4.4 Hardware-Software design issues on embedded system:

Embedded Systems overview

An embedded system is a dedicated computing system designed to perform specific tasks, often within larger mechanical or electrical systems. Unlike general-purpose computers, embedded systems are optimized for efficiency, reliability, and functionality within their intended environment. These systems integrate hardware and software tightly, with the hardware providing processing capability and the software defining the specific functions.
Embedded systems are used in various applications, from consumer electronics (e.g., smartphones, washing machines) to industrial automation, automotive systems, and medical devices. Key characteristics include real-time operation, low power consumption, and compact design. Depending on the application, embedded systems can be simple microcontroller-based setups or complex, multi-processor systems.

Classification of Embedded Systems

Embedded systems can be classified based on their functionality, performance, and application:
Real-Time Embedded Systems: Operate under strict timing constraints. Examples include industrial robots and medical monitoring devices.
Stand-Alone Embedded Systems: Perform a specific function independently, such as calculators and MP3 players.
Networked Embedded Systems: Connected to a network for communication and data sharing. Examples include smart home devices and IoT applications.
Mobile Embedded Systems: Compact and battery-powered systems like smartphones and portable gaming consoles.
Hybrid Embedded Systems: Combine features of the above categories, such as modern cars with real-time, stand-alone, and networked subsystems.

Custom Single-Purpose Processor Design

Custom single-purpose processors are designed to execute specific tasks efficiently, making them ideal for embedded systems where performance and resource optimization are critical. The design involves:
Defining Specifications: Identifying the specific task and performance requirements.
Creating Data Path: Designing the computational logic, including ALUs, registers, and memory components, tailored to the task.
Implementing Control Logic: Developing finite state machines (FSMs) or microprograms to control data flow and operation sequencing.
Custom processors are preferred for applications where general-purpose processors cannot meet the performance, power, or cost constraints, such as in signal processing or cryptographic systems.

Optimizing Custom Single-Purpose Processors

Optimization focuses on improving speed, power efficiency, and area (size). Techniques include:
Pipelining: Splitting the task into stages to increase throughput.
Parallelism: Adding multiple functional units to perform operations simultaneously.
Hardware Specialization: Using dedicated hardware components for frequently used operations.
Low-Power Design: Minimizing energy consumption through techniques like clock gating and dynamic voltage scaling. Optimized processors are often implemented using hardware description languages (HDLs) like VHDL or Verilog and synthesized into ASICs (Application-Specific Integrated Circuits).

Basic Architecture

The architecture of embedded systems varies based on the application but

generally includes:
Processor: Microcontroller, microprocessor, or custom-designed processor.
Memory: Includes ROM for firmware storage and RAM for temporary data.
Input/Output Interfaces: Enable communication with external devices like sensors and actuators.
Timers and Counters: Support real-time operations and synchronization.
Communication Interfaces: Protocols like UART, SPI, and I2C for data exchange.
This architecture is often embedded on a single chip (System-on-Chip or SoC) to minimize size and power consumption.

Operation and Programmer's View

From the programmer's perspective, embedded systems operate at two levels:
Low-Level Operation: Involves direct manipulation of hardware registers and peripherals, typically through assembly language or C. This provides precise control over timing and resource usage.
High-Level Operation: Abstracted programming using real-time operating systems (RTOS) or middleware, enabling easier development and modularity. Programmers interact with hardware via drivers and APIs, simplifying complex hardware tasks.

Development Environment

The development of embedded systems involves specialized tools and workflows:
Integrated Development Environments (IDEs): Combine code editors, compilers, and debugging tools. Examples include Keil µVision, MPLAB, and Arduino IDE.
Simulators and Emulators: Allow testing and debugging without requiring physical hardware. Simulators mimic the system's behavior, while emulators replicate hardware functionality.
Cross-Compilers: Compile code on a host system for execution on the embedded target.
Hardware Debuggers: Tools like JTAG or SWD interfaces enable real-time debugging of the hardware.
A well-designed development environment improves productivity and reduces errors during system development.

Application-Specific Instruction-Set Processors

ASIPs are tailored processors designed for a specific application domain, offering a balance between flexibility and performance. They extend the standard instruction set with custom instructions optimized for the target application.
For example, ASIPs in digital signal processing include specialized instructions for fast Fourier transforms (FFTs) and convolution operations. Designing ASIPs involves:
Instruction Set Extension: Adding new instructions to support specific operations.
Compiler Support: Modifying compilers to recognize and utilize the extended instructions.
Hardware Implementation: Creating hardware units to execute the new instructions efficiently. ASIPs provide better performance than general-purpose processors and more flexibility than custom single-purpose processors, making them ideal for domains like telecommunications, multimedia, and encryption.