Asymmetric Multiprocessing and Multiprocessor Systems on Chip

Real-Time Industrial Systems

Marcello Cinque

Roadmap

- AMP and MPSoCs for real-time: why?
- AMP and MPSoCs basics
- OpenAMP intro

- References:
  - Notes on “MPSoC and Virtualization for Real-Time and Safety Critical Systems” by Daniele Ottaviano
  - Arm Cortex-R: https://developer.arm.com/ip-products/processors/cortex-r
  - OpenAMP: https://www.openampproject.org

Asymmetric multiprocessing systems

- Systems with heterogeneous unrelated cores, not treated equally
  - both at hardware level
    - different ISAs, speed, cache architectures, interconnection, I/O access, etc.
  - and at software level
    - different runtimes and/or operating systems, no peer execution of kernels, etc.

- It was the first form of multiprocessing, before the advent of SMP
  - Mainly due to the complexity of porting existing OSes on multiprocessors
- It is re-gaining interest today in modern industrial mixed-criticality systems… why?

AMP and real-time mixed-criticality

- The challenge of real-time mixed-criticality system (MCS) is to face contrasting requirements on:
  - Time, space and fault isolation
  - Resource sharing
- SMPs are designed for sharing
  - Shared last level cache
  - Shared access to memory
  - Shared interconnect between cores
  - Shared access to peripherals
- Good for performance and SMP OS design
- Bad for isolation and the differentiated requirements typical of MCS!

AMP and real-time mixed-criticality

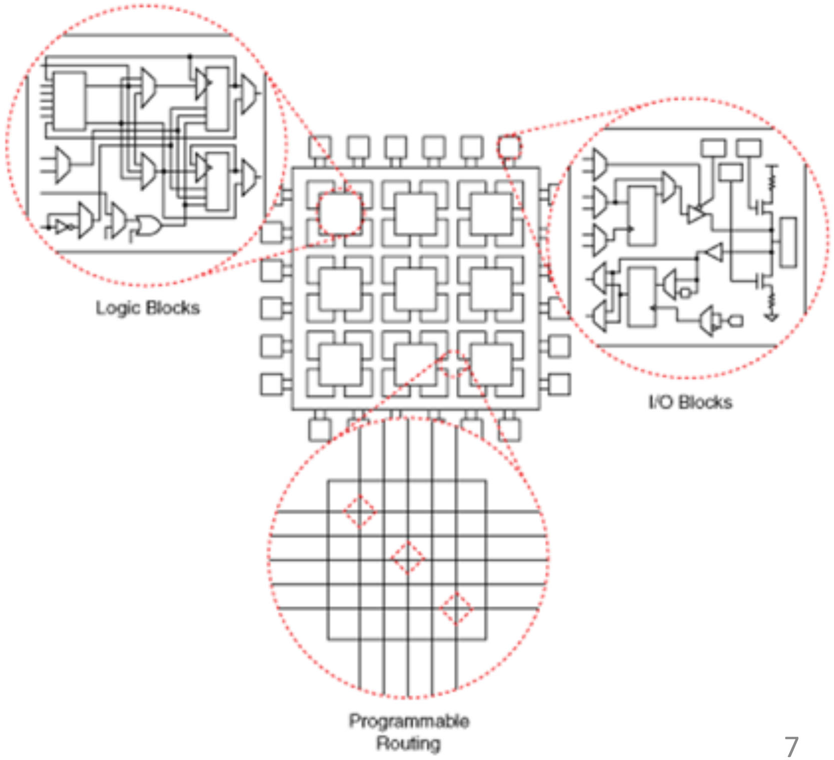
- So, why do not use heterogenous processors for differentiated tasks?
  - APUs (Application Processing Units) for performance-driven non-real time general purpose computing
  - GPUs (Graphical Processing Units) for multimedia, video rendering and AI
  - RPUs (Real-time Processing Units) for predictable computing
  - Secure core, for trusted execution environments
  - PL (Programmable Logic) for custom high-performance and/or highly deterministic tasks or for soft cores
- With different caching mechanisms and many I/O interfaces, driven by applications

GPUs

- Special many-core processors that make extensive use of parallelism techniques adopted by SIMD (Single Instruction Multiple Data) and heavily oriented to vector calculations.
- Traditionally designed for graphic elaboration, nowadays GPUs are used also to accelerate general-purpose calculations
  - On cloud servers, the parallelism is exploited to train machine learning models, e.g., deep neural networks, which are intrinsically parallel systems
  - The same can be done on embedded mixed-criticality systems, to run machine learning tasks in parallel with time-critical tasks on RPUs!

