LEC 1: Introduction to Computing

Embedded Systems

- **Definition**: Special-purpose computer systems designed for dedicated functions, often with real-time constraints. Embedded within complete devices including hardware and mechanical parts.
- **Examples**: DVD players, fax machines, home security systems, photocopiers, TVs.
- **Characteristics**:
  - Size & Weight
  - Speed
  - Power Dissipation
  - Reliability
  - Accuracy
  - Adaptability

Classification of Embedded Systems

- **Functionality**:
  - **Stand-alone Embedded Systems**: Self-contained devices (e.g., MP3 players, digital cameras).
  - **Real-time Embedded Systems**: Provide outputs within specified time constraints.
    - **Hard Real-time Systems**: Critical failure if time constraints are violated (e.g., missile control systems).
    - **Soft Real-time Systems**: Degraded quality but continue to operate (e.g., washing machines).
  - **Networked Embedded Systems**: Access resources via network interfaces (e.g., home security systems).
  - **Mobile Embedded Systems**: Portable devices with memory and resource limitations (e.g., mobile phones).
- **Performance**:
  - **Small Scaled Embedded Systems**: Single 8–16 bit microcontroller with on-chip RAM and ROM.
  - **Medium Scaled Embedded Systems**: 16–32 bit microcontroller/microprocessor with external RAM and ROM.
  - **Large Scaled Embedded Systems**: 32-64 bit multiple chips performing distributed tasks.

Major Application Areas

- **Home Appliances**: Dishwashers, washing machines, microwaves, security systems.
- **Office Automation**: Fax machines, printers, smart phones.
- **Security**: Face recognition, building security systems.
- **Academia**: Smart boards, OCR.
- **Instrumentation**: Signal generators, process instrumentation.
- **Telecommunication**: Routers, cellular phones.
- **Automobile**: Fuel injection controllers, GPS.
- **Entertainment**: MP3 players, video games.
- **Aerospace**: Navigation systems, flight attitude controllers.
- **Industrial Automation**: Assembly lines, monitoring systems.
- **Personal**: PDAs, smartphones.
- **Medical**: CT scanners, ECGs, glucose monitors.
- **Banking & Finance**: ATMs, cash registers.
- **Miscellaneous**: Elevators, smart cards.

Features of Embedded Systems

- **Specific Task**: Designed for particular applications.
- **Resource Limitations**: Limited memory, small or no keyboard/screen.
- **Performance Efficiency**: Optimized for maximum performance with minimal size and weight.
- **Reliability**: Highly dependable operations.
- **Real-time Constraints**: Operates within strict time limitations.

Hardware Elements in Embedded Systems

- **Basic Computer System**: Requires memory (program and data storage), support logic, and I/O devices.
- **Data Bus**: Bidirectional transfer of data.
- **Address Bus**: Carries addresses to memory locations.
- **Control Bus**: Transfers control signals.

Microprocessors and Microcontrollers

- **Microprocessor**:
    - General-purpose CPU focused on flexibility and performance.
    - Used in PCs, PDAs, cell phones.
- **Microcontroller**:
    - Microprocessor with integrated memory on a single chip.
    - Emphasizes size and cost reduction.

- Tailored user interfaces (e.g., TV remote controls).

Number Systems and Logic Gates

- **Binary System**: Used by computers (0 and 1 representing voltage levels).
- **Hexadecimal System**: Base 16, used for representing binary numbers.
- **ASCII Code**: Binary patterns for numbers, letters, control codes, and punctuation.
- **Logic Gates**: AND, OR, Tri-state buffer, XOR, NAND, NOR gates.
- **Adders**: Half Adder and Full Adder for arithmetic operations.

Computer System Components

- **CPU (Central Processing Unit)**: Brain of the computer, executes instructions.
- **Memory**: Stores program instructions and data.
- **I/O Devices**: Input and output information.

Inside the CPU

- **Registers**: Temporary storage within the CPU.
- **Arithmetic and Logic Unit (ALU)**: Performs arithmetic and logic functions.
- **Program Counter**: Points to the next instruction.
- **Instruction Decoder**: Interprets instructions and generates control signals.

