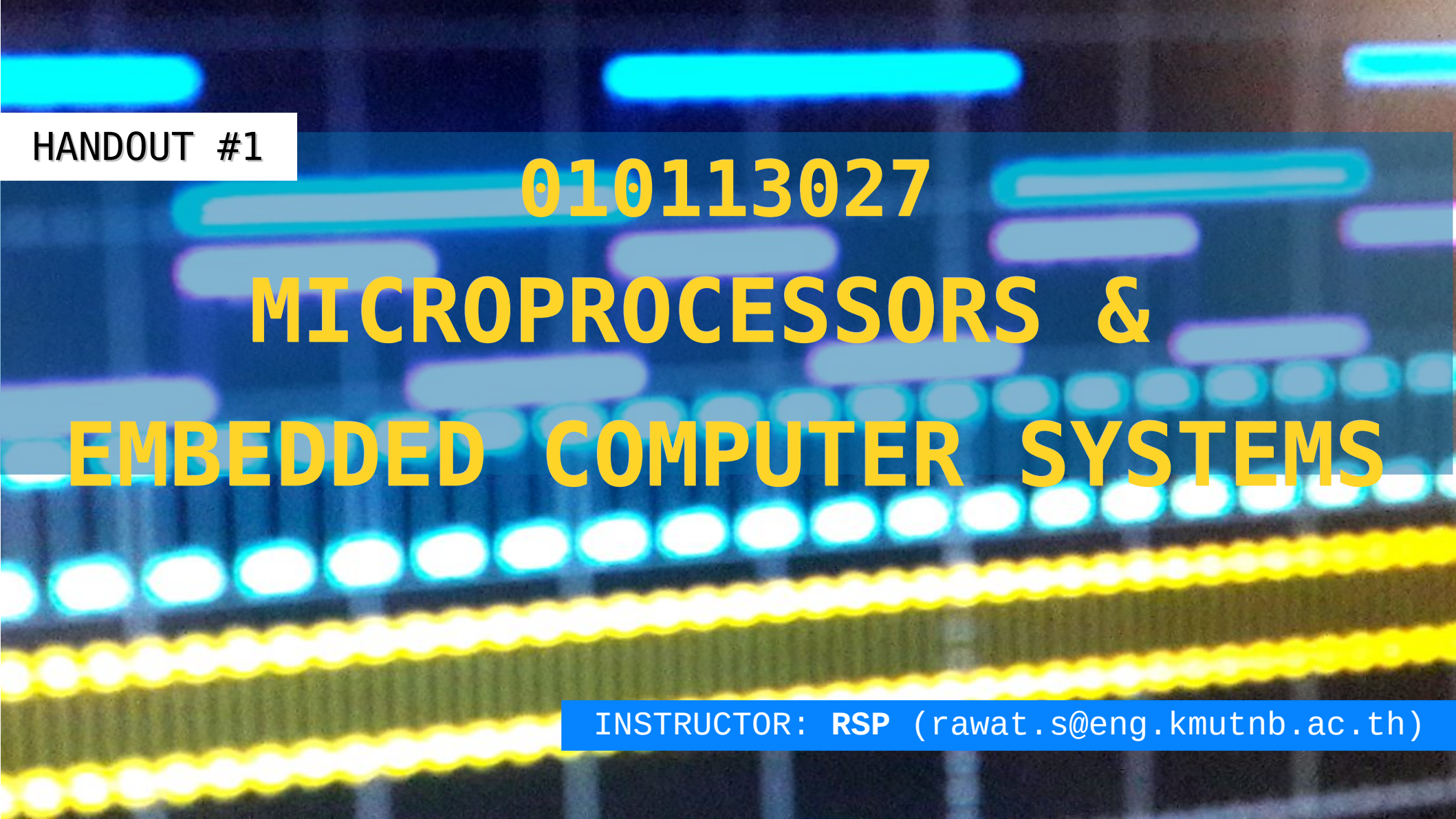


HANDOUT #1

010113027
MICROPROCESSORS & EMBEDDED COMPUTER SYSTEMS

INSTRUCTOR: **RSP** (rawat.s@eng.kmutnb.ac.th)

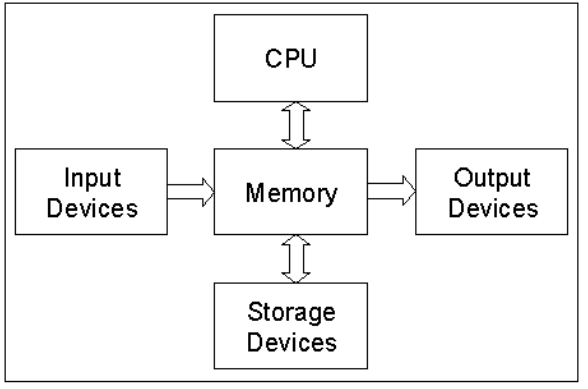
Intro to Embedded Systems Engineering

🔵 What is a computer?

- A computer is an **electronic system** that processes data according to a set of **instructions** (programs).
- Computer exist in many different forms: ranging from high-performance computers in a data center to tiny embedded chips inside everyday devices.

🔵 Main components of a computer:

- **CPU**: used to perform operations (instructions).
- **Memory & storage**: used to save data (memory, disks)
- **I/O**: the input and output of the system



🔵 Computers: Devices vs. Systems

- **Device** = A single electronic unit performing specific functions (e.g., a sensor module or a microcontroller board)
- **System** = A complete setup combining multiple devices / subsystems to perform complex task

CPU Architectures and Organizations

🔵 What is a CPU (Central Processing Unit)?

- The CPU is the "brain" of a computer. It executes **instructions** and manages data flow between function units, registers, memory and peripherals. It mainly consists of the ALU (Arithmetic Logic Unit), control unit, and registers.

🔵 What is an Instruction Set?

- This is the collection of all instructions that a CPU can understand and execute, like ADD, SUB, LOAD, STORE, and JUMP. Each instruction tells the CPU what operation to perform.

🔵 What is an Instruction Set Architecture (ISA)?

- ISA is the interface between software and hardware.
- It defines the CPU's instruction set, data types, registers, memory addressing modes, and how instructions are executed.
- The CPU inside an MPU / MCU implements a specific ISA.
- Programs (binary executables) compiled for a different ISA cannot run on that MCU.

CPU Architectures and Organizations

- **CPU Architecture** refers to the functional design and the instruction set of a CPU. It essentially defines the interface visible to software, with a focus on:
  - What instructions the CPU can execute (ISA)
  - Number and type of registers
  - Memory addressing modes
  - Data types supported (8-bit, 16-bit, 32-bit, 64-bit, floating-point)

- **CPU Organization** refers to the internal implementation of the CPU, i.e., how the architecture is physically realized inside the chip. It focuses on how to implement the ISA, such as:
  - Number and types of functional units (ALU, FPU, multipliers)
  - Control unit design and datapath design
  - Pipeline stages (fetch, decode, execute, memory, write-back)
  - Cache structure, bus connections, and internal data paths

CPU Architectures and Organizations

🔵 Differences between Processors, CPUs, and Microprocessors

- **Processor**: A general term for any software-programmable device that processes data.
- **CPU**: It is the main processor in a computer responsible for executing instructions, performing arithmetic, logic, control, and I/O operations.
- **Microprocessor**: It is a CPU implemented on a single integrated circuit (IC), which combines ALU, control unit, and registers on one chip. It requires external memory and peripherals to function.

🔵 Differences between Microprocessors / Microcontrollers

- **MPU = Microprocessor Uni**t: Focused on computation, relies on external components.
- **MCU = Microcontroller Unit**: All-in-one chip for control tasks, suitable for embedded applications.

Microcontrollers and Embedded Systems

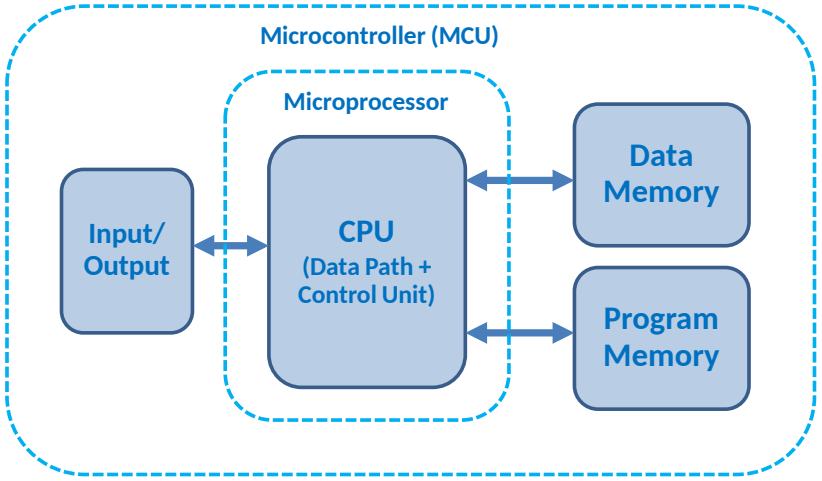
What is a Microcontroller?

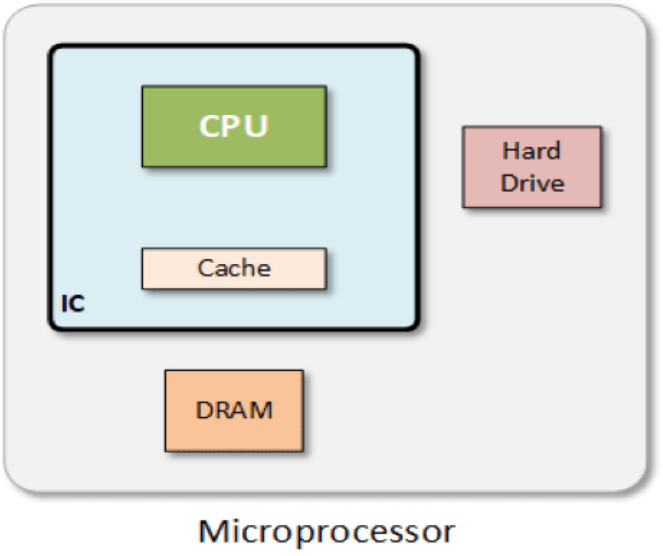
- A **microcontroller (MCU)** is a small, self-contained computer on a single chip.
- It's designed to control specific tasks in an **embedded system**.
- Think of it as the "brain" for a device, handling operations without the need for a full-sized computer.
- Common microcontroller architectures include **ARM Cortex-M**, **RISC-V**, **AVR**, and **PIC/dsPIC, MIPS, Xtensa**.

What is an Embedded System?

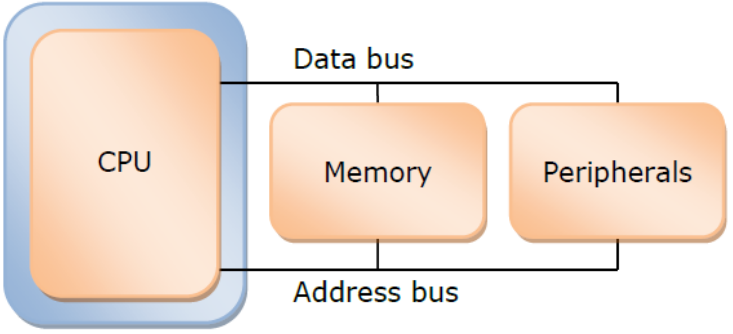
- An embedded system is a **specialized computer** built to perform one or a few dedicated tasks, "embedded" or integrated a larger device or machine.
- Unlike a **general-purpose computer**, like a PC or laptop, which can run a wide variety of software programs, an embedded system is optimized for a specific task.

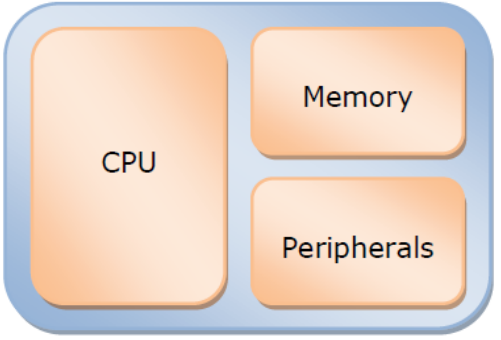
MPU vs. MCU





Source: https://semiengineering.com/mpu-vs-mcu/





MPU vs. MCU

- Over recent years the lines have *blurred* between microcontrollers (MCUs) and microprocessors (MPUs).
- Originally, an MCU is a self-contained system with peripherals, memory, and a CPU that is designed to perform specific tasks.
- Nowadays, although this is still the case, it's very common to attach additional external memory, as the MCUs are powerful enough to support more sophisticated applications.
- One key difference between MCUs and MPUs is software and development.
- An MPU (32-bit or 64-bit) will support rich OS like Linux or Android and the related software stack(bootloader, OS kernel, device drivers, etc.), while an MCU traditionally will focus on bare metal and RTOSes.
- It is up to the software developer to decide which software environment and ecosystem fits best for their application.

Embedded Systems vs. General-Purpose Computers

- *Modern embedded systems are often based on microcontrollers, but microprocessors (using external chips for memory and peripheral interface circuits) are also common, especially in more complex systems.*
- *Embedded systems are dedicated to specific tasks and optimized to reduce the size and cost of the product and increase the reliability and performance.*

- *Compared with general-purpose computers with multiple functionalities, embedded systems are often dedicated to specific tasks.*
- *For example, an embedded airbag control system is only responsible for detecting collision and inflating the airbag when necessary.*
- *Another example is an embedded controller in an air conditioner (or a digital thermostat), which is only responsible for monitoring and regulating the room temperature.*

Complexity of Embedded Systems

- Embedded systems differ in both hardware and software complexities.
  - They can use microprocessors or microcontrollers ranging from 8-bit, 16-bit, 32-bit and 64-bit options.
  - Embedded systems can be classified based on their complexity and performance into small-scale, medium-scale and large-scale.
- A general-purpose system has full-scale operating system (OS) support, while embedded systems may or may not have OS support at all.
- Many small-sized embedded systems are designed to perform simple tasks and thus do not need operating system support.
- However, many real-time embedded systems have a real-time kernel or real-time OS (RTOS).

Examples of MCUs with an 8-bit CPU

- An 8-bit CPU is defined by the width of its **ALU and general-purpose registers**, which handle 8-bit data at a time.

  Examples: AVR ATmega, PIC16F / PIC18F, STM8

- The **instruction word** (opcode + operands) may be larger than 8 bits or variable-length.
- **Memory addressing** is often wider than 8 bits.

