SNS COLLEGE OF TECHNOLOGY

(An Autonomous Institution)

COIMBATORE – 641035

DEPARTMENT OF MECHATRONICS ENGINEERING

Distributed Embedded Architecture

Distributed Embedded Architecture refers to a system design approach where embedded systems are distributed across multiple nodes or devices, interconnected by communication networks to perform coordinated tasks. These systems are often found in applications such as automotive systems, industrial automation, IoT (Internet of Things) devices, and smart infrastructure. Below are the key concepts:

1. Node-Based System Design

- **Nodes**: A distributed embedded system consists of multiple embedded nodes, each with specific functionalities (e.g., sensors, actuators, control units). Each node usually has its own microcontroller or processor to process data locally.
- **Autonomy**: Each node may operate independently, performing specific tasks like data acquisition, processing, or control. However, these nodes must communicate and collaborate to achieve the overall system's objectives.

2. Communication Networks

- **Interconnection of Nodes**: The nodes in a distributed system are connected via communication networks such as **CAN bus**, **Ethernet**, **I2C**, **SPI**, or **wireless protocols** (e.g., Wi-Fi, Zigbee).
- **Message Passing**: Nodes communicate with each other using message-passing techniques, exchanging data or control signals. In real-time systems, this communication needs to be deterministic and meet timing requirements.
- **Protocols**: Different communication protocols are used depending on the system requirements. For instance, CAN bus is widely used in automotive systems for real-time communication between Electronic Control Units (ECUs).

3. Decentralized Control

- **Distributed Control**: Instead of a single central controller, control logic is distributed across various nodes. For example, in a smart factory, multiple controllers may handle different processes such as machine control, quality inspection, or robotic coordination.

- **Synchronization**: In many distributed embedded systems, there is a need for synchronization between nodes, especially when tasks are dependent on each other (e.g., timing coordination in automotive systems like ABS and engine control).

4. Fault Tolerance and Redundancy

- **Redundancy**: Many distributed embedded systems incorporate redundant nodes or communication paths to ensure fault tolerance. This is especially important in safety-critical systems like aerospace, automotive, or medical devices.
- **Fault Detection**: Mechanisms like heartbeat signals or watchdog timers help detect failures in individual nodes, and backup nodes or processes can take over if needed.

5. Real-Time Operation

- **Real-Time Constraints**: Distributed embedded systems often operate in real-time environments where tasks must be executed within strict time bounds. Real-time communication protocols and Real-Time Operating Systems (RTOS) are typically used to ensure timely coordination of tasks between nodes.
- **Priority Management**: Communication protocols in real-time systems often support priority-based message passing to ensure that high-priority tasks (e.g., brake control in a car) are executed before less critical tasks.

6. Scalability and Modularity

- **Scalability**: Distributed architectures are scalable; additional nodes or devices can be added to the network without drastically altering the existing system. This makes the architecture adaptable to growing or changing requirements.
- **Modularity**: Each node can be developed and tested independently, which simplifies system integration and maintenance. This modular approach also allows for easier upgrades and replacements of specific nodes.

7. Energy Efficiency

- **Distributed Processing**: In many cases, processing tasks are offloaded to local nodes, reducing the need for centralized processing power and lowering overall energy consumption. This is particularly relevant in battery-powered systems or remote IoT devices.
- **Power Management**: Techniques like local data processing (edge computing) and efficient

communication protocols reduce power consumption, which is a critical factor in embedded systems.

OSI Model in Embedded Applications

The **OSI (Open Systems Interconnection)** model is a conceptual framework used to understand network communication across various layers. In the context of embedded systems, the OSI model provides a way to structure communication between distributed nodes or devices. While not all layers are explicitly used in every embedded system, key elements of the OSI model are applicable to embedded applications, especially when network communication is involved.

1. Layer 1: Physical Layer

- **Role in Embedded Systems**: The physical layer in embedded systems refers to the hardware used for transmission of raw data between nodes. This includes cables, wireless transceivers, and physical media like Ethernet, CAN bus wiring, or RF components.
- **Example in Embedded Systems**: In an automotive system, the CAN bus connects multiple ECUs (Electronic Control Units) using twisted pair cables, which form the physical medium for data transmission.

2. Layer 2: Data Link Layer

- **Role in Embedded Systems**: This layer ensures reliable data transfer between nodes on the same physical network. It handles error detection, frame synchronization, and flow control.
- **Embedded Example**: In the CAN protocol, the data link layer manages frame arbitration (prioritizing messages), error handling, and acknowledgment, ensuring reliable transmission of data between ECUs.
- **Protocols Used**: CAN, I2C, SPI.

3. Layer 3: Network Layer

- **Role in Embedded Systems**: The network layer is responsible for routing data between different network segments. In some distributed embedded systems, this layer is used to manage data communication between different networked devices.
- **Embedded Example**: In IoT applications, embedded devices might use IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN) to route packets across large-scale, distributed sensor networks.

- **Protocols Used**: IP, 6LoWPAN.

4. Layer 4: Transport Layer

- **Role in Embedded Systems**: The transport layer ensures reliable data delivery, maintaining end-to-end communication, managing flow control, and error recovery. It's not commonly used in simple, resource-constrained embedded systems but is important in more complex systems.
- **Embedded Example**: In IoT devices communicating over the internet, TCP (Transmission Control Protocol) ensures that messages are reliably transmitted between sensors and a cloud server.
- **Protocols Used**: TCP, UDP.

5. Layer 5: Session Layer

- **Role in Embedded Systems**: The session layer manages communication sessions between devices, establishing, maintaining, and terminating sessions.
- **Embedded Example**: In distributed control systems, session management can be used for communication between supervisory control units and local controllers over a communication network.
- **Protocols Used**: MQTT (often for maintaining sessions in IoT systems).

6. Layer 6: Presentation Layer

- **Role in Embedded Systems**: The presentation layer is responsible for data formatting, encryption, and translation between different data representations. In embedded systems, this may not be a separate layer but is relevant in systems requiring data encoding or encryption.
- **Embedded Example**: In systems where sensor data needs to be transmitted securely (e.g., smart meters or automotive diagnostics), encryption might be applied at this layer before transmission.
- **Protocols Used**: JSON (data formatting in IoT), encryption protocols like SSL/TLS.

7. Layer 7: Application Layer

- **Role in Embedded Systems**: The application layer is where the communication between devices and users happens, including the protocols that provide network services.
- **Embedded Example**: In a smart home system, the application layer might include protocols like **CoAP** (Constrained Application Protocol) for resource-constrained devices to communicate with

a central server, or **HTTP** for IoT devices interacting with web-based applications.

- **Protocols Used**: HTTP, CoAP, MQTT, and other IoT-specific protocols.