LEC 2: Embedded System Architecture

Components of Embedded Systems

1. Hardware:

- **Processor/Controller:** Central Processing Unit (CPU), timers, interrupt controllers, I/O devices, memories, and ports. These components form the core of the embedded system, allowing it to process inputs and generate outputs.
- **System Memory:** Program Memory stores software programs, while Data Memory stores processed data.
- **I/O Ports:** Facilitate communication between the CPU and external devices or systems (e.g., USB, wireless ports).
- **Communication Ports:** Enable information exchanges with other devices or systems (e.g., USB ports, printer ports, wireless RF, and infrared ports).
- **User Interfaces:** Keypads, switches, buzzers, audio outputs, and lights for human interaction.
- **Sensors and Actuators:** Interact with the external environment (e.g., temperature, pressure sensors, actuators for displacement, acceleration).
- **ADC/DAC:** Analog-to-Digital Converters and Digital-to-Analog Converters for interfacing with analog sensors and actuators.
- **Diagnostics and Testing Components:** Ensure robust and reliable system operation.
- **System Support Components:** Provide essential services for system operation (e.g., power supply, clock, timers, interrupts).

2. Application Software:

- Performs various tasks, potentially concurrently. This software is essential for the functionality of the embedded system.

3. Real-Time Operating System (RTOS):

- Manages the application software, ensuring that tasks are executed in a timely and orderly manner. While not always necessary for small-scale systems, an RTOS is crucial for more complex systems.

Structure of Embedded Systems

1. Central Processing Unit (CPU):

- Executes software instructions, processes system inputs, and makes decisions to guide system operation. CPU selection can vary from 8-bit to 64-bit, depending on the application's requirements.

2. System Memory:

- Stores programs and data necessary for system operation. This includes program memory for software programs and data memory for processed data.

3. I/O Ports:

- Facilitate communication between the CPU and external devices or systems. Examples include communication ports (USB, wireless), user interfaces (keypads, switches), sensors, actuators, and data converters (ADC and DAC).

Hardware Design Factors

1. Processing Power:

- The speed and power consumption of the microprocessor are critical. It is measured in millions of instructions per second per milliwatt (MIPS/mW). More powerful processors offer higher computational capacities but at a higher cost.

2. Memory Requirements:

- The types and amounts of RAM and ROM needed for both the evaluation board and the target system. Adequate memory is essential for project success, though excessive memory increases production costs.

3. Peripherals:

- The selection of necessary peripherals is based on performance rather than cost. Built-in peripherals in MCUs are considered, with performance prioritized over the expense.

4. Reliability:

- Determining if the system should be fail-proof or if occasional failures are acceptable. This factor influences the design's robustness and reliability.

5. Future Upgrades:

- Consideration of how field upgrades will be performed to ensure the system can be updated and maintained over time.

CPU Architectures

1. Von Neumann Architecture:

- Uses a single storage memory for both data and program instructions. It is less efficient and does not support pipelining, making it suitable for simpler systems.

2. Harvard Architecture:

- Separates storage for data and program instructions, allowing for more efficient operation and pipelining. It is preferred for more complex and high-performance systems.

Software Components

1. Definition and Storage:

- Software components include programs necessary for functionality, stored in non-volatile memory (firmware).

2. Software Structure in Embedded Systems:

- **System Tasks:** Small programs handling distinct actions within the system, utilizing specific system resources, and requesting services via the kernel.
- **System Kernel:** Manages resources like memory, I/O devices, CPU, and other hardware components.
- **Services:** Service routines providing functionality to system resources, activated by polling or interrupts.

Classification of Embedded Systems

1. Small Embedded Systems:

- Consist of an MCU and a few components. Operate with minimal maintenance and very low cost. Software is single-tasked and rarely requires RTOS. Examples: Tire pressure monitoring, microwave controllers, electronic toy controllers.

2. Distributed Embedded Systems:

- Components like CPU, memory, and I/O are spread across several chips. Require maintenance and updates. Manage multiple tasks, often without an RTOS. Applications: Video processors, video game controllers, network processors.

