwatch to a smartphone or other compatible devices.

Q11. What is the process of designing an Embedded system. Specify different stages of these with explanation.

Ans: Designing an embedded system involves several stages, each crucial for the successful development and deployment of the system. Here are the different stages of designing an embedded system, along with explanations:

1. Requirement Analysis:

- In this stage, the requirements and specifications of the embedded system are gathered and analysed. This involves understanding the purpose of the system, its intended use cases, functional requirements, performance constraints, and any regulatory or standards compliance requirements. Stakeholder inputs, user feedback, and market research play a vital role in defining the system requirements accurately.

2. System Architecture Design:

- Based on the requirements analysis, the system architecture is designed. This involves defining the overall structure of the system, including hardware components, software modules, communication interfaces, and data flow. The architecture design phase establishes the high-level design decisions, such as selecting the appropriate microcontroller or processor, choosing communication protocols, and identifying key system components.

3. Hardware Design: - In this stage, the hardware components of the embedded system are designed and developed. This includes selecting and integrating microcontrollers, sensors, actuators, power supplies, and other electronic components based on the system requirements and architecture. Schematic design, PCB layout, prototyping, and testing are key activities in the hardware design process. Cost, size, power consumption, and environmental factors are considered during hardware design.

4. Software Design:

- Concurrently with hardware design, the software components of the embedded system are designed. This involves defining the software architecture, including the firmware's structure, device drivers, application software, and middleware components. Design decisions such as the choice of programming language, software libraries, real-time operating system (RTOS), or bare-metal programming are made based on the system requirements and hardware capabilities.

5. Integration and Testing:

- Once the hardware and software components are developed, they are integrated into a complete system. Integration involves connecting hardware modules, configuring software interfaces, and ensuring compatibility between different components. Testing is

performed at various levels, including unit, integration, and system testing, to verify that the system meets the specified requirements and functions correctly under different operating conditions.

6. Verification and Validation:

- In this stage, the embedded system is subjected to verification and validation processes to ensure it meets the intended requirements and performs as expected. Verification involves checking whether the system meets its design specifications, while validation confirms that it meets user needs and expectations. Testing, simulation, prototyping, and user feedback validate the system's functionality, performance, reliability, and safety.

7. Deployment and Maintenance:

- Once the embedded system is verified and validated, it is deployed in its intended environment. Deployment involves installing, configuring, and commissioning the system, followed by user training and documentation. Maintenance activities such as software updates, bug fixes, hardware upgrades, and troubleshooting are performed throughout the system's lifecycle to ensure its continued operation and performance.

By following these stages systematically, embedded system designers can develop reliable, efficient, and cost-effective solutions that meet the needs of their intended users and applications.

Q. 12 A GPIO of the MCU powered at 3.3V needs to communicate on the digital interface with an external device that provides 5V signals. What design considerations need to be taken for effective communication between the devices? (*Marking scheme: 1 Mark each)*

Ans: When interfacing a GPIO of a microcontroller unit (MCU) powered at 3.3V with an external device providing 5V signals, several design considerations need to be considered to ensure effective communication between the devices. Here are some key considerations:

1. Voltage Level Compatibility:

- The most critical consideration is ensuring compatibility between the voltage levels of the MCU and the external device. Since the MCU operates at 3.3V and the external device provides 5V signals, voltage level translation or level shifting is necessary to ensure that the MCU GPIO pins can safely receive and interpret the signals from the external device.

2. Bidirectional Communication:

- Determine whether the MCU and external device communication is bidirectional. If bidirectional communication is required, bidirectional level shifting techniques, such as using a bidirectional voltage level translator or a bidirectional buffer, should be employed to handle both input and output signals effectively.

3. Level Shifting Circuitry:

- Implement level-shifting circuitry between the MCU GPIO pins and the external device to translate the 3.3V signals to 5V levels and vice versa. This can be achieved using dedicated level shifter ICs, voltage divider networks, or MOSFET-based level shifters, depending on the application's specific requirements.

4. Signal Integrity:

- Ensure that the level-shifting circuitry preserves the integrity of the signals, including their timing characteristics and noise immunity. Proper signal conditioning techniques such as impedance matching, signal buffering, and filtering may be required to maintain signal integrity and minimise the risk of signal corruption or distortion during transmission.

5. Input Protection:

- Implement input protection measures on the MCU GPIO pins to prevent damage from overvoltage or electrostatic discharge (ESD) events that may occur due to the higher voltage levels of the external device. This can include using series resistors,

clamping diodes, or transient voltage suppressors (TVS) to limit the voltage and protect the GPIO pins.

6. Power Supply Considerations:

- Ensure that the power supplies for both the MCU and the external device are stable and properly regulated to minimise voltage fluctuations and ensure reliable operation of the level-shifting circuitry. Proper decoupling capacitors should be placed near the power supply pins of the MCU and the level shifter ICs to filter out high-frequency noise and ensure stable voltage levels.

7. Testing and Validation:

- Test and validate the level-shifting circuitry under various operating conditions and signal scenarios to verify its performance and reliability. This includes testing for signal integrity, timing accuracy, noise immunity, and compatibility with different input/output configurations to ensure that the communication between the MCU and the external device meets the desired specifications.