QEMU for Xilinx ZynqMP

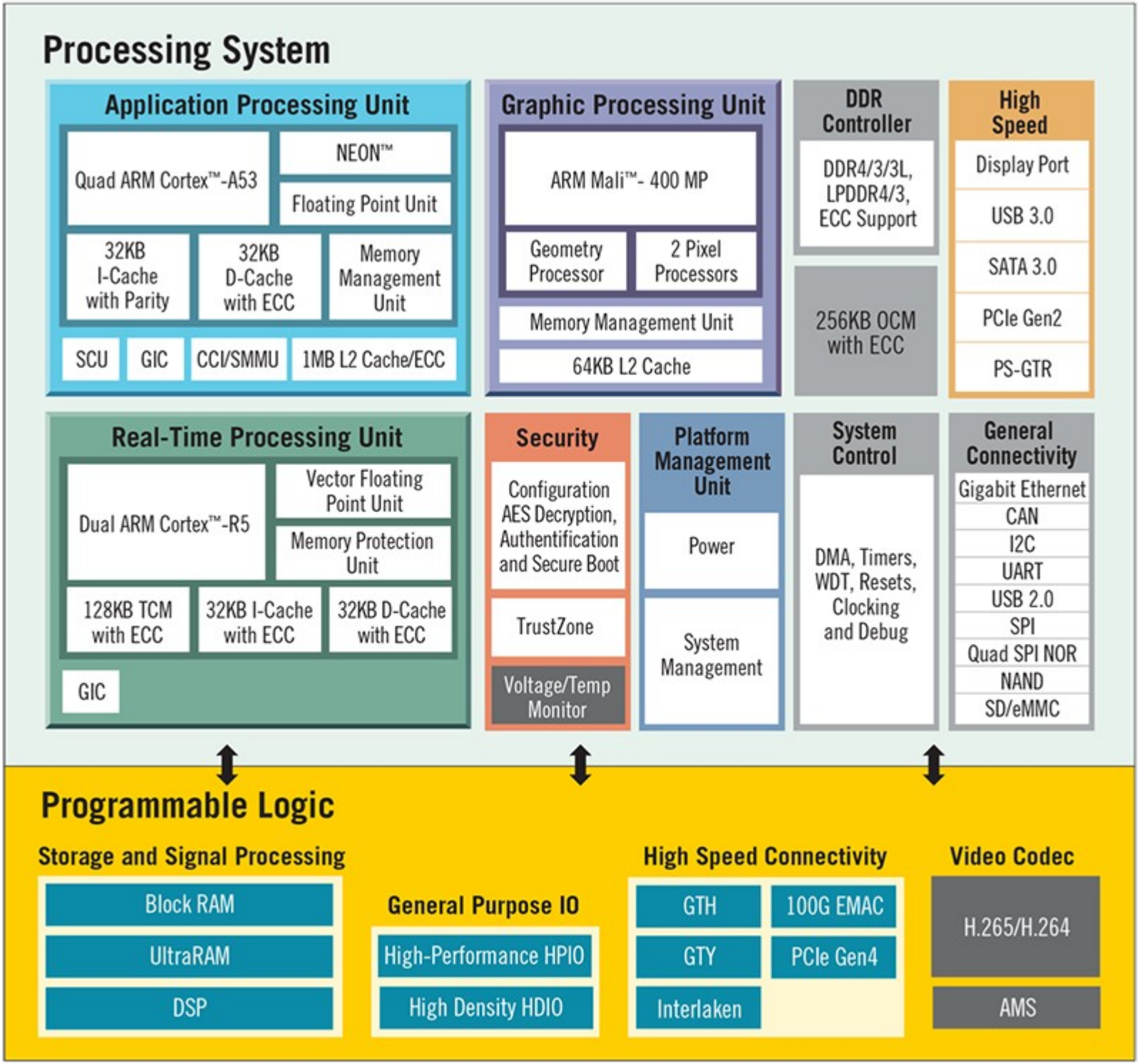
Edgar E. Iglesias *<edgar.iglesias@xilinx.com>* *V2 2015-Aug-20*

ZynqMP SoC

- New Chip (Zynq NG)
- Aggressive target for QEMU as early SW platform emulating WiP chip
  - BootROMs, Boot-loaders, Firmware, Hypervisors, OS ports etc
  - Schedules and Secrecy
- QEMU is fast and scales to large user-base compared to RTL based simulations