AVR ATmega (Atmel / Microchip)
- fixed-length instruction
- instruction size: 2 or 4 bytes (Most instructions are two bytes long)

STM8 (STMicroelectronics)
- variable-length instruction
- instruction size: 1-5 bytes (The average instruction is 2 bytes long)

Examples of AVR Instruction Set

Opcode	Mnemonics	Operands	Description	Operation	Flags	#Clocks
Arithmetic and Logic Instructions						
0000 11rd dddd rrrr	ADD	Rd, Rr	Add without Carry	Rd ← Rd+Rr	Z,C,N,V,S,H	1
0001 11rd dddd rrrr	ADC	Rd, Rr	Add with Carry	Rd ← Rd+Rr+C	Z,C,N,V,S,H	1
1001 0110 KKdd KKKK	ADIW	Rd, K	Add Immediate to Word	Rd ← Rd+1:Rd+K	Z,C,N,V,S	2
0001 10rd dddd rrrr	SUB	Rd, Rr	Subtract without Carry	Rd ← Rd-Rr	Z,C,N,V,S,H	1
0101 KKKK dddd KKKK	SUBI	Rd, K	Subtract Immediate	Rd ← Rd-K	Z,C,N,V,S,H	1
0000 10rd dddd rrrr	SBC	Rd, Rr	Subtract with Carry	Rd ← Rd-Rr-C	Z,C,N,V,S,H	1
0100 KKKK dddd KKKK	SBCI	Rd, K	Subtract Immediate with Carry	Rd ← Rd-K-C	Z,C,N,V,S,H	1
1001 0111 KKdd KKKK	SBIW	Rd, K	Subtract Immediate from Word	Rd+1:Rd ← Rd+1:Rd-K	Z,C,N,V,S	2
0010 00rd dddd rrrr	AND	Rd, Rr	Logical AND	Rd ← Rd & Rr	Z,N,V,S	1
0111 KKKK dddd KKKK	ANDI	Rd, K	Logical AND with Immediate	Rd ← Rd & K	Z,N,V,S	1
0010 10rd dddd rrrr	OR	Rd, Rr	Logical OR	Rd ← Rd|Rr	Z,N,V,S	1
0110 KKKK dddd KKKK	ORI	Rd, K	Logical OR with Immediate	Rd ← Rd|K	Z,N,V,S	1
0010 01rd dddd rrrr	EOR	Rd, Rr	Exclusive OR	Rd ← Rd^Rr	Z,N,V,S	1
1001 010d dddd 0000	COM	Rd	One's Complement	Rd ← $FF-Rd	Z,C,N,V,S	1
1001 010d dddd 0001	NEG	Rd	Two's Complement	Rd ← $00-Rd	Z,C,N,V,S,H	1
0110 KKKK dddd KKKK	SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd or K	Z,N,V,S	1
0111 KKKK dddd KKKK	CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd & ($FFh-K)	Z,N,V,S	1
1001 010d dddd 0011	INC	Rd	Increment	Rd ← Rd+1	Z,N,V,S	1
1001 010d dddd 1010	DEC	Rd	Decrement	Rd ← Rd-1	Z,N,V,S	1
0010 00dd dddd dddd	TST	Rd	Test for Zero or Minus	Rd ← Rd & Rd	Z,N,V,S	1
0010 01dd dddd dddd	CLR	Rd	Clear Register	Rd ← Rd xor Rd	Z,N,V,S	1
1110 1111 dddd 1111	SER	Rd	Set Register	Rd ← $FF	None	1

Source: https://github.com/nlitsme/arduino/blob/master/AVRInstructionTable.html

Examples of MCUs with a 32-bit CPU

ARM Cortex-M Series (Cortex-M0 /M0+/ M3 / M4 / M7 / M23 / M33)

- **Thumb 1 Instructions:**
  - fixed 16-bit instruction size (first introduced with ARM7TDM)
- **Thumb 2 Instructions:**
  - Mixed instruction set: 16-bit (like Thumb-1) and 32-bit (first introduced with Cortex-M3)
  - It provides a trade-off between high code density and high performance.
- **Data Size**
  - ARM Cortex-M processors are 32-bit processors with **32-bit internal registers**, **32-bit data paths**, and **32-bit bus interfaces**.
  - All Cortex-M family processors (M0, M0+, M1, M3, M4, M7, M23, M33) have a **32-bit address bus**, providing a 4GB address space

Exception for the Cortex-M7

- The Cortex-M7 has 64-bit instruction and data buses for the TCM (Tightly Coupled Memory) and internal interfaces.
- External AXI buses can be 64-bit or 128-bit depending on the chip vendor’s implementation.

RISC vs. CISC

RISC and **CISC** are traditionally seen as two opposite **ISA (Instruction Set Architecture) design philosophies**:

- RISC = *Reduced Instruction Set Computing* (e.g. ARM, RISC-V, MIPS)

CISC = *Complex Instruction Set Computing* (e.g. x86 / x86-64)

1. Instruction Complexity

- RISC: simple, small, uniform instructions
- CISC: complex, variable-length instructions (some do multiple operations)

2. Instruction Count

- RISC: fewer instructions
- CISC: larger instruction set

3. Execution Time

- RISC: most instructions complete in 1 cycle
- CISC: some instructions take multiple cycles

4. Instruction Encoding

- RISC: fixed-length (e.g., 32-bit); this makes instruction decoding simpler and faster.
- CISC: variable-length (e.g., 1–15 bytes for x86)

5. Memory Access

- RISC: load/store architecture
- CISC: instructions may perform memory + compute operation together.

RISC vs. CISC

🔵 What is RISC?

- **RISC (Reduced Instruction Set Computing)** is a type of **ISA design philosophy** that uses a small, simple, and efficient set of instructions.
- The main goal of RISC is to improve performance by simplifying the hardware and making instruction execution fast and predictable.

🔵A RISC-based ISA follows these principles:

- Simple, uniform instructions
- Typically Fixed-length encoding
- Load/store architecture
- Most instructions finish in one cycle
- Designed for instruction pipelining

Load/Store Architecture

- A **load/store architecture** is a CPU design where only two types of instructions can access memory:
  - **Load**: reads data from memory and puts it into a register
  - **Store**: writes data from a register back to memory
- All other instructions (add, subtract, AND, OR, compare, shift, multiply, etc.) operate only on registers, not directly on memory.
- Memory access is separated from computation. This simplifies the CPU pipeline and data paths (pipeline-friendly).

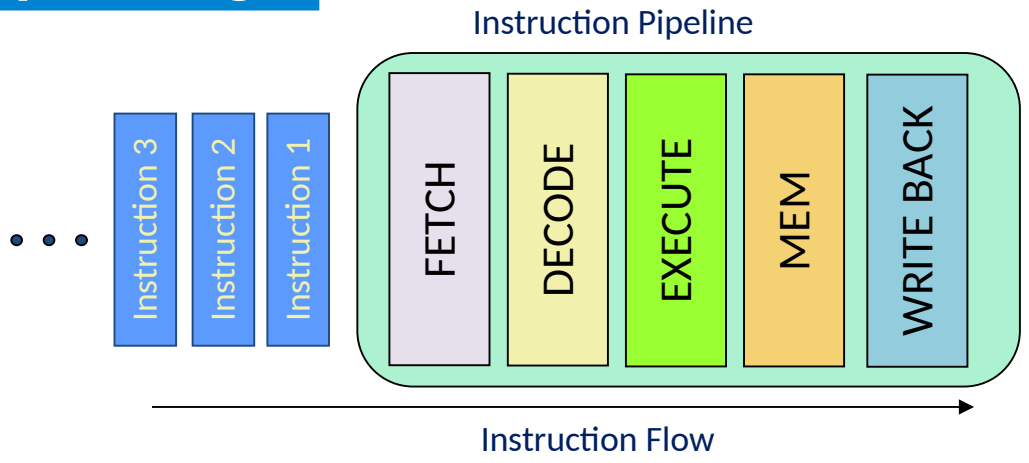
Instruction Pipelining

- **Instruction pipelining** is a **CPU implementation technique** where the processor does not wait for one instruction to complete all of its execution steps before starting the next one.
- Instead, each instruction is broken into several smaller **CPU stages**. When an instruction completes a stage and moves on, the CPU immediately begins processing the next instruction in the now-available stage.
- This allows several instructions to be processed simultaneously — each at a different pipeline stage — significantly increasing overall throughput.

Instruction Pipelining

- Pipelining is especially effective in RISC architectures, where instructions are simple, fixed-length, and easy to divide into predictable stages.
- Well-known RISC families that use pipelining include:
  - **ARM Cortex-M** (3-stage pipeline in M3/M4: Fetch, Decode, and Execute)
  - **ARM Cortex-A** (multi-issue, deeply pipelined superscalar designs)
  - **RISC-V cores** such as SiFive E-series (2–8 stages)
  - **MIPS processors** (classic 5-stage pipeline, widely taught in computer architecture)
    - **IF = Instruction Fetch** (instruction-cache/program memory access)
    - **ID = Instruction Decode** (instruction decode & register file read)
    - **EX = Execute** (perform ALU operation)
    - **MEM = Memory access** (data memory access for load/store operation)
    - **WB = Register write back** (write the result to the register file)

Instruction Pipelining



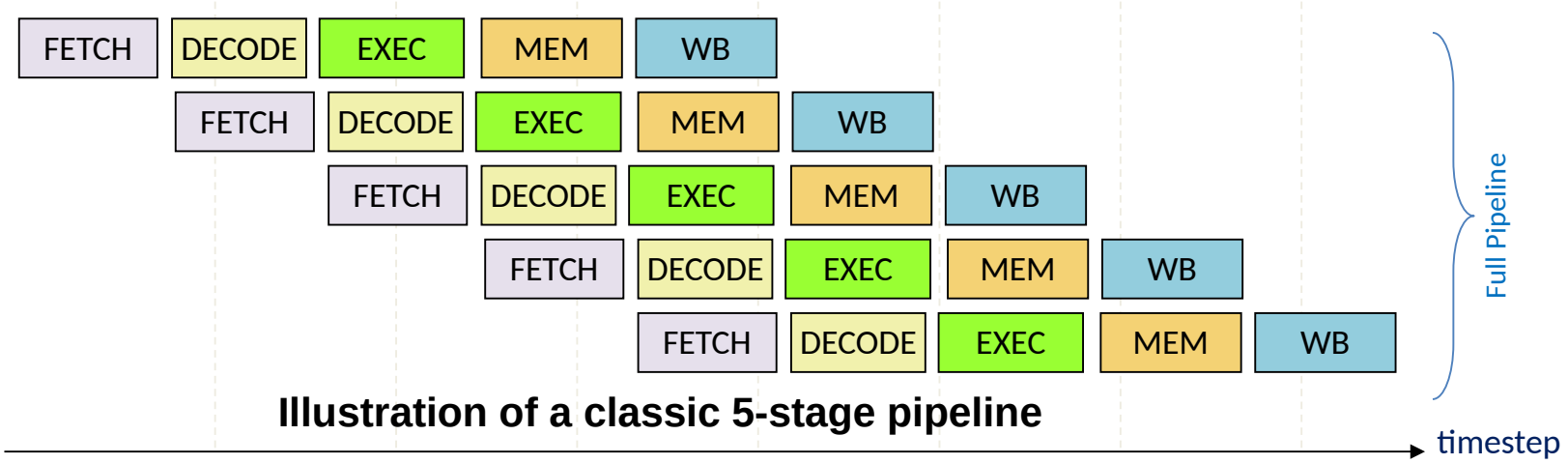


Illustration of a classic 5-stage pipeline

Source: https://en.wikipedia.org/wiki/Classic_RISC_pipeline

Instruction Pipelining

Superscalar pipeline

- A superscalar pipeline is a CPU design where multiple instructions are processed in parallel during each clock cycle using multiple execution pipelines.
- There are multiple ALUs, FPUs, load/store units. The CPU fetches, decodes, and attempts to run multiple instructions at the same time.
- This allows the CPU to issue and execute more than one instruction per cycle.
- This technique increases **instruction-level parallelism (ILP)** inside a single thread.
- "***Multi-issued***" describes how many instructions per cycle the CPU can dispatch or issue to execution units.

In-order vs Out-of-order Execution

- A CPU with in-order execution processes instructions strictly in the order they appear in the program.
- An out-of-order CPU dynamically reorders instruction execution to improve performance, as long as program correctness is preserved.

Separate Program and Data Memory Space

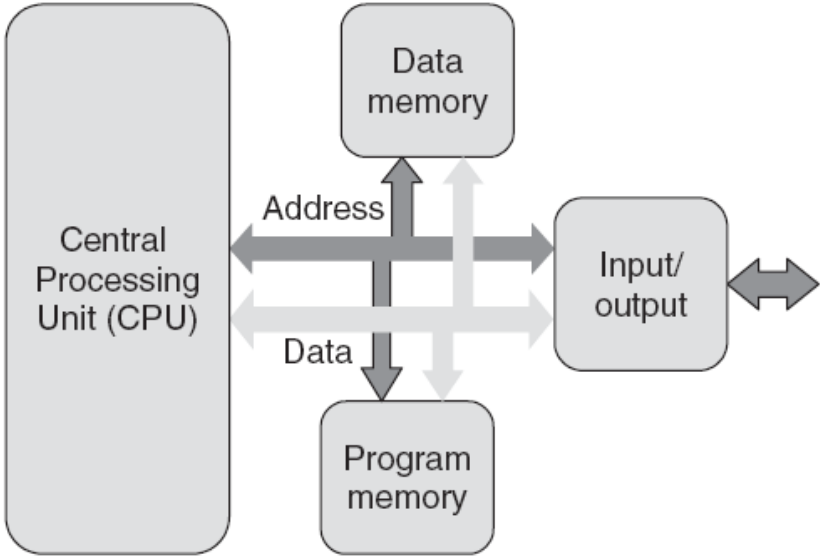
🔵 Von Neumann Architecture

- Uses one shared memory and one bus for both instructions and data.
- Simple and flexible, common in general computers.
- Main drawback: the Von Neumann bottleneck—the CPU often waits because everything shares the same path.

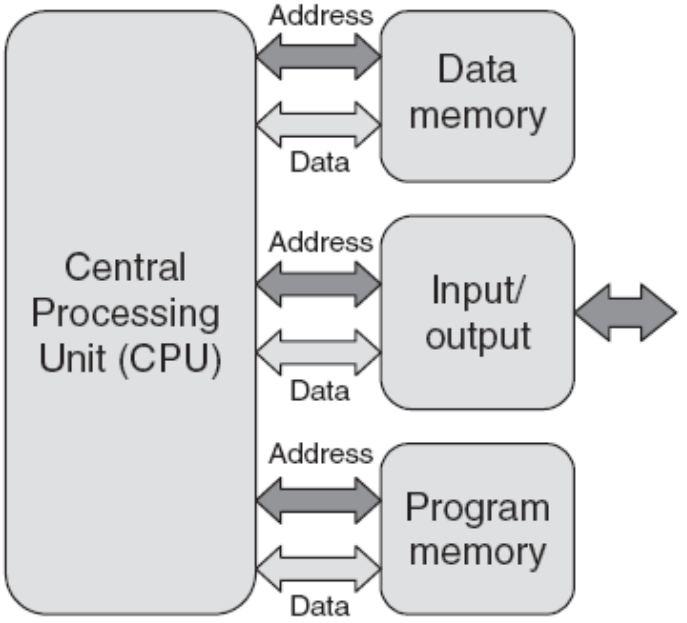
🔵 Harvard Architecture

- Separate memory and buses for instructions and data.
- Removes the bottleneck: faster, more predictable execution.
- Great for real-time systems and MCUs, though more complex to design.
- Modern CPUs use a **Modified Harvard approach**: split instruction / data paths near the core, unified when accessing main memory.

