OpenAMP – Intro

Feb 2016

Tomas Evensen, Xilinx

Intro

With todays sophisticated SoCs there is often a need to integrate multiple runtime environments with multiple operating systems. This raises a lot of issues, such as:

- Lifecycle issues
  - Boot order, start one OS from another, tear down OS, reboot, …
- Communication
  - Message passing, …
- Resource handling
  - Memory, devices, interrupts, …
- Power management
- …

OpenAMP aims to address these and other issues in a standardized way, both through an open source project and through standardization by MCA.

What is Needed to Be Able to Mix OSes?

- A standard “protocol” so OSes can interact without dependencies
  - On a given HW, any OS can interact with any other OS
    - Without special adaptation
  - On a shared memory system this is a set of data structures and conventions
    - I.e. the ring buffers in *virtio*
- A standard set of APIs for OS interactions
  - A low level API that abstracts underlying OS and HW
  - A set of lifecycle APIs
  - Messaging APIs
  - Other potential features
    - Proxy capabilities to make remote OS look like Linux process
    - Remote procedure calls, power management, device configuration, debug, …
- Upstream Linux support for protocol and APIs
  - Linux is increasingly becoming the main OS in a multi-OS system
- Open Source implementation
  - Quickest way to adoption
  - Standardization by reference implementation

What is OpenAMP?

- OpenAMP standardizes how Operating Systems interact
  - In particular between Linux and RTOS/bare-metal
  - In particular in a multicore heterogeneous systems
  - Includes:
    - Shared memory protocol for OS interactions (virtio)
    - Lifecycle APIs to start/stop/… other OSes (rproc)
    - Communication APIs to share data (rpmsg)
    - More to come
- Both a standardization effort and an open source project
- Guiding principles
  - Open Source implementations for Linux and RTOSes
  - Prototype and prove in open source before standardizing
  - Business friendly APIs and implementations to allow proprietary solutions

