**MCU Architecture**

**1. MCU Packages**

* Every microcontroller unit (MCU) comes in a **package** – the physical form with pins that connect to a circuit board.
* Package names follow **industry standards**. The number at the end of the name tells you how many pins it has.
  + Example: a package ending in *64* has **64 pins** (16 pins per side on a square chip).
* On an **STM32 Nucleo board**, the MCU sits on the **main board**, but there’s also a smaller **ST-Link sub-board**.
  + ST-Link = used only for **debugging and programming**.
  + Main board = runs your software.
* Different package options exist for the same MCU:
  + **Mounting scheme**: how the part is soldered (surface mount vs through-hole).
  + **Pin count**: ranges from ~48 to 100 pins.
  + **Physical size**: from ~3 mm to ~14 mm per side.
  + **Resources**: some packages even differ in the number of timers or peripherals inside.
* **Why package choice matters**:
  + More pins = more peripheral options.
  + Larger package = takes more PCB space.
  + Manufacturing process may limit mounting schemes.

**2. System-on-a-Chip (SoC)**

* An MCU is a **System-on-a-Chip**.
  + This means CPU, memory, and peripherals are integrated into one silicon die.
* **Advantages**: smaller, cheaper, uses less power.
* **Disadvantages**: harder to expand — you can’t just “add RAM” like in a PC.
* **Solution**: manufacturers sell many versions of the same MCU family with different memory sizes and peripheral options.
* A diagram of a computer

  AI-generated content may be incorrect.Historical note: decades ago, all these blocks were **separate chips** on a board. Today, they’re merged into one part.

**3. CPU and Memory**

* **CPU** = heart of the MCU, executes your software.
* **Flash memory**:
  + Stores **machine code (software instructions)**.
  + **Persistent**: keeps data even when powered off.
  + **Limitations**:
    - Slow to write.
    - Can “wear out” after many writes.
  + Best use: program storage or configuration data that changes rarely.
  + Example: product settings stored once during installation.
* **RAM**:
  + Stores **dynamic data** (variables, stack, buffers).
  + **Fast** and doesn’t wear out.
  + **Volatile**: data is lost when power is off.
  + Some MCUs allow **battery backup RAM**, so it survives short power losses.
* **Example sizes (STM32 Nucleo)**:
  + 512 KB Flash
  + 96 KB RAM
* Compared to a laptop or phone, this is tiny. But for MCU tasks (like reading sensors, controlling motors, handling communication), it’s often plenty.
* If **internal memory isn’t enough**, you can connect **external memory chips**. This is slower but sometimes necessary.

**4. System Bus**

* Inside the MCU, all parts must communicate → this is handled by the **system bus**.
* Think of it like a **city bus system**: passengers (data) move between different stops (CPU, RAM, Flash, peripherals).
* **Parallel bus**: moves many bits at once, very fast, but requires lots of internal wires.
* Larger systems (PCs) also use buses, but often external.
* Real MCUs usually have **several buses** connected together to avoid bottlenecks.
* You usually don’t worry about buses directly, except with **DMA controllers**, which rely on bus efficiency.

**5. Peripherals**

* **Peripherals** are hardware modules that extend the CPU’s basic capabilities.
* They connect to the **system bus** (for configuration by the CPU) and to **I/O pins** (to interact with the outside world).
* Examples:
  + **UART (Universal Asynchronous Receiver/Transmitter)**
    - Sends and receives data serially (1 bit at a time).
    - Example: talking to a PC console or GPS module.
    - USART adds synchronous mode (rarely used).
  + **Timers**
    - *System timer*: provides a tick (e.g., 1 ms).
    - *General-purpose timers*: measure time, count events, or generate PWM.
  + **GPIO (General Purpose I/O)**
    - Digital pins you can configure as input (switch) or output (LED).
  + **ADC (Analog to Digital Converter)**
    - Converts analog voltage (e.g., microphone signal) into a digital value.
  + **I²C (Inter-Integrated Circuit)**
    - Serial bus for communication with sensors, displays, or other chips.
  + **SPI (Serial Peripheral Interface)**
    - Another serial bus, often faster than I²C. Example: external flash memory.
  + **DMA (Direct Memory Access)**
    - Moves data automatically without CPU involvement.
    - Example: sending data from RAM to SPI buffer.
    - Frees CPU to do “smarter” work.

👉 An MCU often has **multiple instances** of each peripheral. Example: 3 UARTs, 4 timers, several ADC channels.

**6. Pin Multiplexer and Electrical Config**

* **Problem**: MCU has more peripheral signals than pins available.
* **Solution**: Pin multiplexer → maps internal signals to external pins.
* Not fully flexible: each pin can only support a limited set of functions.
* IDE tools help avoid conflicts.
* **GPIO exception**: any pin can always be used as GPIO.
* **Electrical configuration**: each pin can also be set as input or output, with optional pull-up/pull-down resistors.
* **Package choice**:
  + More pins = more mapping options.
  + Example: 100-pin package makes it easier to use multiple peripherals simultaneously.

**7. Interrupt Controller**

* An **interrupt** is a way for hardware to get the CPU’s attention quickly.
* CPU has limited interrupt inputs (e.g., 1–2).
* MCU might have **50+ interrupt sources** (UART, GPIO, timers).
* **Interrupt controller**:
  + Collects all interrupt requests.
  + Tells CPU which source triggered.
  + Supports **priorities**, so urgent events (like emergency stop) can override less urgent ones.
* Example: UART receives a character → raises interrupt → CPU runs “UART handler code” → then goes back to its previous work.

**8. Clocks**

* **Digital hardware runs step by step** → the clock signal sets the speed.
* MCUs often have **multiple clocks** for different modules (CPU, bus, peripherals).
* **External crystal**: provides very accurate timing (good for precise time measurement).
* **Internal oscillator**: cheaper, less accurate, sometimes “good enough.”
* **Power trade-off**:
  + Higher frequency = faster, but uses more power.
  + Lower frequency = slower, but saves power.
* **Trick**: Turn off clocks to unused modules to save power.
* **RTC (Real Time Clock)**:
  + Keeps calendar time (hours, minutes, seconds).
  + Often has battery backup so it runs even when the MCU is off.

**9. Debug Interface**

* Essential for software development.
* Functions:
  + Control MCU from debugger (step, pause, inspect).
  + Read/write memory.
  + Program flash with your code.
* Standards:
  + **JTAG**: uses 5 signals.
  + **SWD (Serial Wire Debug)**: ARM standard, uses only 2 signals (common in STM32).
  + **SWIM**: single-wire debug (other MCU families).
* By default, **debug pins are reserved** on startup so you can always connect a debugger.
* Later, you can reassign those pins for other functions if needed.

**10. Software Setup**

* When MCU powers on, CPU starts executing code from **flash**.
* First steps of your software: **configure hardware** (clocks, peripherals, pins, interrupts).
* IDE can help by generating setup code, but ultimately **you are responsible**.
* If code misbehaves, people blame you, not the IDE.

✅ **Key Takeaway**:  
An MCU is like a **mini-computer in a single chip**.

* CPU runs your code.
* Flash holds the program, RAM holds the data.
* A system bus ties everything together.
* Peripherals handle communication, timing, analog signals, etc.
* The pin multiplexer decides what each pin does.
* Interrupts make the CPU responsive.
* Clocks set speed and power usage.
* Debug interfaces make programming possible.