On-Chip Bus Architectures



Single System Bus



Separate System Buses

Miniaturization of Computers

🔵 **Why do we need to make computers small?**

- **Portability**: suitable for mobile devices, wearables, and handheld devices.
- **Integration**: Computers can be embedded in appliances and IoT devices where space is limited.
- **Energy Efficiency**: Smaller circuits consume less power and produce less heat.
- **Speed and Reliability**: Shorter electrical paths, faster operation and fewer signal delays.
- **Cost Reduction**: Smaller chips use less material and are cheaper to produce.
- Innovation: Miniaturization enables new technologies.

Level of Integration

🔵 Chips or ICs (Integrated Circuits)

- The most basic building block level.
- ICs mainly contain transistors, and often include resistors, and capacitors, all integrated into a silicon die, with different types of IC packages.
- Used on PCBs (Printed Circuit Boards) along with other ICs and discrete parts.
- Examples: Logic gate ICs, micrcontroller chips, etc.

🔵 SoC (System-on-a-Chip)

- A complete computing system on one silicon die.
- Different types of components are integrated, such as CPU, peripherals, memories, etc.

🔵 SiPs (System-in-a-Package)

- Multiple ICs (or chiplets) packaged together in a single module or substrate.
- Each chip performs a different role (CPU, RAM, Flash, etc.).

🔵 PCB (Printed Circuit Board) assembly

- The assembly level, where chips, SoCs, memory, and connectors are mounted.
- The PCB connects all components electrically and mechanically.

Examples of Leading MCU Manufacturers

- Microchip Technology (USA)
- STMicroelectronics or STM (Switzerland / France)
- NXP Semiconductors or NXP (Netherlands)
- Renesas Electronics (Japan)
- Texas Instruments or TI (USA)
- Infineon Technologies (Germany)
- Nuvoton Technology (Taiwan)
- Silicon Labs (USA)
- Espressif Systems (China)
- Nordic Semiconductor (Norway)
- GigaDevice Semiconductor (China)
- Raspberry Pi Trading (UK)

Examples of SoC Manufacturers
(for Embedded Applications)

- Intel (USA)
- Broadcom (USA)
- Qualcomm (USA)
- Microchip (USA)
- Texas Instruments (USA)
- Renesas Electronics (Japan)
- NXP Semiconductors or NXP (Netherlands)
- STMicroelectronics or STM (France)
- Samsung (South Korea)
- MediaTek (Taiwan)
- RealTek (Taiwan)
- RockChips (China)

Other Chip Makers acquired by other companies

- Freescale Semiconductor (acquired by NXP in 2015)
- Atmel Corp. (acquired by Microchip in 2016)
- Cypress Semiconductor (acquired by Infineon, 2020)
- Maxim Integrated (acquired by Analog Devices Inc. in 2021)

Why MCUs Are Trending Toward SoC Designs Today?

MCUs are increasingly designed as **Systems on Chip (SoC)**.
This trend is driven by the integration of multiple functions:

- Features such as timers, ADC/DAC, communication interfaces (UART, SPI, I²C, CAN, USB, Wi-Fi, Bluetooth), and security modules are now often built directly into the chip.
- Combining all of these functions on a single chip reduces the number of separate components required, simplifying design and lowering costs.

Why 8-bit/16-bit MCUs Are Being Replaced by 32-bit MCUs?

🔵 **More performance for similar cost**

- For many common applications, the unit cost difference between a low-end 32-bit ARM MCU and an 8-bit MCU has become negligible, depending on the specific features and order volume.

🔵 **Larger address space**

- 32-bit MCUs can directly address 4 GB memory, compared to 64 KB–1 MB for 8/16-bit MCUs.
- This allows for bigger programs, complex stacks, RTOSes, and libraries like FreeRTOS, mbed OS.

🔵 **Complex on-chip peripherals**

- 32-bit MCUs integrate advanced functions such as DMA, USB, CAN, Ethernet, ADC/DAC, timers, security modules, wireless (Wi-Fi/BLE).

Note: 8-bit MCUs are still widely used in extremely cost-sensitive, high-volume products (like toys or simple appliances) where their simplicity, predictable low latency, and minimal external circuitry needs are required.

Single- vs. Multi-Core MCUs

🔵 **Single-Core CPU**
- Most MCUs have a single-core CPU architecture.

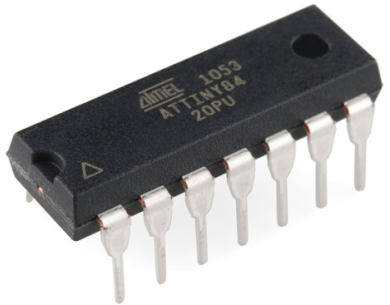
🔵 **Multiple CPU cores of the same architecture**
- known as **Symmetric Multi-Processing (SMP)**
- All cores share the same ISA and memory hierarchy.
- Tasks can be scheduled on any core interchangeably.

🔵 **Multiple CPU cores of different ISA architecture**
- known as **Asymmetric Multi-Processing (AMP)**
- Cores may have different ISAs, performance, or roles
- Software must explicitly target the specific core/ISA

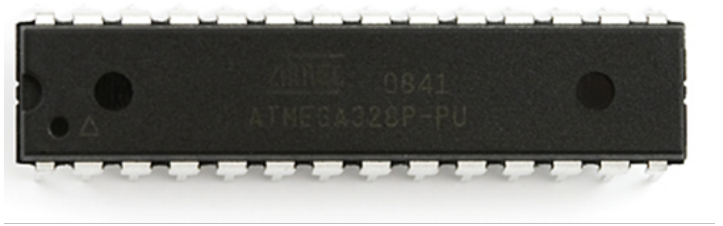
- *Multi-core MCUs fundamentally enable task-level parallelism.*
- *Each CPU core can execute a different task or thread simultaneously, which can significantly improve overall system performance and efficiency compared to a single-core architecture.*

Examples of MCUs / MPUs / SoCs with multi-core different ISAs
- **Raspberry Pi RP2350** (Arm Cortex-M33 and Hazard3 RISC-V / RV32IMC ISA)
- **STM32H747XI**: Arm Cortex-M7 (32-bit) @480MHz + Cortex-M4 (32-bit) @240MHz
- **NXP i.MX 8M Mini MPU**: Quad-core ARM Cortex-A53 @1.8GHz + ARM Cortex-M4 @400MHz

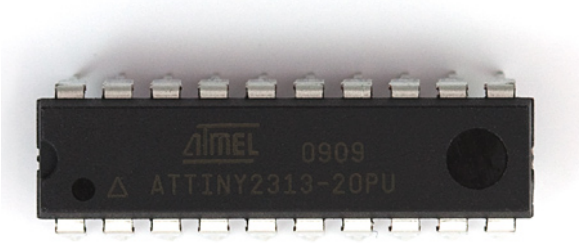
IC Packages for MCUs (Low-Pin Count)



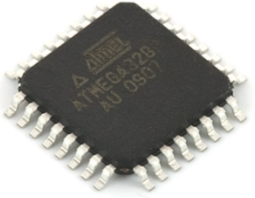
ATtiny84, PDIP-14,
8KB Flash, 512B SRAM, 512B EEPROM,
12 I/O lines, up to 20MHz



ATmega328P, PDIP-28,
32KB Flash, 2KB SRAM, 1KB EEPROM,
23 I/O lines, up to 20MHz



ATtiny2313, PDIP-20,
2KB Flash, 128B SRAM, 128B EEPROM,
15 I/O lines, up to 20MHz



ATmega328P-AU, TQFP -32,
32KB Flash, 2KB SRAM, 1KB EEPROM,
23 I/O lines, up to 20MHz

Raspberry Pi MCUs

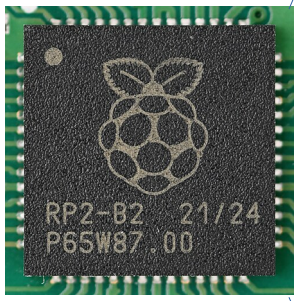
🔵 RP2040

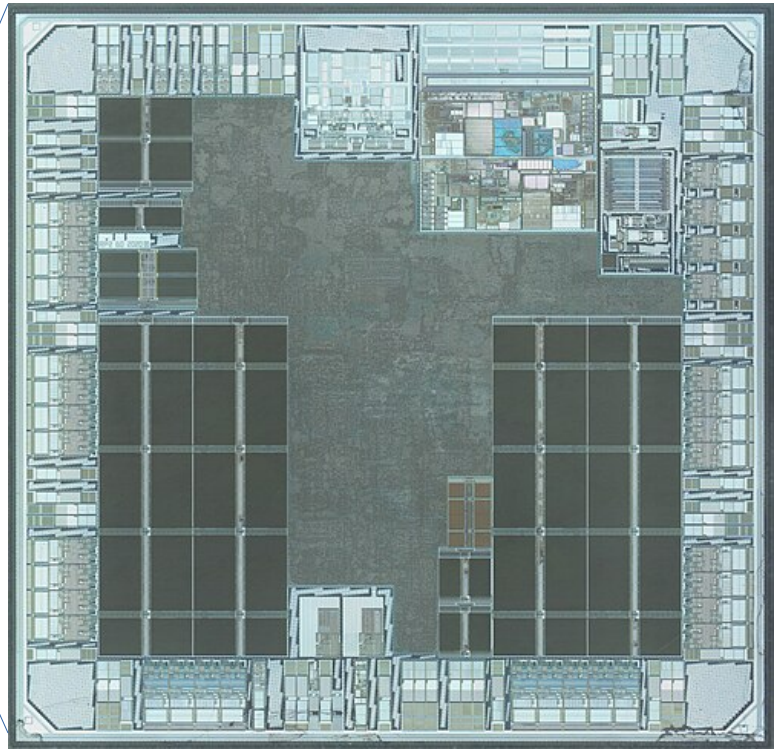
- Designed by Raspberry Pi Foundation
- Release Date: January 2021
- CPU: Dual-core Arm Cortex-M0+ (32-bit)
- Clock Speed: Up to 133 MHz (overclock possible)
- Memory: on-chip RAM (264KB), no built-in flash; uses external QSPI flash
- Instruction Set: Arm Cortex-M0+ (32-bit)
- Single ISA: both cores always run the same architecture

🔵 RP2350 MCU (successor of RP2040): Dual-ISA, Dual-Core Architecture

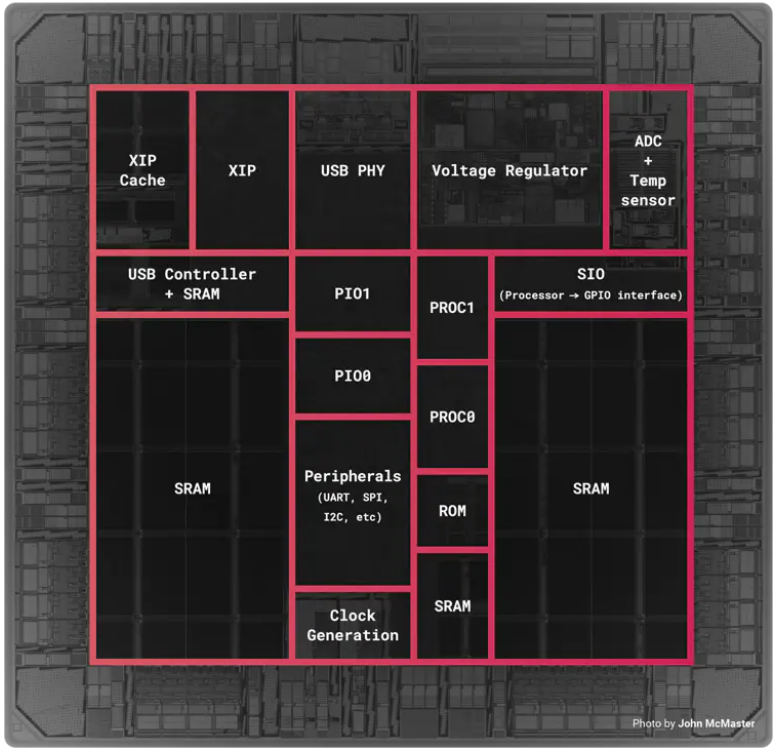
- Release Date: August 2024
- CPU: Dual-ISA, Dual-core configuration:
  - 2× Arm Cortex-M33 cores
  - 2× RISC-V cores (based on open-source Hazard3 implementation)
- Clock Speed: Up to 150Mz (overclock possible)
- Memory: on-chip RAM (520KB), no built-in flash (RP2350A)
- Boot-Time Selection: Can run either ARM cores or RISC-V cores (not both at once)
- Dual-ISA flexibility; software is ISA-specific