Thanks Xilinx

- John Williams and Peter Ryser
- Peter C, Alistair Francis, Sai Pavan

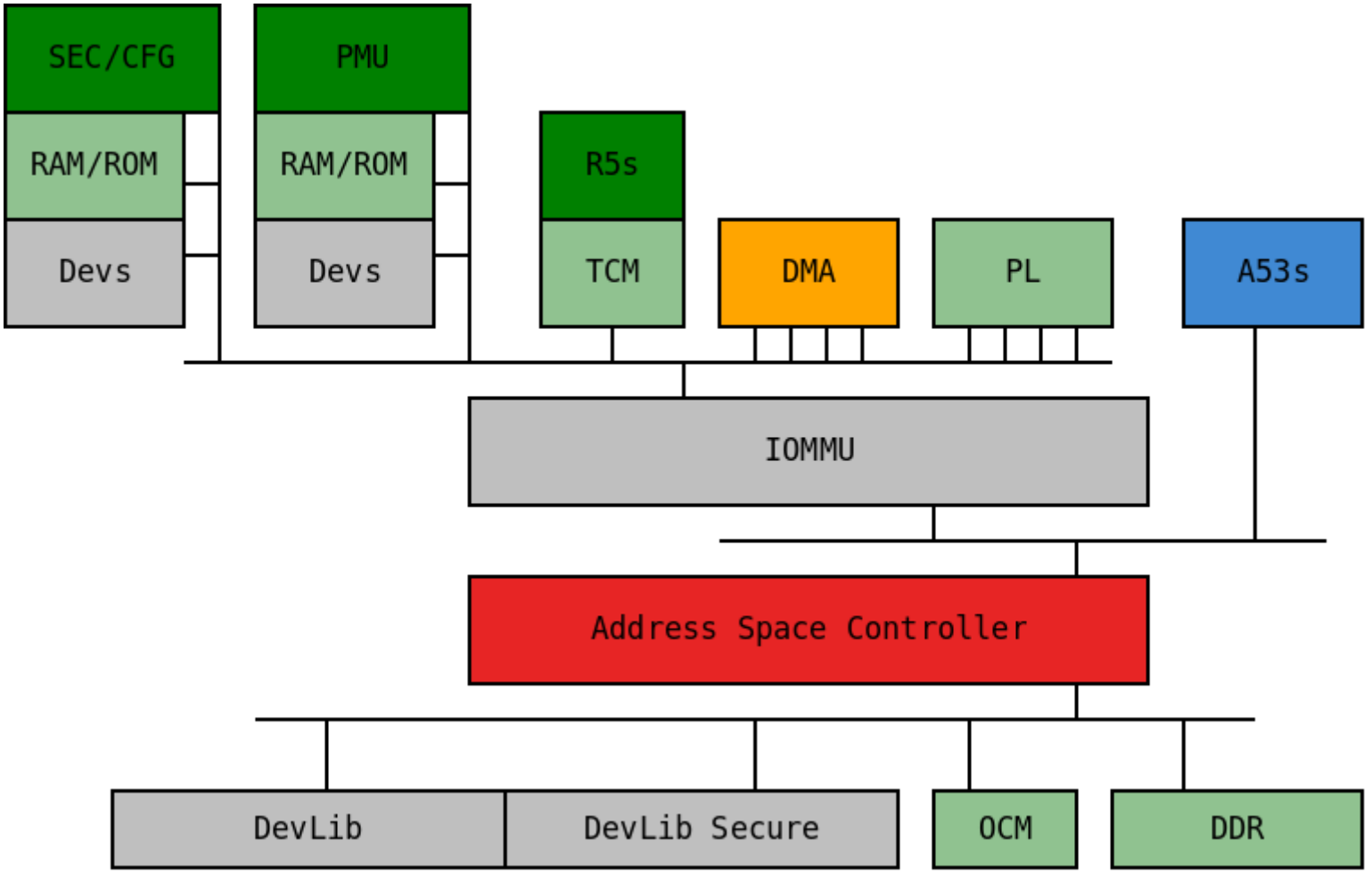


Note: Illustration not drawn to scale.

