Unit I: Embedded Systems - Quick Revision Notes

---

1. Overview of Embedded Systems

Definition

- **A combination of hardware and software designed for a specific function.**
- **Examples: Washing machines, ATMs, pacemakers, and industrial robots.**

Characteristics

- **Real-time operation: Works within strict time limits.**
- **Power-efficient: Optimized for low energy consumption.**
- **Small size: Designed to fit specific applications.**
- **Dedicated function: Performs a single or limited set of tasks.**
- **Reliability & Stability: Should work continuously without failure.**

---

2. Design Challenges & Metrics

Common Design Challenges

- **Cost constraints: Limited budget for development and manufacturing.**
- **Performance vs. power: Must balance processing speed and energy efficiency.**
- **Reliability: Must function without errors for long periods.**
- **Security: Protection against cyber threats in connected devices.**

Common Design Metrics

- **Performance: Execution speed, response time.**
- **Power consumption: Battery life, heat generation.**

- **Size: Must fit within physical constraints.**
- **Flexibility: Ability to upgrade or modify.**
- **Time-to-market: Speed of development and deployment.**

---

3. Processor Technologies

Types of Processors

- **General-Purpose Processors (GPPs): Multi-use CPUs (e.g., Intel, AMD).**
- **Single-Purpose Processors: Dedicated for specific tasks (e.g., timers, motor controllers).**
- **Application-Specific Instruction Set Processors (ASIPs): Custom processors optimized for specific applications (e.g., network processors).**

---

4. IC (Integrated Circuit) Technologies

Types of IC Technologies

- **Full Custom (VLSI - Very Large Scale Integration): High-performance, expensive, designed from scratch.**
- **Semi-Custom (ASIC - Application-Specific Integrated Circuit): Pre-designed modules with some customization.**
- **PLD (Programmable Logic Devices): User-programmable chips (e.g., FPGA - Field Programmable Gate Arrays).**

---

5. Design Technologies

Compilation/Synthesis

- **Converts high-level code (C, Python) into machine code or hardware description.**

Libraries/IP (Intellectual Property)

- **Pre-designed, reusable hardware/software components to speed up development.**

Test/Verification

- **Testing: Ensures correct functionality under different conditions.**
- **Verification: Confirms the design meets specifications before manufacturing.**

---

6. General-Purpose Processors

Basic Architecture

- **Data Path: Handles data movement, includes ALU, registers, and buses.**
- **Control Unit: Directs CPU operations using control signals.**
- **Memory: Stores programs and data.**

---

7. Operations & Instruction Execution

- **The instruction cycle (Fetch → Decode → Execute → Store) determines how a processor executes commands.**

---

8. Advanced Processor Architectures

Pipelining

- **Divides the instruction execution process into stages, increasing speed.**

Superscalar & VLIW (Very Long Instruction Word)

- **Superscalar: Executes multiple instructions in parallel.**

- **VLIW: Groups multiple instructions into a single long instruction for parallel execution.**

---

9. Programmer's View of a Processor

Instruction Set

- **Defines the commands that the processor can execute (e.g., ADD, SUB, LOAD).**

Memory & Registers

- **Program Memory: Stores instructions.**
- **Data Memory: Stores variables.**
- **Registers: Small, fast storage inside the CPU for quick access.**

---

10. Input/Output (I/O) & Interrupts

I/O

- **Allows communication between the processor and external devices (e.g., keyboard, sensors).**

Interrupts

- **Mechanism that allows the processor to pause current execution and handle urgent tasks (e.g., hardware failures, user inputs).**

---

11. Development Environment & Tools

Design Flow

1. **Requirement analysis**
2. **System design**

3. **Implementation**
4. **Testing & debugging**
5. **Deployment**

Debugging & Testing

- **Debuggers: Software tools to find errors in code.**
- **Emulators: Simulate the hardware to test software.**