PL: Field Programmable Gate Arrays (FPGAs)

- Integrated circuits that consists of internal hardware blocks with user-programmable interconnects to customize operation for a specific application.
- Can be used to implement custom functions in hardware that would run inefficiently on a CPU
  - high-speed packet processing (networking)
  - cryptography
  - signal and image processing
  - high speed search for data centers
  - machine learning
  - deterministic calculations (hw-clock driven)



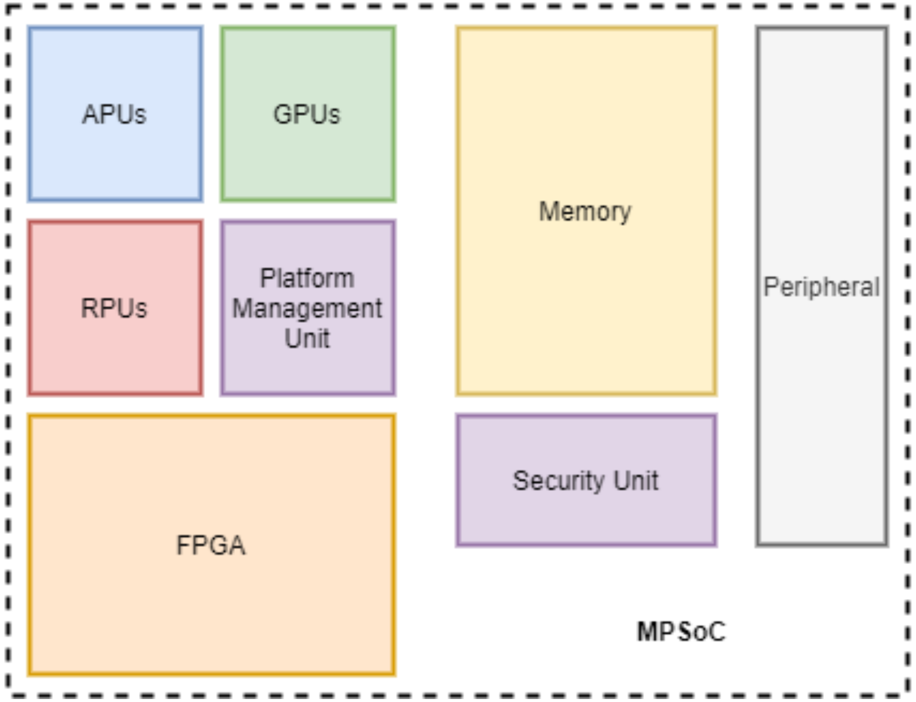
RPUs

- Processors specifically designed for real-time performance
- Regarding **processing**:
  - Instructions with known WCET (e.g., “in order” superscalar pipelines)
  - Make long instructions (load/store) interruptible
- Regarding **memory**:
  - Traditional cache memories can be disabled
  - Tighly Coupled Memories (TCMs) can be used: fast memories assigned to processors for low-latency (but TCM-aware) applications
  - MPU instead of MMU: for static memory partitioning

    Memory Protectiong Unit instead of Memory Management Unit
- Regarding **I/O**:
  - Low latency interrupt handling with dedicated interrupt controller

Multiprocessor Systems on Chip (MPSoCs)

- Bringing all together on the same chip, driven by today's chips high-level of integration, low power capabilities and industry needs to reduce TCO and hardware footprint
- Non traditional cores (GPUs, RPUs, PLs) can be regarded as "accelerators" by APUs and managed by the OS run on APUs in an asymmetric way



An example of ARM-based MPSoC: Xilinx Zynq

- Zynq-7000 SoC (2011)
  - Processing System
    - Application Processor Unit (APU)
    - Interconnects and memory interfaces (CAN, I2C, USB, ..)
    - I/O Peripherals
  - Programmable Logic (used for custom as well as programmable cores like Microblaze)
  - PS Frequency: up to 1GHz; PL Frequency up to 741GHz
- Zynq MPSoC (2015)
  - Dual or Quad APU, Dual RPU (opt. an Arm Mali GPU), General Purpose and Video domains
  - PS: Arm Cortex-A53 for APU, Cortex-R5 for RPU, VCU IP supporting H.265 and H.264
  - PL: Xilinx Ultrascale+, up to 1M+ Flip-Flops and 500k+ LUTs
  - PCIe Gen2, USB3.0, SATA 3.1, DisplayPort, Gigabit Ethernet, ..
  - Configuration and Security Unit, Platform Management Unit, ..
  - PS Frequency: up to 1.5GHz; PL Frequency up to 891GHz
- Application domains:
  - mobile base-station signal processing, video compression/decompression, broadcast camera equipment, navigation and radar systems, high speed switching, routing infrastructure for data centres, Advanced Driver Assistance Systems (ADAS), and even big data analytics, ..