Machine description

- Flat Device Tree
- Devices matched with compat props
- Links between devs
  - DMA stream connections
- GPIOs, Interrupts, Memory Regions, etc
- Issue: QEMU specific device-tree
- Started simple but improved by Peter C and others QoM related work

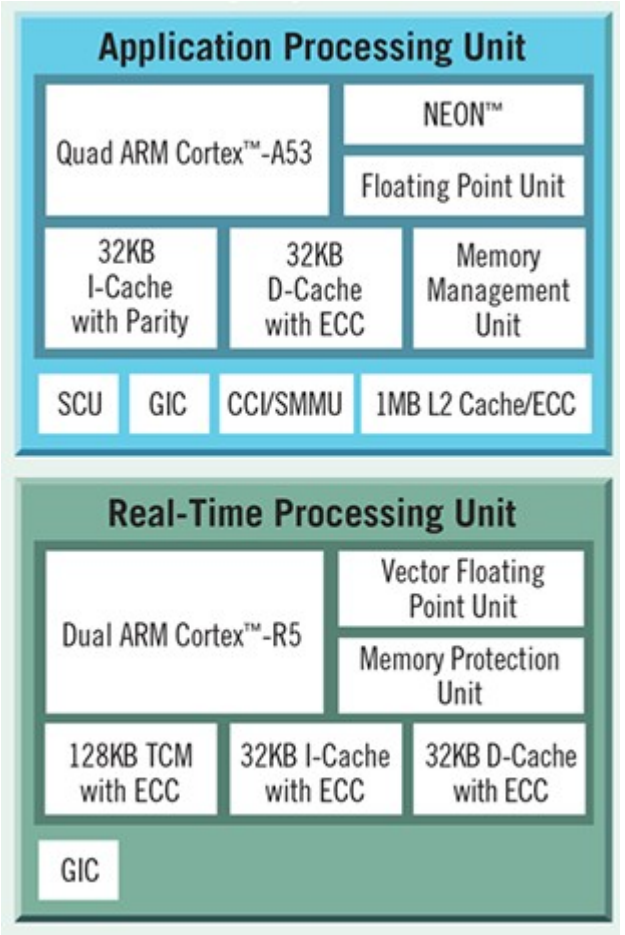
Simplified bus approximation



PMA - Per Master AddressSpace(s)

- Per CPU AdressSpaces
- Useful to avoid and easily debug DMA related AS errors
- Memory regions are described in FDT syntax
- Common setup mechanism for devs/cpus missing upstream
- Multiple AS:s per CPU (Maydell)

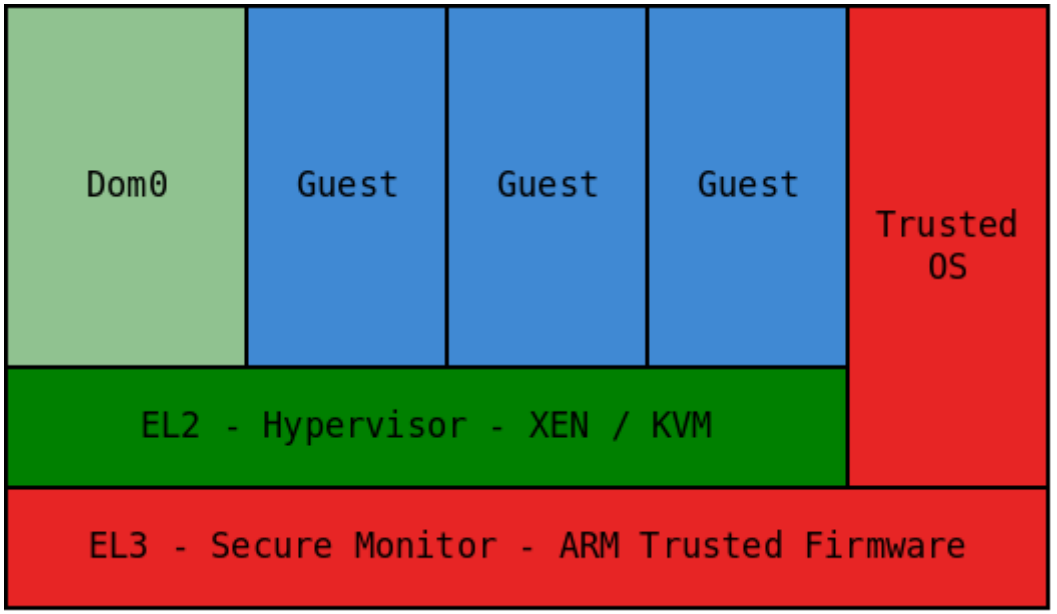
Cortex A53 and Cortex R5



Cortex A53 ARMv8

- Thanks to Alexander Graf, Peter Maydell and others EL1 and EL0 were functional and quite stable
- Kernel and User-space