---

12. Selecting a Microprocessor

Factors to Consider

- **Processing speed: How fast it executes instructions.**
- **Power consumption: Battery life and heat generation.**
- **Cost: Budget constraints.**
- **Software compatibility: Must support required applications.**
- **I/O capability: Ability to interface with required peripherals.**

Unit II: Custom Processors:

Custom Single-Purpose Processors

A **custom single-purpose processor** is a processor designed for a specific task rather than general computing. Unlike general-purpose processors, which handle multiple tasks, custom processors are optimized to perform one function efficiently. These are commonly found in embedded systems, consumer electronics, and industrial applications.

Custom Single-Purpose Processor Design

The design of a custom single-purpose processor involves the following key aspects:

1. **Defining the Functionality** – Understanding the exact task it needs to perform.
2. **Hardware Design** – Creating logic circuits, data paths, and control paths to implement the function.
3. **Memory and Registers** – Incorporating necessary storage for computation.
4. **Optimization** – Reducing power consumption, increasing processing speed, and minimizing the required chip area.

Optimizing Custom Single-Purpose Processors

To ensure efficient operation, optimization techniques include:

- **Minimizing Power Consumption** – Using low-power circuits and clock-gating techniques.
- **Reducing Area Usage** – Implementing optimized logic circuits to minimize chip space.
- **Enhancing Speed** – Using pipeline techniques and parallel processing.
- **Balancing Cost and Efficiency** – Choosing an optimal trade-off between performance and hardware complexity.

Standard Single-Purpose Processors: Peripherals

Standard single-purpose processors, also called **peripherals**, are pre-designed hardware units that handle specific functions in an embedded system. They work alongside general-purpose processors to offload specialized tasks.

Timers

- **Purpose**: Measure time intervals, create time delays, and schedule tasks.
- **Modes**:

    - **Up-counting** – Increments the timer value.
    - **Down-counting** – Decrements the timer value.
    - **Free-running mode** – Runs continuously without stopping.
- **Applications**: Used in operating systems for task scheduling and real-time processing.

Counters

- **Purpose**: Count the number of events or pulses in a system.
- **Types**:
    - **Binary Counters** – Count in binary format.
    - **Decade Counters** – Count up to 10 before resetting.
- **Applications**: Frequency measurement, digital clocks, and event counting.

Watchdog Timers

- **Purpose**: Detect and recover from software failures.
- **How It Works**:
    - A system must periodically reset the watchdog timer.
    - If the system hangs and fails to reset the timer, the watchdog initiates a reset.
- **Applications**: Used in embedded systems to prevent system crashes.

UART (Universal Asynchronous Receiver/Transmitter)

- **Purpose**: Enables serial communication between devices.
- **Functionality**:
    - Converts parallel data from a microcontroller into serial data for transmission.
    - Receives serial data and converts it back into parallel data.

- **Features**:
    - Supports multiple baud rates (data transmission speeds).
    - Includes error detection mechanisms.
- **Applications**: Used in communication between computers, sensors, and embedded systems.

Pulse Width Modulator (PWM)

- **Purpose**: Controls the power delivered to devices like motors and LEDs.
- **How It Works**:
    - Modifies the duty cycle (ON/OFF time ratio) of a pulse signal.
    - A higher duty cycle increases power output, while a lower duty cycle reduces power.
- **Applications**: Used in motor speed control, LED dimming, and power regulation.

LCD Controller

- **Purpose**: Manages communication between a microcontroller and an LCD display.
- **Functions**:
    - Converts digital commands into display signals.
    - Controls brightness, contrast, and refresh rate.
- **Applications**: Found in digital watches, smartphones, and industrial display systems.

Keypad Controller

- **Purpose**: Detects and interprets keypresses from a keypad.
- **How It Works**:
    - Uses a matrix of rows and columns to scan for key presses.

    - Converts key presses into binary values for processing.
- **Applications**: Security systems, ATMs, and consumer electronics.