3. High-Performance Embedded Systems:

- Require fast computations, robustness, fault tolerance, and high maintainability. Typically distributed and often use RTOS for managing multiple tasks. Produced in small quantities with high costs. Used in military and aerospace applications (e.g., flight controllers, missile guidance systems).

Life Cycle of an Embedded System

Design Constraints:

- **Functionality:** Ability to perform designed functions.
- **Cost:** Resources needed for conception, design, production, maintenance, and disposal.
- **Performance:** Ability to perform functions on time.
- **Size:** Physical space required.
- **Power and Energy:** Energy required to perform functions.
- **Time to Market:** Time from conception to deployment.
- **Maintainability:** Ability to remain functional throughout its mature life.

LEC 3: IoT and Interfacing with Peripherals

IoT Architecture

- **Nodes**: Devices such as sensors, actuators, processors, and memory that interact with the environment. Each node typically has a network interface and may or may not run the Internet Protocol (IP).
- **Cloud Servers**: Provide computational services and storage for the IoT system. They mediate between nodes and users by offering a variety of services.

Components of an IoT Node

- **Sensors**: Used to monitor and collect data from the environment.
- **Microcontrollers (MCU)**: Process sensor data and interface with wireless devices for connectivity.

IoT Applications

- **Smart Homes**: Automate tasks like switching on air conditioning before arrival, turning off lights remotely, or unlocking doors for temporary access.
- **Smart Cities**: Address urban challenges like pollution, traffic congestion, and energy shortages. Sensors help in finding parking slots, detecting tampering issues, and managing electricity systems.
- **Smart Healthcare**: Empower people to live healthier lives through wearable devices that collect health data for personalized analysis and tailored health strategies.

IoT Challenges

- **Power Management**: IoT devices often operate on rechargeable batteries. Energy harvesting from ambient sources (solar, wind, thermal, etc.) is crucial.
- **Security**: Ensuring data encryption, authentication, and protection against side-channel attacks and privacy breaches is essential.

I/O Interfacing in Embedded Systems

- **Serial Communication Interfaces**: Includes RS-232, RS-422, and RS-485.
- **Synchronous Serial Communication**: Interfaces such as I2C, SPI, SSC, and ESSI.
- **USB and Multimedia Cards**: For data transfer and storage.
- **Network Interfaces**: Ethernet, LonWorks, etc.
- **Fieldbuses**: CAN-Bus, LIN-Bus, PROFIBUS.

- **Timers and Discrete IO**: Includes PLL, Capture/Compare Units, General Purpose Input/Output (GPIO).
- **Analog/Digital Converters**: ADC and DAC for analog signal processing.
- **Debugging Ports**: JTAG, ISP, ICSP, BDM, BITP, and DP9 ports.

Digital System Performance

- **Computation Performance**: Enhanced by advancements in CMOS technology.
- **Communication Performance**: Increasingly significant with higher computation capabilities. On-chip and off-chip communication needs to be efficient.

On-chip Communication Architectures

- **Point-to-Point**: Direct connection between two components.
- **Bus Architectures**: Shared communication pathways (e.g., AMBA, CoreConnect).
- **Network-on-Chip (NoC)**: Advanced, scalable communication networks within a chip.

Interconnection Requirements

- **High Data Transfer Speed**: Essential for both serial and parallel data.
- **Fault Tolerance**: Ability to detect and correct errors during transmission.
- **Low Power Consumption**: Critical for energy efficiency.

Data Transfer Types

- **Simplex**: One-way communication.
- **Half-Duplex**: Two-way communication but not simultaneously.
- **Full-Duplex**: Simultaneous two-way communication.
- **Serial vs. Parallel Transfer**: Serial sends data one bit at a time, while parallel sends multiple bits simultaneously.
- **Synchronous vs. Asynchronous**: Synchronous transfers bits without start/stop bits, whereas asynchronous uses start/stop bits for each byte.

Specific Data Transfer Methods

- **Asynchronous Serial Transfer**: Includes start and stop bits, gaps between bytes, ensuring bits are synchronized in duration.
- **Synchronous Serial Transfer**: Continuous bit stream without start/stop bits, requiring the receiver to group bits appropriately.