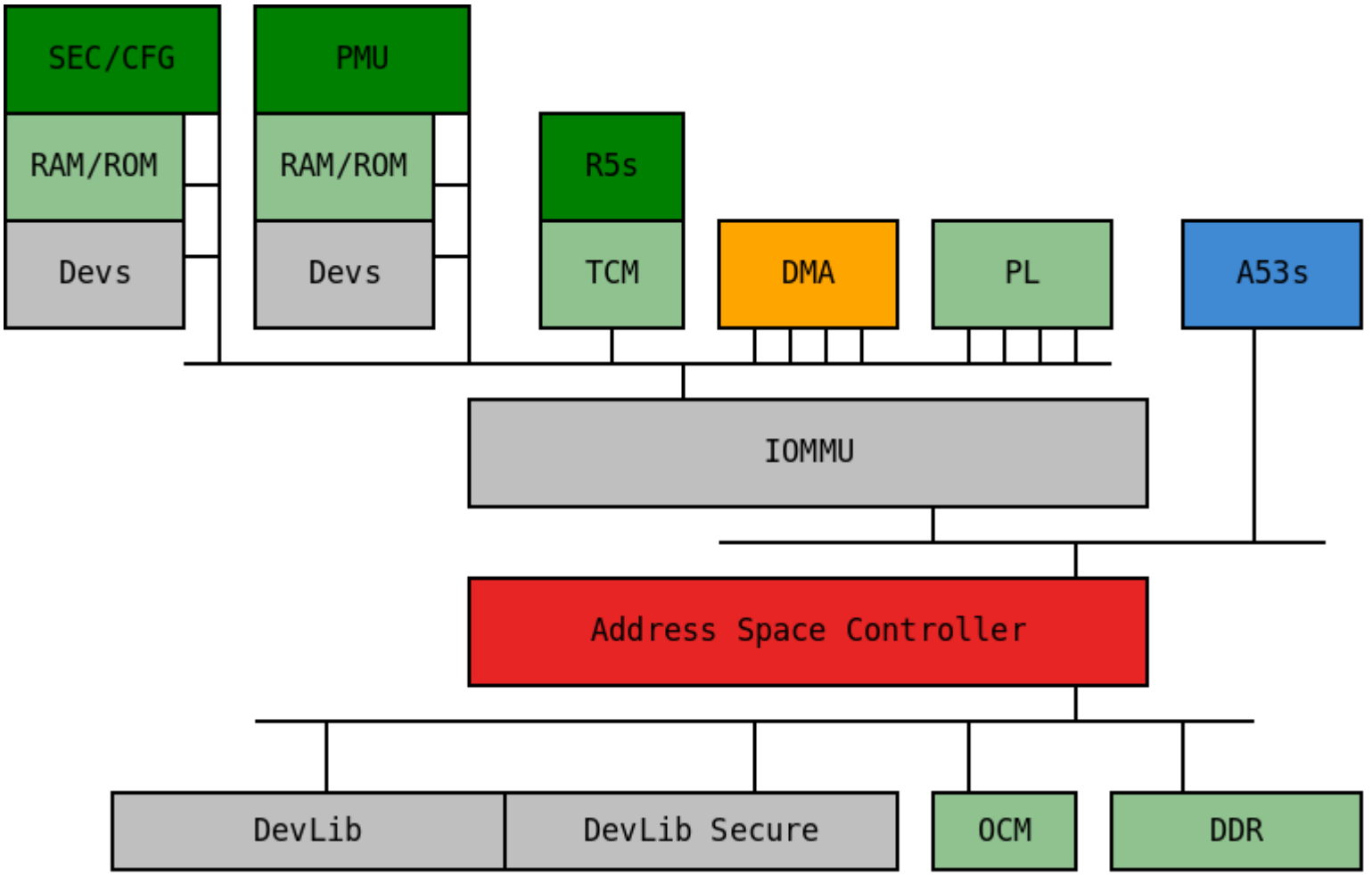
A53 Software stack
EL3 and 2 missing



EL3 and EL2

- Xilinx focused on AArch64 modes
- GIC virtualization, virtual interrupts, 2-stage MMU, exception model, virtual timers
- Community effort (Maydell, Greg, Fabian, Sergey and more)
    - Still lots missing upstream (Huge spec)
    - Xilinx tree limited but runs emulated XEN/KVM/ATF