Analog-to-Digital Converter (ADC)

- **Purpose**: Converts analog signals (continuous values) into digital values (discrete numbers).
- **Process**:
    - Samples an analog signal at regular intervals.
    - Converts each sample into a digital number using quantization.
- **Applications**: Sensors (temperature, pressure, sound), medical devices, and control systems.

Real-Time Clock (RTC)

- **Purpose**: Provides accurate timekeeping even when the system is powered off.
- **Features**:
    - Runs on a small backup battery to maintain time.
    - Tracks time, date, and alarms.
- **Applications**: Used in computers, embedded systems, and IoT devices for scheduling tasks.

UNIT-3 Application-Specific Instruction Set Processors (ASIPs):

Application-Specific Instruction Set Processors (ASIP)

An **Application-Specific Instruction Set Processor (ASIP)** is a processor designed to execute a specific class of applications efficiently. It provides a balance between

flexibility (like general-purpose processors) and **high performance (like custom single-purpose processors).** ASIPs are widely used in **signal processing, telecommunications, multimedia, and embedded systems.**

ASIP Design Methodologies

ASIP design follows structured methodologies to ensure optimal performance and efficiency. The major approaches include:

1. **Top-Down Design:**
    - Starts from application requirements and designs the instruction set accordingly.
    - Ensures optimization for power, performance, and area.
2. **Bottom-Up Design:**
    - Begins with an existing processor and modifies it to fit the target application.
    - Provides flexibility but may not be fully optimized.
3. **Template-Based Design:**
    - Uses pre-designed processor templates and customizes them.
    - Reduces design time but offers limited customization.

Steps in ASIP Design

Designing an ASIP involves several key steps:

1. Application Analysis

- Identifying the computational needs of the target application.
- Determining required data types, operations, and performance constraints.
- Profiling the application to find bottlenecks.

2. Design Space Exploration (DSE)

- Evaluates different processor architectures to find the best trade-offs between power, performance, and area.
- Considers factors like instruction set complexity, memory usage, and pipeline depth.

3. Generation of Software Tools

- **Compiler:** Translates high-level code into ASIP instructions.
- **Debugger:** Helps detect and fix errors in the code execution.
- **Instruction Set Simulator:** Emulates ASIP behavior before hardware implementation to test functionality and performance.

4. Synthesizing the Processor

- Hardware description of ASIP is created using **HDL (Hardware Description Language).**
- The design is implemented on **ASIC (Application-Specific Integrated Circuit) or FPGA (Field-Programmable Gate Array).**
- Performance verification is done to ensure the processor meets design goals.

Design Space Exploration Techniques

Design Space Exploration (DSE) helps in selecting the best ASIP configuration based on performance, power, and area constraints. The two major techniques are:

1. Simulation-Based Design Space Exploration

- Uses software simulators to evaluate various processor configurations before hardware implementation.
- **Advantages:**

- Provides detailed performance insights.
- Allows extensive testing without hardware fabrication.

- **Disadvantages:**
  - Time-consuming due to repeated simulations.

2. Scheduler-Based Design Space Exploration

- Uses **task scheduling algorithms** to predict processor performance.
- **Advantages:**
  - Faster than simulation-based methods.
  - Reduces the number of configurations to test.
- **Disadvantages:**
  - Less accurate compared to detailed simulation.

Comparison of Simulation-Based and Scheduler-Based DSE

Feature	**Simulation-Based DSE**	**Scheduler-Based DSE**
Accuracy	High	Moderate
Speed	Slow	Fast
Hardware Dependency	No (software-based)	Minimal
Use Case	Detailed evaluation	Quick estimation

Conclusion

ASIPs provide an efficient way to design processors optimized for specific applications. The **design process** involves application analysis, tool generation, and processor synthesis. **Design Space Exploration (DSE)** ensures the best balance of power, performance, and cost, using either **simulation-based** or **scheduler-based** techniques.

