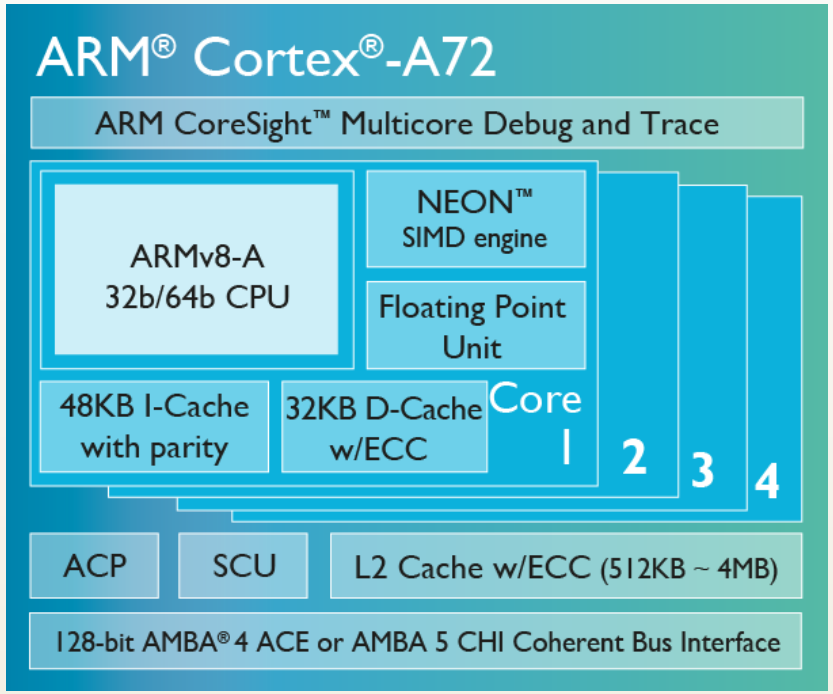
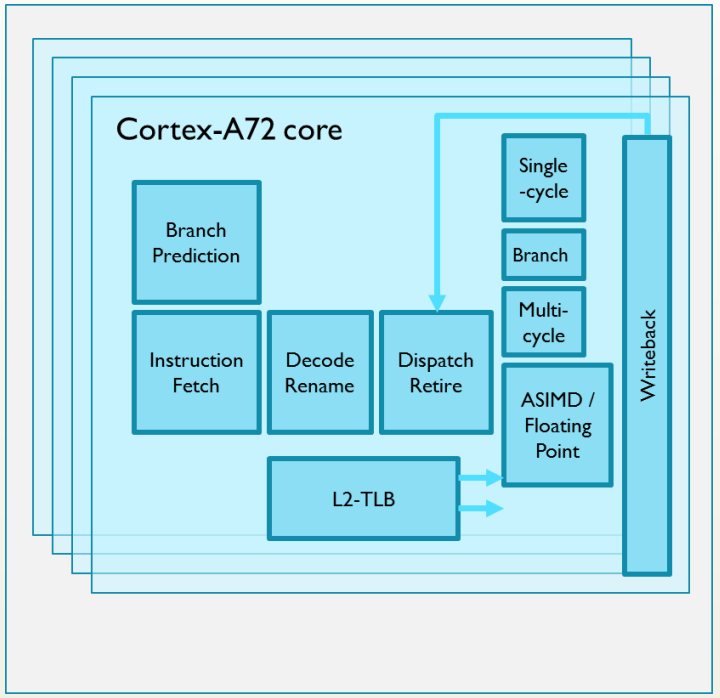
# Embedded Design with Microprocessors



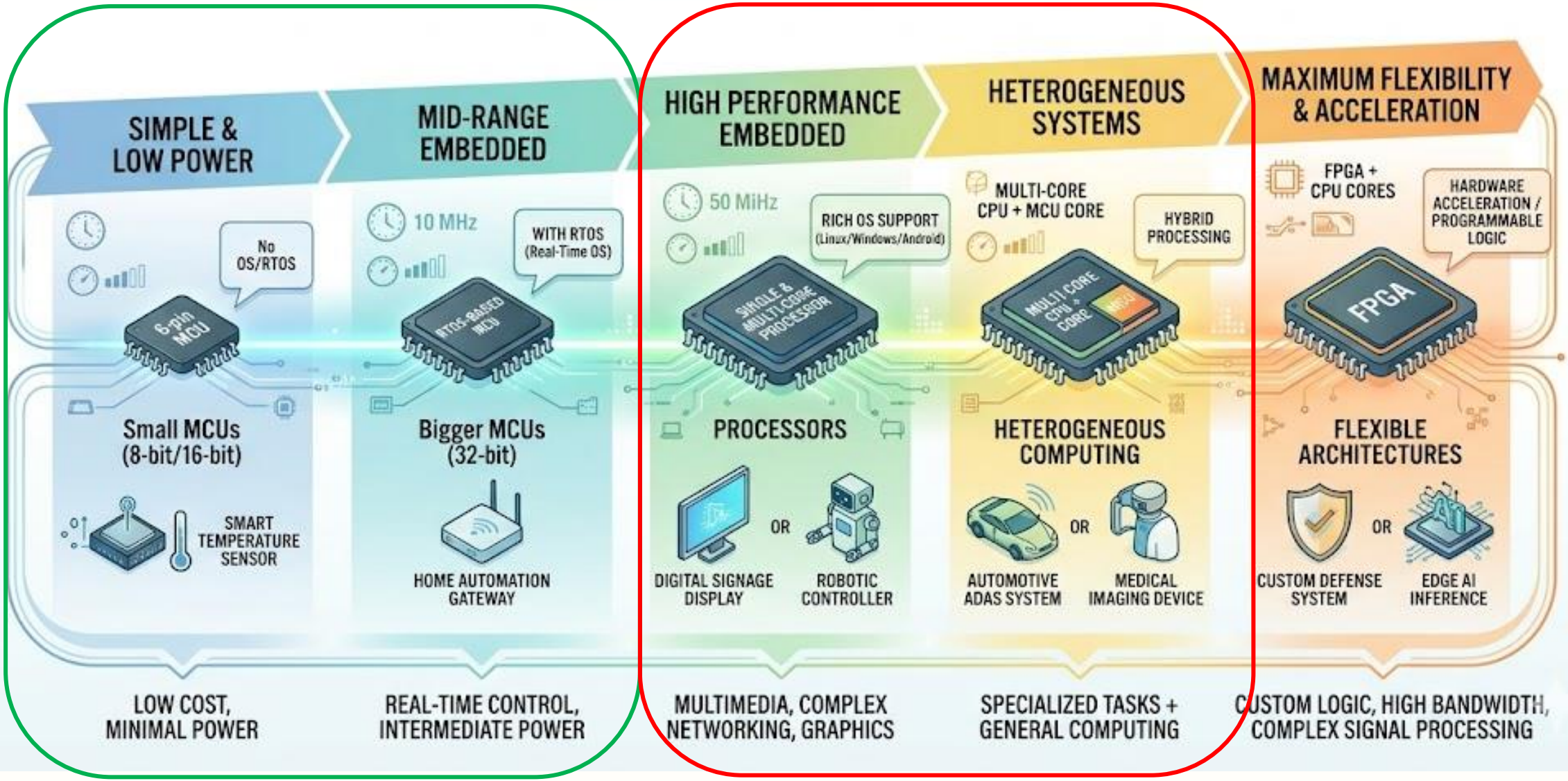




## Session #5

## Microprocessors with Operating System

# SPECTRUM OF CHOICES FOR EMBEDDED PRODUCT DEVELOPMENT

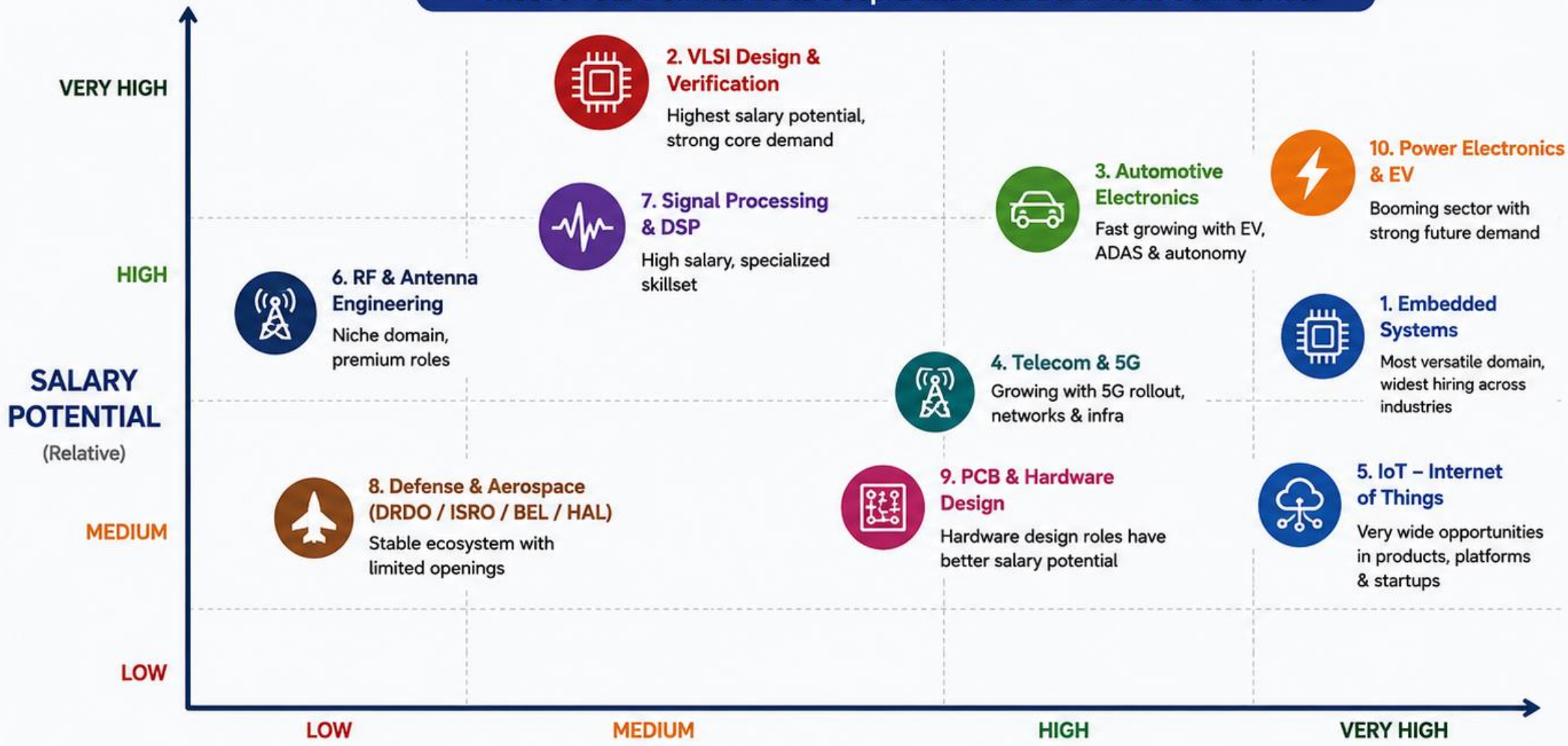
SELECTING THE RIGHT HARDWARE AND SOFTWARE FOR APPLICATIONS



# ECE CAREER DOMAINS IN INDIA

## Job Opportunities vs. Salary Potential (2026 Outlook)

**Choose Your Domain. Build Deep. Build Real. Build With Confidence.**




### HOW TO READ THIS GRAPH

- Right = More Job Opportunities in India
- Up = Higher Salary Potential
- Bubble color represents the domain category

### DOMAIN CATEGORIES

- Core Electronics & Systems
- Semiconductors
- Automotive & EV
- Telecom & Networks
- Specialized / Advanced
- Hardware Design
- Defense & Aerospace
- IoT & Emerging Tech

- This positioning is based on current industry trends in India (2026 outlook).
- Domain overlap exists. Many roles require skills from multiple domains.
- Build depth in your primary domain and gain working knowledge in a secondary domain.
- Skills + Projects + Internships = Real employability.

**Seekers Signpost**
Build Deep. Build Real. Build With Confidence.


# Recap – Sessions 1 - 4

## Why we went through these Sessions?

- 1) As we wanted to embark into the learning about Embedded System Design aspects & Fundamentals.
- 2) We started with MCUs as they are the key components in one end of the spectra of Embedded products design. Most of the electronic products use one or more MCUs.

## What are the topics we covered?

- 1) How the integrated circuits design evolved into Logics, ALU, Flipflops and memory;
- 2) Silicon aspects of the CPU design of an MCU; other hardware elements inside an MCU such as Memories, Cache, DMA and Bus Matrix interfaces;
- 3) Importance of ISA, ISA Extensions and Accelerators; and
- 4) How the silicon vendors create MCUs and differentiate in the market.