Serial Communication Examples

- **Single-Ended (RS-232C)**: Common for simple, short-distance communication.

- **Differential (RS-422/RS-485)**: Enhanced noise immunity, with RS-485 supporting multidrop capability for connecting multiple devices.

LEC 4: Addressing Modes in Embedded Systems

Addressing Modes

- **Definition**: Addressing mode specifies the location of data required by an operation.
- **Importance**: Understanding addressing modes is crucial for efficient programming and operation of microcontrollers and processors.

Simple Addressing Modes

- **Register Addressing Mode**:
  - Operands are in processor registers.
  - Example: `MOV AX, BX` where **AX** and **BX** are registers.
  - **Efficiency**: Most efficient mode due to fast access to register data.
- **Immediate Addressing Mode**:
  - Operand is explicitly specified in the instruction.
  - Example: `MOV AX, 5` where **5** is an immediate value.
  - **Usage**: Commonly used for constants and initial values.
  - **Efficiency**: Data is readily available as part of the instruction.

Memory Addressing Modes

- **Direct Addressing Mode**:
  - The address of the operand is given explicitly within the instruction.
  - Example: `MOV AX, [1234H]` where **1234H** is a direct memory address.
  - **Usage**: Simple variable access.
- **Indirect Addressing Mode**:
  - The address of the operand is held in a register.
  - Example: `MOV AX, [BX]` where **BX** contains the memory address of the operand.
  - **Flexibility**: Allows for dynamic address calculation and efficient use of memory.
- **Based Addressing Mode**:
  - Effective address is calculated as base register + displacement.
  - Example: `MOV AX, [BX + 04H]` where **BX** is the base register and **04H** is the displacement.
  - **Usage**: Accessing array elements and structure fields.

- **Indexed Addressing Mode**:
  - Effective address is calculated as (index register * scale factor) + displacement.
  - Example: **`MOV AX, [SI + 2*DI]`** where **`SI`** is the index register, **`2`** is the scale factor, and **`DI`** is the displacement.
  - **Usage**: Accessing elements of arrays, particularly with elements of size 2, 4, or 8 bytes.
- **Based-Indexed Addressing Mode**:
  - Effective address is computed as base register + index register + displacement.
  - Example: **`MOV AX, [BX + SI + 04H]`** where **`BX`** is the base register, **`SI`** is the index register, and **`04H`** is the displacement.
  - **Usage**: Accessing multi-dimensional arrays and complex data structures.
- **Based-Indexed with Scale Factor**:
  - Effective address is computed as base register + (index register * scale factor) + displacement.
  - Example: **`MOV AX, [BX + 2*SI + 08H]`** where **`BX`** is the base register, **`SI`** is the index register, **`2`** is the scale factor, and **`08H`** is the displacement.
  - **Usage**: Efficiently accessing two-dimensional arrays and arrays of records.

Operand and Address Size Override Prefixes

- **16-bit vs. 32-bit Addressing**:
  - **Default Size**: Determined by the D bit in the CS segment descriptor.
    - **`D = 0`**: Default size is 16 bits.
    - **`D = 1`**: Default size is 32 bits.
  - **Override Prefixes**:
    - **`66H`**: Operand size override prefix.
    - **`67H`**: Address size override prefix.
  - **Usage**: Mix 16-bit and 32-bit data and addresses within the same program.

Examples and Applications

- **Insertion Sort**:
  - Example of sorting an integer array using the insertion sort algorithm in assembly.
  - Demonstrates use of addressing modes to manipulate array elements.
- **Binary Search**:
  - Efficient search algorithm implementation in assembly.
  - Uses addressing modes to access array elements during search operations.

- **Array Manipulation**:
    - **One-Dimensional Arrays**: Summing elements, accessing by displacement calculation.
    - **Multi-Dimensional Arrays**: Calculating displacement for row-major and column-major order, accessing specific elements based on calculated addresses.

Recursion in Assembly

- **Concept**: Recursive procedures call themselves, either directly or indirectly.
- **Applications**: Naturally expressed algorithms like factorial computation and quicksort.
    - **Factorial**: Recursive calculation of factorial using stack for activation records.
    - **Quicksort**: Recursive sorting algorithm selecting partition elements and sorting subarrays.