Bus approximation
MemAttrs NS, MasterID, etc



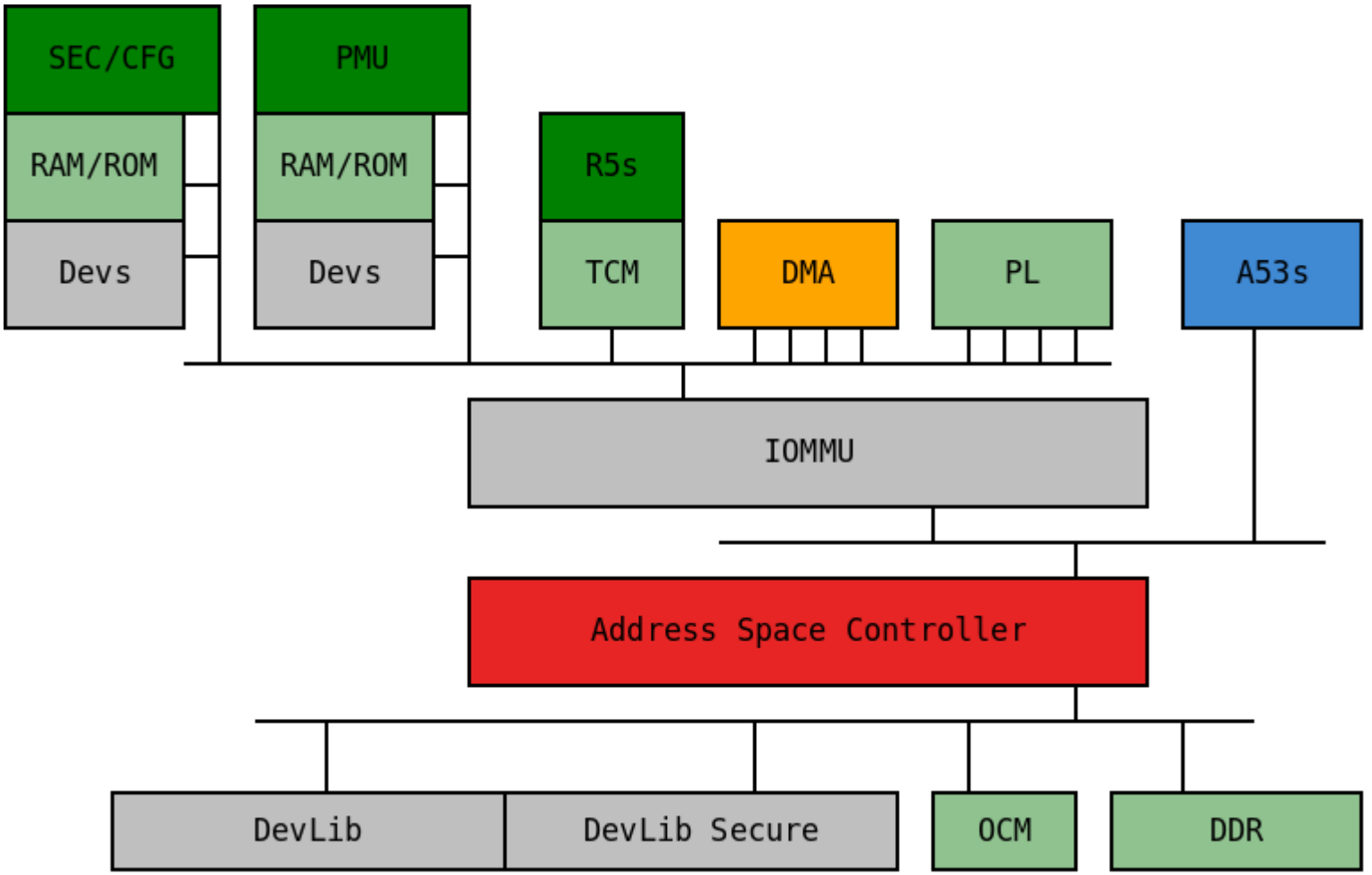
Memory Attributes

- AS based first attempt
  - Very messy with the IOMMUs
- Xilinx MemAttr changes to the MemAPI
- Peter Maydell did a better implementation of the same that eventually got merged
- Xilinx: MemAttrs are QoM objects, making it easy to create them and assign them with FDT syntax

Model of IOMMUs

- ARM SMMU model (limited and fixed cfg)
- R5s and Control processors are behind the IOMMUs
- Turned out to be fairly complex but also very helpful in SW driver bring-up
    - Helped find and fix errors in the XEN SMMU drivers and in our RTL!