## How does it help an engineer / Embedded system developer?

- 1) Having a strong knowledge on MCU core/CPU design aspects and complete MCU, lays a great hardware & Software foundation for one to select the right MCU optimizing the development time & cost;
- 2) Convert all it's features into benefits for the end application
- 3) Also helps one to understand the other spectra of embedded design aspects easier.

# Where do MCUs “Truly Shine?”

1. Single-chip simplicity — no external RAM, no OS boot time, instant-on

2. Cost-optimized for high-volume production

3. Deterministic real-time control with interrupt latency in single-digit microseconds

4. Ultra-low power operation — down to nanoamperes in sleep modes

# Where do MCUs may “Hit a Wall?”

1. Many applications may be dynamically loaded and run and hence there is a need for a full-fledged OS (not a thin RTOS)

2. Data throughput exceeds what a single CPU core can process in time. The algorithm is too complex for even a Cortex-M85 with Helium

3. Hardware-level parallelism is required — not software threading

May need an MPU / FPGA / Heterogenous solution

# What are we going to cover in Session #5

1. When MCUs Are Not Enough
2. What are Microprocessors (MPUs) where do we need them?
3. Anatomy and Characteristics of MPU
4. Popular MPU SOMs in the market for Embedded Systems
5. MPU Selection Criteria
6. Embedded System-Linux Development flow example
7. MPUs for Real Time applications

# When do I need more than an MCU for my embedded system Design?

# The Four Axes That Drive Your Choice Beyond a simple MCU

| Raw Compute Performance | Hardware Parallelism | OS & Ecosystem Need | Hardware Flexibility & Custom Logic |
|---|---|---|---|
| • Need exceeds what even a Cortex-M85 MCU at 1 GHz with Helium can deliver<br>• Examples: H.264 video encode, complex AI inference, 3D graphics rendering all together<br>• Solution direction: Application Processor (MPU) | • Multiple independent data streams must be processed simultaneously<br>• Examples: Multi-channel signal acquisition, custom communication protocols, radar processing<br>• Software threading is not a substitute — you need true concurrent hardware execution<br>• Solution direction: FPGA | • Application requires Linux, Android, file systems, networking stack, or rich UI frameworks<br>• An RTOS on MCU is not sufficient — you need a full memory-managed OS environment<br>• Solution direction: MPU with Linux / Android | • Protocol does not exist as standard silicon — must be implemented in reconfigurable logic<br>• Timing constraints require nanosecond-level determinism beyond what any CPU can guarantee<br>• Solution direction: FPGA |

# EMBEDDED SYSTEMS – A SPECTRUM OF CHOICES

From simple control to highly parallel, programmable and heterogeneous architectures

**SIMPLEST**
Lower Cost, Lower Power, Lower Performance

**INCREASING PERFORMANCE, COMPLEXITY & FLEXIBILITY**

**MOST COMPLEX**
Higher Performance, Higher Flexibility

## 1 MCU (Bare Metal to RTOS)

6-pin to 100's of pins

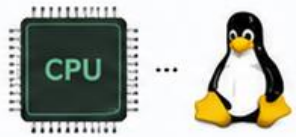
- Single core MCU
- Bare metal or RTOS
- 8/16/32-bit
- Timers, ADC, GPIO, UART, I²C, SPI, CAN...

Examples: AVR, PIC, STM32, NXP LPC, TI MSPM0

**Use Cases:**
Sensor nodes, simple control, appliances, wearables, toys, low cost devices

## 2 MPU / Application Processor (OS Based)

Single core to Multi core



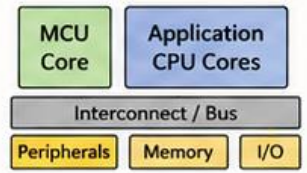

- Runs OS (Linux, RTOS, Android, etc.)
- Single or multi-core CPUs
- Higher memory & peripherals
- Rich connectivity (Ethernet, USB, PCIe, Wi-Fi, etc.)

Examples: ARM Cortex-A series, Raspberry Pi, i.MX, TI AM62x

**Use Cases:**
HMI, gateways, data logging, network devices, multimedia, industrial HMIs

## 3 Heterogeneous SoC (CPU + MCU)

Multi-core CPU + MCU core

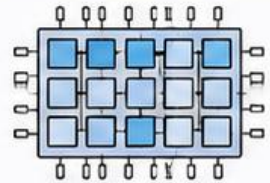



- Combines real-time MCU with high-performance CPU
- Optimized for both control and complex processing
- Shared peripherals & memory

Examples: NXP i.MX RT series, TI AM243x, Renesas RZ/G

**Use Cases:**
Motor control + HMI, robotics, ADAS, industrial automation, smart devices

## 4 FPGA (Programmable Logic)



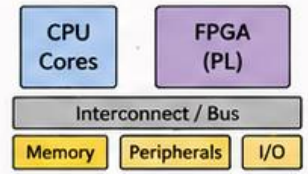
- Custom hardware in programmable logic
- Massive parallelism
- Deterministic, low latency
- Steep learning curve (HDL)
- No inherent OS (soft-core optional)

Examples: Xilinx Artix/Kintex/ Versal, Intel (Altera) Cyclone

**Use Cases:**
High-speed processing, custom protocols, image/video pipelines, instrumentation, SDR

## 5 FPGA + CPU (Hybrid SoC / MPSoC)

CPU cores + Programmable Logic




- CPU runs OS & applications
- FPGA accelerates critical tasks
- Best of both worlds
- Highest complexity & flexibility

Examples: Xilinx Zynq / Zynq UltraScale+, Intel (Altera) SoC FPGA

**Use Cases:**
Machine vision, AI/ML acceleration, high-performange networking, aerospace, defense

| | | |
|---|---|---|
| Compute Performance | Low | Very High |
| Determinism / Real-Time | Very High | Medium |
| Power Consumption | Very Low | High |
| Flexibility / Customization | Low | Very High |
| Development Complexity | Low | Very High |

There is no "one size fits all". The right choice depends on your application requirements for performance, power, cost, time-to-market and skill set.

# How Do You Decide? — MCU vs MPU vs FPGA

1. Question 1 — **Does your application need a full OS?** (Linux / Android / Windows)
   - YES → You need an MPU → Go to Question 3NO → Continue to Question 2
2. Question 2 — **Does your application require true hardware parallelism or custom timing logic?**
   - YES → You need an FPGA → Go to Question 4NO → Continue with MCU evaluation → Go to Question 5
3. Question 3 — MPU Selection trigger confirmed.
   - Ask: Does it also need real-time I/O alongside the OS? → Consider MPU + MCU co-design
   - Ask: Does it need ML acceleration? → Consider MPU with NPU (Jetson, i.MX 8)
4. Question 4 — FPGA Selection trigger confirmed.
   - Ask: Do you also need a CPU running alongside? → Consider SoC FPGA (Zynq, Intel Cyclone SoC)Is power/cost critical? → Consider smaller FPGA families (Lattice iCE40, ECP5)
5. Stay with MCU.
   - Ask: Does it need heavy DSP / signal processing? → Add DSP extensions or dedicated DSP chip
   - Ask: Does it need ML inference at the edge? → Cortex-M55/M85 with Helium + Ethos-U NPU

# The Definitive Three-Way Comparison Table: #1

| Feature | MCU | MPU | FPGA |
|---|---|---|---|
| **Architecture** | Fixed CPU + **integrated peripherals** | Fixed CPU + cache + **MMU** | Reconfigurable logic fabric |
| **Execution Model** | Sequential — instruction by instruction | Sequential — out-of-order per core | Spatial — all operations simultaneous |
| **Clock Speed** | 48 MHz to 1 GHz | 800 MHz to 3+ GHz | 100 MHz to 700 MHz fabric |
| **Compute Throughput** | Low to moderate | High to very high | Very high — parallelism multiplies throughput |
| **On-chip Flash** | Yes — 16 KB to 2 MB | No — external eMMC | No — external SPI Flash for bitstream |
| **On-chip RAM** | Yes — 4 KB to 8 MB | On chip and external LPDDR | Yes — BRAM/URAM — fixed blocks |
| **External RAM** | Optional | Mandatory | Optional |
| **MMU** | No — MPU only | Yes — full virtual memory | No — but SoC FPGA has ARM with MMU |
| **OS Support** | Bare metal or RTOS | Full OS — Linux/Android | Bare metal — soft CPU for RTOS/Linux |
| **Real-Time Determinism** | Excellent — nanosecond ISR | Poor without PREEMPT-RT | Perfect — sub-nanosecond guaranteed |
| **Interrupt Latency** | 12–200 ns deterministic | Microseconds variable | 1 clock cycle — deterministic |
| **True Parallelism** | No — single instruction stream | No per core — SMP multicore only | Yes — thousands of simultaneous operations |
| **Custom Logic** | No — fixed peripheral set | No — fixed peripheral set | Yes — any digital circuit describable in RTL |
| **Reconfigurability** | Never — silicon fixed | Never — silicon fixed | Yes — new bitstream = new circuit |

# The Definitive Three-Way Comparison Table: #2

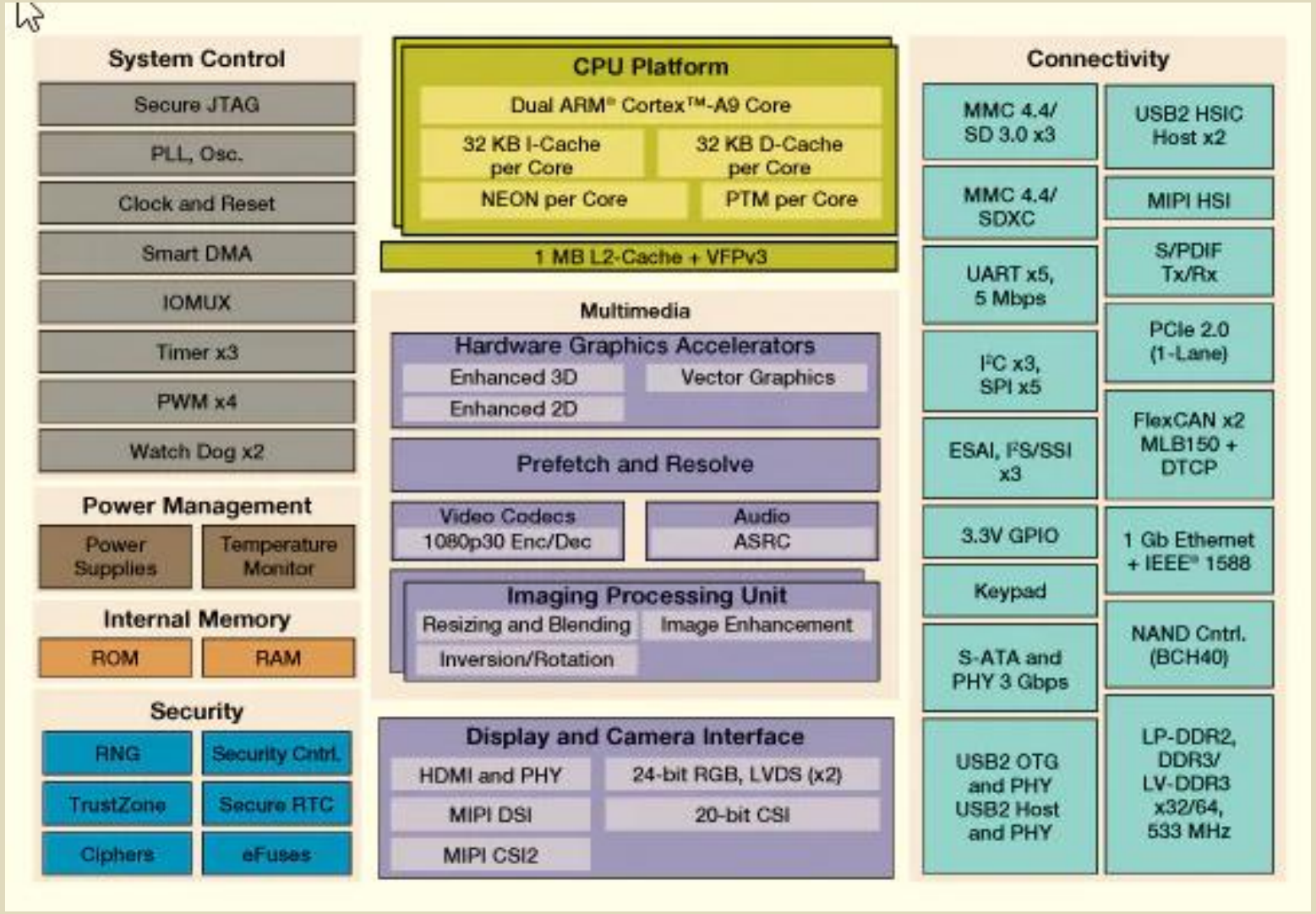
| Feature | MCU | MPU | FPGA |
|---|---|---|---|
| **Boot Time** | Microseconds — instant on | Seconds — OS boot | Milliseconds — bitstream load |
| **Power — Sleep** | nA to μA — excellent | mW — always on | mW — static power always present |
| **Power — Active** | mW range | Hundreds of mW to W | Tens of mW to W |
| **Development Language** | C / C++ / Assembly | C / C++ / Python / Linux apps | VHDL / Verilog / HLS |
| **Development Complexity** | Low to moderate | Moderate to high | High — paradigm shift required |
| **Debug Tools** | GDB + JTAG — mature | GDB + Linux tools — excellent | ILA/ChipScope — different mindset |
| **Iteration Time** | Seconds — flash and run | Minutes — build and boot | Minutes to hours — synthesize and implement |
| **Unit Cost** | $0.30 to $15 (IC) | $5 to $200+ (module) | $1 to $3,000 |
| **BOM Complexity** | Low — few components | High — DDR, eMMC, PMIC | Moderate — bitstream Flash, power regulators |
| **Ecosystem Maturity** | Excellent — decades of tools | Excellent — Linux ecosystem | Good — improving rapidly |
| **AI / ML Capability** | TinyML — small models only | Full frameworks — large models | Custom accelerators — unmatched throughput |
| **Volume Cost Scaling** | Excellent — aggressive pricing | Moderate | Poor to moderate — FPGA cost stays high |
| **Supply Chain Longevity** | 10–15 years typical | 5–15 years — varies | 8–15 years typical |