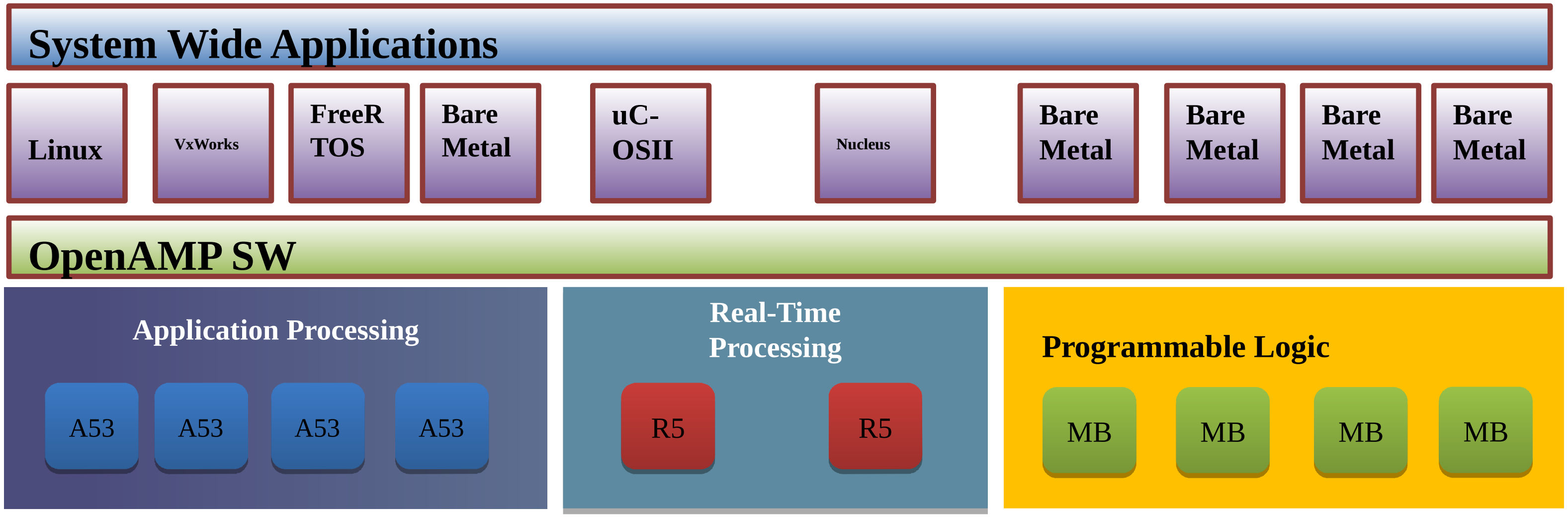
Open Source and Standardization

- Same players but different focus
  - Value in separating standardization and implementation
- Open Source project
  - Focus on implementation and testing new ideas
  - Today driven by Mentor Graphics and Xilinx
    - Also used by Freescale, Wind River and Micrium
  - Based on existing open source technologies in Linux
    - Shared memory protocol – virtio, Lifecycle APIs – remoteproc, Messaging – rpmsg
  - Clean-room implementation for non-Linux kernel code
    - Multiple licenses (GPL for Linux kernel, BSD for rest)
  - Proxy technologies to emulate Linux processes
  - More in future, please join!
- MCA OpenAMP Working Group
  - Focus on formally standardizing APIs used by open source project
    - Allows for proprietary solutions like hypervisors, certified systems, commercial OSes, …
  - Discuss future directions
    - Implement in open source – standardize when proven

Working Group Mission and Structure

- Mission statement (strawman):
  - “To formally standardize the interactions between Operating Systems in an Asymmetric Multiprocessing (AMP) multicore environment by adopting APIs used in Open Source solutions.”
- Structure:
  - Chair: Tomas Evensen – Xilinx
  - Vice Chair: Felix Baum – Mentor
  - Secretary: Tina Hutchens – Xilinx
  - Participants (Feb 2016) – Xilinx, Mentor, Feescale/NXP, TI, Wind River, Micrium, Express Logic, Airbus, Siemens
- Meetings:
  - Weekly meetings Thu 9 AM Pacific Time

OpenAMP SW Simplifies Heterogeneous Systems

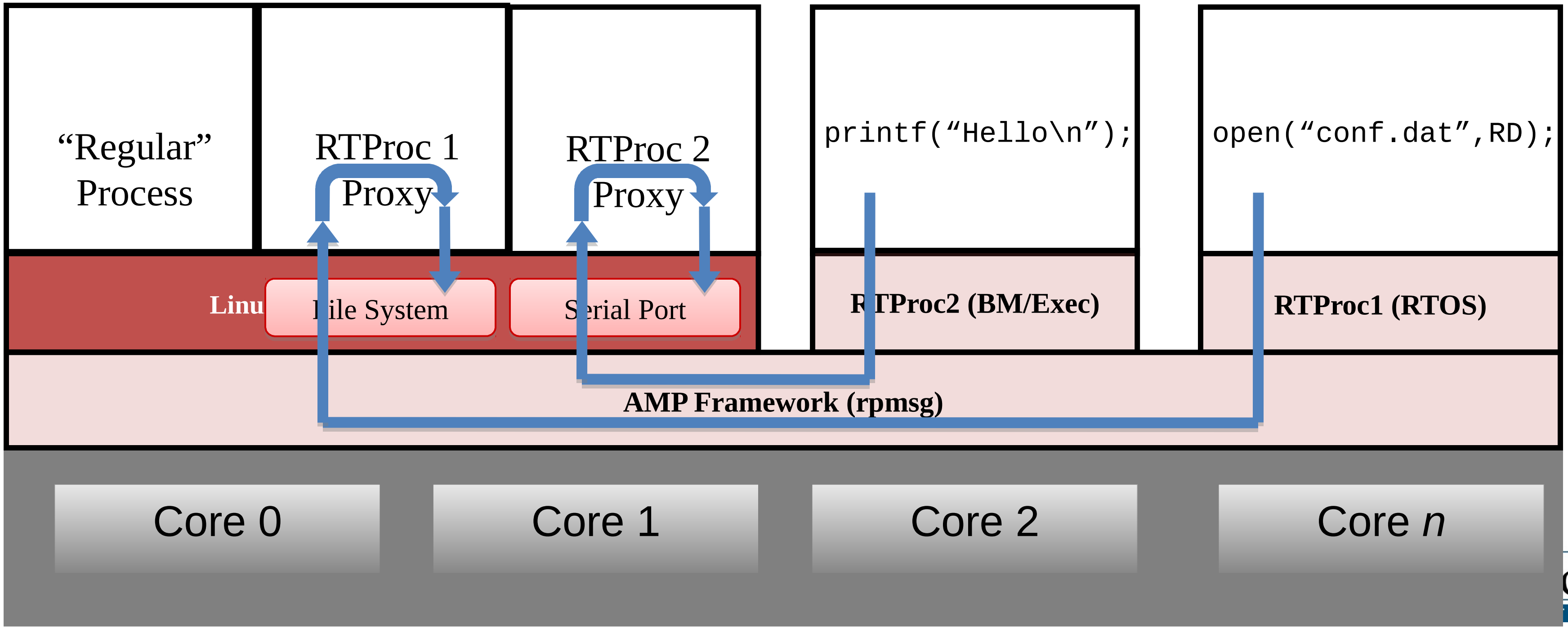


- Provides a Layer for Applications
  - Standard API's that allow applications to be ported across processors and operating systems, including hypervisors
- System Development
  - Provides a wide rage of capabilities needed to deploy applications across asymmetric computing elements