MCU / SoC: Raspberry Pi RP2040 Chip





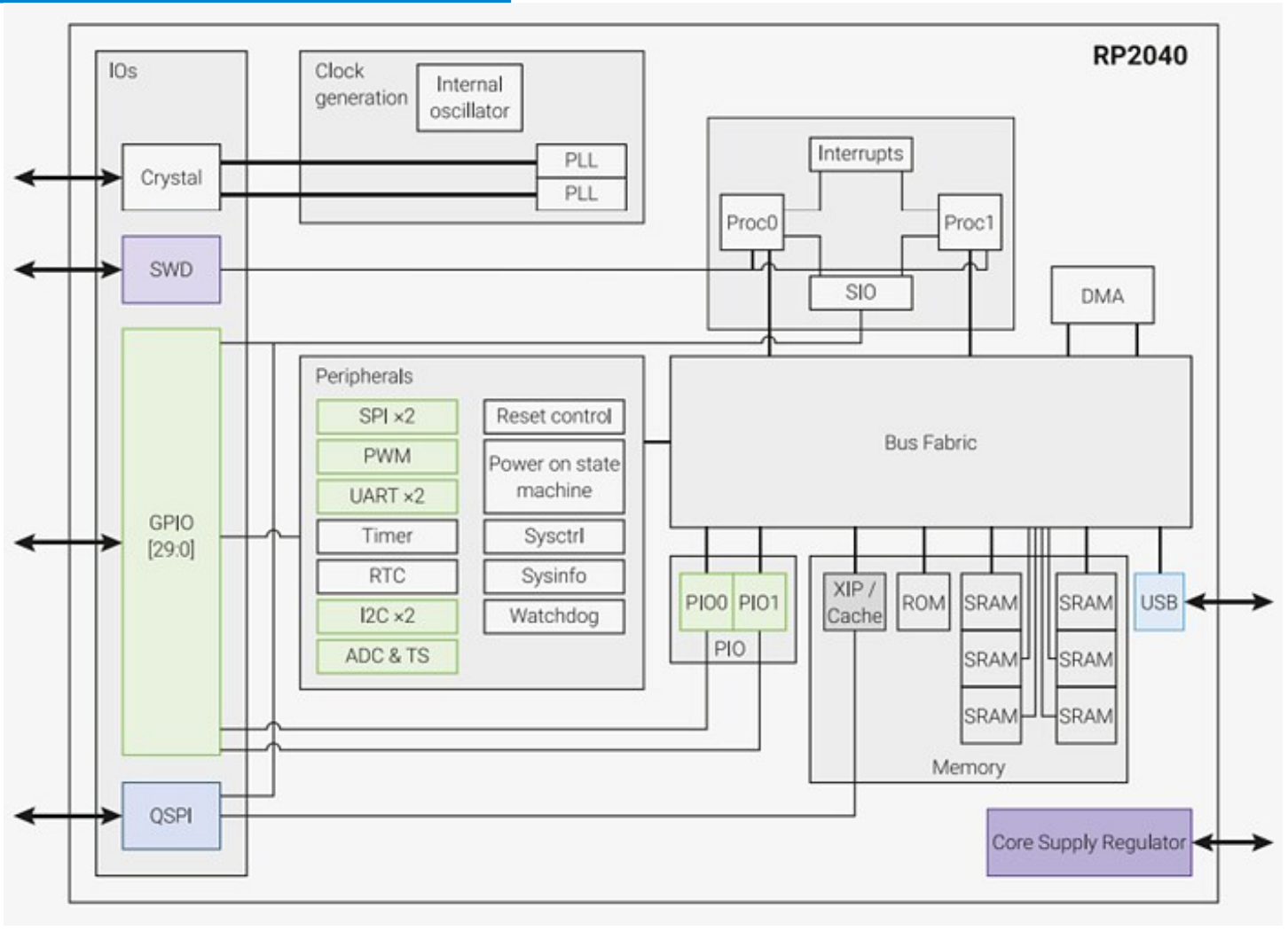
Die Photo



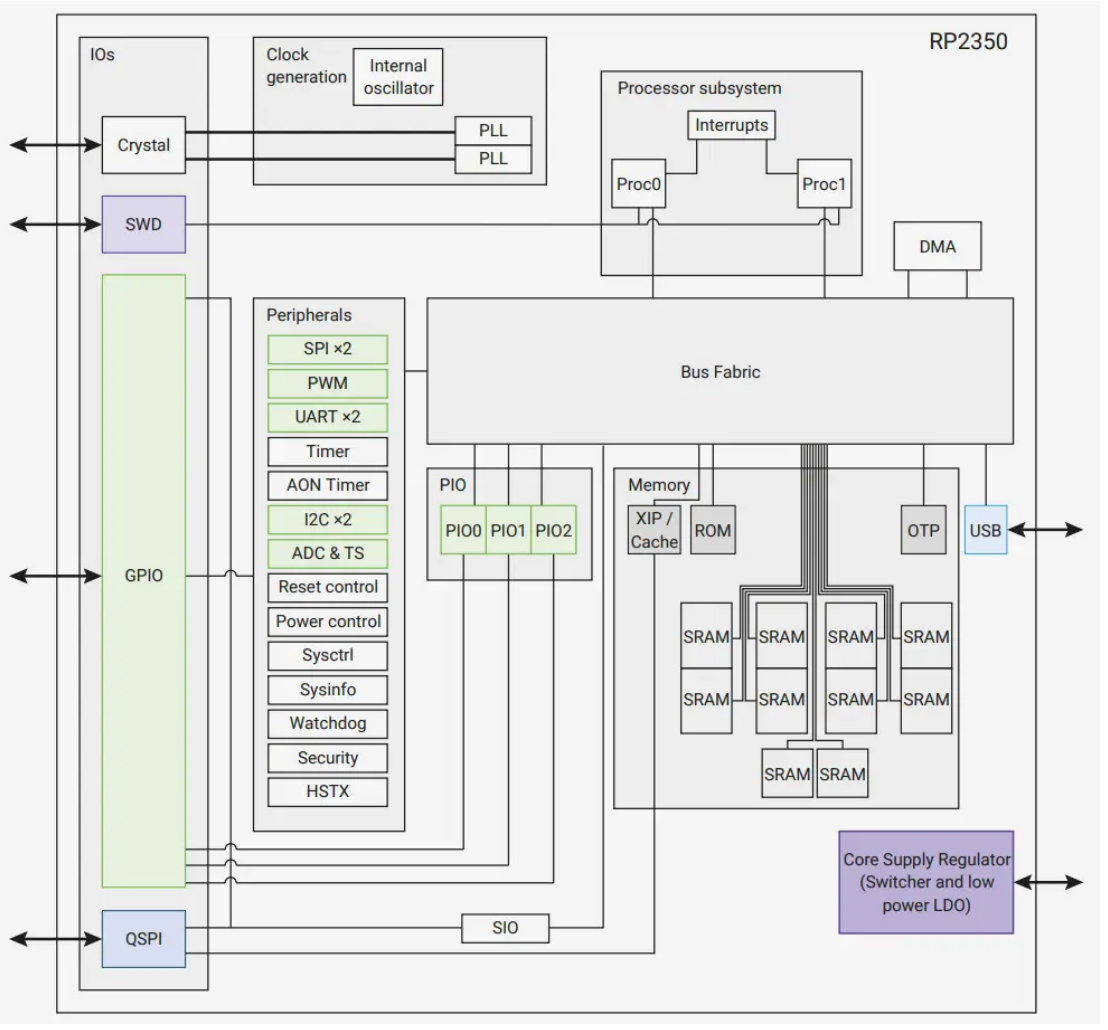
Floorplan Illustration

Source: https://en.wikipedia.org/wiki/RP2040

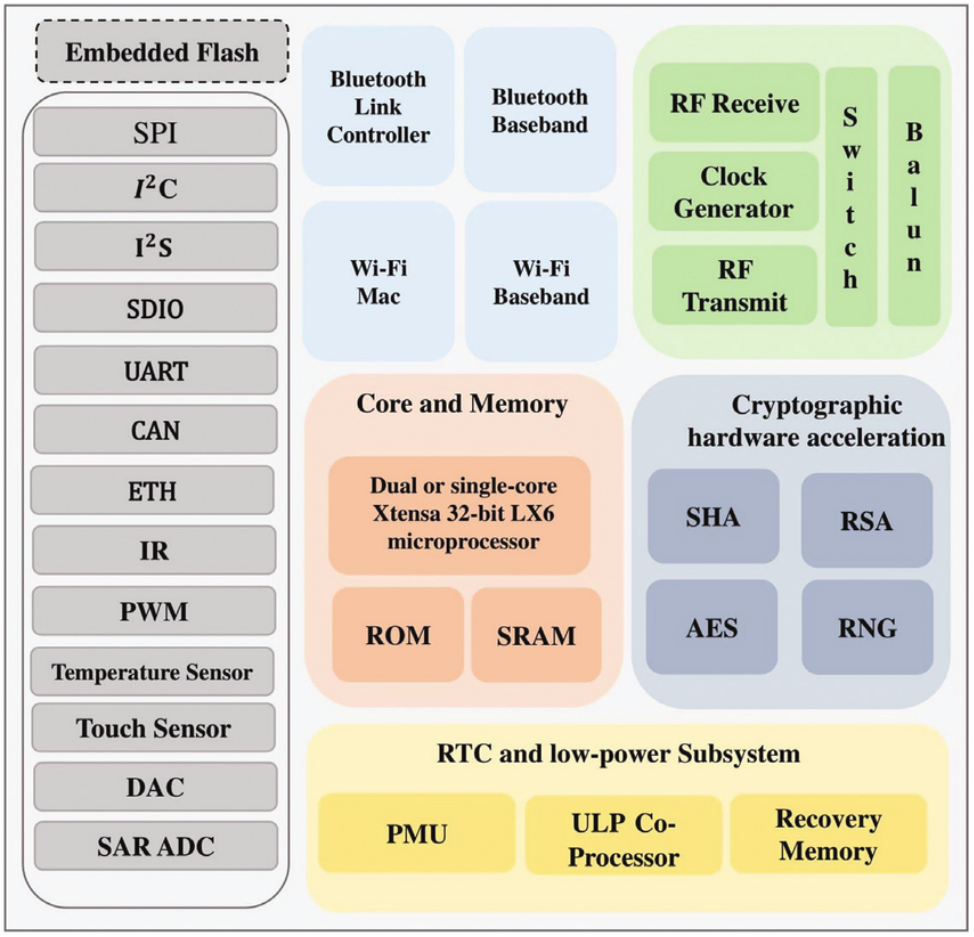
Block Diagram of RP2040



Block Diagram of RP2350



Block Diagrams of ESP32 vs. STM32F411CE



ESP32

STM32F411CE

Key Criteria for MCU Selection

Choosing the right Microcontroller (MCU) balances technical needs with development and long-term production feasibility.

1. Performance and Resource Requirements (The "What It Can Do")

How to meet the task's computing needs efficiently and cost-effectively?

a) Core Performance

- **Speed / Processing Power**: Clock speed (MHz), core architecture (8-bit, 16-bit, 32-bit ARM), instruction set efficiency, and presence of hardware accelerators (e.g., DSP).
- **Memory**: Sufficient RAM (volatile data) and ROM/Flash (program code, non-volatile data). Evaluate need for external memory.
- **Power Consumption:** Crucial for battery devices. Check current draw in Active, Sleep, and Low-Power Modes. Look for Power Management Units (PMUs).

b) Integrated Peripherals

- **I/O Pins**: Required number and type (GPIO).
- **Timers/Counters**: For timing, event control, and PWM generation.
- **Communication Interfaces**: Necessary protocols (e.g., UART, SPI, I^2C, CAN, USB, Ethernet).
- **Analog Features**: ADC/DAC channels, resolution (bits), and sampling rate.

Key Criteria for MCU Selection

2. Development Ecosystem & Support (The "How Easy It Is to Use")

A strong ecosystem speeds up development and reduces long-term maintenance costs.

a) Software Tools

- **IDE & Compilers**: Availability of reliable, code-efficient C/C++ compilers and a comprehensive Integrated Development Environment (IDE).
- **Debugging**: Support for In-Circuit Debugging (ICD) via standards like JTAG or SWD. Availability of reliable debuggers/emulators.

b) Support & Longevity

- **Third-Party & Community Support:** A large community means more open-source libraries, RTOS support, sample code, and faster troubleshooting (e.g., STM32, ESP32 families).
- **Documentation**: Quality and completeness of datasheets and application notes.

Key Criteria for MCU Selection

3. Production & Supply Chain (The "How We Get It Made")

How to ensure manufacturability and long-term product viability?

a) Cost and Scaling

- **Cost per Unit**: The final price at the expected production volume. Consider the entire Bill of Materials (BOM).
- **Ease of Upgrade/Migration**: Ability to scale performance (up or down) within the same product family without major redesigns.

b) Logistics and Form Factor

- **Packaging:** Physical format appropriate for the stage: DIP (prototyping) vs. small QFN/QFP/BGA (mass production) .
- **Availability & Sourcing**: Wide availability from multiple, reliable distributors. Check current lead times.
- **Product Longevity:** Manufacturer's guarantee for long-term supply (essential for industrial/commercial products).

Other Types of Processing Elements for Embedded Systems

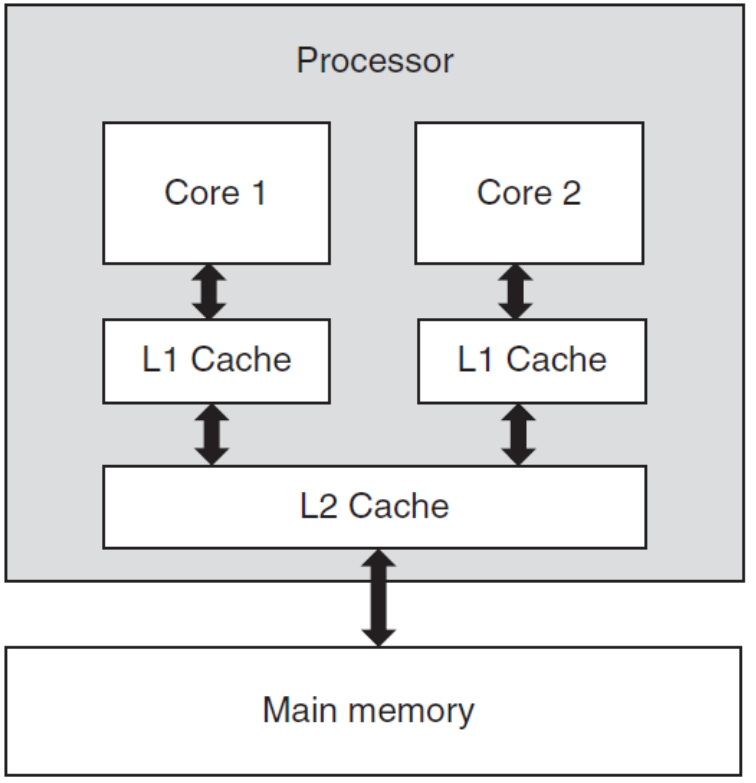
- An **ASIC (Application-Specific IC)** is a highly specialized device and constructed for one specific-purpose application only.
- It integrates several logic functions into a single chip and thus reduces the number of overall circuits needed.
- **ASICs** are very expensive to manufacture, and once it is made, there is no way to modify or improve and its development are the most expensive and of fixed cost.
- An **FPGA** is a **bitstream-programmable IC**, which basically contains a regular grid of logic cells that can be rapidly reconfigured, which facilitates fast prototyping of embedded systems.
- **FPGAs** are commonly used during system design or for small-unit production.
- When reconfigurability is an essential part of the functionality of an embedded system, **FPGAs** can be used in the final product.

Other Types of Processing Elements for Embedded Systems

- Typically, **ASIPs** consist of custom integrated circuitry that is integrated with an instruction set tailored to benefit a specific application.
- This specialization of the CPU core provides a trade-off between the flexibility (via software programmability) of a general-purpose processor and the performance of an ASIC.
- Advantages of **ASIPs** include high performance and increased design flexibility.
- **DSPs** are designed for high-data-rate computations.
  - used to implement algorithms with repetitive and numerically intensive tasks.
  - usually faster than the general-purpose microprocessors in signal processing applications, including audio, video, and communication applications.
- Most **DSPs** use **Harvard architecture** in which data memory and program memory are stored physically in different locations and accessed separately.