Arm subsystem within the Zynq MPSoC

- Arm Intellectual Property (IP) licensed to Original Equipment Manufacturers (OEMs) such as Xilinx:
  - Cortex-A53 and Cortex-R5 processor + additional Arm IPs in the MPSoC
  - can be partly customized by the OEM

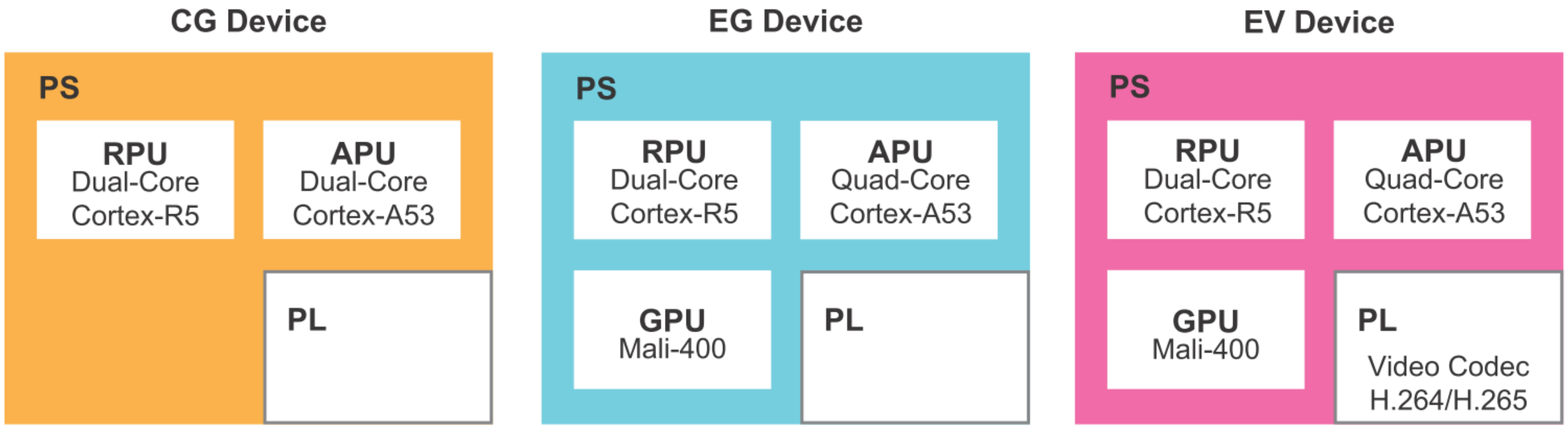
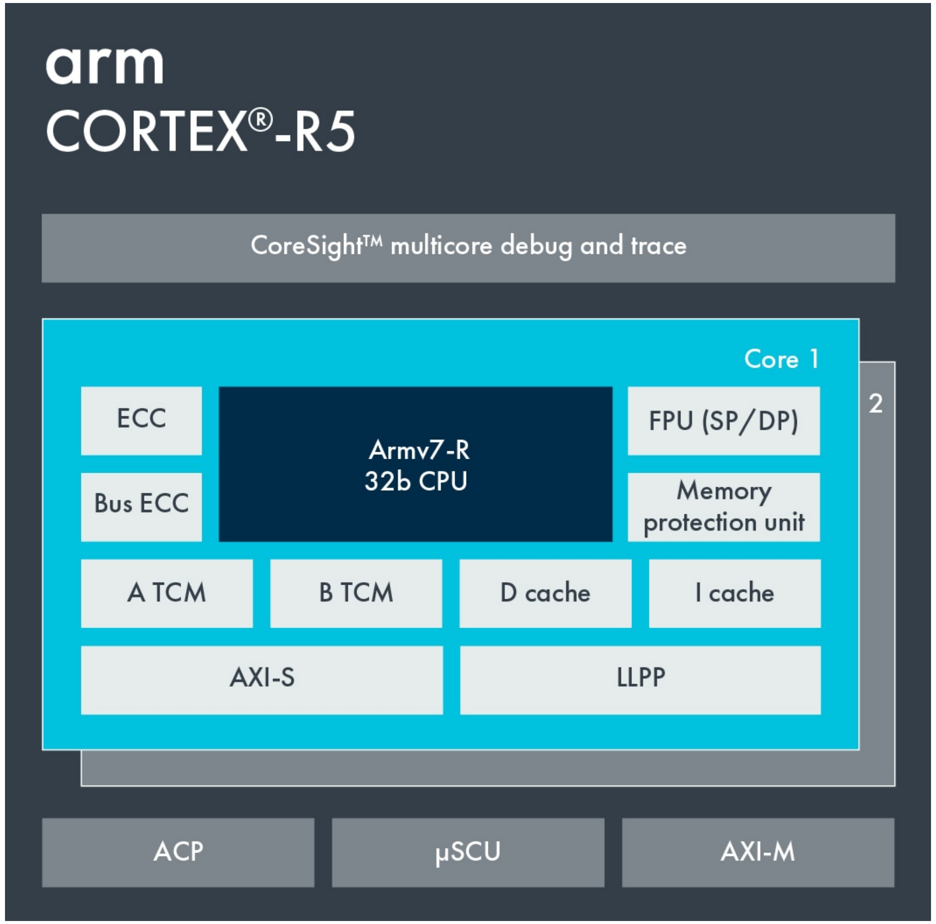