- Inter-OS & Inter Processor Communication
  - Send messages back and forth
- OS Management
  - Provides booting/rebooting of processors
- Linux proxy processes
- More features in the future

RT Process – Make AMP Look Like Linux

- Use standard Linux process concept to manage other OS
  - Compile application meant to run on RTOS linking to RTOS and OpenAMP libraries
  - Put application executable (ELF file) in the Linux file system
  - Boot Linux as normal on main processors
  - Start application through special OpenAMP launch program, runs in supervisor mode
  - Handles select system(i.e. open/close/read/write/…) calls through RPC
  - Terminate RT process like any other process



“Regular” Process

RTProc 1 Proxy

RTProc 2 Proxy

`printf(“Hello\n”);`

`open(“conf.dat”,RD);`

Linux

File System

Serial Port

RTProc2 (BM/Exec)

RTProc1 (RTOS)

AMP Framework (rpmsg)

Core 0

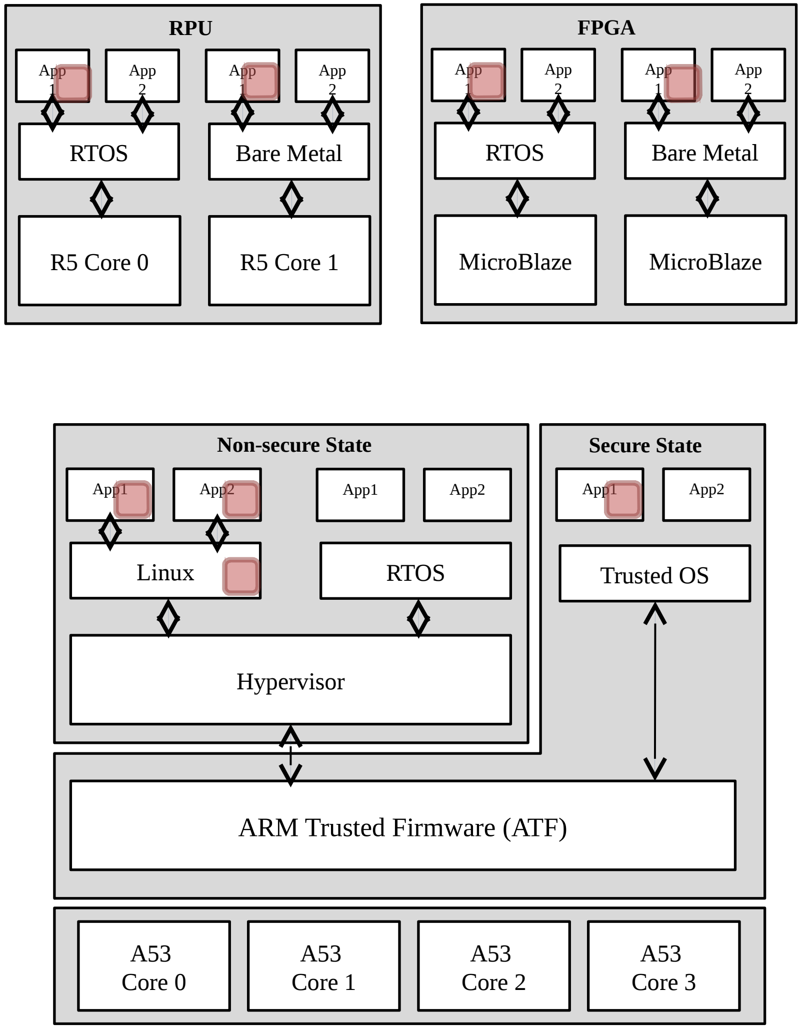
Core 1

Core 2

Core *n*

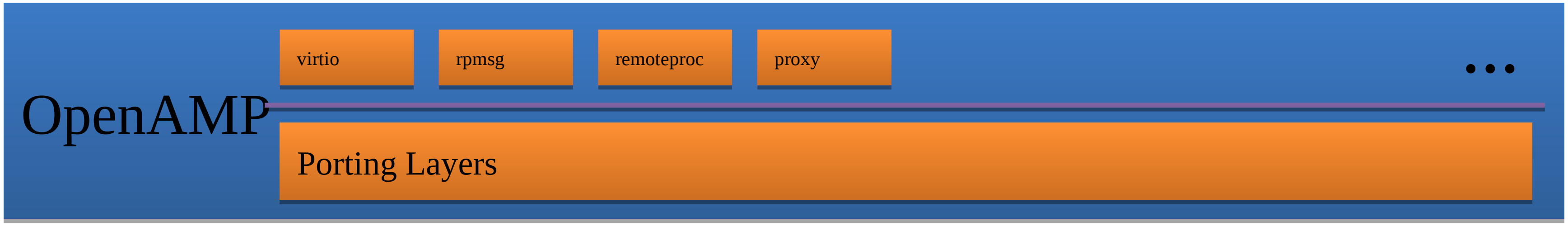
OpenAMP Use Cases for Zynq MPSoC

- OpenAMP applications can run in many places
  - A53, R5, MicroBlaze cores
  - In Linux kernel, Linux user space, RTOS, Bare Metal
  - OS’s can run directly on core(s), in hypervisor, in trusted mode, …
- A lot of different operating systems and hypervisors
  - Linux is used in majority of use cases
  - Most free and commercial RTOS’s are being used
  - Bare metal (no OS) is common on smaller cores