# What is the Key challenge for working on Embedded Processors? or what is the entry barrier for development?

- Embedded Firmware Development (Bare-Metal / RTOS): No OS (Bare metal C Code) or minimal OS (OS Port generally available ready) – Engineering students can do.
- Linux Application Development (PC/Desktop) : No need to work on OS as it is preloaded in PC, Free. User can create applications running on top of OS – Students can do.
- Embedded Linux System Development: This sits between firmware and desktop Linux—**and is the most complex**. You are building both the OS and the application – To learn this, need experienced personnel guidance once to show the path!

Here is a great pool to dive into for mastering Embedded System Linux https://bootlin.com/docs/

# Microprocessor / MPU

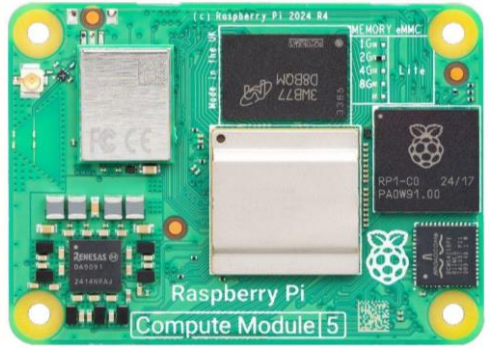



i.MX 6

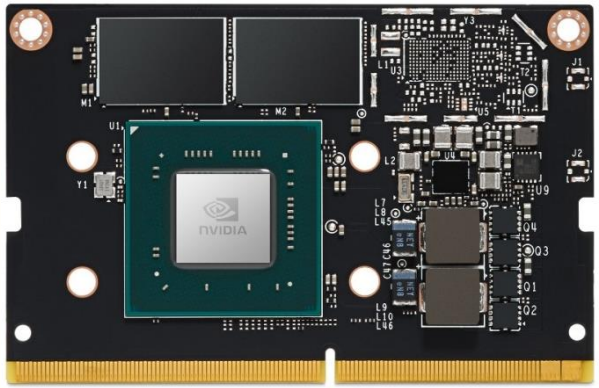
# What this section covers

- What makes an MPU fundamentally different from an MCU
- The architecture inside a modern application processor
- Real MPUs used in embedded products today — with selection guidance


NXP i.MX 8QM Computer on Module - Apalis iMX8




Raspberry Pi Compute Module 5 | Crowd Supply




NVIDIA - Jetson Nano


Qualcomm TRIA SM2S-IQ615

# What Is a Microprocessor / MPU?

- **An MCU is a complete computer on a chip including memories and peipherals.**
- **An MPU is a powerful CPU that expects the rest of the system to be built around it.**
  - One or more high-performance CPU cores — Cortex-A, x86, or RISC-V application class
  - Large L1, L2 and sometimes L3 cache hierarchies — megabytes, not kilobytes
  - **A Memory Management Unit (MMU) — mandatory for running a full OS like Linux**
  - High-speed memory interfaces — LPDDR4, LPDDR5 — connecting to external DRAM
  - No or minimal on-chip Flash — program code lives in external storage (eMMC, SD, QSPI)

# Where Do MPUs Show Up in Embedded Products? #1, 2

**Domain 1 — Industrial Human Machine Interface (HMI)**

- High resolution touchscreen displays — 1080p, multi-touch, gesture recognition
- Requires GPU for smooth UI rendering, OpenGL ES, Qt framework
- Boot into a full graphical environment in under 5 seconds
- **Example products: CNC machine panels, PLC operator terminals, medical device displays**
- Typical MPU: NXP i.MX 8M Plus, STM32MP1, Renesas RZ/G2

**Domain 2 — Edge AI & Machine Vision**

- Running full OpenCV, TensorFlow, or PyTorch inference pipelines
- Requires GBs of RAM — model weights alone can exceed MCU total Flash
- Real-time camera feed processing — object detection, OCR, defect inspection
- **Example products: Smart cameras, quality inspection systems, retail analytics**
- Typical MPU: NVIDIA Jetson Orin NX, NXP i.MX 8M Plus with NPU, Rockchip RK3588

# Where Do MPUs Show Up in Embedded Products? #3, 4

## Domain 3 — Automotive Infotainment & ADAS

- Android Automotive OS, navigation, voice recognition, camera fusion
- Functional safety requirements — ISO 26262 ASIL-B/D
- Multiple camera inputs processed simultaneously — surround view, lane detection
- **Example products: Head units, digital instrument clusters, ADAS domain controllers**
- Typical MPU: Qualcomm Snapdragon Ride, NXP S32G, Renesas R-Car H3

## Domain 4 — Networking & Communications Equipment

- Packet processing, deep packet inspection, protocol termination
- Requires hardware accelerators for encryption, compression alongside CPU
- Line-rate processing at multi-gigabit speeds
- **Example products: Industrial routers, 5G gateways, managed switches**
- Typical MPU: NXP Layerscape LS1046A, Marvell OCTEON, Intel Atom C3000

# Where Do MPUs Show Up in Embedded Products? #5, 6

**Domain 5 — Medical Imaging & Diagnostics**

- Ultrasound image reconstruction, CT scan processing, ECG analysis
- Requires certified OS (Linux with IEC 62304), data security, audit trails
- High resolution display output alongside real-time data acquisition
- **Example products: Portable ultrasound, patient monitoring systems, diagnostic workstations**
- Typical MPU: AMD Ryzen Embedded V2000, Intel Core i7 Embedded, NXP i.MX 8QuadMax

**Domain 6 — Single Board Computers & Gateway Devices**

- IoT gateways aggregating data from hundreds of sensor nodes
- Running containerized applications — Docker, Kubernetes edge deployments
- Over-the-air update capability, cloud connectivity, VPN termination
- **Example products: Industrial IoT gateways, energy management controllers, smart grid nodes**
- Typical MPU: Raspberry Pi CM4, Toradex Verdin i.MX 8M Plus, Variscite DART-MX8M-PLUS

# Microprocessor (MPU) Architecture

# The anatomy of a modern MPU — Simplified five-layer diagram

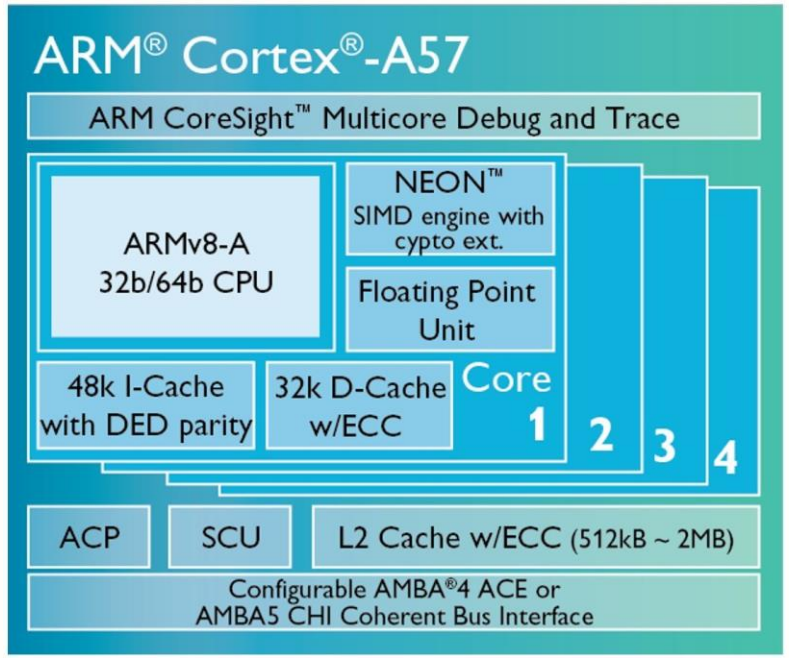
CPU Core Cluster (Cortex-A cores)

Cache Subsystem (L1/L2/L3)

MMU + TLB + Page Table Walker

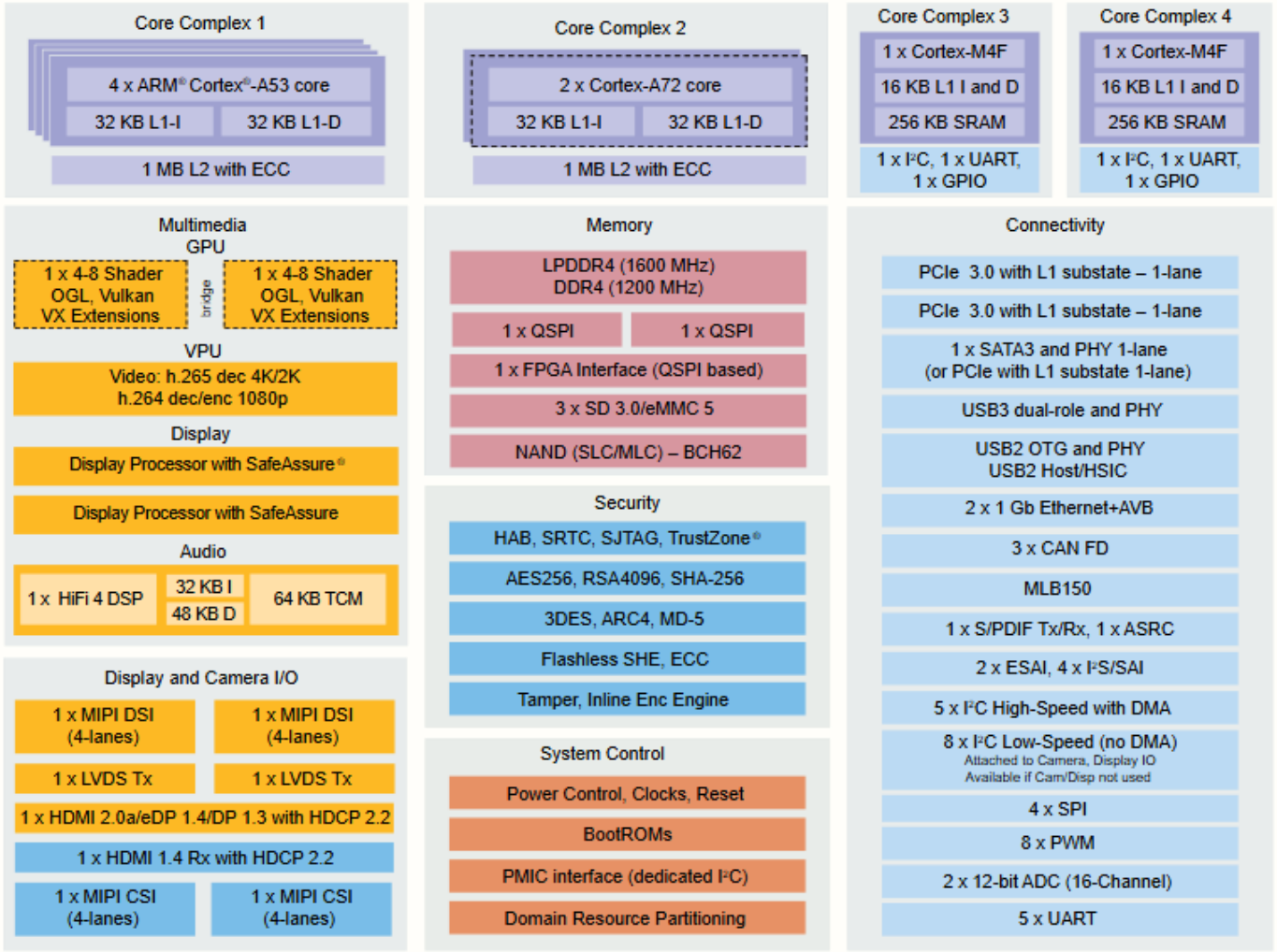
Interconnect Fabric (AXI / CoreLink)