Image from
C. H Louise, N. David, R. Craig, *Exploring Zynq MPSoC: With PYNQ and Machine Learning Applications*, Strathclyde Academic Media, 2019

ARM Cortex-R5



ECC= Error Correction Code

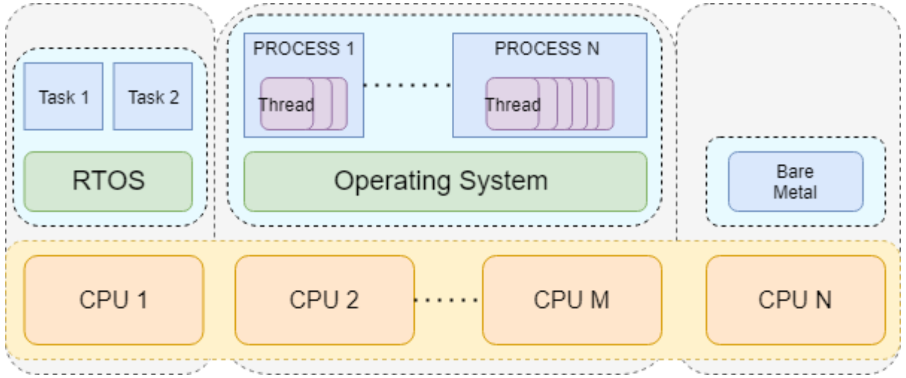
- Caches and TCMs protected with ECC
- LLPP: Low-Latency Peripherial Port to integrate latency-sensitive peripherals more tightly with the processor.
- Dual-core processor configurations:
  - **lock-step mode (or *safety mode*)** : redundant fault tolerant/fault detecting execution mode, for dependable systems
  - **split mode (or performance mode)** : cores running independently, each executing its own program with its own bus interfaces, interrupts, and so on.

Cortex-R Safety mode

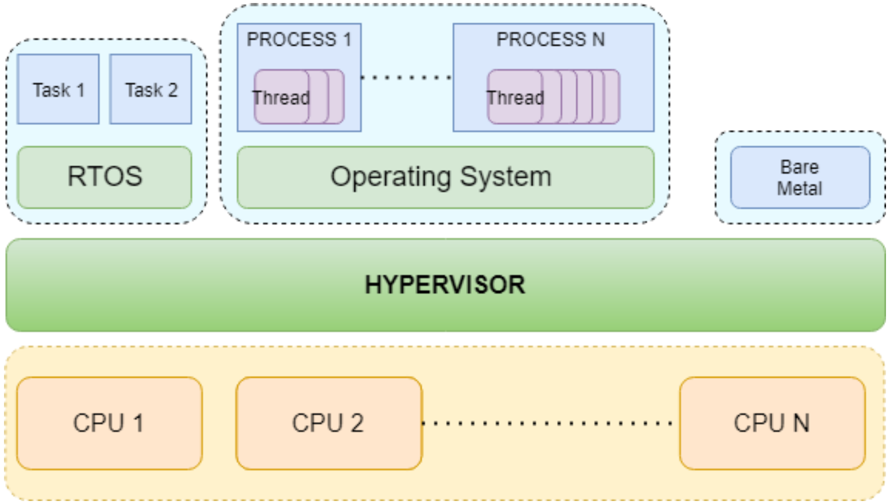
- The dual cores execute the same code in parallel, with a couple of cycles of differences between each other
- The outputs from the cores are compared every cycle, and if at any point they differ, an error is signaled
- The memories are protected by using ECC schemes (capable of detecting double bit errors, or correcting single bit errors)
- The aim is to protect the system from transient errors, for instance bit-flips caused by particle strikes (especially useful in space applications)