- Communication typically through shared memory
  - Though dedicated HW often used for data plane
  - Interprocess “kick” done in various ways
    - Inter Processor Interrupt (IPI), kernel call, hyper call, …
- Boot order varies
  - “Independent” boot
    - Through uboot, hypervisor, …
  - OpenAMP master starts slave
    - “Big” starts “small” (e.g. Linux app on A53 starts RTOS on R5)
    - “Small” starts “big” (e.g. Safe RTOS starts Linux)
- Safety and Security issues common
  - Affects boot order, messaging implementation, …



🟥 - Examples of OpenAMP applications

Evolving OpenAMP Architecture (Current)



OpenAMP

virtio

rpmsg

remoteproc

proxy

…

Porting Layers

Virtio – a communication protocol typically using shared memory and IPI

rpmsg – a set of messaging APIs:

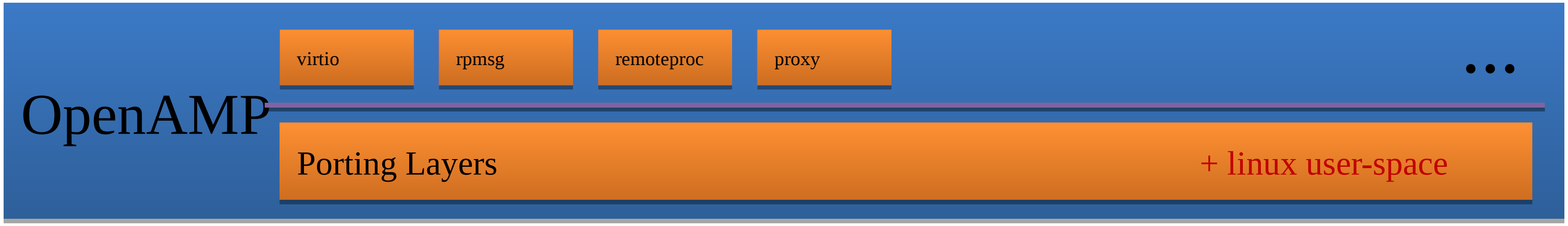
```
int rpmsg_send(struct rpmsg_channel *rpdev, void *data, int len);
int rpmsg_sendto(struct rpmsg_channel *rpdev, void *data, int len, u32 dst);
int rpmsg_send_offchannel(struct rpmsg_channel *rpdev, u32 src, u32 dst,
int rpmsg_trysend(struct rpmsg_channel *rpdev, void *data, int len);
int rpmsg_trysendto(struct rpmsg_channel *rpdev, void *data, int len, u32 dst);
…
```

Remoteproc – a set of APIs dealing with startup/shutdown:

```
int rproc_boot(struct rproc *rproc);
void rproc_shutdown(struct rproc *rproc);
struct rproc *rproc_get_by_phandle(phandle phandle);
```