Memory Types for Embedded Systems

- On-chip vs. Off-chip (external)
- Non-volatile vs. Volatile
- Main memory vs. Cache memory (multi-levels: L1/L2/L3)
- ROM (Read-Only Memory) vs. RAM (Random Access Memory)
  - **ROM**: EEPROM vs. Flash
  - **RAM**: SRAM (static RAM) vs. DRAM (dynamic RAM)



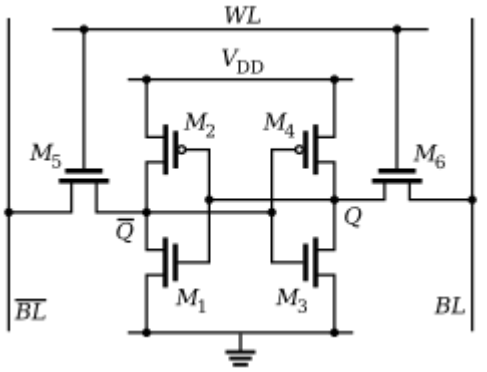
Source/Credit: “Real-Time Embedded Systems”, Jiacun Wang, John Wiley & Sons, Inc., 2017

On-Chip Memory Type: Cache

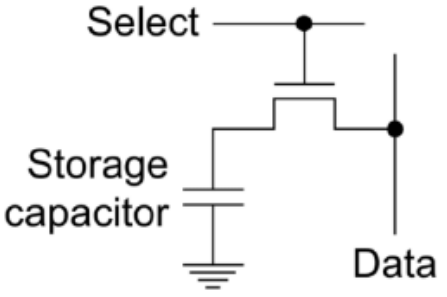
- Cache memory is a type of RAM that a microprocessor can access significantly faster than it can access regular RAM.
- Cache memory is used to store program instructions that are frequently re-referenced by software during operation, whereas less frequently used data is stored in a big and low-speed memory device.
- Some CPUs have multilevel cache memory, referred to as **L1** for level one, **L2** for level two, **L3** for level three, and so on.
  - A CPU directly works with L1 cache.
  - The L1 cache typically ranges in size from 8 to 64KB and uses the high-speed SRAM.
  - The L2 cache is expected to feed data to the L1 cache every few processor cycles, while the L3 cache can feed data to the L2 cache at a further slower rate.

Volatile Memory Types

- **SRAM (Static Random Access Memory)**
  - Fast read / write operations (access time)
- **DRAM (Dynamic RAM with refresh cycle)**
  - With read / write overhead, cheaper than SRAM
  - asynchronous DRAM vs. synchronous DRAM (SDRAM)
- **DDR (Double-Rate) SDRAM**
  - A Double Data Rate (DDR) SDRAM is the advanced version of SDRAM.
  - Data is transferred on both the rising and falling edges of the clock signal.



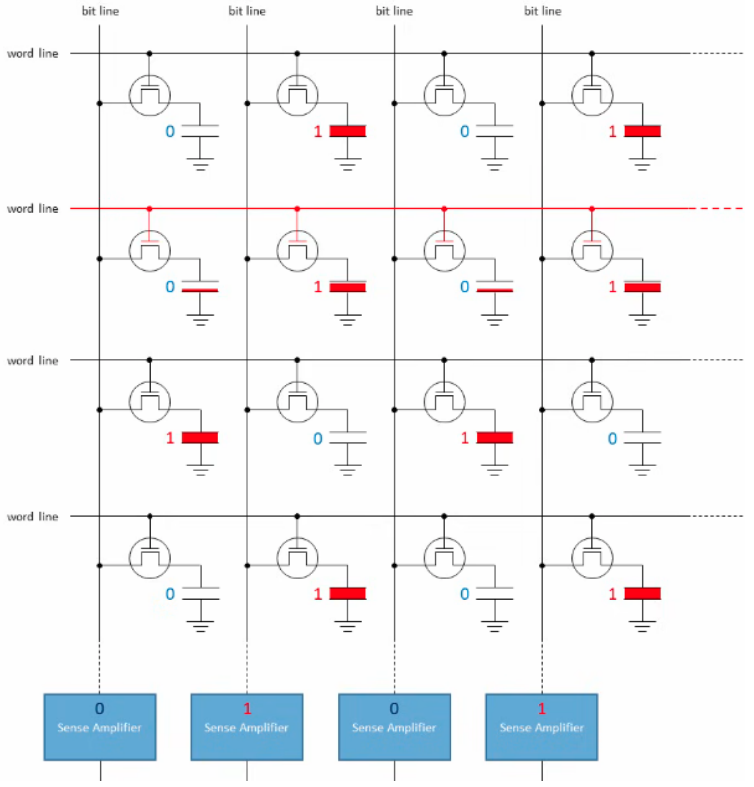
6-transistor SRAM cell



Single DRAM cell

DRAM

- DRAM is a type of volatile semiconductor memory that stores each bit of data
  in a memory cell typically consisting of a tiny capacitor and a MOSFET transistor.
- The data is stored as an electric charge in the capacitor.
- Since the charge on the capacitor gradually leaks away, DRAM is called "dynamic" because it requires periodic refreshing to retain its data.
- It's widely used as the main system memory (RAM) in computers due to its high density and low cost.



Source: https://www.youtube.com/watch?v=-Df09el4yDU

Refresh cycle: DRAM must perform regular refresh cycles to prevent data loss. Because the charge stored in each memory cell gradually leaks over time, the DRAM periodically reads and rewrites (restores) the data to keep the charge intact. This refresh operation is essential for maintaining the stored information.

Volatile Memory Types

The main advantages of DRAM over SRAM are:

- **Higher Density:**
  - DRAM uses only one transistor and one capacitor per bit (1T1C), while SRAM uses six transistors (6T). This simpler cell structure allows DRAM to pack many more cells into the same physical space, leading to greater storage capacity per chip.
  - SRAM uses flip-flop circuits (typically six transistors) to store each bit, which means it retains its data as long as power is supplied, without needing a refresh.
- **Lower Cost:** Due to its simpler cell structure and high density, DRAM is significantly more affordable per bit than SRAM.

SDRAM

- **SDRAM** (Release Year):
  - **SDRAM** (1993), **DDR** (1998), **DDR2** (2003), **DDR3** (2007), **DDR4** (2014), **DDR5** (2020)
  - Lowering voltage: 3.3V, 2.5V, 1.8V to 1.2V (lower power)
  - Increased data size (words) per clock cycle: 1,2,4,8
  - Higher clock rates: and memory data transfer rates

Names	Memory clock	I/O bus clock	Transfer rate	Theoretical bandwidth
DDR-200, PC-1600	100 MHz	100 MHz	200 MT/s	1.6 GB/s
DDR-400, PC-3200	200 MHz	200 MHz	400 MT/s	3.2 GB/s
DDR2-800, PC2-6400	200 MHz	400 MHz	800 MT/s	6.4 GB/s
DDR3-1600, PC3-12800	200 MHz	800 MHz	1600 MT/s	12.8 GB/s
DDR4-2400, PC4-19200	300 MHz	1200 MHz	2400 MT/s	19.2 GB/s
DDR4-3200, PC4-25600	400 MHz	1600 MHz	3200 MT/s	25.6 GB/s
DDR5-4800, PC5-38400	300 MHz	2400 MHz	4800 MT/s	38.4 GB/s
DDR5-6400, PC5-51200	400 MHz	3200 MHz	6400 MT/s	51.2 GB/s

Key Topics in Embedded Systems

1. Processor/Microcontroller Selection

- **Definition**: Selecting appropriate MCU/MPU architecture (8-bit, 16-bit, 32-bit) and specific chip family for the application.
- **Importance**: The processor determines system performance, power consumption, cost, peripheral availability, and development ecosystem. Incorrect selection leads to project failure or costly redesigns.

2. Operating System & RTOS Integration

- **Definition**: Determining OS requirements and selecting between commercial, open-source, or custom operating system solutions.
- **Importance**: OS choice directly impacts real-time performance, multitasking capabilities, driver availability, development time, and licensing costs.
  An OS provides hardware abstraction enabling code portability across platforms.

Key Topics in Embedded Systems

3. Security Design & Implementation

- **Definition**: Implementing protective measures against IP theft, tampering, cloning, unauthorized access, and cyberattacks using hardware and software security features.
- **Importance**: Connected devices face serious vulnerabilities that can compromise user data, enable counterfeiting, or create liability issues. Security requires design-phase integration rather than post-development addition.

4. Wireless Connectivity & IoT Integration

- **Definition**: Integrating wireless communication protocols (Wi-Fi, Bluetooth, cellular, LoRa) and connecting devices to IoT infrastructure.
- **Importance**: Modern embedded products require remote monitoring, control, data collection, and cloud connectivity. Wireless integration enables new business models and features while introducing complexity in protocol selection, power management, and security.

Key Topics in Embedded Systems

5. Power Management & Energy Efficiency

- **Definition**: Minimizing power consumption through hardware selection, software optimization, sleep modes, and efficient power supply design.
- **Importance**: Battery-powered applications demand careful power budgeting. Poor power design reduces battery life, increases operating costs, generates excess heat, and diminishes product competitiveness. This is critical for IoT, wearables, and remote sensors.

6. Functional Safety Standards

- **Definition**: Compliance with industry safety standards (ISO 26262 for automotive, IEC 61508 for industrial, DO-178C for aerospace) ensuring safe operation during failures.
- **Importance**: Safety-critical applications require certified development processes and fail-safe designs to prevent injury, death, or catastrophic failures. Non-compliance results in legal liability, product recalls, and market rejection.

Key Topics in Embedded Systems

7. AI/ML & Embedded Intelligence

- **Definition**: Integrating artificial intelligence and machine learning algorithms for local processing, pattern recognition, predictive analytics, and autonomous decision-making.
- **Importance**: Edge AI enables real-time processing without cloud dependency, reducing latency, protecting privacy, and lowering bandwidth costs. Increasingly required for smart devices, vision systems, and predictive maintenance applications.

8. Software Development Tools & Debugging

- **Definition**: Utilizing compilers, IDEs, debuggers, emulators, version control, and testing tools for firmware and software development.
- **Importance**: Debugging represents the most time-consuming development activity. Quality tools directly impact development speed, code quality, bug detection, and time-to-market. Poor toolchain selection creates persistent development friction.

Key Topics in Embedded Systems

9. Code Reuse & Hardware IP Reuse

- **Definition**: Leveraging existing software code, drivers, middleware, and hardware designs from previous projects or third-party sources.
- **Importance**: Reuse substantially reduces development time, cost, and risk through proven, tested components. This enables faster time-to-market and improved reliability while allowing focus on differentiating product features.

10. System Integration & Testing

- **Definition**: Combining hardware, firmware, software, and third-party components into complete working systems with comprehensive verification.
- **Importance**: Integration issues represent major sources of project delays and defects. Complex systems with multiple subsystems and real-time requirements demand systematic integration approaches. Testing validates functional requirements, performance, reliability, and safety before production.

Market Surveys & Reports

- There are several **publicly available market surveys** that provide valuable insights into **embedded systems development and technology trends**, including:

  1) Embedded Market Study 2023 – AspenCore Media
  URL: https://www.embedded.com/embedded/embedded-survey/

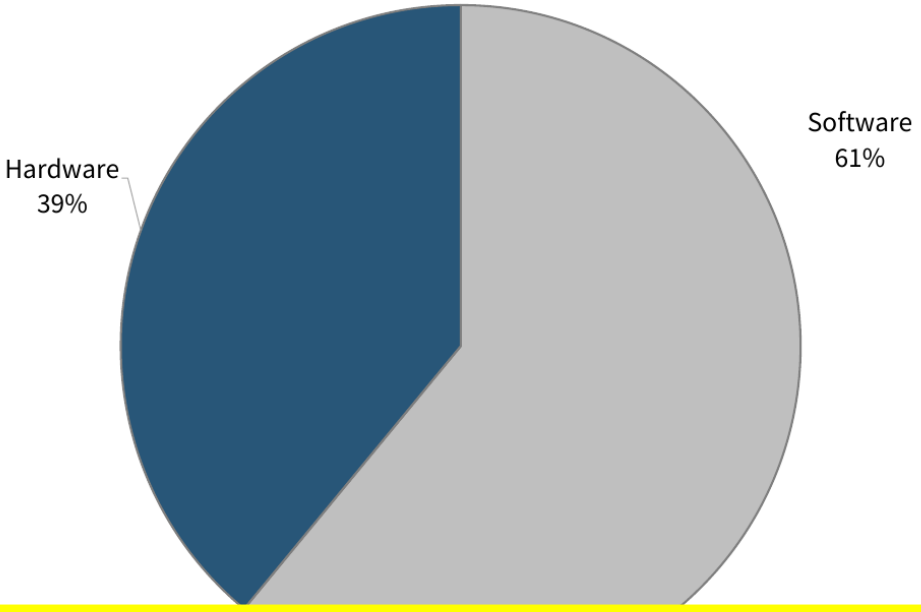
  2) Eclipse Foundation – IoT & Edge Developer Survey
  URL: https://iot.eclipse.org/community/resources/iot-surveys/

- Market surveys help engineers, students, and companies understand the real direction of the **embedded and IoT industry**.
- They reveal how technologies are evolving, what tools professionals actually use, and where future opportunities or challenges lie.
- This helps guide learning, career planning, platform selection, and product strategy.