AMP approaches

- Unsupervised AMP:
  - Each core has its own software stack
  - there is no central software that manages and coordinates the cores
    - Complications with shared resources, interrupt management, memory access, etc.
  - Applications must be aware of AMP

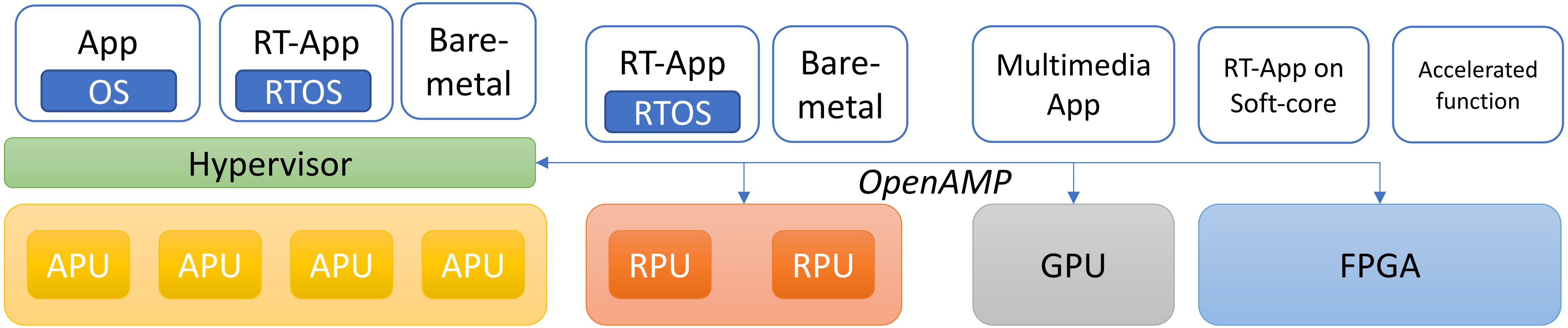


- Supervised AMP:
  - All processors run independently with their own software stack,
  - there is a centralized software layer (a **hypervisor**) that coordinates the cores
  - Applications are not aware of AMP
  - More heavyweight



AMP on MPSoC

- The use of heterogeneous processors (with different ISAs and not all equipped with hw virtualization) makes it impossible to run a hypervisor on all cores
- A possible approach is to mix supervised and unsupervised AMP
  - Supervised AMP with hypervisor and VMs on APUs
  - Unsupervised AMP for the other cores (or *accelerators*)
- The hypervisor can perform task assignment on accelerators, manage their shared use from VMs, and communicate with them using **OpenAMP**



OpenAMP

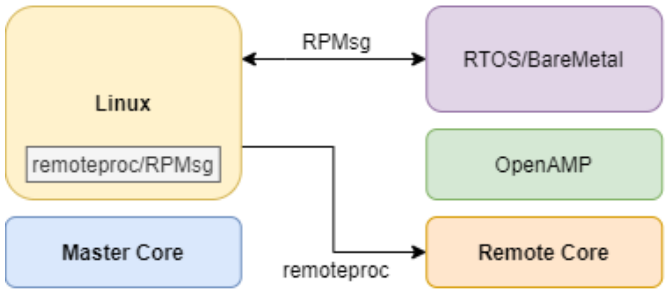
“OpenAMP (Open Asymmetric Multi-Processing) seeks to standardize the interactions between operating environments in a heterogeneous embedded system through open source solutions for Asymmetric MultiProcessing (AMP)”

(www.openampproject.org)

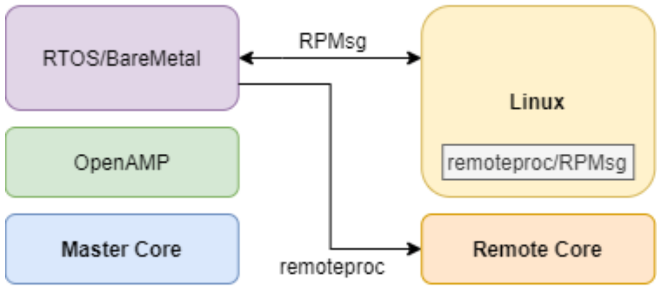
- A framework based on open source software components to simplify the development of applications on AMP systems
- Used to perform:
  - **Lifecycle management (LCM)** of cores and applications
  - **Inter-process communication (IPC)** between tasks running on heterogeneous processors
- Relies on two key technologies: **`remoteproc`** for LCM and **`rpmsg`** for IPC
  - Preexisting packages in Linux, adapted to be run from an RTOS or bare-metal