LEC 5: ARM Architecture in Embedded Systems

Introduction to ARM Architecture

- **ARM (Advanced RISC Machines)**:
  - Develops and licenses its architecture to other companies.
  - Other companies design products like SoCs (System-on-Chips) and SoMs (System-on-Modules) using ARM's architecture.
  - ARM cores are integrated into various products.

ARM vs. 8086

- **ARM**:
  - 32-bit CPU.
  - RISC (Reduced Instruction Set Computing) design.
  - Load/store architecture: instructions operate on registers, not directly on memory.
  - Large register set.
  - Pipelined execution.
  - Conditional execution of all instructions.
  - Multiplication and load/store multiple instructions are complex.
  - No traditional stack; uses link register.
  - PC (Program Counter) is a regular register.
- **8086**:
  - 16-bit CPU.
  - CISC (Complex Instruction Set Computing) design.
  - Instructions operate on both registers and memory.
  - Smaller register set.
  - Sequential execution of shifts/rotations.
  - It has a traditional stack.

Data Sizes and Instruction Sets

- **Data Sizes**:
  - Byte: 8 bits.
  - Halfword: 16 bits (two bytes).
  - Word: 32 bits (four bytes).
- **Instruction Sets**:
  - 32-bit ARM Instruction Set.
  - 16-bit Thumb Instruction Set.

  - ARM processors commonly use both instruction sets for different operational efficiencies.

ARM Processor Modes

- **Operating Modes**:
  - **User Mode**: Unprivileged mode for most tasks.
  - **FIQ (Fast Interrupt Request)**: High priority interrupt mode.
  - **IRQ (Interrupt Request)**: Low priority interrupt mode.
  - **SVC (Supervisor Call)**: Entered on reset and software interrupts.
  - **Abort**: Handles memory access violations.
  - **Undef**: Handles undefined instructions.
  - **System**: Privileged mode with same registers as user mode.

ARM Registers

- **General-Purpose Registers**:
  - 37 registers, all 32-bits long.
  - 30 general-purpose registers.
  - Specific sets accessible depending on the current processor mode.
- **Special Registers**:
  - **PC (Program Counter, R15)**: Holds the address of the next instruction.
  - **LR (Link Register, R14)**: Stores return addresses for subroutine calls and interrupts.
  - **SP (Stack Pointer, R13)**: Points to the current stack top.
  - **CPSR (Current Program Status Register)**: Holds the current state of the processor.
  - **SPSR (Saved Program Status Register)**: Stores a copy of the CPSR during exceptions.

ARM Instruction Set Features

- **3-Operand Instructions**:
  - Example: **`ADD Rd, Rn, Operand2`** where Rd is the destination register, Rn is the first operand, and Operand2 is the second operand.
  - Efficient for arithmetic and logical operations.
- **Load/Store Architecture**:
  - Operations are performed on registers rather than directly on memory.
  - Memory access is handled by specific load/store instructions, improving performance and simplicity.
- **Conditional Execution**:

    - All instructions can be conditionally executed based on the status of the CPSR flags.
    - Reduces the need for branching, thus optimizing execution flow.
- **Pipelined Execution**:
    - ARM processors use pipelining to improve instruction throughput.
    - Multiple instructions are processed simultaneously at different stages.

ARM Development in Embedded Systems

- **Prototype Development**:
    - ARM processors are commonly used in developing embedded system prototypes.
    - Involves programming, debugging, and simulating assembly language programs.
    - Focus on interfacing techniques and peripheral integration.
- **Peripheral Interfaces**:
    - Common interfaces include USB, I2C, SPI, CAN, and LIN.
    - ARM processors are designed to handle multiple peripheral interfaces efficiently.

LEC 6: Embedded Systems Interfacing

Microcontrollers and Interfacing

Microcontrollers:

Microcontrollers are integral components of embedded systems, designed to manage hardware interfaces and execute specific tasks efficiently. They integrate various peripherals, such as timers, communication interfaces, and ADCs (Analog-to-Digital Converters), which allow them to interact with other hardware components and perform diverse functions.