Bus approximation: Multi-arch?



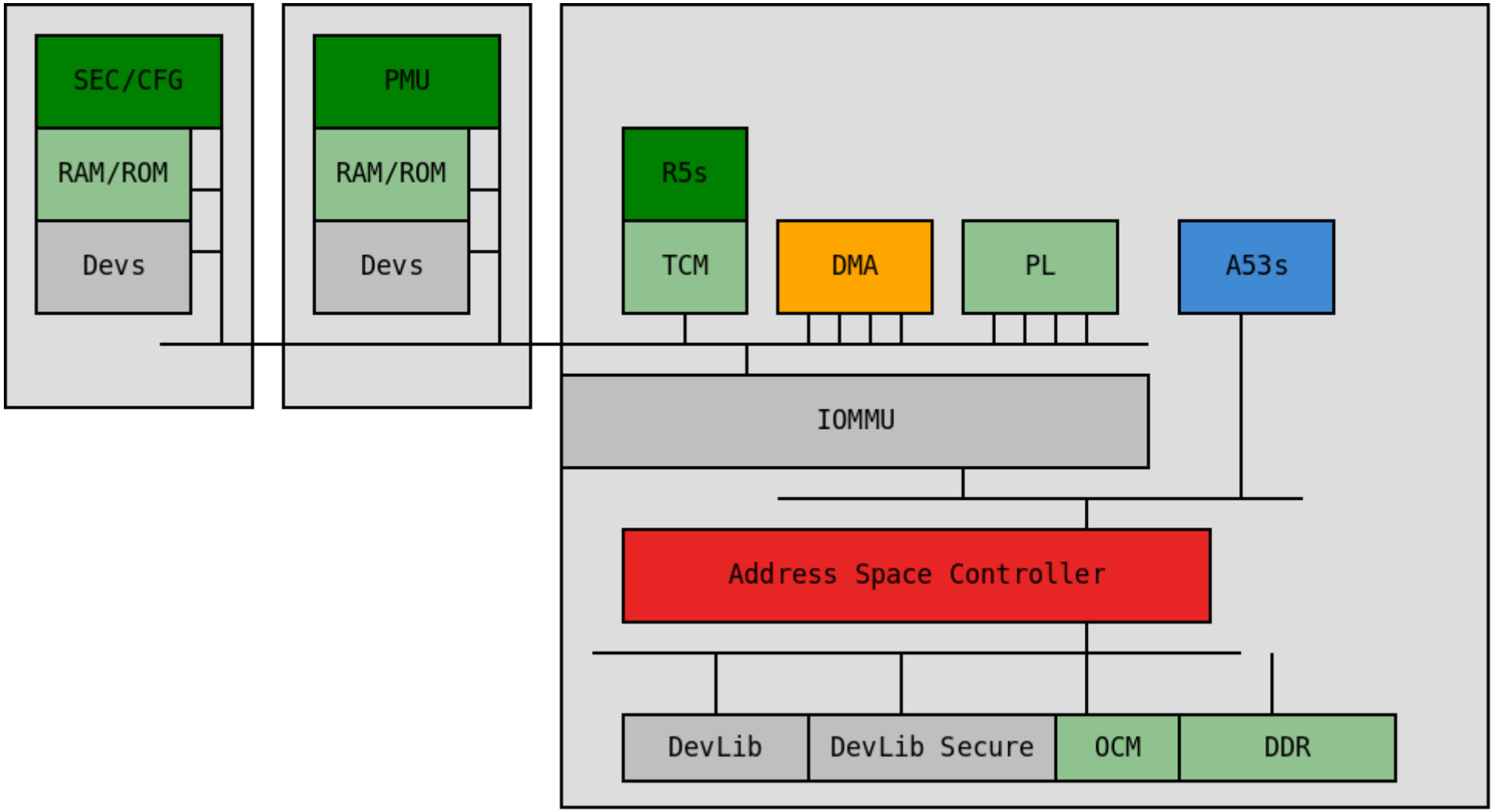
Multi-arch

- June 2013
  - QEMU didn't seem to be close enough for single build with multi-arch (Despite good efforts from multiple people including Andreas F)
  - Looking ahead, MMTCG and MultiArch look promising