Maser/Remote cores configurations

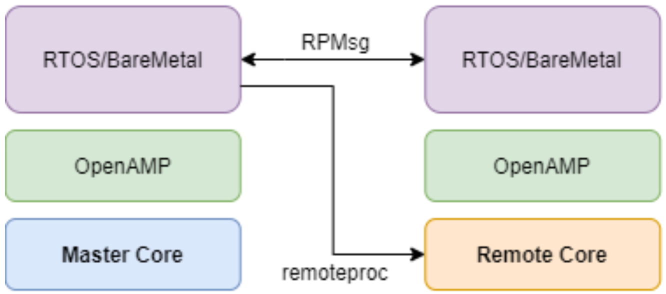
- OpenAMP identifies a Master core and one or more Remote cores
- Multiple configurations are possible



Linux Master (default)
RTOS/BareMetal Remote



RTOS/BareMetal Master
Linux Remote

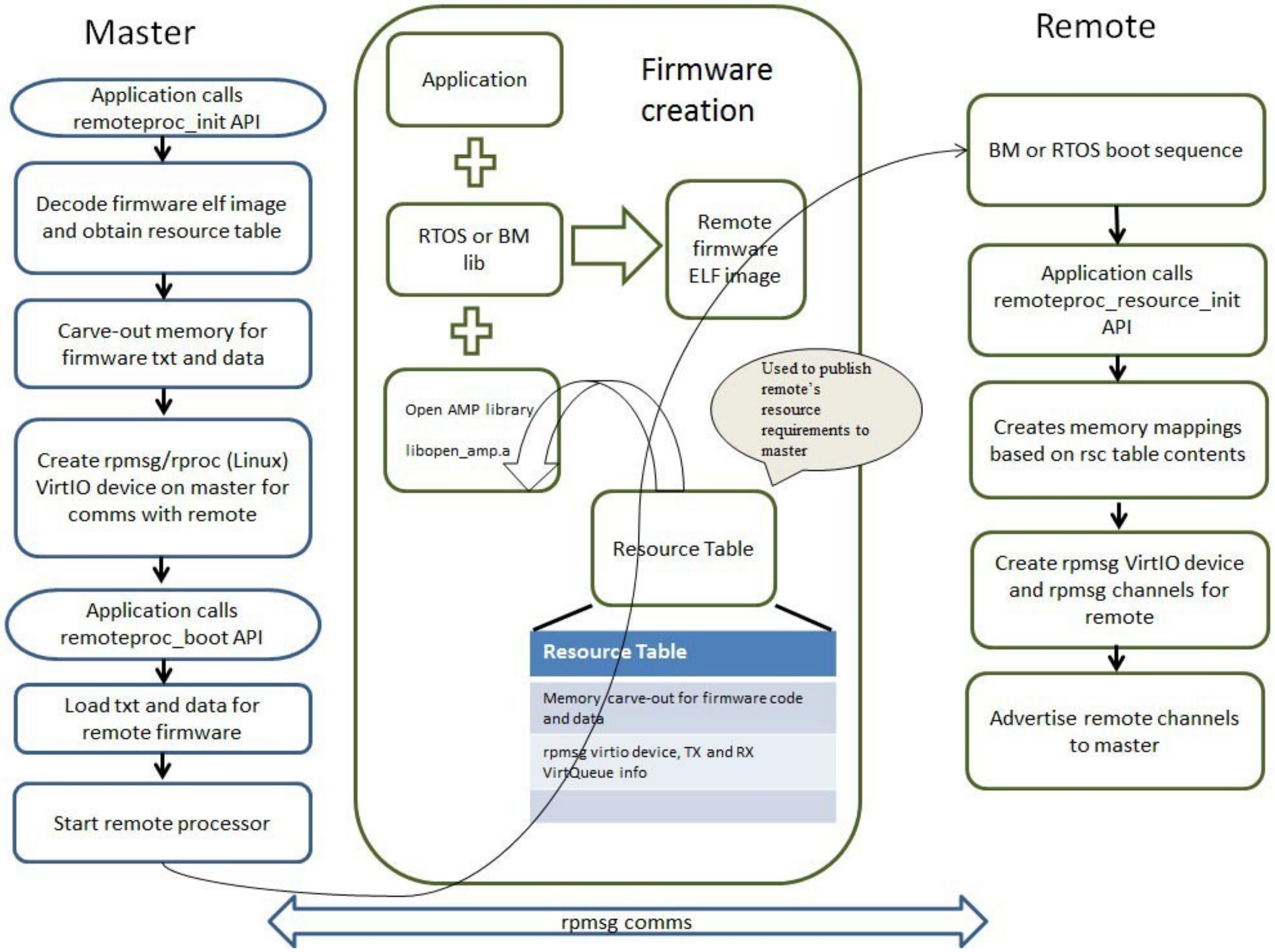


RTOS/BareMetal Master
and Remote

LCM with `remoteproc`

- It provides five essential functions:
  - allow the master software applications to load the code and data sections of the remote firmware image to appropriate locations in memory for in-place execution (**`remoteproc_init`** API)
  - establish RPMsg communication channels for run-time communications with the remote context
  - allows the remote applications to seamlessly initialize the remoteproc system on the remote side and establish communication channels with the master context (**`remoteproc_resource_init`** API)
