Module 5:

EMBEDDED SYSTEM DESIGN

1. Processor Technology

Processor Technology refers to the development and enhancement of the hardware architecture of microprocessors and microcontrollers used in embedded systems. This technology is crucial for improving the performance, power efficiency, and functionality of embedded devices.

Key Concepts:CPU (Central Processing Unit): The primary component of a processor that performs arithmetic and logic operations, as well as controls the flow of data and instructions.

- Performance Metrics: Processor performance is often measured in terms of clock speed (MHz or GHz), MIPS (Million Instructions Per Second), and power consumption.
- Power Efficiency: Modern embedded processors focus on low-power consumption, critical for portable and battery-operated devices.
- Microarchitecture: The implementation of an instruction set architecture (ISA) within a processor, which can vary across different processors.

Emerging Trends:

- Multi-core processors: These processors include multiple cores (processing units) that can run tasks in parallel, improving performance and energy efficiency.
- Specialized Processors: Processors designed for specific tasks, such as Digital Signal Processors (DSP) for audio and video processing, and Graphics Processing Units (GPU) for visual computations.

2. IC Technology (Integrated Circuit Technology)

IC Technology involves the fabrication and design of integrated circuits (ICs), which are used to create compact, reliable, and efficient electronic components in embedded systems.

Key Concepts:

- IC Fabrication: The process of creating integrated circuits involves multiple stages, such as photolithography, doping, etching, and deposition. This process allows multiple components (resistors, capacitors, transistors) to be integrated into a single chip.
- Types of ICs:
    - Analog ICs: Process analog signals (e.g., amplifiers, sensors).
    - Digital ICs: Process binary data (e.g., microcontrollers, processors).
    - Mixed-Signal ICs: Combine both analog and digital components (e.g., ADC/DAC converters).

Emerging Trends:

- System-on-Chip (SoC): Integrating the processor, memory, and peripherals on a single chip, reducing size, cost, and power consumption.
- Nanotechnology: The use of nanometer-scale manufacturing processes to create smaller, more efficient, and faster ICs.

3. Design Technology

Design Technology focuses on the techniques and tools used to design and optimize embedded systems and their hardware/software components. It involves the methodology and technologies used for creating functional, efficient, and cost-effective designs.

Key Concepts:

- Hardware Design: The creation of circuit boards, processors, and integrated circuits using tools like CAD (Computer-Aided Design) and VHDL (VHSIC Hardware Description Language).
- Software Design: The development of embedded software, often using languages like C, C++, and Assembly, tailored to the specific hardware.
- FPGA (Field-Programmable Gate Array): An IC that can be programmed after manufacturing, enabling hardware flexibility for custom designs.
- Design Automation Tools: Software tools that automate the design process, optimizing for factors like power, performance, and area (PPA).

Challenges:

- Design Complexity: As systems become more integrated, designing the hardware and software efficiently becomes more complex.
- Design Verification: Ensuring the correctness of the design through simulations and testing.
- Time-to-market: Reducing the development time to ensure that the design meets market demands.

4. Tradeoffs in Embedded System Design

In embedded system design, there are often tradeoffs between competing factors such as performance, power, cost, and time. These tradeoffs must be carefully evaluated to ensure an optimal solution.

Key Tradeoffs:

- Performance vs. Power Consumption: High-performance systems typically consume more power. In battery-powered devices, power efficiency is prioritized.
- Cost vs. Features: Adding more features (e.g., more I/O pins, communication protocols) increases the cost of the system.

- Speed vs. Complexity: A faster system may require more complex design and greater power consumption.
- Development Time vs. Robustness: A rapid design may lead to less robust systems, while a thorough design process can take more time but produce more reliable results.

5. Model vs. Language

In embedded systems design, modeling and programming languages serve different purposes in the design process:

Modeling:

- Modeling is the process of creating a high-level representation of a system's behavior and structure. It helps in understanding the system before implementation.
- Examples of modeling tools include UML (Unified Modeling Language) and State Machine Diagrams.

Programming Language:

- Programming languages are used to write the code that will run on the embedded system, defining its actual behavior.
- Common languages in embedded systems include C, C++, Assembly, and specialized languages like VHDL for hardware description.

6. System Modelling in Embedded Systems

System modeling in embedded systems allows designers to represent and simulate the behavior of a system before implementing it in hardware and software. Several models are used depending on the complexity and requirements of the system.

6.1 Data Flow Model (DFM)

A Data Flow Model (DFM) represents the flow of data between components in the system, without explicitly specifying the sequence of operations. It focuses on how data is processed as it moves through various stages.

- Components: These can be processing units or operations that transform the data.
- Edges: Data flows between components in the form of edges.
- Use Case: Ideal for describing systems that process continuous data streams, such as digital signal processing systems.

6.2 Finite State Machine (FSM)

A Finite State Machine (FSM) is a mathematical model of computation consisting of a finite number of states. It transitions between states based on inputs and performs specific actions in each state.

- Components:
    - States: The possible conditions of the system.
    - Transitions: The conditions under which the system moves from one state to another.
    - Inputs: External signals that influence the transitions between states.
    - Outputs: Actions that occur when in a particular state.
- Use Case: Ideal for systems with distinct modes of operation, such as traffic lights, vending machines, and communication protocols.

6.3 Finite State Machine with Datapath (FSMD)

An FSMD is an extension of the FSM, incorporating a datapath that performs operations on data. It is useful for designing systems that need to handle complex data manipulations alongside state transitions.

- Datapath: A set of registers and operations that modify the data.
- Control Logic: FSM that controls the flow of data through the datapath.
- Use Case: FSMDs are commonly used in processors, digital controllers, and hardware implementations.

6.4 High-Level Control Finite State Machine (HCFSM)

An HCFSM is an advanced version of FSM, incorporating hierarchical state machines. It allows a system to operate at multiple levels of abstraction, managing complex behavior more effectively.

- Hierarchical States: States can have sub-states, leading to a more organized and scalable design.
- Modularity: Each sub-state machine can be designed and tested independently.

6.5 Process Sequence Model (PSM)

A Process Sequence Model (PSM) is used to describe the sequence of operations or processes that occur within a system. This model emphasizes the order in which tasks or operations occur, often used to visualize system workflows.

- Components: Steps or tasks that need to be completed in a specific sequence.

- Use Case: Used in control systems, workflows, or manufacturing processes where the sequence of actions is important.

6.6 Concurrent Process Model

In a Concurrent Process Model, multiple processes run simultaneously, either in parallel or in an interleaved fashion, without any specific order. This is used for systems with real-time requirements or multitasking.

- Processes: Independent units that execute simultaneously.
- Synchronization: Mechanisms like semaphores or message queues are used to ensure proper communication and synchronization between processes.

7. Implementation of System Models

After system modeling, the next step is to implement the model in hardware or software. Implementation typically involves:

- Hardware Description: Using languages like VHDL or Verilog to describe hardware behavior.
- Software Development: Writing the control logic using languages like C, C++, or Python.
- Simulation: Simulating the behavior of the system using tools like ModelSim or MATLAB to ensure correctness before implementation.