LPDDR Controller | eMMC | QSPI | PCIe



# The anatomy of a modern MPU — #1 CPU Core Clusters

- One or more Cortex-A cores — each with its own private L1 I-Cache and D-Cache
- Each core has its own register file, pipeline, branch predictor, and execution units
- big.LITTLE pairing — high performance cores + high efficiency cores on same die
- Example: Cortex-A72 (performance) + Cortex-A53(efficiency) + M4F (MCU Core) in NXP i.MX 8M Plus



# How do I develop  firmware for multi cores?

# Multicore Processor – OS role in managing cores

1. **Dynamic Scheduling:** The OS scheduler (e.g., in Linux or Android) treats all cores as a pool of resources. It dynamically migrates tasks between cores to ensure no single core is overwhelmed.
2. **ARM big.LITTLE/DynamIQ:** Modern Cortex-A systems have high-performance ("big") cores and energy-efficient ("LITTLE") cores. The OS identifies the intensity of a task (e.g., video editing vs. background sync) and schedules it accordingly.
3. **Core Power Management:** The OS can power down unused cores to save energy and power them up when workload increases

# MPU with Multicore Processors – Symmetric Multiprocessing

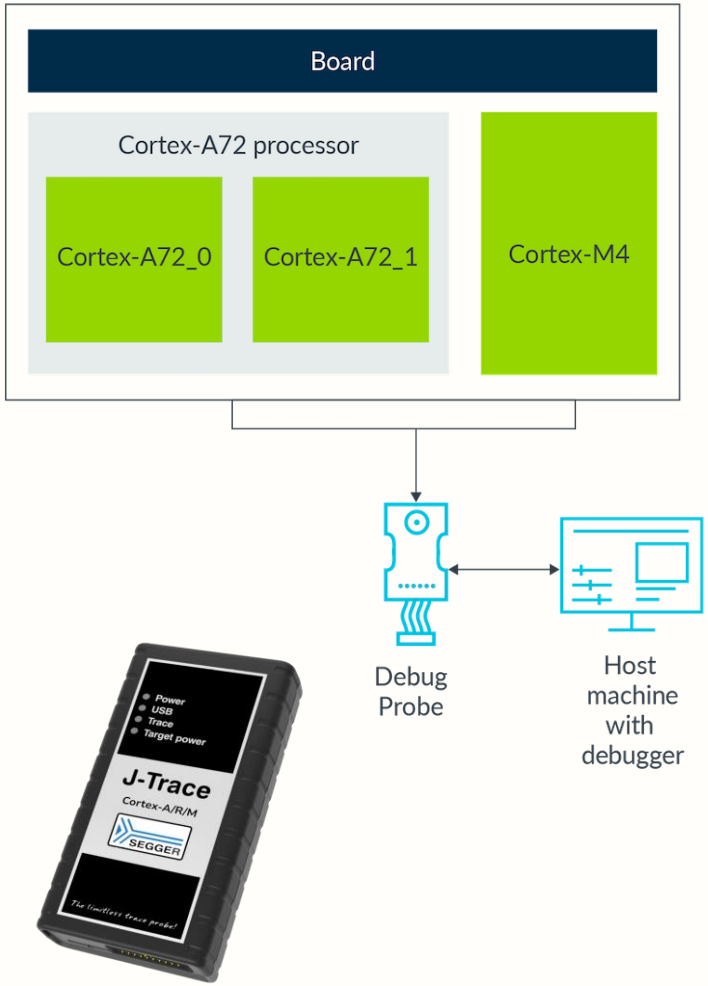
Symmetric Multiprocessing (SMP) is the **software architecture** that makes this dynamic, OS-led management of cores possible.

- **Equal Access:** In an SMP system, all processor cores are identical in functionality and have equal access to memory (RAM) and peripherals.
- **Single OS Instance:** A single OS instance manages all cores. It does not dedicate a core to a specific task but treats them as interchangeable compute resources.
- **Hardware Coherency:** ARM Cortex-A processors use a **Snoop Control Unit (SCU)** to maintain data cache coherency between cores automatically, which is a requirement for SMP.

# MPU with Multicore processor & MCU – Asymmetric Multiprocessing (AMP)

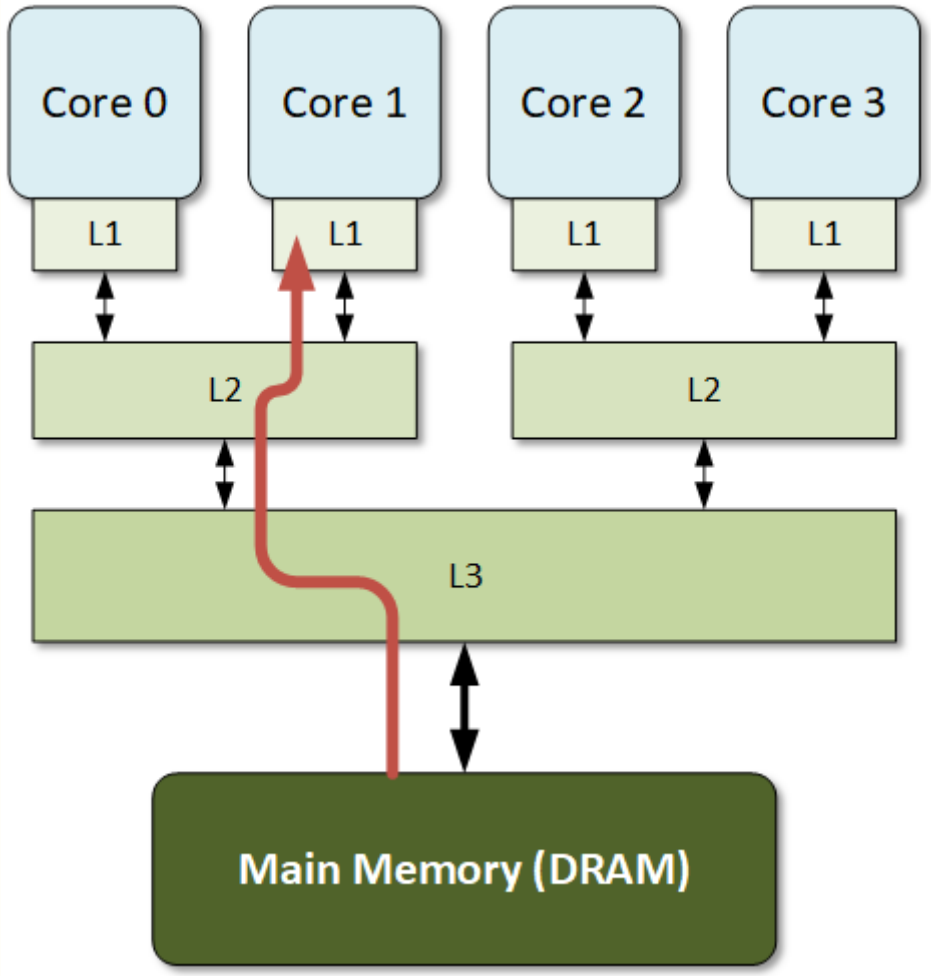
- Similar to an SMP connection, an Asymmetric Multiprocessing (AMP) connection typically involves the debugger establishing simultaneous links to multiple cores or processors.
- Consider, for example, an SoC featuring a Cortex-A72 processor dedicated to application execution alongside a Cortex-M4 core. Given their distinct architectures, neither the same program nor identical debug hardware is shared between the Cortex-A72 and the Cortex-M4 within this heterogeneous platform.
- Observing the state of all these cores calls for an AMP connection. Such a connection enables simultaneous access to both the Cortex-A72 and the Cortex-M4, while maintaining a looser coupling between the two devices from the debugger's perspective.

Learn the architecture - Debugger usage on Armv8-A Guide

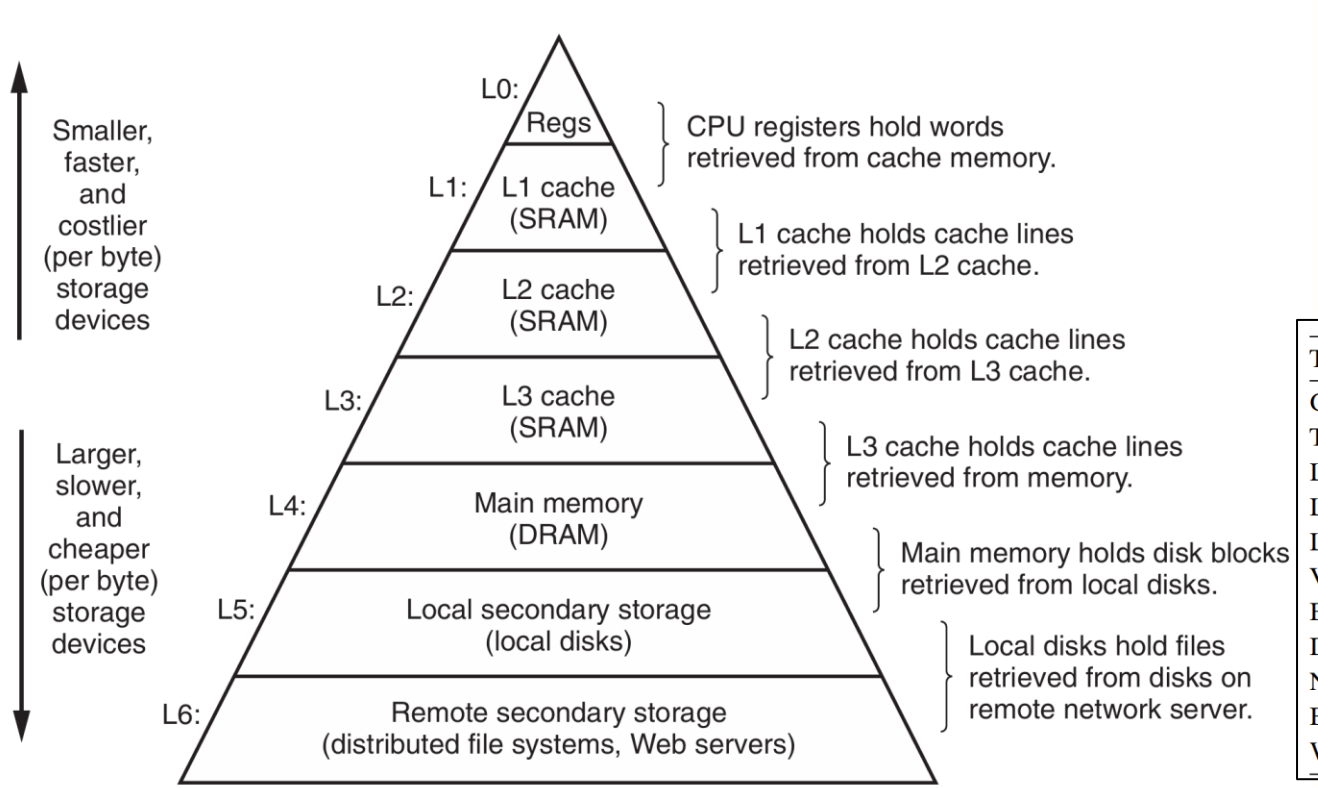


# The anatomy of a modern MPU — #2 Cache Subsystem

- L1: Private per core — 32–64 KB each — single cycle latency — invisible to software
- L2: Per core or per cluster — 256 KB to 1 MB — hardware managed — ~10 cycles
- L3: Shared across all cores — 2–8 MB — last stop before DRAM — ~30–40 cycles
- Cache coherency hardware ensures all cores see consistent data — critical for SMP Linux



# An Example of Memory Hierarchy




| Type | What cached | Where cached | Latency (cycles) | Managed by |
|---|---|---|---|---|
| CPU registers | 4-byte or 8-byte words | On-chip CPU registers | 0 | Compiler |
| TLB | Address translations | On-chip TLB | 0 | Hardware MMU |
| L1 cache | 64-byte blocks | On-chip L1 cache | 4 | Hardware |
| L2 cache | 64-byte blocks | On-chip L2 cache | 10 | Hardware |
| L3 cache | 64-byte blocks | On-chip L3 cache | 50 | Hardware |
| Virtual memory | 4-KB pages | Main memory | 200 | Hardware + OS |
| Buffer cache | Parts of files | Main memory | 200 | OS |
| Disk cache | Disk sectors | Disk controller | 100,000 | Controller firmware |
| Network cache | Parts of files | Local disk | 10,000,000 | NFS client |
| Browser cache | Web pages | Local disk | 10,000,000 | Web browser |
| Web cache | Web pages | Remote server disks | 1,000,000,000 | Web proxy server |

Courtesy: Computer Systems A Programmer's Perspective, Randal E. Bryant, David R. O'Hallaron, Carnegie Mellon University

# The anatomy of a modern MPU — Snoop Control Unit (SCU)

**Core Purpose:** **Cache Coherency** management

**Typical Placement:** The SCU is physically located within a **MPCore cluster**. It sits between the individual CPU cores and the shared L2 cache or the system bus (like the AXI interface).