  - release the remote processor from reset to start execution of the remote firmware (**`remoteproc_boot`** API)
  - shut down the remote software context and processor when its services are not needed (**`remoteproc_shutdown`** and **`remoteproc_resource_deinit`** APIs);

`remoteproc` API usage

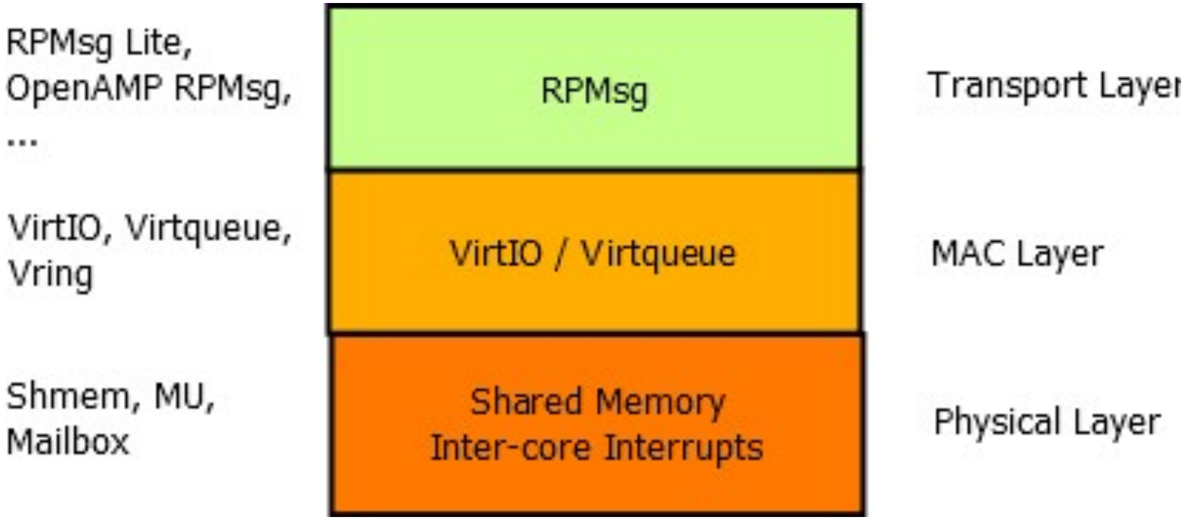


Resource table

- The **resource table** data structure is addedd to the Remote core firmware and contains entries that define
  - the system resources required by the remote firmware (for example, contiguous memory carve-outs required by the remote firmware’s code and data sections),
  - features/functionality supported by the remote firmware (like virtio devices and their configuration information required for RPMsg-based IPC).

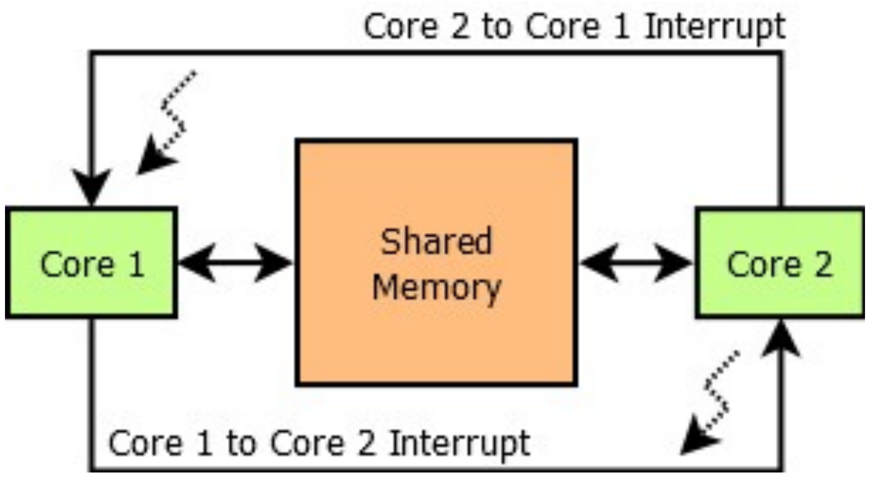
RPMsg

- A protocol to standardize to communication between cores, implementing a networking-like communication stack
  - Transport, MAC, and Physical layers