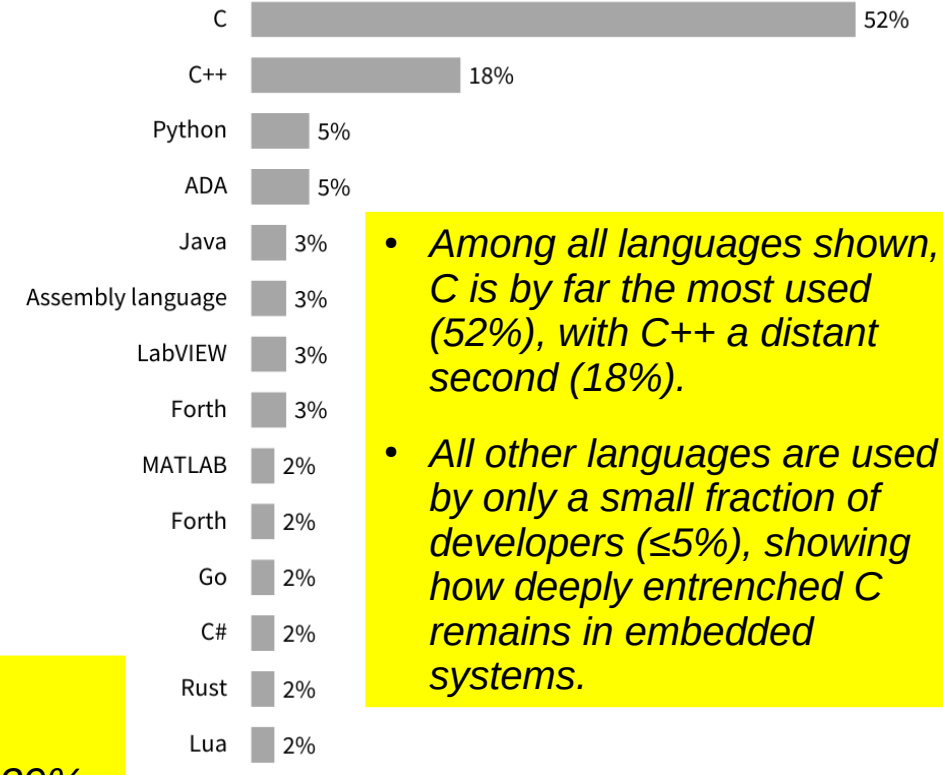
Embedded Market Survey 2023

Software development requires more cycle time

"C" dominates other languages for embedded software programming



Software dominates embedded development effort. The pie chart shows that 61% of development time goes into software, while hardware accounts for only 39%.



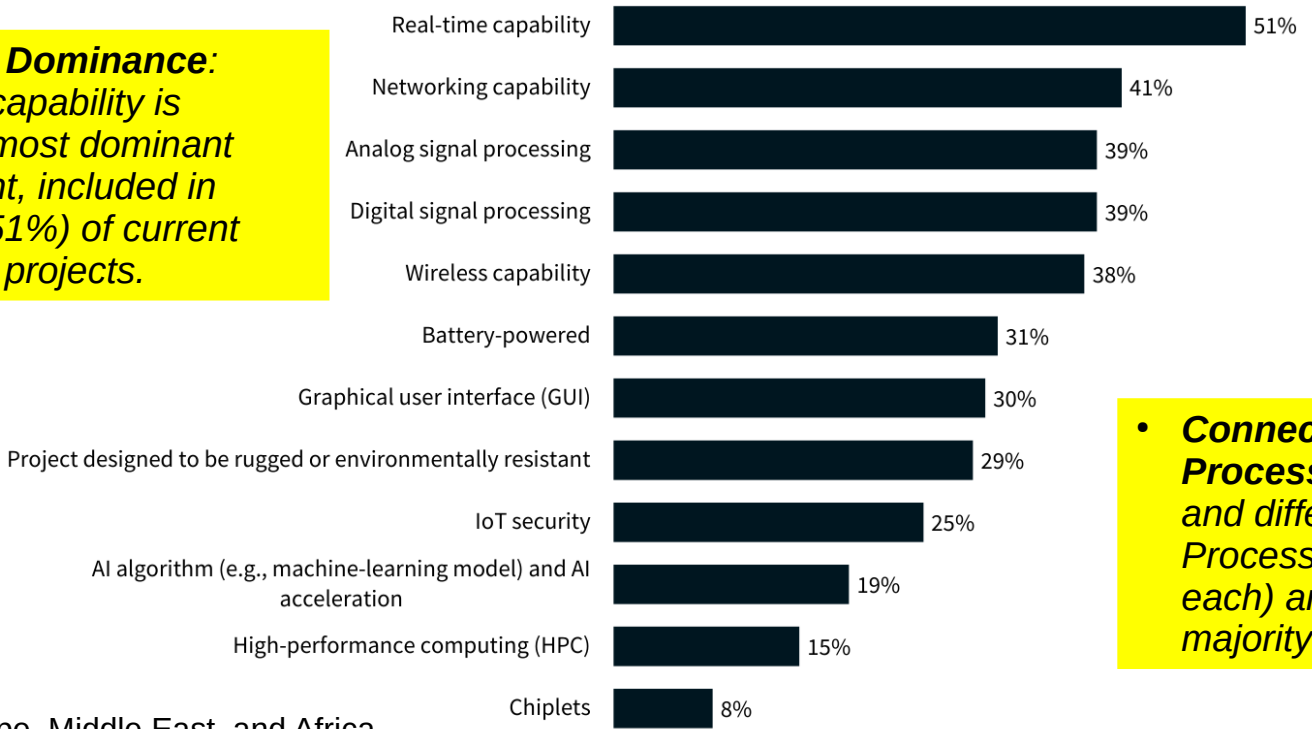
- *Among all languages shown, C is by far the most used (52%), with C++ a distant second (18%).*
- *All other languages are used by only a small fraction of developers (≤5%), showing how deeply entrenched C remains in embedded systems.*

Embedded Market Survey 2023

Current embedded development devotes considerable attention to performance, connectivity, power efficiency and signal processing

EMEA and APAC teams are particularly interested in these capabilities

- ***Real-Time Dominance**: Real-time capability is the single most dominant requirement, included in over half (51%) of current embedded projects.*



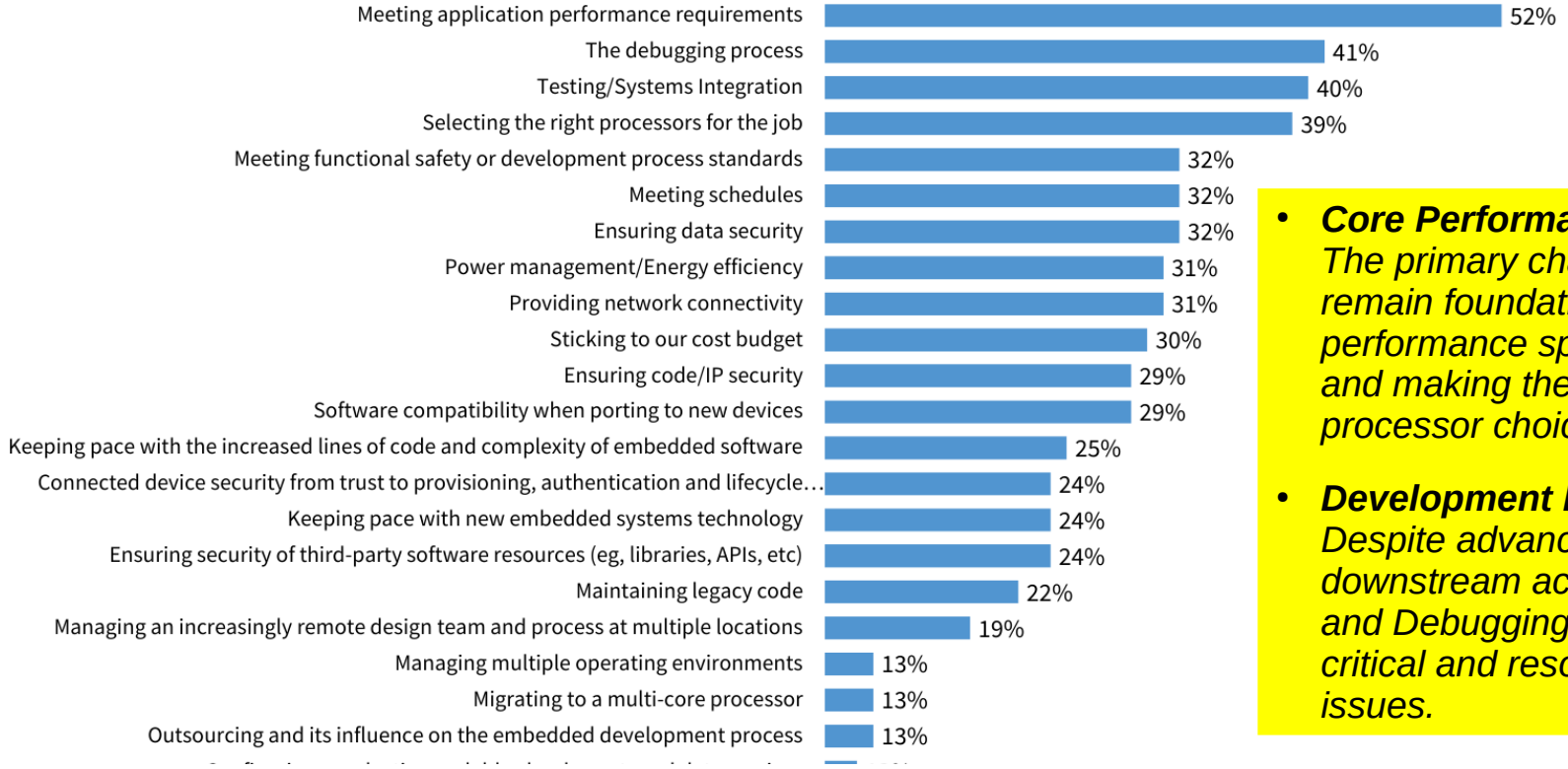
- ***Connectivity & Signal Processing**: Networking (41%) and different forms of Signal Processing (Analog/Digital, 39% each) are critical features in a vast majority of new designs.*

EMEA = Europe, Middle East, and Africa
APAC = Asia–Pacific

Embedded Market Survey 2023

Meeting performance specs, processor choice and test/debugging are critical issues

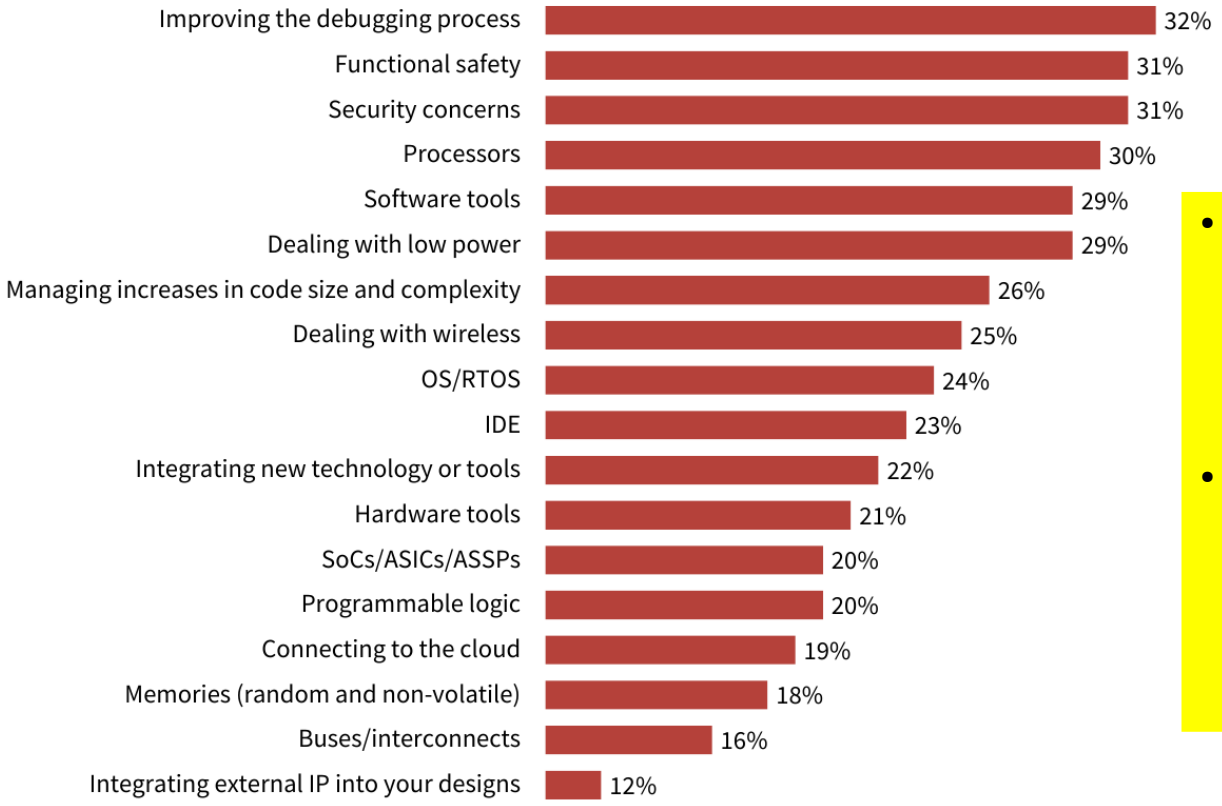
Safety, security and power management are also high on the agenda (especially for EMEA and APAC designers)



- ***Core Performance Hurdles**: The primary challenges remain foundational: Meeting performance specifications and making the optimal processor choice.*
- ***Development Bottleneck**: Despite advances, the downstream activities of Test and Debugging continue to be critical and resource-intensive issues.*

Embedded Market Survey 2023

Better debugging and SW tools, improved safety and security and power join processor selection as most critical design challenges



- ***Top Design Hurdles**: Meeting performance specs and processor selection remain the most critical, enduring challenges.*
- ***The Emerging Focus**: Debugging/SW tools, Safety/Security, and Power Management are now equally important and demanding design priorities.*