**Why is it Necessary?** In a single-core system, the CPU just looks at its own cache. In a multicore system, if two cores are working on the same block of memory:

- **Core 0** reads a variable and changes it in its L1 cache.
- **Core 1** reads the same variable from main memory.
- Without an SCU, **Core 1** would receive the old value because **Core 0** hasn't written its changes back to the main memory yet.

The SCU prevents this by "snooping" the bus using snoop protocols ((like MESI or MOESI – Modified, Owned, Exclusive, Shared, Invalid )and signaling to Core 1 that a newer version exists in Core 0's cache.

**Limitations :** It is important to note that the SCU typically manages coherency **within a single cluster** (e.g., 4 cores). In modern "Many-Core" or "Big.LITTLE" systems with multiple clusters, a higher-level interconnect (like the **CoreLink CCI - Cache Coherent Interconnect**) is used to maintain coherency between the different clusters.

# The anatomy of a modern MPU
## — Accelerator Coherency Port (ACP)

**Core Purpose:** An AXI slave interface on the **Snoop Control Unit (SCU)** that enables non-CPU masters (DMA, Crypto engines, FPGAs) to access memory coherently with the CPU's L1 caches.

**Typical Placement:** Located physically on the **SCU** within the Cortex-A processor cluster, acting as a bridge between the system-on-chip (SoC) logic and the private CPU memory subsystem.

**Why is it Necessary?**

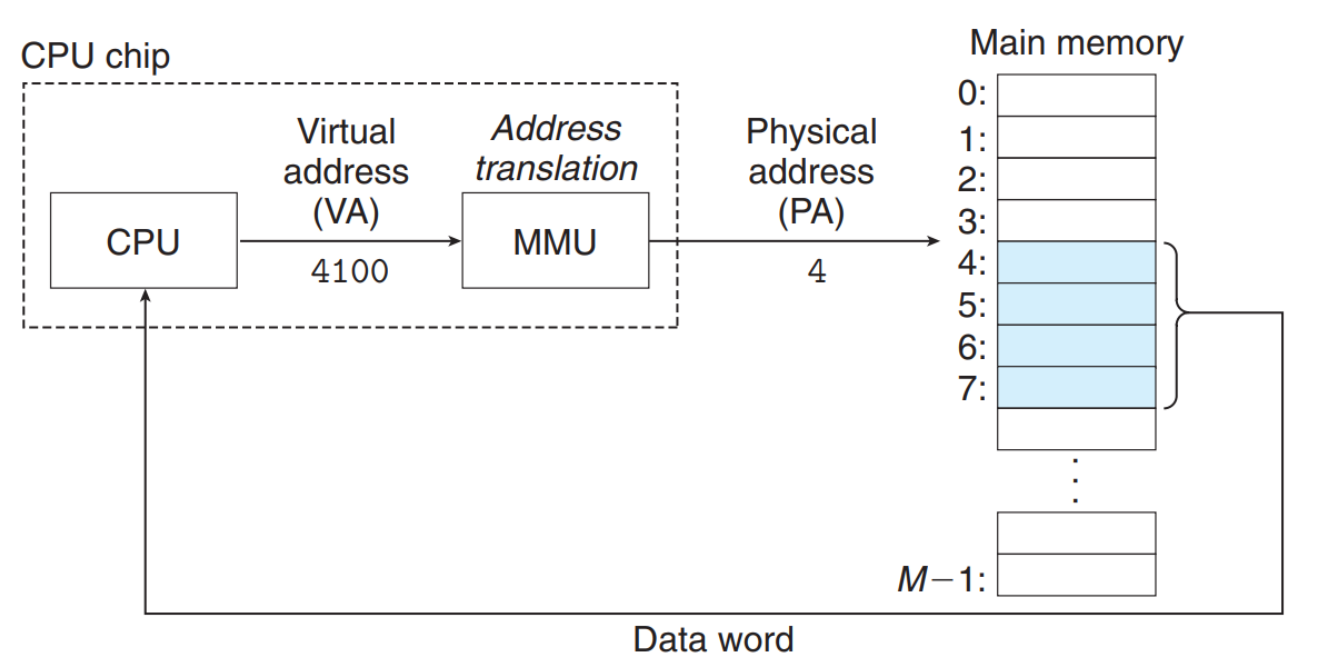
- **Standard DMA Issue:** Normally, if a DMA writes to RAM, the CPU doesn't know. The CPU might read its own "stale" L1 cache instead of the new data in RAM.
- **ACP Solution:** The ACP allows the DMA to "talk" to the SCU. The SCU automatically snoops the CPU caches and handles the data update, ensuring the CPU and DMA always see the same data.

**Key Benefits:**

- **Zero Software Overhead:** Eliminates the need for "Cache Clean" or "Invalidate" instructions in code, saving CPU cycles.
- **Reduced Latency:** External masters can read "dirty" data directly from the CPU L1 cache via the SCU, avoiding slow trips to the main DDR memory.
- **One-Way Coherency:** Provides a simple path for peripherals to remain coherent with the CPU cluster without the complexity of a full multi-way coherent bus.

# The anatomy of a modern MPU
## — #3 Memory Management Unit (MMU) + TLB + Page Table Walker

- Sits between CPU and memory bus — translates every virtual address to physical address
- Translation Lookaside Buffer (TLB) cache — avoids page table walk penalty
- Page table walk hardware — automatically handles TLB misses in hardware
- Enforces memory permissions per page — read/write/execute/no-access per process



# Memory Architecture & MMU — The Foundation of Every Full OS

**MCU world**

On a typical MCU (e.g., ARM Cortex-M):

- All tasks share the **same physical memory**
- Even with an RTOS like FreeRTOS:
  - Tasks are just scheduled functions
  - A bad pointer can corrupt *anything*
- Memory protection Unit (if present) gives **limited guardrails**, but not full isolation

So, here the mindset is:

"I trust my code not to break other code."

**Microprocessor world**

On a microprocessor running Linux:

- The system runs **multiple independent processes**
- Each process gets its **own virtual address space**
- The MMU enforces **strict isolation**. Hence a process may crash but never a system

Here the mindset is:

"Even if code is buggy, it should not break other programs."

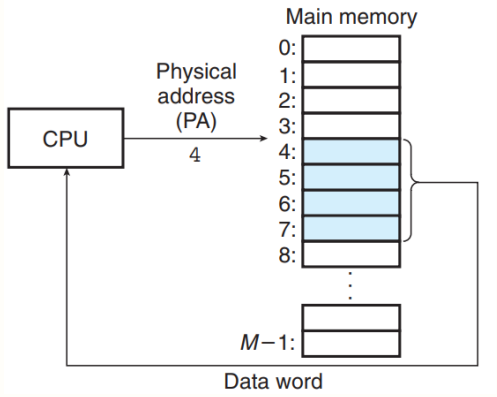
# Need for Memory Management Unit (MMU) in a Microprocessor

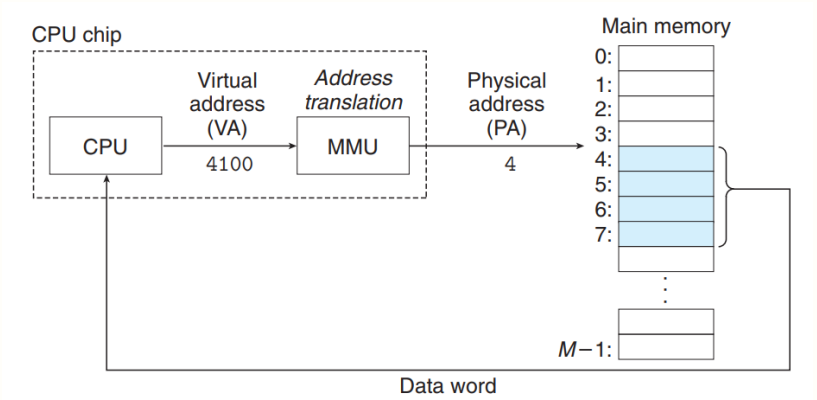
The primary advantage of an **MMU** over an **MPU** is its ability to create **Virtual Memory**, which provides three critical benefits that an MPU cannot offer:

- **Memory Isolation:** Every application runs in its own private virtual address space. A crash or malicious act in one process cannot physically access or corrupt the memory of another process or the kernel.
- **Physical Address Abstraction:** The OS can map fragmented pieces of physical RAM into a single, contiguous block of virtual memory. This eliminates the "fragmentation" problem where you might have enough total RAM for a task, but not in one continuous physical chunk.
- **Expanded Memory Space:** Through "swapping" or "paging," an MMU allows a system to run applications that are larger than the actual physical RAM by moving unused data to a storage drive (like an SSD) and bringing it back only when needed.

# Memory Protection Unit vs. Memory Management Unit: Cortex-M vs. Cortex-A

| Feature | Memory Protection Unit (Cortex-M) | Memory Management Unit (Cortex-A) |
|---|---|---|
| **Deterministic Timing** | **Yes:** No page-table walks; access time is constant. | **No:** TLB misses cause unpredictable delays. |
| **Software Flexibility** | **Low:** Programs must be compiled for specific physical locations. | **High:** Multiple instances of the same program can run at the same virtual address. |
| **System Reliability** | Prevents illegal access to memory. | Prevents illegal access **and** enables complex multitasking. |



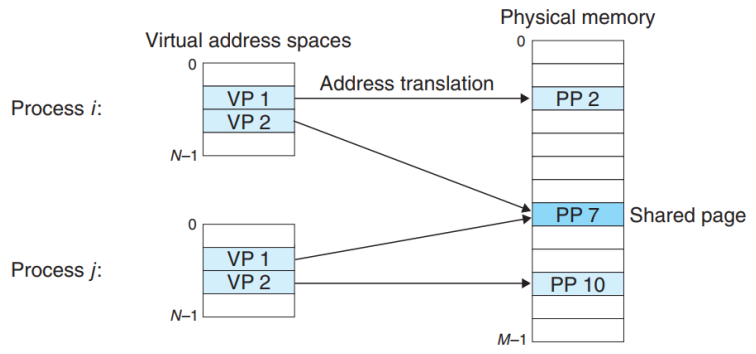



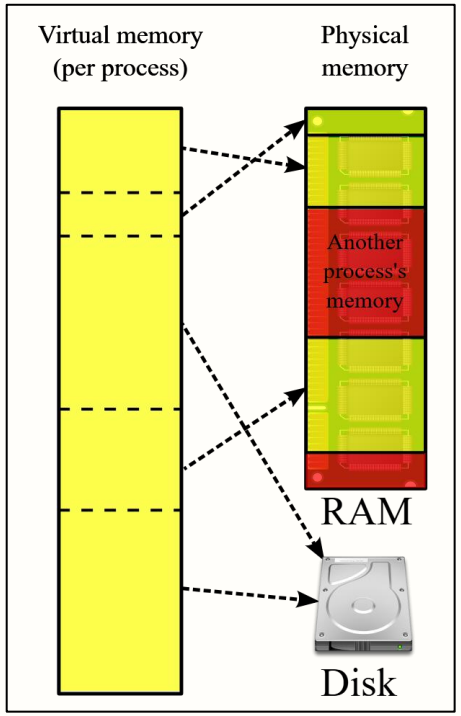


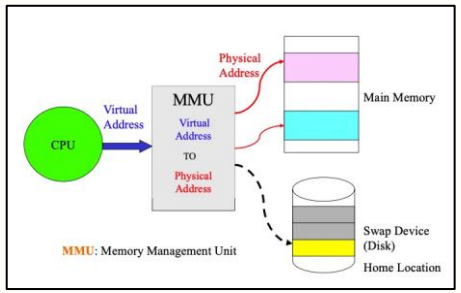

Image Courtesy: Computer Systems A Programmer's Perspective, Randal E. Bryant, David R. O'Hallaron, Carnegie Mellon University