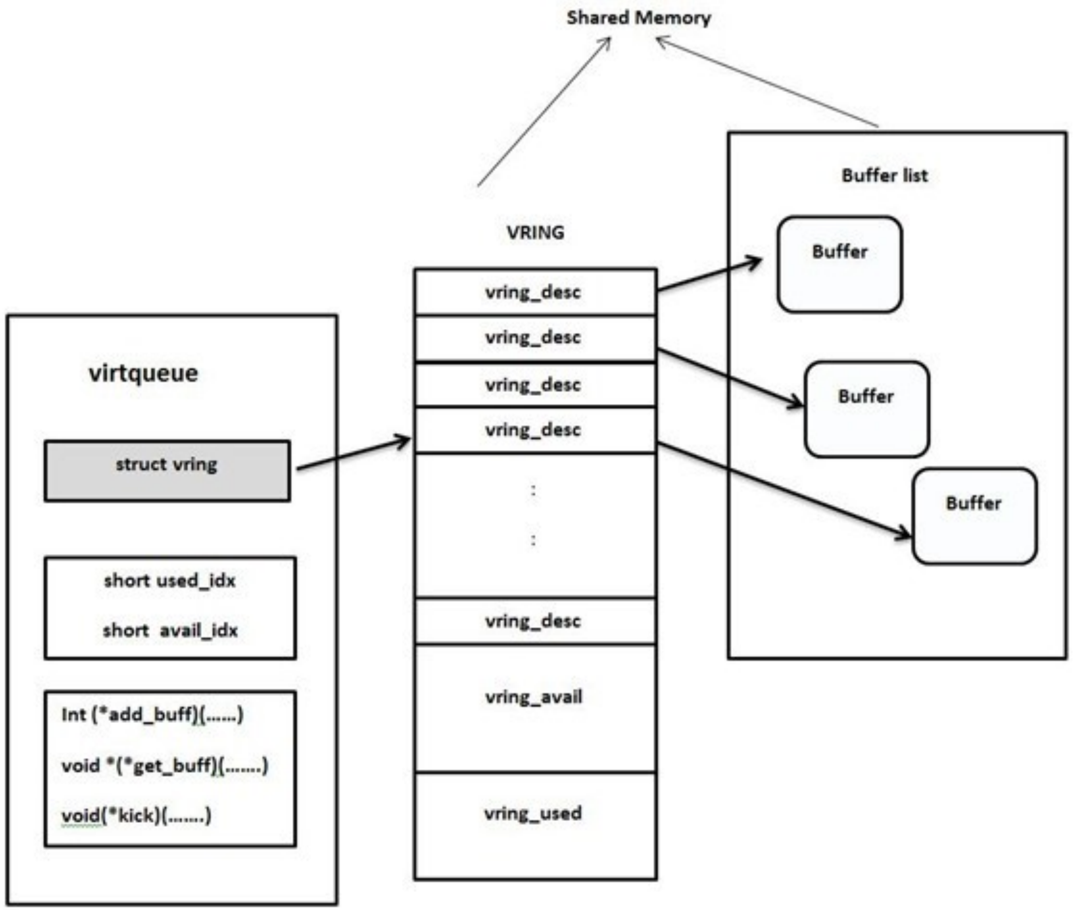
Physical layer

- The ”physical” communication media is a shared memory region dedicated to inter-core communication
- Inter-core interrupts used to signal the presence of data
- No synchronization implemented at this layer (left to MAC)
- Relies on the `libmetal` library, that can be used by OS or bare-metal



Media Access Control Layer: VirtIO

- Based on **VirtIO**, an abstraction mechanism originally developed for I/O para-virtualization of Linux-based guests
- In this way the shared memory is abstracted as a virtual I/O device
- The communication and synchronization is performed trough ring buffers with a queuing mechanism (`virtqueue` and `vring` structures implemented by VirtIO)



Transport layer: RPMsg

- Provides transport-layer APIs (e-g., `rmpsg_send()`, `rpmsg_receive()`, …)
- Abstracts the message structure with header and payload
- Introduce the notions of **channels** and **endpoints**
  - One channel per abstracted core
  - Multiple endpoints for each channel