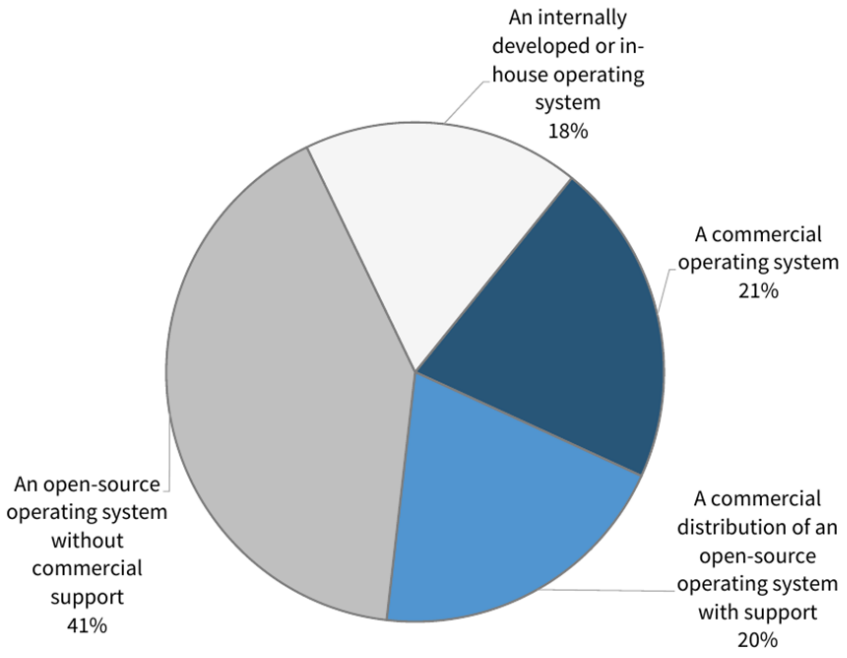
Embedded Market Survey 2023

Most embedded projects utilize an operating system

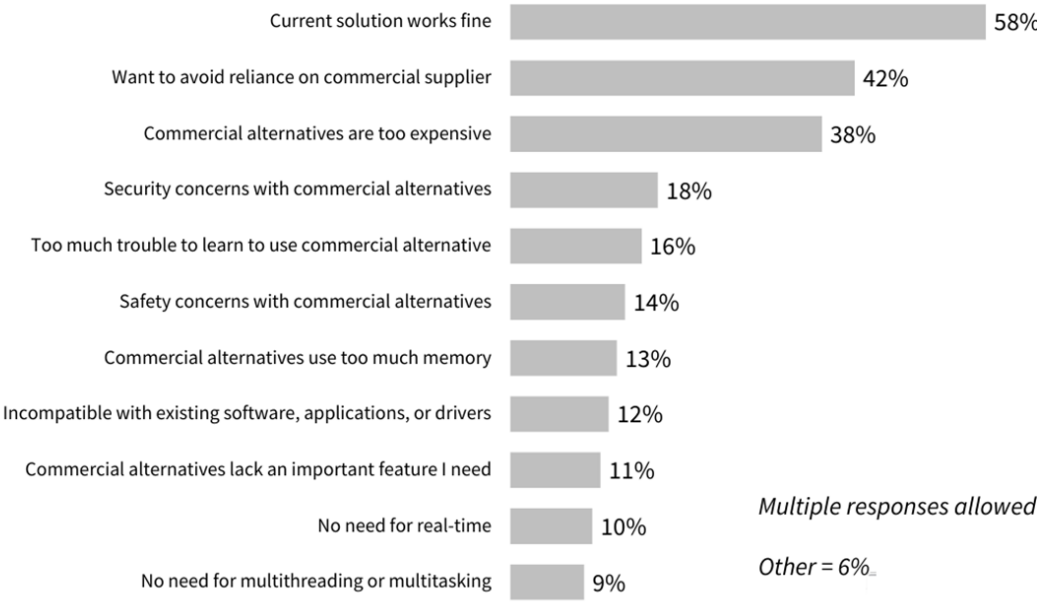
Although open source is popular, four in ten use either commercial OS or open-source OS distributed commercially

74% use an OS in current embedded project

OS Used in Current Embedded Project



Reasons for not using commercial OS

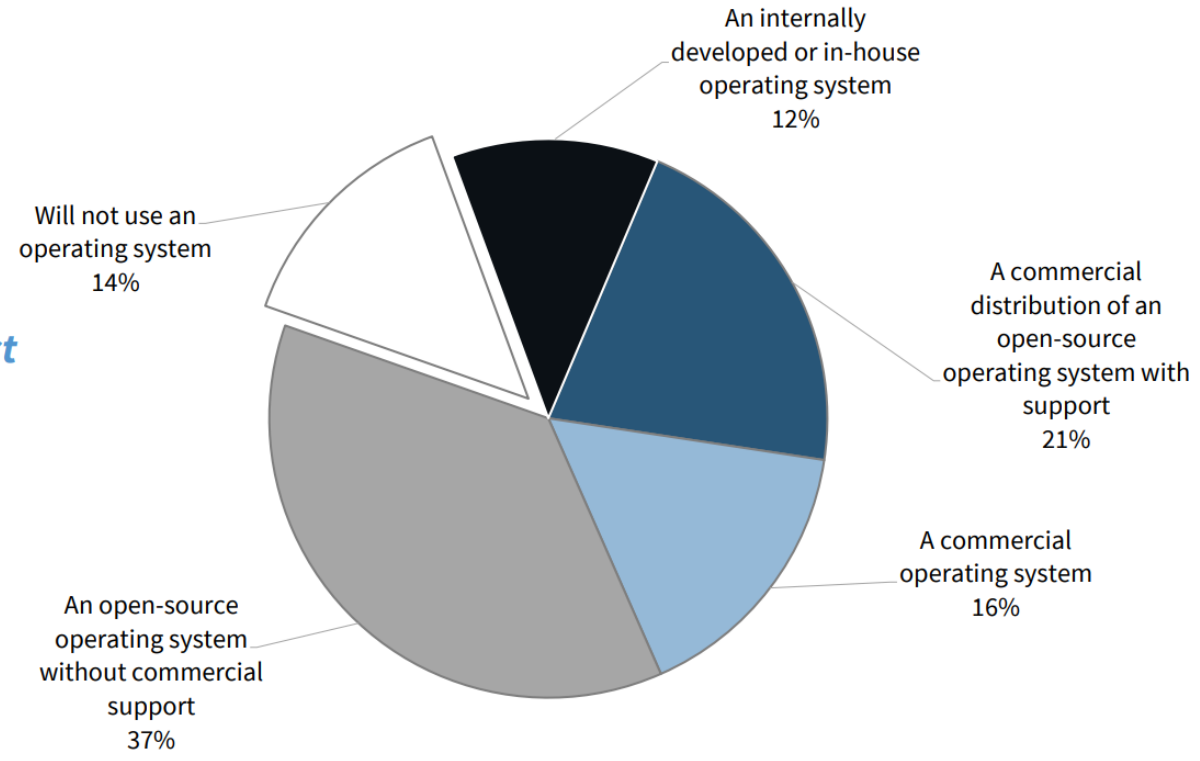


Embedded Market Survey 2023

OS use will increase, but open-source share will grow

Nearly 30% of those now using commercial OS are considering open-source alternatives

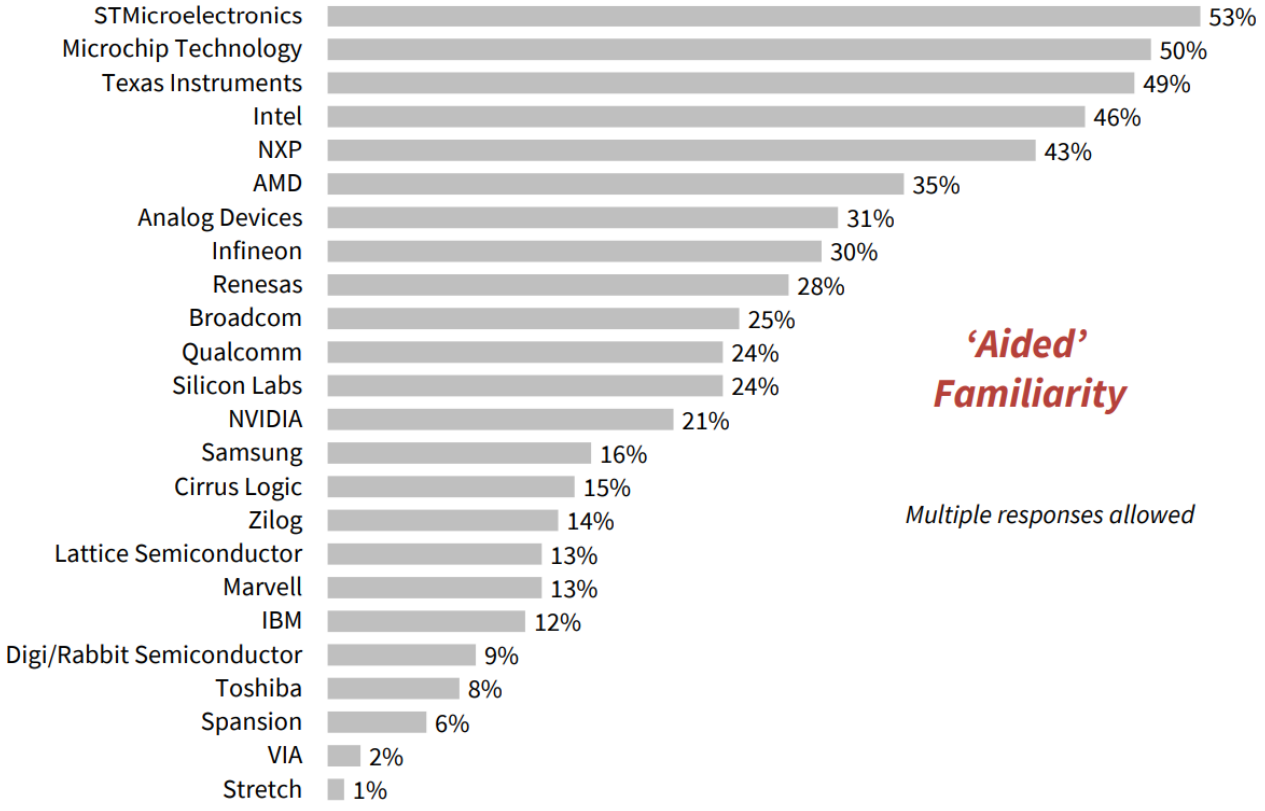
86% will use an OS for next embedded project



Embedded Market Survey 2023

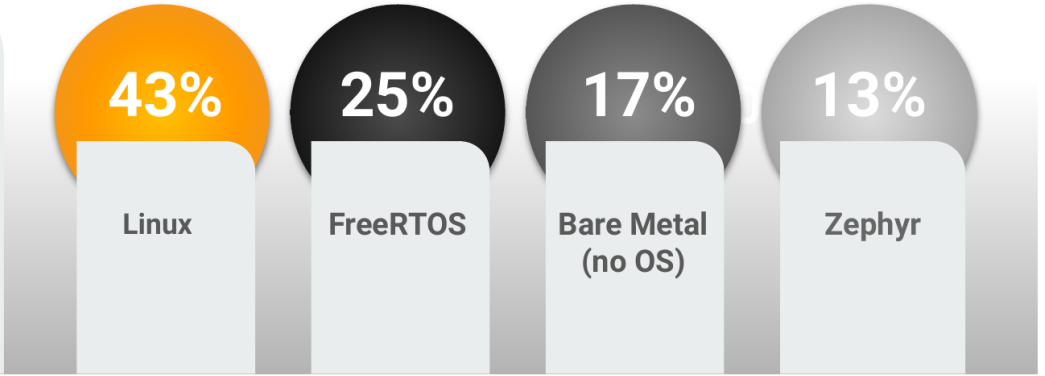
Familiarity with MPU/MCU vendors

STMicro, Microchip, TI, Intel, and NXP are the most well-known processor vendors



Eclipse Foundation – IoT & Edge Developer Survey 2023

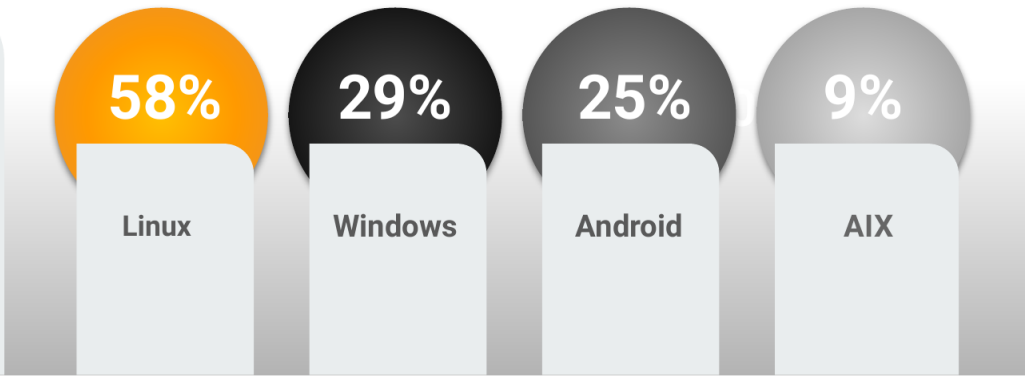
Linux (43%), and **FreeRTOS** (25%) are the top embedded OS choices for constrained devices. A solid 17% of developers prefer no OS at all, while **Zephyr** enjoys a respectable 13%, compared to only 8% in 2022.



OS Preferences for constrained devices

Q: Which embedded operating system(s) do you use for your constrained devices? (select up to three)

Linux (58%, up from 51%), **Windows** (29%, down from 42%) and **Android** (25%) are the top OS choices for IoT gateways and edge nodes, with **AIX** maintaining at 9% (same as 2022). **Azure Sphere**, at 8%, dropped out of the top four.

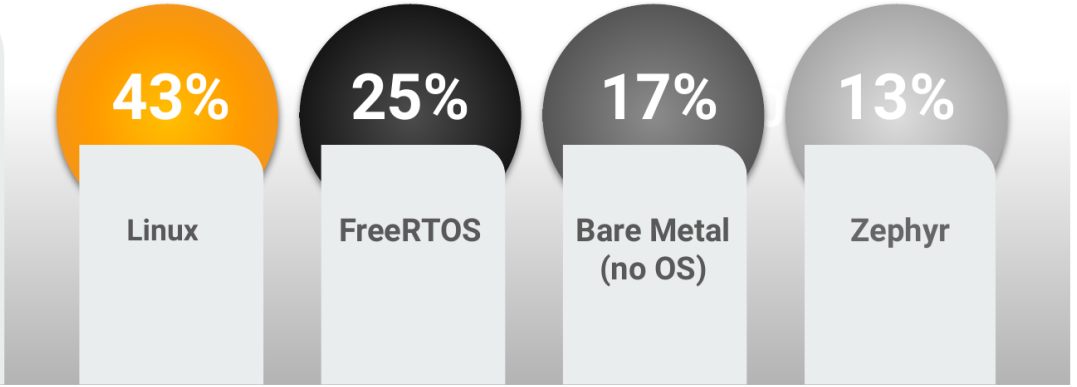


OS Preferences for IoT Gateways & Edge Nodes

Q: Which operating system(s) do you use for your IoT gateways & edge nodes? (select up to three)

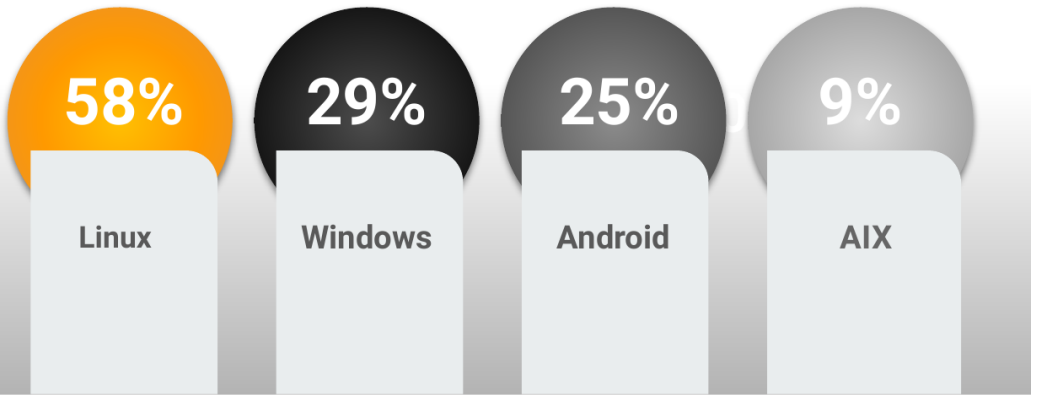
Eclipse Foundation – IoT & Edge Developer Survey 2023

Linux (43%), and **FreeRTOS** (25%) are the top embedded OS choices for constrained devices. A solid 17% of developers prefer no OS at all, while **Zephyr** enjoys a respectable 13%, compared to only 8% in 2022.