# What the MMU enables — four capabilities no MCU can match: 1, 2

## Capability 1 — Virtual Address Space per Process

- Every Linux process sees identical virtual address map — 0x00000000 to 0xFFFFFFFF
- Physical RAM location is completely hidden from the process
- Process A at virtual 0x1000 and Process B at virtual 0x1000 map to different physical pages
- Hardware enforces this — no software overhead per memory access




## Capability 2 — Memory Protection & Process Isolation

- Each page tagged with permissions — read / write / execute / no-access
- Attempt to execute data page → hardware fault → OS kills the offending process
- One crashed process cannot affect any other process or the kernel
- Foundation of Linux stability — why a crashed app does not reboot the system



Image Courtesy: https://medium.com/@jason890418123/virtual-memory-whats-virtual-address-and-how-it-translate-to-physical-address-7ccb582cf337

# What the MMU enables — four capabilities no MCU can match: 3, 4

## Capability 3 — Demand Paging & Virtual Memory

- OS can map more virtual memory than physical RAM exists
- Pages not currently needed swapped to storage — brought back on demand
- Enables running applications larger than available physical RAM
- Critical for embedded Linux systems with constrained DRAM budgets

## Capability 4 — Shared Libraries & Copy-on-Write

- Single physical copy of libc shared across all processes simultaneously
- Each process sees its own virtual copy — MMU maps all to same physical pages
- Copy-on-write — physical copy made only when a process actually writes to shared page
- Dramatically reduces RAM consumption in multi-process embedded Linux systems

# MMU Aspects: The Translation Lookaside Buffer (TLB)

In an MMU-based system (like a Cortex-A), the processor works with **Virtual Addresses**, but the actual data sits at a **Physical Address** in RAM. The "Page Table Walk" and "TLB" are the mechanisms used to bridge this gap.

1. **The Translation Lookaside Buffer (TLB)**

2. Think of the TLB as a **high-speed cache for address translations**. It lives inside the CPU and stores the most recently used mappings between Virtual and Physical addresses.

3. **TLB Hit:** When the CPU needs a piece of data, it first checks the TLB. If the translation is found, it happens nearly instantaneously (usually in a single clock cycle).

4. **TLB Miss:** If the translation is not in the TLB, the hardware must look it up in the main system memory. This brings us to the "Walk."

# MMU Aspects: Page Table Walk

When a TLB miss occurs, the MMU performs a **Page Table Walk**. Since storing a single flat table for 4GB of memory would be massive and wasteful, ARM uses a **Multi-level Page Table** (often 2 to 4 levels).

The "Walk" is the process of navigating these levels:

1. **Level 1 (Translation Table Base Register):** The CPU looks at a special register to find the start of the "Master" table in RAM.

2. **Indexing:** The MMU uses the top bits of the Virtual Address as an index to find an entry in this first table. This entry points to the next-level table.

3. **Traversing:** It repeats this for Level 2 and Level 3 until it reaches a "Descriptor" that contains the actual Physical Address.

4. **Completion:** Once the physical address is found, the MMU updates the TLB so it won't have to "walk" for this address again anytime soon.



Reference: https://gabrieletolomei.wordpress.com/miscellanea/operating-systems/virtual-memory-paging-and-swapping/

# The anatomy of a modern MPU — #4 Interconnect & Bus Fabric (AXI / CoreLink)

**Core Purpose:** The central communication "fabric" that routes data between CPU clusters, Memory, and Peripherals.

- **AXI (Bus Protocol):** The high-performance, point-to-point communication standard used for data transfer.
- **CoreLink (The Hardware):** The physical interconnect (e.g., CCI or NIC) that manages system-wide traffic.

**Key Functions:**

- **Arbitration:** Acts as a traffic controller when multiple masters (CPU, GPU, DMA) access the same memory simultaneously.
- **System Coherency:** Maintains data consistency across multiple clusters (e.g., Big.LITTLE) via the **Cache Coherent Interconnect (CCI)**.
- **Bridge & Route:** Decodes memory addresses to send data to the correct destination (DDR, SRAM, or I/O).
- **Domain Crossing:** Connects components running at different speeds/voltages (e.g., 2GHz CPU to 400MHz Peripheral).

**The Difference in One Sentence:** While the **SCU** manages data integrity *inside* a 4-core cluster, the **Interconnect** manages data flow and coherency *across* the entire chip.

# The anatomy of a modern MPU

## — #5 Memory & Storage Interfaces (LPDDR Controller | eMMC | QSPI )

**Core Purpose:** Dedicated hardware controllers that manage the physical electrical signaling and protocol logic required to communicate with external memory chips.

### 1. Dynamic Memory Interface (DRAM)

- **Controller:** Multi-Port Memory Controller (e.g., **CoreLink DMC**).
- **Purpose:** Connects the SoC to high-speed **LPDDR4/5** or **DDR4** RAM.
- **Key Task:** Manages complex timing, refresh cycles, and row/column addressing to provide the CPU with low-latency workspace.

### 2. Flash Storage Interfaces (Non-Volatile)

- **eMMC / SD:** Managed by the **SD/MMC Controller**. Used for mobile storage and boot code.
- **UFS (Universal Flash Storage):** A faster, full-duplex serial interface that replaces eMMC in high-performance Cortex-A systems.
- **QSPI (Quad SPI):** Often used for small "NOR Flash" chips to store the primary bootloader or firmware.

### Key Responsibilities:

- **Protocol Translation:** Converts internal **AXI** bus commands into the specific signaling required by the memory chip (e.g., JEDEC standards).
- **Data Integrity:** Implements **ECC** (Error Correction Code) and CRC to detect and fix bit-flips in storage.
- **Efficiency:** Uses **DMA** (Direct Memory Access) to move large blocks of data without taxing the CPU.

# Other Key Characteristics of an MPU - Multi-GHz Clocking and Frequency scaling

**Core Purpose:** To provide peak processing power for demanding tasks while maintaining thermal efficiency for mobile and embedded environments.

## Key Concepts:

- **High-Frequency Design:** Modern Cortex-A cores utilize deep, multi-stage pipelines to reach speeds of 2.0GHz to 3.0GHz+, enabling faster execution.
- **DVFS (Dynamic Voltage and Frequency Scaling):** The hardware adjusts clock speed and voltage in real-time based on workload. It "boosts" for heavy tasks and "throttles" during idle to save battery.
- **Clock Gating:** Automatically shuts off the clock signal to unused logic blocks within the CPU to eliminate unnecessary power consumption.

## The Multi-GHz Challenge:

- **Thermal Throttling:** Running at max frequency generates significant heat. The SoC must balance clock speed with the physical cooling limits of the device.
- **Power/Performance Curve:** Power consumption increases exponentially with voltage ($P \approx V^2 f$). Doubling the frequency often more than doubles the power draw.

# Other Key Characteristics of an MPU - Superscalar Architecture

**Core Purpose:** To increase performance by executing **multiple instructions per clock cycle** (IPC > 1) through redundant functional units.

**How it Works:**

- **Instruction-Level Parallelism (ILP):** The processor's hardware identifies instructions that do not depend on each other and executes them simultaneously.
- **Multiple Execution Pipelines:** Instead of one single path, the CPU has several parallel units (ALUs, FPUs, Load/Store units) to process different instructions at the same time.
- **Dispatch & Issue:** A central "dispatcher" looks ahead in the instruction stream and "issues" multiple instructions to available functional units in every cycle.

**Superscalar:** Multiple execution units allow >1 instruction per cycle.

**Out-of-Order execution:** Bypasses stalls by executing independent instructions while waiting for slow data.

**Result:** Maximum hardware utilization and significantly higher performance than older "In-Order" designs.

# The anatomy of a modern MPU
## — Generic Interrupt Controller

**Core Purpose:** The central engine that collects, prioritizes, and routes hardware and software interrupts to the appropriate CPU core.

**Key Functions:**

- **Prioritization:** Assigns urgency levels to interrupts;
- **Routing:** Distributes interrupts across the multicore cluster to balance the processing workload.
- **Virtualization:** Provides hardware support for hypervisors to switch interrupts between virtual machines securely.

**Interrupt Types:**

- **SGI (Software Generated):** Used for **Inter-Processor Communication (IPC**
- **PPI (Private Peripheral):** Core-specific interrupts, such as a local timer or private watchdog.
- **SPI (Shared Peripheral):** Global interrupts (e.g., UART, DMA, or GPIO)that can be routed to any core.

# Cortex M7 Features – Not available in Cortex A72

| Cortex-M7 Feature | Why Not in Cortex-A72 |
|---|---|
| TCM (ITCM/DTCM) | A72 uses cache hierarchy, no single-cycle TCM |
| MPU (no MMU) | A72 has full MMU with virtual addressing |
| NVIC | A72 uses GIC for interrupt management |
| WFI/WFE low-power sleep | A72 targets performance, not MCU-class power |
| Bit-Band region | Not part of AArch64 memory architecture |
| HardFault/BusFault handlers | A72 relies on OS-level exception handling |
| Thumb-2 only ISA | A72 supports full AArch64/AArch32 |
| No HW cache coherency | A72 has full SCU-based hardware coherency |

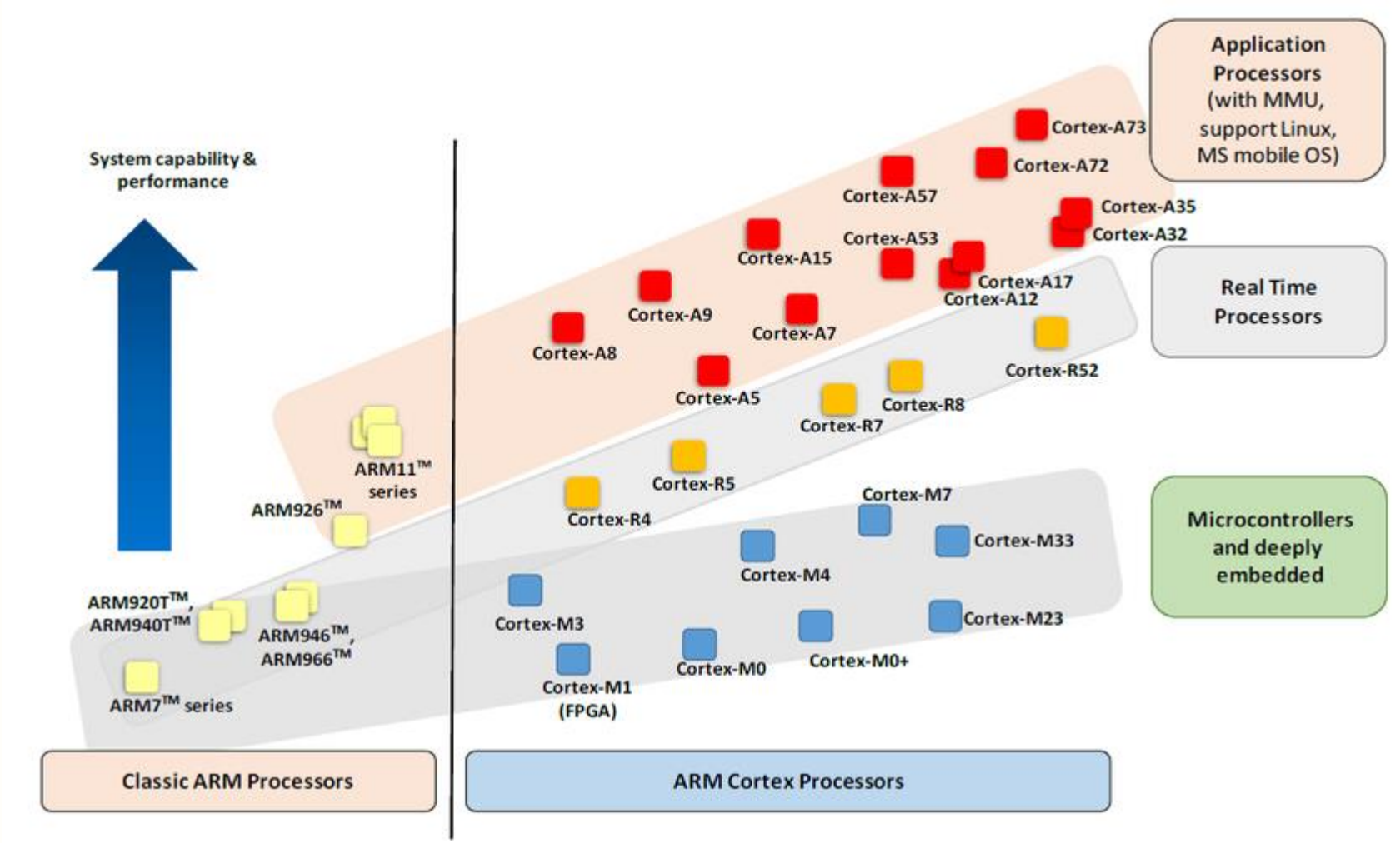
# ARM x86 and RISC V Processors in the market for Embedded applications