Unit IV: Memory and Interfacing:

Memory

Memory is a critical component in computing systems, responsible for storing and retrieving data. Different types of memory are used based on speed, capacity, and functionality.

Memory Write Ability and Storage Performance

- **Write Ability:** The ease with which data can be written or modified in memory.
- **Storage Performance:** Measured in terms of access time, latency, and bandwidth.
- **Trade-off:** Faster memory (e.g., cache) is smaller and expensive, while slower memory (e.g., hard drives) is larger and cheaper.

Common Memory Types

1. **Read-Only Memory (ROM):** Non-volatile memory used for firmware storage.
2. **Random-Access Memory (RAM):** Volatile memory for fast data access.
3. **Flash Memory:** Non-volatile storage used in SSDs and USB drives.

Composing Memories

- **Combining different types of memory** to achieve a balance between speed, size, and cost.
- Example: Using **SRAM for cache, DRAM for main memory, and SSD for storage.**

Memory Hierarchy and Cache

- **Registers → Cache → RAM → Secondary Storage** (Hierarchy based on speed and size).
- **Cache Memory:** A small, high-speed memory that stores frequently accessed data to improve performance.

Advanced RAM Types

1. **DRAM (Dynamic RAM):** Stores data using capacitors, requires frequent refreshing.
2. **FPM DRAM (Fast Page Mode DRAM):** Improves read/write speed by accessing entire memory pages.
3. **EDO DRAM (Extended Data Out DRAM):** Faster than FPM by keeping data available longer.
4. **SDRAM (Synchronous DRAM):** Synchronizes with CPU clock for faster access.
5. **RDRAM (Rambus DRAM):** High-speed memory with increased bandwidth for demanding applications.

Memory Management

- **Paging:** Divides memory into fixed-size pages to manage virtual memory.
- **Segmentation:** Divides memory into variable-sized segments based on program needs.
- **Cache Management:** Algorithms like **LRU (Least Recently Used)** determine which data to keep in cache.

Interfacing

Interfacing involves connecting processors with memory, input/output devices, and communication systems.

Arbitration

- Used in **multi-master systems** to resolve conflicts when multiple devices request access to a shared resource.
- **Methods:** Fixed priority, round-robin, time-division.

Multi-Level Bus Architectures

- **Single-Level Bus:** One bus shared by CPU, memory, and peripherals (simple but slow).
- **Multi-Level Bus:** Separate buses for different components (e.g., system bus, memory bus, I/O bus).

Serial Protocols

Serial communication protocols transfer data **one bit at a time** over a single wire.

1. **I^2C Bus (Inter-Integrated Circuit):**
    - Used for short-distance communication between microcontrollers and peripherals.
    - Supports multiple devices with simple wiring.
2. **CAN Bus (Controller Area Network):**
    - Used in automotive and industrial systems.
    - Allows multiple controllers to communicate without a central host.
3. **FireWire Bus (IEEE 1394):**
    - High-speed data transfer for multimedia applications.
    - Supports peer-to-peer communication.
4. **USB (Universal Serial Bus):**
    - Widely used for connecting peripherals like keyboards, mice, and storage devices.
    - Supports hot-swapping and plug-and-play.

Parallel Protocols

Parallel communication transmits **multiple bits at a time** over multiple wires.

1. **PCI (Peripheral Component Interconnect):**
    - High-speed interface for connecting expansion cards (e.g., graphics, network cards).
    - Supports plug-and-play.
2. **ARM Bus (Advanced RISC Machine Bus):**
    - Used in ARM-based embedded systems.
    - Optimized for low power and high efficiency.

Wireless Protocols

Wireless protocols enable communication without physical connections.