Interfacing:

Interfacing involves the techniques used to connect microcontrollers with peripherals and other devices. Effective interfacing ensures smooth communication and control over hardware components.

Communication Protocols:

Serial Communication:

Serial communication protocols involve data transmission over a single data line, making them suitable for long distances due to lower capacitance issues. Common serial communication protocols include:

- **I2C (Inter-IC):**
  - A two-wire protocol (SDA and SCL) for short-distance communication between ICs.
  - Supports multiple speed modes (standard, fast, high-speed) and various addressing schemes.
- **SPI (Serial Peripheral Interface):**
  - A four-wire protocol (MISO, MOSI, SCLK, SS) suited for high-speed data transfer over short distances.
  - Commonly used for communication between microcontrollers and peripherals like sensors and memory devices.
- **UART (Universal Asynchronous Receiver/Transmitter):**
  - Used for asynchronous serial communication between microcontrollers and peripheral devices.
  - Does not require a clock line; relies on agreed-upon baud rates for data transmission.

- **RS-232:**
  - A standard for serial communication, traditionally used for connecting PCs and peripherals.
  - Supports longer distances and higher voltage levels.
- **USB (Universal Serial Bus):**
  - Provides high-speed data transfer and supports multiple devices through a tiered star topology.
  - Commonly used for connecting peripherals to computers.

Parallel Communication:

Parallel communication involves transmitting multiple bits simultaneously over multiple data lines, which is suitable for short distances due to potential signal degradation over longer cables.

- **PCI (Peripheral Component Interconnect):**
  - A high-performance parallel communication protocol used in computers for interconnecting chips, boards, and memory subsystems.
  - Supports synchronous bus architecture with multiplexed data/address lines.

Wireless Communication:

Wireless communication enables data transfer without physical connections, using protocols like:

- **Bluetooth:**
  - Supports low-cost, short-range communication within a 10-meter range.
  - Commonly used for connecting personal devices like phones, headphones, and keyboards.
- **IEEE 802.11 (Wi-Fi):**
  - Provides high-speed data transfer over longer distances, commonly used for wireless LANs.
  - Defines parameters for PHY (Physical Layer) and MAC (Media Access Control) layers.
- **IrDA (Infrared Data Association):**
  - A protocol for short-range, point-to-point infrared data transmission.

Error Detection and Correction:

Protocols often include mechanisms to ensure data integrity during transmission. Common techniques include:

- **Parity Bits:**
  - Simple error detection mechanism adding an extra bit to data for parity checking (even or odd).
- **Checksums:**
  - A calculated value based on data content used for error detection by comparing sent and received checksum values.
- **CRC (Cyclic Redundancy Check):**
  - An advanced error detection method using polynomial division to generate a checksum.

Sensor and Actuator Interfacing:

Analog Input and Output:

- **Analog Input:**
  - ADCs (Analog-to-Digital Converters) convert analog signals from sensors (e.g., temperature, pressure) to digital data for processing by the microcontroller.
  - Important parameters include resolution and sampling rate.
- **Analog Output:**
  - Typically simulated using PWM (Pulse Width Modulation) to create analog-like signals from digital outputs.

Digital Input/Output (I/O):

- Techniques for handling binary data (0s and 1s) involve configuring microcontroller pins as either input or output for various applications, such as controlling LEDs, reading button states, or interfacing with digital sensors.

Interfacing Techniques:

- **UART (Universal Asynchronous Receiver/Transmitter):**
  - A serial communication protocol for asynchronous data exchange, commonly used for debugging and communication with peripherals.
- **SPI (Serial Peripheral Interface):**
  - A synchronous serial communication protocol used for short-distance communication, especially in microcontroller systems.

Lab Exercises and Practical Applications:

- Practical exercises involving digital I/O, analog I/O, and serial communication help solidify the understanding of interfacing techniques. These may include:
    - Controlling LEDs and reading button states.
    - Interfacing with sensors to read environmental data.
    - Implementing communication modules for data exchange between microcontrollers and other devices.