# ARM Cortex-A Family — The Cores Powering Embedded MPUs Today

**The Cortex-A family follows the same ISA-compatible lineage as Cortex-M — but optimized for maximum throughput, not minimum power:**

**The key distinction within Cortex-A — three tiers:**

- Cortex-A "Classic" cores — A53, A55 — efficiency focused — embedded and mobile
- Cortex-A "Performance" cores — A72, A76, A78 — maximum compute throughput
- big.LITTLE pairing — Performance core + Efficiency core on same SoC — scheduler assigns tasks dynamically



# Cortex-A Architecture Version Comparison Table

| Core | Arch | Pipeline | Issue Width | Relative IPC | Efficiency | Primary Use |
|---|---|---|---|---|---|---|
| **Cortex-A53** | ARMv8-A | 8-stage in-order | 2-wide | Baseline | Excellent | Low-power embedded |
| **Cortex-A55** | ARMv8.2-A | 8-stage in-order | 2-wide | +15% vs A53 | Best in class | Efficiency cluster |
| **Cortex-A72** | ARMv8-A | 15-stage OoO | 3-wide | 3× vs A53 | Moderate | Performance embedded |
| **Cortex-A76** | ARMv8.2-A | 13-stage OoO | 4-wide | 4.2× vs A53 | Good | High performance |
| **Cortex-A78** | ARMv8.2-A | 13-stage OoO | 4-wide | 4.5× vs A53 | Good | Edge AI compute |

# Practical guidance — which ARM core for which embedded application

- Battery-powered embedded Linux device → Cortex-A55 based SoC
- Industrial HMI with Qt UI → Cortex-A53 or A55 quad-core
- Edge AI inference platform → Cortex-A76 + A55 big.LITTLE
- Automotive ADAS domain controller → Cortex-A78 cluster with NPU
- Cost-sensitive IoT gateway → Cortex-A53 dual-core

*"Clock speed comparison between Cortex-M and Cortex-A is meaningless — a Cortex-A55 at 1.2 GHz completes more work per second than a Cortex-M85 at 1 GHz due to out-of-order execution, wider issue, and deeper cache hierarchy"*

# Key x86 Embedded Processors — Comparison Table

| Processor | Cores/ Threads | TDP | GPU | Key Feature | Primary Application |
|---|---|---|---|---|---|
| **Intel Atom x6000E (Elkhart Lake)** | 4C/4T | 4.5–12W | Intel UHD | Ultra-low power x86 | IoT gateway, fanless HMI |
| **Intel Core i5/i7 (Alder Lake-PS)** | 4–8C | 15–28W | Intel Iris Xe | High performance embedded | Medical imaging, industrial PC |
| **Intel Core Ultra 5/7 (Meteor Lake-PS)** | 6–16C | 15–45W | Intel Arc + NPU | Built-in AI NPU | Edge AI, smart vision systems |
| **AMD Ryzen Embedded R1000** | 2C/4T | 12–25W | Radeon Vega 3 | Cost-optimized | Digital signage, thin clients |
| **AMD Ryzen Embedded V2000** | 4–8C | 10–25W | Radeon RX Vega 7 | Balanced perf/power | Industrial HMI, medical |
| **AMD Ryzen Embedded V3000** | 4–8C | 15–54W | Radeon 610M | Maximum compute | High-end vision, simulation |

# x86 Vs ARM in embedded — Decision criteria

- Existing validated software stack must run unchanged → x86
- Windows IoT Enterprise is a hard requirement → x86
- Standard PCIe peripherals must be used without driver porting → x86
- Multi-threaded compute exceeds ARM embedded SoC capability → x86

- New design, power-constrained, ARM ecosystem sufficient → ARM MPU
- Battery or ultra-low power operation required — ARM wins decisively
- BOM cost is critical — x86 modules significantly more expensive than ARM SoCs
- Boot time under 2 seconds required — ARM embedded Linux boots faster
- High volume consumer product — ARM licensing economics more favorable

# Key RISC-V Application Processor Cores

| Processor | Core | Cores | Clock | MMU | Linux Ready | Key Feature | Status |
|---|---|---|---|---|---|---|---|
| SiFive U74 | RV64GCB | 1–4 | 1.5 GHz | Sv39 | Yes — mature | In-order efficiency | Production |
| SiFive P670 | RV64GCB | 1–8 | 3.0 GHz | Sv48 | Yes | OoO performance | Early sampling |
| T-Head TH1520 | C910 RV64GCV | 4 | 2.0 GHz | Sv39 | Yes — active community | GPU integrated | Production |
| SpacemiT K1 | RV64GCVB | 8 | 1.6 GHz | Sv39 | Yes | AI NPU 2 TOPS | Production |
| Starfive JH7110 | U74 RV64GC | 4+1 | 1.5 GHz | Sv39 | Yes — VisionFive 2 | Most documented | Production |

# Popular System On Modules (SOM) in the market

# Popular MPU SoCs in Embedded Designs #1

## SoC 1 — NXP i.MX 8M Plus

- Core cluster: 4× Cortex-A53 @ 1.8 GHz + 1× Cortex-M7 @ 800 MHz
- RAM: LPDDR4 up to 6 GB
- Key accelerators: 2.3 TOPS NPU, HiFi4 DSP, 4K H.265 VPU, Vivante GPU
- Connectivity: Dual GbE, PCIe, USB 3.0, MIPI CSI/DSI
- Real-time companion: Cortex-M7 runs FreeRTOS alongside Linux on Cortex-A53
- Primary applications: Industrial HMI, smart cameras, edge AI gateways, medical devices
- OS support: Linux (Yocto, Buildroot), Android
- Why it wins: Best-in-class industrial ecosystem, NXP long-term supply guarantee, rich BSP

# Popular MPU SoCs in Embedded Designs #2

## SoC 2 — Raspberry Pi BCM2711 (CM4)

- Core cluster: 4× Cortex-A72 @ 1.5 GHz
- RAM: LPDDR4 up to 8 GB
- Key accelerators: VideoCore VI GPU, H.265 4K decode
- Connectivity: GbE, PCIe Gen 2, USB 3.0, MIPI CSI/DSI, WiFi/BT
- Form factor: Compute Module 4 — 55×40mm — designed for product integration
- Primary applications: Industrial gateways, digital signage, kiosks, maker-to-product designs
- OS support: Raspberry Pi OS (Debian), Ubuntu, many Linux distributions
- Why it wins: Largest community, lowest software bring-up cost, widest peripheral support

# Popular MPU SoCs in Embedded Designs #3

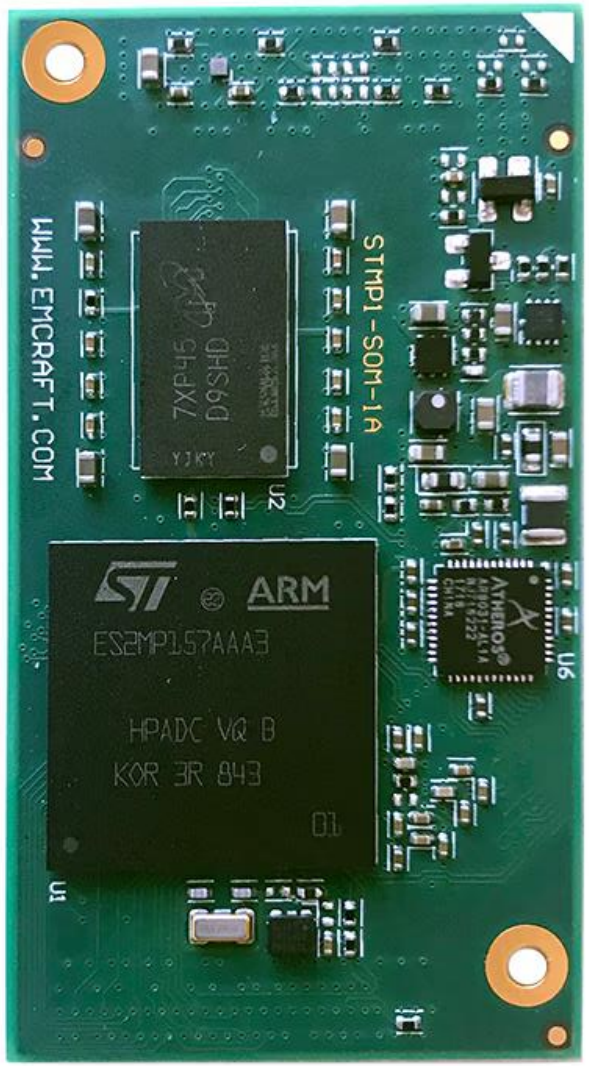
## SoC 3 — NVIDIA Jetson Orin NX

- Core cluster: 6–8× Cortex-A78AE @ 2.0 GHz
- RAM: LPDDR5 up to 16 GB
- Key accelerators: Ampere GPU (1024 CUDA cores), 2× NVDLA, PVA — up to 100 TOPS total
- Connectivity: PCIe Gen4, USB 3.2, 10GbE, MIPI CSI (up to 8 cameras)
- Primary applications: Edge AI inference, autonomous robots, ADAS, smart cameras
- OS support: JetPack (Ubuntu-based Linux), CUDA, TensorRT, DeepStream SDK
- Why it wins: Unmatched AI inference performance at the edge — no embedded competitor close

# Popular MPU SoCs in Embedded Designs #4

## SoC 4 — STM32MP1 (STMicroelectronics)

- Core cluster: 2× Cortex-A7 @ 650 MHz + 1× Cortex-M 209 MHz
- RAM: DDR3/DDR3L up to 1 GB
- Key accelerators: 3D GPU (OpenGL ES 2.0), display controller
- Connectivity: GbE, USB OTG, CAN, SDIO, MIPI DSI
- Primary applications: Entry-level industrial HMI, smart home controllers, building automation
- OS support: Linux (STM32MP1 OpenSTLinux), FreeRTC on M4 core simultaneously
- Why it wins: STM32 ecosystem familiarity — MCU engineers transition to MPU without changing vendor



# Popular MPU SoCs in Embedded Designs #5

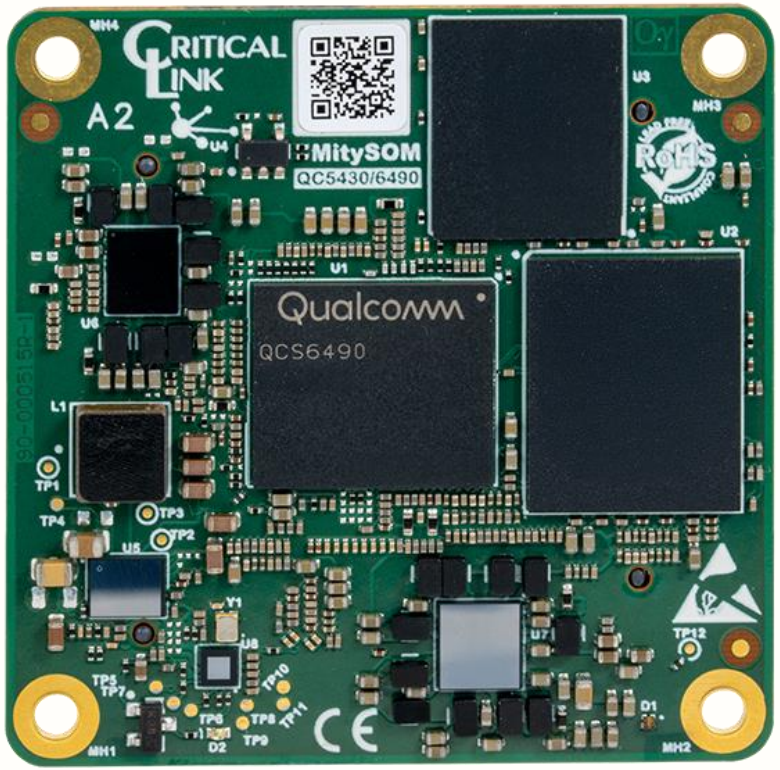
## SoC 5 — Rockchip RK3588

- Core cluster: 4× Cortex-A76 @ 2.4 GHz + 4× Cortex-A55 @ 1.8 GHz
- RAM: LPDDR4/5 up to 32 GB
- Key accelerators: Mali-G610 GPU, 6 TOPS NPU, 8K VPU
- Connectivity: PCIe Gen3, USB 3.1, 10GbE option, HDMI 2.1, MIPI CSI/DSI
- Primary applications: High-end edge AI, NVR/DVR systems, ARM servers, digital signage
- OS support: Linux (Armbian, Debian, Ubuntu), Android 12
- Why it wins: Highest compute density per dollar in embedded ARM SoC market today