1. **IrDA (Infrared Data Association):**
    - Uses infrared light for short-range communication (e.g., remote controls).
2. **Bluetooth:**
    - Low-power wireless technology for short-range communication.
    - Used in mobile devices, headphones, and smart appliances.
3. **IEEE 802.11 (Wi-Fi):**
    - Standard for wireless LAN communication.
    - Enables high-speed internet access over long distances.

Conclusion

- **Memory systems** involve different storage types, speed optimizations, and management techniques.
- **Interfacing** connects components using **serial, parallel, and wireless protocols** to ensure smooth communication.

Unit V: Case Study – Digital Camera:

Case Study: Embedded System – Digital Camera

A **digital camera** is an embedded system that captures images and videos digitally. It consists of a microcontroller, image sensors, memory, and display components. The design of a digital camera involves both **user's perspective** (features and usability) and **designer's perspective** (hardware and software implementation).

User's Perspective

From a user's point of view, a digital camera should provide:

- **Ease of Use** – Simple interface with buttons for capture, zoom, and settings.
- **Image Quality** – High resolution, good color accuracy, and low noise.
- **Battery Life** – Efficient power consumption for long usage.
- **Storage** – Sufficient memory for photos and videos.
- **Connectivity** – USB, Wi-Fi, or Bluetooth for transferring images.

Designer's Perspective

From a designer's perspective, the digital camera must be optimized for:

- **Processing Speed** – Fast image capture and processing.

- **Sensor Integration** – Efficient use of CCD (Charge-Coupled Device) or CMOS sensors.
- **Memory Management** – Quick access to RAM and Flash storage.
- **Display and User Interface** – LCD integration for previewing images.
- **Power Optimization** – Minimizing battery consumption.

Requirement Specification

1. Non-Functional Requirements

These define the system's quality attributes, such as:

- **Performance** – Low-latency image processing.
- **Power Efficiency** – Optimized battery usage.
- **Reliability** – Ability to function in different lighting conditions.
- **Cost** – Affordable hardware and manufacturing.

2. Informal Functional Specification

An initial, high-level description of the camera's functionality:

- Capture, store, and display digital images.
- Support zoom, flash, and white balance adjustment.
- Save images in standard formats (JPEG, PNG).
- Allow image transfer via USB or wireless connection.

3. Refined Functional Specification

A detailed specification with precise technical requirements:

- **Sensor:** CCD or CMOS with a resolution of at least 12MP.
- **Microcontroller:** 32-bit processor for image processing.

- **Memory:** 2GB RAM, expandable storage via SD card.
- **Display:** 3-inch LCD with touch controls.
- **Battery:** Rechargeable Li-ion with 2000mAh capacity.

Design Alternatives

Different architectures can be used to implement the digital camera:

1. Microcontroller Alone

- A basic design using only a **microcontroller (MCU)** for image processing and storage.
- **Pros:** Simple, low cost, and easy to implement.
- **Cons:** Limited processing power, slow image processing, and lower quality.

2. Microcontroller and CCDPP (Charge-Coupled Device Processing Pipeline)

- Uses a **microcontroller** along with a **CCDPP** to handle image capture and processing.
- **Pros:** Faster processing than MCU alone, improved image quality.
- **Cons:** Increased cost and complexity.

3. Microcontroller and CCDPP with Fixed-Point DCT (Discrete Cosine Transform)

- Adds **Fixed-Point DCT**, an efficient image compression technique, to **reduce storage size** without losing much quality.
- **Pros:** Saves memory, faster image storage and retrieval.
- **Cons:** Requires additional processing hardware.

4. Microcontroller, CCDPP, and DCT (Full Implementation)

- The most advanced design, using **MCU + CCDPP + DCT** for high-speed image capture, processing, and compression.

- **Pros:** Best image quality, fast processing, and efficient storage.
- **Cons:** Highest cost and power consumption.

Conclusion

A **digital camera** is a complex embedded system requiring careful design decisions based on performance, cost, and power constraints. Different **design alternatives** balance trade-offs between simplicity, processing power, and efficiency.