Q: Which embedded operating system(s) do you use for your constrained devices? (select up to three)

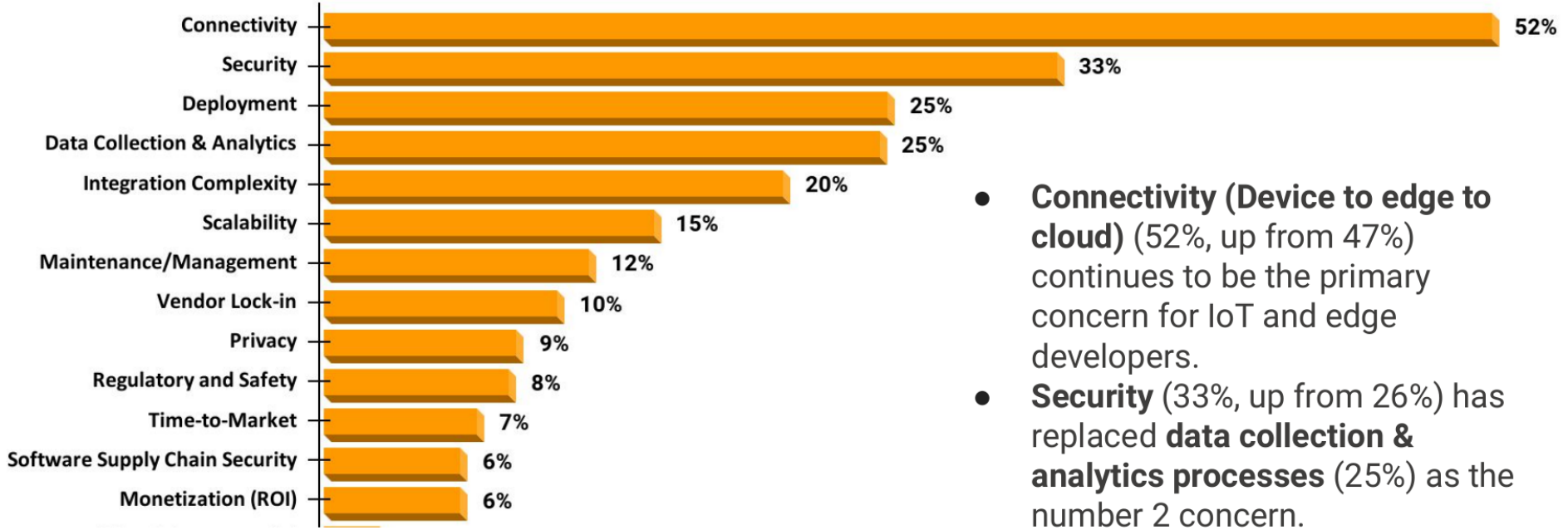
Linux (58%, up from 51%), **Windows** (29%, down from 42%) and **Android** (25%) are the top OS choices for IoT gateways and edge nodes, with **AIX** maintaining at 9% (same as 2022). **Azure Sphere**, at 8%, dropped out of the top four.



Q: Which operating system(s) do you use for your IoT gateways & edge nodes? (select up to three)

Eclipse Foundation – IoT & Edge Developer Survey 2023

Connectivity Continues to be Top Developer Concern

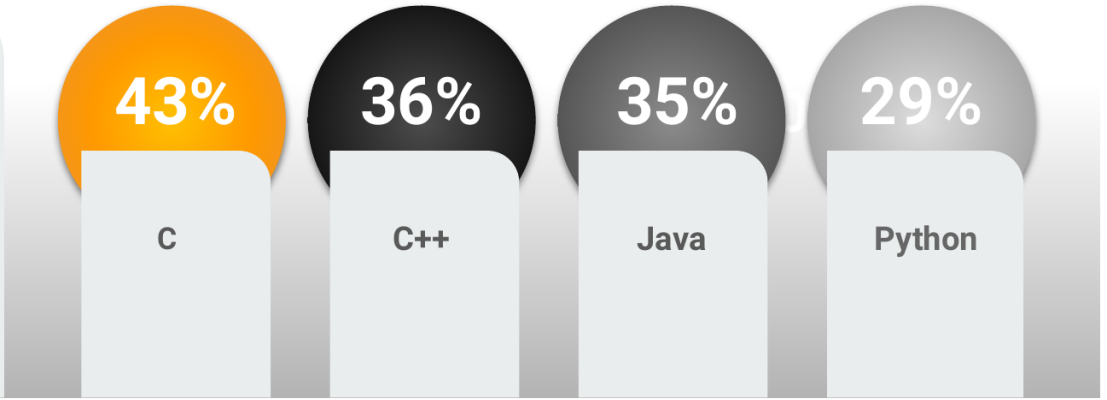


- **Connectivity (Device to edge to cloud)** (52%, up from 47%) continues to be the primary concern for IoT and edge developers.
- **Security** (33%, up from 26%) has replaced **data collection & analytics processes** (25%) as the number 2 concern.

Q: What are your primary IoT development concerns? (Select up to 3)

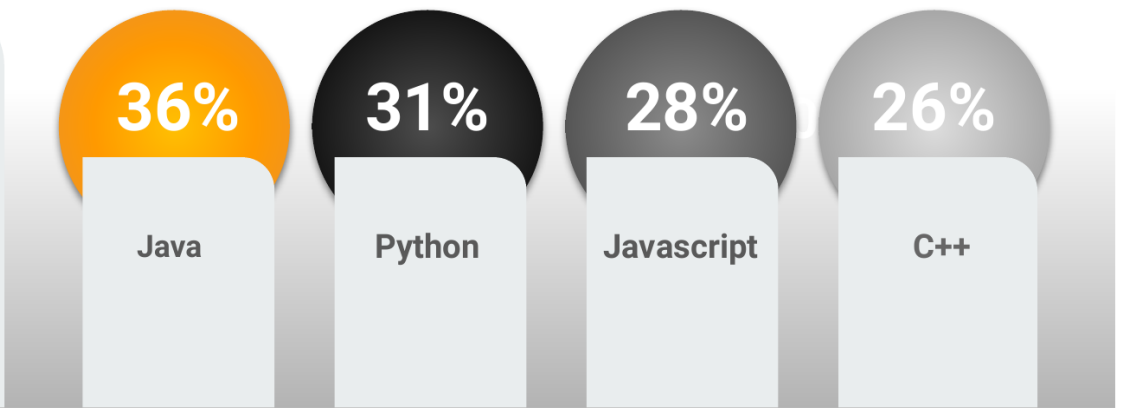
Eclipse Foundation – IoT & Edge Developer Survey 2023

C, C++, Java, and **Python** are the programming languages of choice for development on constrained devices.



Q: Which of the following programming languages do you use to build on constrained devices? (Select up to three)

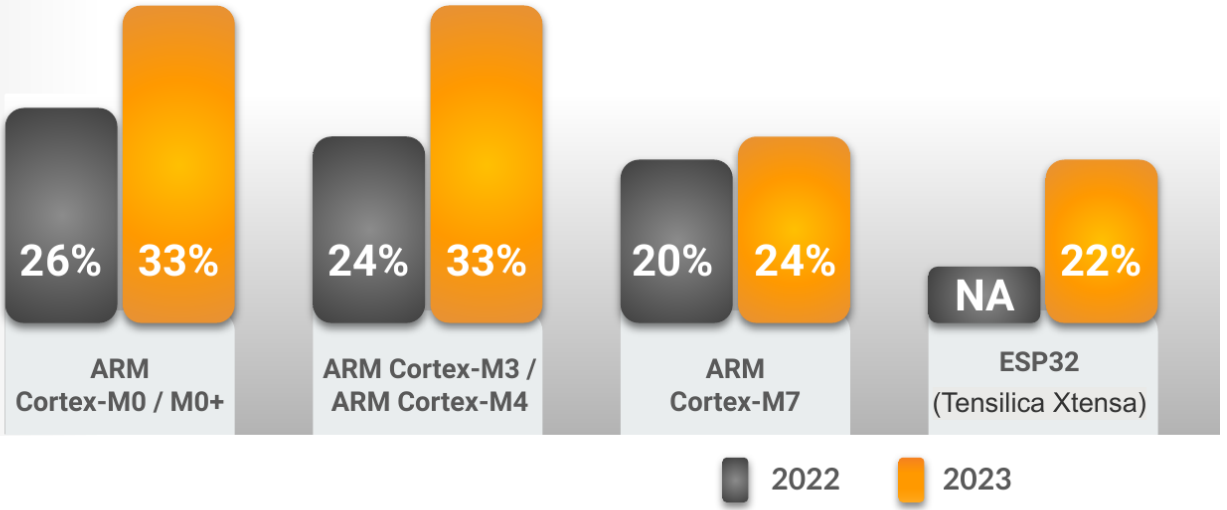
Java is the preferred development language for IoT Gateways and Edge Nodes, followed by **Python, Javascript,** and **C++**.



Q: Which of the following programming languages do you use to build on IoT Gateways & Edge Nodes? (Select up to three)

Eclipse Foundation – IoT & Edge Developer Survey 2023

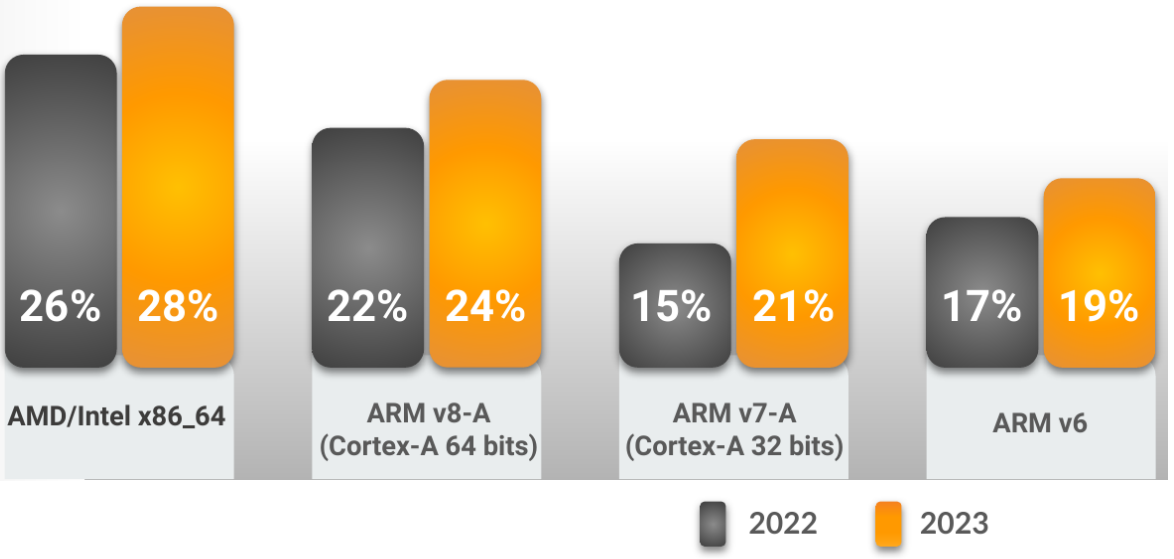
- ARM continues to dominate the constrained device landscape. In particular, **ARM Cortex-M0 / M0+** and **ARM Cortex-M3 / M4** usage shows substantial increases over 2022.
- New to this survey, the Espressif Systems **ESP32 (Tensilica Xtensa)** is used by an impressive 22% of respondents.
- **RISC-V architectures from OpenHW Group** (CORE-V CVE2, CVE4) are gaining momentum with 7%.



Q: What hardware architecture(s) do you use for constrained devices? (Select up to three)

Eclipse Foundation – IoT & Edge Developer Survey 2023

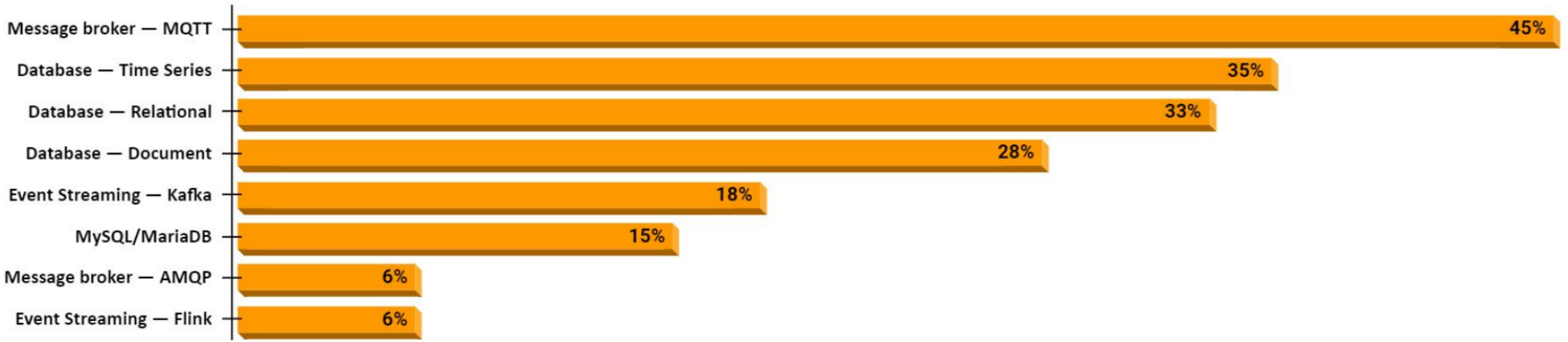
- 64-bit architectures continue to gain ground in gateways and edge nodes.
- **AMD/Intel x86-64** usage shows marginal growth from 26% in 2022 to 28%.
- Overall, ARM architectures are dominant, with the 64-bit **v8-A,** the **v7-A** and **v6** all showing up in the top 4 and other family members (v7-M, v8.2A, V9, v7-R) lagging not too far behind.
- The new OpenHW Group CORE-V (CVA6) makes its debut at 4%.



Q: What hardware architecture(s) do you use for IoT gateways and edge nodes? (Select up to three)

Eclipse Foundation – IoT & Edge Developer Survey 2023

MQTT is preferred for messaging infrastructure.



Eclipse Foundation – IoT & Edge Developer Survey 2023