# Popular MPU SoCs in Embedded Designs #6

## SoC 6 — Qualcomm QCS6490 (Snapdragon Embedded)

- Core cluster: 4× Kryo Gold (Cortex-A78) + 4× Kryo Silver (Cortex-A55) @ up to 2.7 GHz
- RAM: LPDDR5 up to 12 GB
- Key accelerators: Adreno 643 GPU, Hexagon DSP, Qualcomm AI Engine — 12 TOPS
- Connectivity: WiFi 6E, BT 5.3, 5G modem option, USB 3.1, DisplayPort
- Primary applications: Smart retail, industrial handhelds, ruggedized tablets, robotics
- OS support: Android, Linux (Qualcomm Linux), Windows IoT
- Why it wins: Best wireless connectivity stack, Android ecosystem, camera ISP quality



# Popular MPU SOM suppliers

| Vendor | SOM Portfolio Size | BSP Quality (Yocto/Linux) | India Support | Ecosystem | Best Use Case |
|---|---|---|---|---|---|
| **Toradex** | High | ★★★★★ | ★★★★★ | ★★★★★ | Production-ready products |
| **Variscite** | Very High | ★★★★★ | ★★ | ★★★★ | Industrial / long lifecycle |
| **Digi** | Medium | ★★★★ | ★★★ | ★★★★★ | IoT + cloud devices |
| **Phytec** | Medium | ★★★★★ | ★★ | ★★★★ | Custom engineering |
| **Congatec** | High (esp. x86) | ★★★★ | ★★ | ★★★★ | COM Express / x86 |
| **iWave** | Medium | ★★★ | ★★★★★ | ★★★ | India-focused designs |
| **VVDN** | Growing | ★★★ | ★★★★★ | ★★★ | ODM / AI edge systems |

# SoC recommendation for specific applications

| Application | Recommended SoC |
| --- | --- |
| Industrial reliability + long supply life | NXP i.MX 8M Plus |
| Lowest software bring-up cost | Raspberry Pi CM4 |
| Maximum AI inference performance | NVIDIA Jetson Orin NX |
| Android + wireless connectivity | Qualcomm QCS6490 |
| MCU-to-MPU transition, STM32 familiarity | STM32MP1 |
| Maximum compute per dollar | Rockchip RK3588 |
| Real-time + Linux simultaneously | NXP i.MX 8M Plus or STM32MP1 |

# How to select an MPU?

# The seven criteria for MPU design: #1

## Criterion 1 — Compute Performance Requirement

- Entry level: Cortex-A7/A53 class — simple Linux UI, basic networking, sensor aggregation
- Mid range: Cortex-A55/A72 class — 1080p video, moderate AI, multi-camera
- High performance: Cortex-A76/A78 class — 4K video, heavy AI inference, multi-threaded workloads
- Question to ask: What is the peak DMIPS or CoreMark requirement of your heaviest use case?

# The seven criteria for MPU design: #2

**Criterion 2 — AI / ML Inference Requirement**

- No AI → any MPU sufficient
- TinyML / keyword spotting → integrated NPU on i.MX 8M Plus (2.3 TOPS) sufficient
- Computer vision / object detection → NVIDIA Jetson Orin (100 TOPS) required
- Question to ask: What model size, what frame rate, what latency budget?

# The seven criteria for MPU design: #3

## Criterion 3 — Real-Time I/O Alongside Linux

- Linux alone cannot guarantee deterministic microsecond response times
  - Solution: Heterogeneous SoC with companion real-time core
- i.MX 8M Plus: Cortex-M7 runs FreeRTOS — handles motor control while A53 runs Linux
- STM32MP1: Cortex-M4 handles hard real-time — Cortex-A7 handles UI and connectivity
- Question to ask: Does any subsystem require sub-100 microsecond deterministic response?

# The seven criteria for MPU design: #4

**Criterion 4 — Operating System Requirement**

- Linux (Yocto/Buildroot) → all MPUs supported — most flexible choice
- Android → Qualcomm Snapdragon or Rockchip — best Android BSP ecosystem
- Windows IoT Enterprise → x86 Intel/AMD embedded only
- Real-time Linux (PREEMPT-RT) → any MPU with mainline Linux support
- Question to ask: What OS does your software team have expertise in?

List of OS for Embedded: Embedded Linux /Android Things / Embedded Android / Windows IoT/ QNX Neutrino / VxWorks

# The seven criteria for MPU design: #5

## Criterion 5 — Connectivity & Peripheral Requirements

- Industrial protocols (CAN, Profinet, EtherCAT) → NXP i.MX or TI Sitara
- Wireless (WiFi 6E, 5G, BT 5.3) → Qualcomm Snapdragon
- PCIe expansion (FPGA, GPU, NVMe) → Rockchip RK3588 or x86
- Camera inputs (4–8 MIPI CSI) → NVIDIA Jetson or NXP i.MX 8M Plus
- Question to ask: List every external interface — does the SoC support all natively?

# The seven criteria for MPU design: #6

## Criterion 6 — Supply Chain & Longevity

- Consumer SoCs (Broadcom BCM2711) → 5–8 year availability typical
- Industrial grade MPUs (NXP i.MX 8M Plus) → 10–15 year longevity commitment
- Automotive grade (NXP S32G, Renesas R-Car) → AEC-Q100 qualified — 15+ years
- Question to ask: What is the product lifetime? Medical and industrial need 10+ year supply guarantee

# The seven criteria for MPU design: #7

## Criterion 7 — BSP & Software Ecosystem Maturity

- Yocto BSP quality varies dramatically between vendors
- NXP: Industry-leading BSP — regular updates — strong community
- Raspberry Pi: Largest community — fastest bring-up — consumer-grade longevity
- NVIDIA Jetson: JetPack SDK — best AI framework support — CUDA, TensorRT, DeepStream
- Qualcomm: Strong Android BSP — Linux BSP improving but historically weaker
- Question to ask: How long will BSP bring-up take? Who maintains it after product ships?

# MPU Selection Criteria Matrix

| Criterion | NXP i.MX 8M Plus | RPi CM4 | Jetson Orin NX | Snapdragon QCS6490 | STM32MP1 | RK3588 |
|---|---|---|---|---|---|---|
| Compute Performance | ★★★ | ★★★ | ★★★★★ | ★★★★ | ★★ | ★★★★★ |
| AI Inference | ★★★ | ★ | ★★★★★ | ★★★★ | ★ | ★★★ |
| Real-Time Companion Core | ✅ M7 | ❌ | ❌ | ❌ | ✅ M4 | ❌ |
| Linux Ecosystem | ★★★★★ | ★★★★★ | ★★★★★ | ★★★ | ★★★★ | ★★★★ |
| Android Support | ★★ | ★★ | ★★★ | ★★★★★ | ❌ | ★★★★ |
| Industrial Longevity | ★★★★★ | ★★★ | ★★★★ | ★★★ | ★★★★★ | ★★★ |
| BSP Maturity | ★★★★★ | ★★★★★ | ★★★★★ | ★★★ | ★★★★ | ★★★ |
| Cost | ★★★ | ★★★★★ | ★★ | ★★★ | ★★★★ | ★★★★★ |

# An example of Linux based development for Embedded applications

# Develop HMI application: Discovery kit with STM32MP135F MPU – Step 1 & 2

For the STM32MP135F-DK (based on STM32MP135F-DK), ST actually provides a very complete, production-grade ecosystem. The key is knowing which package/tool corresponds to which layer (bootloader, kernel, rootfs, HMI, etc.).ships? for quick start)

## 1. Start Here:
## ST's Linux Distribution : OpenSTLinux

What you get inside OpenSTLinux:

- Bootloader: U-Boot + TF-A
- Linux kernel (ST-maintained)
- Device tree for STM32MP13 boards
- Yocto layers (for customization)
- Prebuilt images (for quick start)

## 2. Bootloader (U-Boot) Customization

**What you customize:**

Boot source (SD / eMMC)
Environment variables (bootcmd)
Splash screen (for HMI branding)
DDR + low-level init (usually pre-done)

**How:**

Via Yocto (bitbake u-boot)Or manual build using ST instructions



Here is a great pool to dive into for mastering Embedded System Linux https://bootlin.com/docs/

# Develop HMI application: Discovery kit with STM32MP135F MPU – Step 3 & 4

## 3. Linux Kernel + Device Tree

**Source: ST kernel repo:**
https://github.com/STMicroelectronics/linux
**Device Tree (VERY IMPORTANT for HMI)**Location inside kernel:
arch/arm/boot/dts/stm32mp13*.dts
**What you customize:**
Display panel (DSI/LVDS/RGB); Touch controller (I2C/SPI); GPIOs, PWM, backlight; Audio, sensors
**Workflow:**
Modify .dts
Rebuild kernel via Yocto
Deploy .dtb

## 4. Root Filesystem Customization (Yocto)

**Build System: Yocto project**
**ST provides:**

- meta-st layers
- Predefined images:
  - core-image-minimal
  - core-image-weston (for GUI)

**You customize via:**

- local.conf
- bblayers.conf
- Custom layer (recommended)

**Add packages:**

- SSH, networking tools
- Graphics libs
- Your app

Build Command: bitbake core-image-weston

# Develop HMI application: Discovery kit with STM32MP135F MPU – Step 5 & 6

## 5. BSP Customization (Board Support Package)

In STM32MP1 world, BSP = combination of:

- U-Boot config
- Kernel config + Device Tree
- Yocto layers

**Where BSP lives:**

- Inside OpenSTLinux Yocto layers:
  - meta-st-stm32mp

**What you modify:**

- Board config (MACHINE)
- Kernel fragments
- Drivers
- Boot config

**Best practice:**

👉 Create your own custom Yocto layer meta-myboard/

## 6. HMI Development Stack


**Option 1 (Recommended): Qt + Wayland**

- Framework: Qt
- Display server: Wayland
- Compositor: Weston

ST provides:

- core-image-weston (ready to use)

**Option 2: Qt EGLFS (simpler)**

- No window manager; Full-screen UI only

**Option 3: LVGL**

- Lightweight; Less common on MPU (more MCU-focused)

**Where to get Qt:**

- Included via Yocto:
  - meta-qt5 or meta-qt6

**Development flow:**

- Develop on PC (Qt Creator)
- Cross-compile via Yocto SDK
- Deploy to board

# Develop HMI application: Discovery kit with STM32MP135F MPU – Step 7 & 8

## 7. Tools You'll Use Daily

**From ST:**

**STM32CubeProgrammer → flash images**

**STM32CubeIDE → optional**

**Linux tools:**

- **bitbake**
- **Devtool**
- **Menuconfig**
- **evtest, fbtest, modetest**


## 8. Recommended Learning Path (Very Practical)

If you're starting fresh, follow this order:

1) Flash **prebuilt OpenSTLinux image** → verify board works
2) Build image using Yocto (no changes)
3) Modify: Device tree (LED / GPIO first)
4) Enable display + touch
5) Switch to core-image-weston
6) Run Weston demo
7) Add Qt
8) Build simple HMI app
9) Auto-start app on boot

# Enabling Real Time Linux

**What "Real-Time Linux" Means**

Standard Linux is **not fully deterministic**:

- Scheduling latency can vary
- Interrupts may be delayed

Real-time Linux (PREEMPT_RT):

- Makes the kernel **fully preemptible**
- Reduces latency (often < 100 µs)
- Turns interrupts into schedulable threads

**NOTE:**

Linux (even RT) = soft real-time

MCU / RTOS = hard real-time (often < 10 µs)

**To enable real-time Linux:**

- Get ST kernel (OpenSTLinux)
- Apply PREEMPT_RT patch
- Enable CONFIG_PREEMPT_RT
- Integrate via Yocto
- Rebuild + flash
- Tune userspace + CPU

# When Real Time Linux is not enough.. - Heterogeneous SoCs and AMP

## STM32MP Series with A-core + M-core (Heterogeneous SoCs)

## Key features:

- Dual-core Cortex-A7 (runs Linux)
- Cortex-M4 (runs firmware / RTOS)
- Hardware IPC (RPMsg, mailbox)
- Designed for **industrial + HMI + control**

## How A7, M4 Binaries stored?

Typical memory layout (eMMC/SD)

- Bootloader (TF-A + U-Boot)
- Kernel + Device Tree
- RootFS (Linux)
- M4 firmware (optional partition or /lib/firmware)

## Code Execution Flow:

- Linux boots normally
- Kernel loads M4 firmware from filesystem
- Starts M4 core