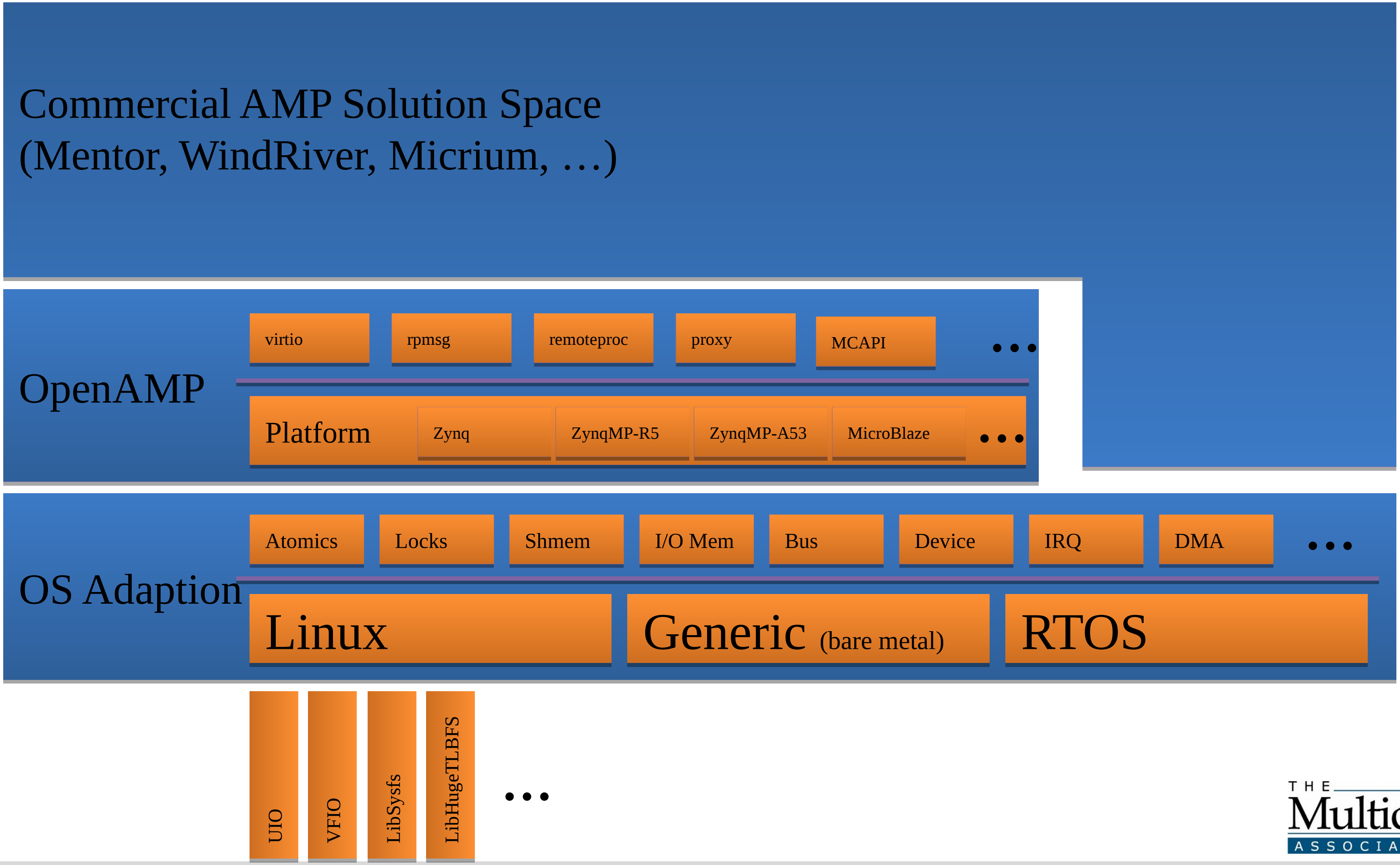
Proxy – a Linux user process that enables a remoteproc instance act similar to a Linux process

Evolving OpenAMP Architecture (Available)



Linux user-space – An efficient implementation in user-space utilizing UIO

Evolving OpenAMP Architecture (Next)



Commercial AMP Solution Space
(Mentor, WindRiver, Micrium, …)

OpenAMP

virtio rpmsg remoteproc proxy MCAPI …

Platform Zynq ZynqMP-R5 ZynqMP-A53 MicroBlaze …

OS Adaption

Atomics Locks Shmem I/O Mem Bus Device IRQ DMA …

Linux Generic (bare metal) RTOS

UIO VFIO LibSysfs LibHugeTLBFS …

THE Multicore ASSOCIATION

Proposed Working Group Tasks

- Define “North-bound” interfaces
  - Rpmsg, remoteproc
  - Discuss future enhancements
    - E.g. attach to running OS, communicate available resources, …
- Define “South-bound” interfaces
  - OS and HW abstraction interfaces
- Discuss MCAPI implementation on top of South-bound interfaces
  - Start with subset?