12th Task in Embedded Systems

🔷 MCU Clock System

The clock system is what controls the speed at which a microcontroller (MCU) runs. It provides timing signals that determine how fast the MCU executes instructions.

Key components:

Clock source: Can be internal (like an RC oscillator) or external (like a crystal oscillator).

PLL (Phase-Locked Loop): Multiplies the input clock to a higher frequency.

Prescalers: Divide the clock frequency to a suitable value for different modules.

Example: If your external crystal is 8 MHz and you use a prescaler of 8, your CPU runs at 1 MHz.

🔷 Interrupt Fundamentals & Architecture

An interrupt is a signal that tells the MCU to stop its current task and handle something more important immediately.

Why interrupts?
Instead of continuously checking (polling) for events like button presses, the MCU can sleep or do something else, and automatically respond when the event happens.

🔷 Architecture basics:

Each interrupt has an interrupt vector (a specific address in memory).

In ARM-based MCUs, the NVIC (Nested Vectored Interrupt Controller) manages the interrupts, priorities, and nesting.

Think of it as someone tapping your shoulder to say "Hey, something just happened!"

🔷 Interrupt Handling & Startup Process

When an interrupt occurs, the MCU follows these steps:

Handling Process:

-Stops the current execution.

-Saves context (register values, program counter, etc.).

-Jumps to the ISR (Interrupt Service Routine) — special function for that interrupt.

-After finishing the ISR, the MCU restores context and resumes what it was doing**.**

🔷 Startup Process:

On reset or power-up, the MCU loads the vector table, which holds the addresses of all ISRs.

The system knows where to jump when each interrupt occurs.

Example: If Timer1 interrupt happens, the MCU jumps to the Timer1 ISR defined in the vector table.

🔷 Software Mechanisms for Managing Interrupts

Software offers tools to help manage and organize interrupts efficiently.

Enabling/Disabling:

You can enable or disable interrupts globally or individually, depending on the critical sections of your code.

Priorities:

Some MCUs allow priority levels, so high-priority interrupts can preempt lower ones.

Writing Efficient ISRs:

- Keep the ISR short and fast.
- Avoid delays or long processing inside the ISR.
- Use volatile keyword for variables shared between ISR and main code.

Clearing flags:

In many MCUs, you must manually clear the interrupt flag inside the ISR to prevent it from being called again immediately.