Remote-Port

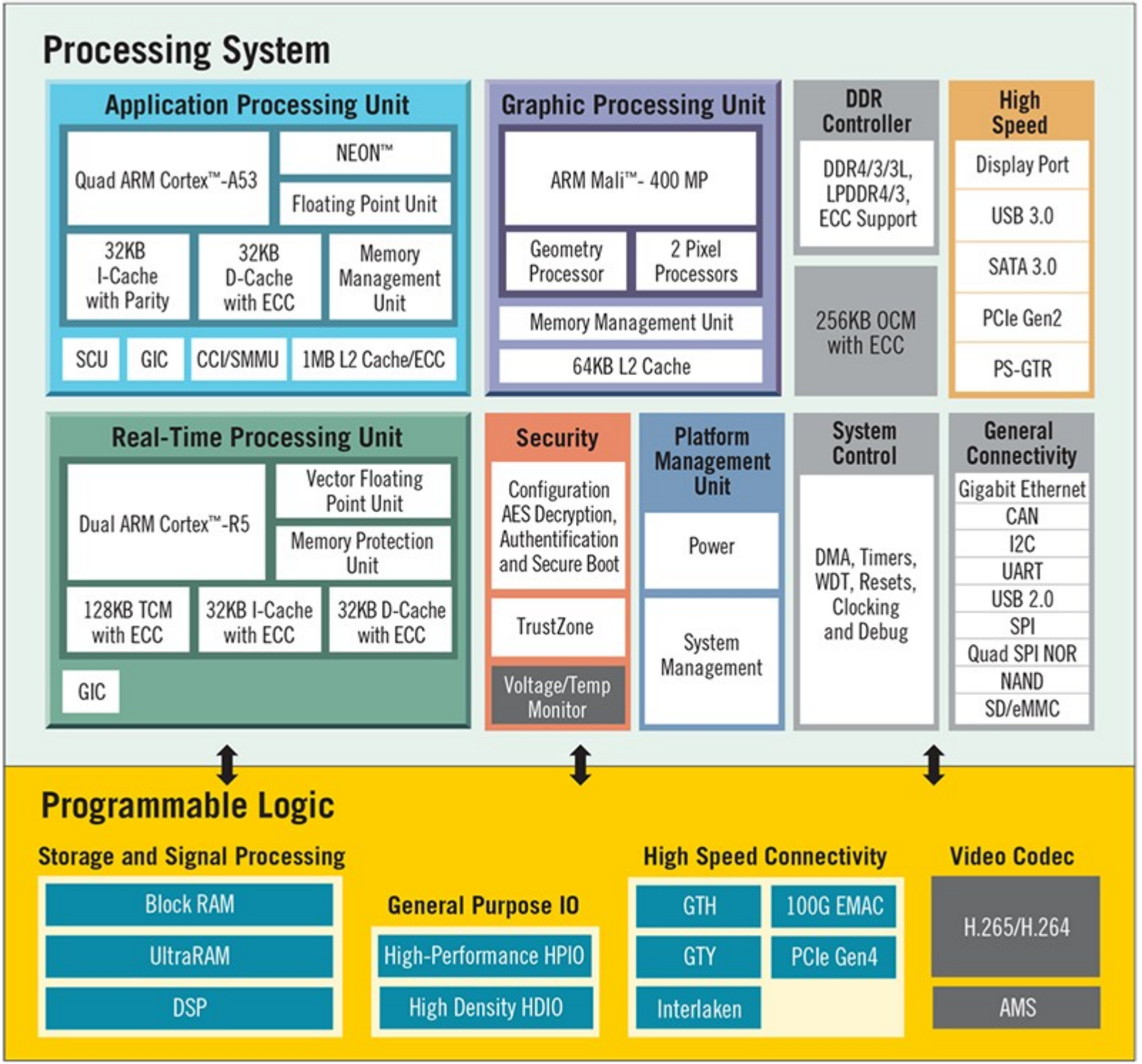
- QEMU to QEMU proxy
- First prototype based on hacked Qtest
- Binary protocol using shared memory maps for RAM sharing
- RP devices, bus master, bus slave, GPIO, DMA Stream, core (setup, sync, atomic ops)
- Multiple QEMUs run in parallel
- Keep it simple, one big QEMU, a few tiny remote ones attaching CPUs and mostly IRQs.

Bus approximation: Remote Port



Remote-Port

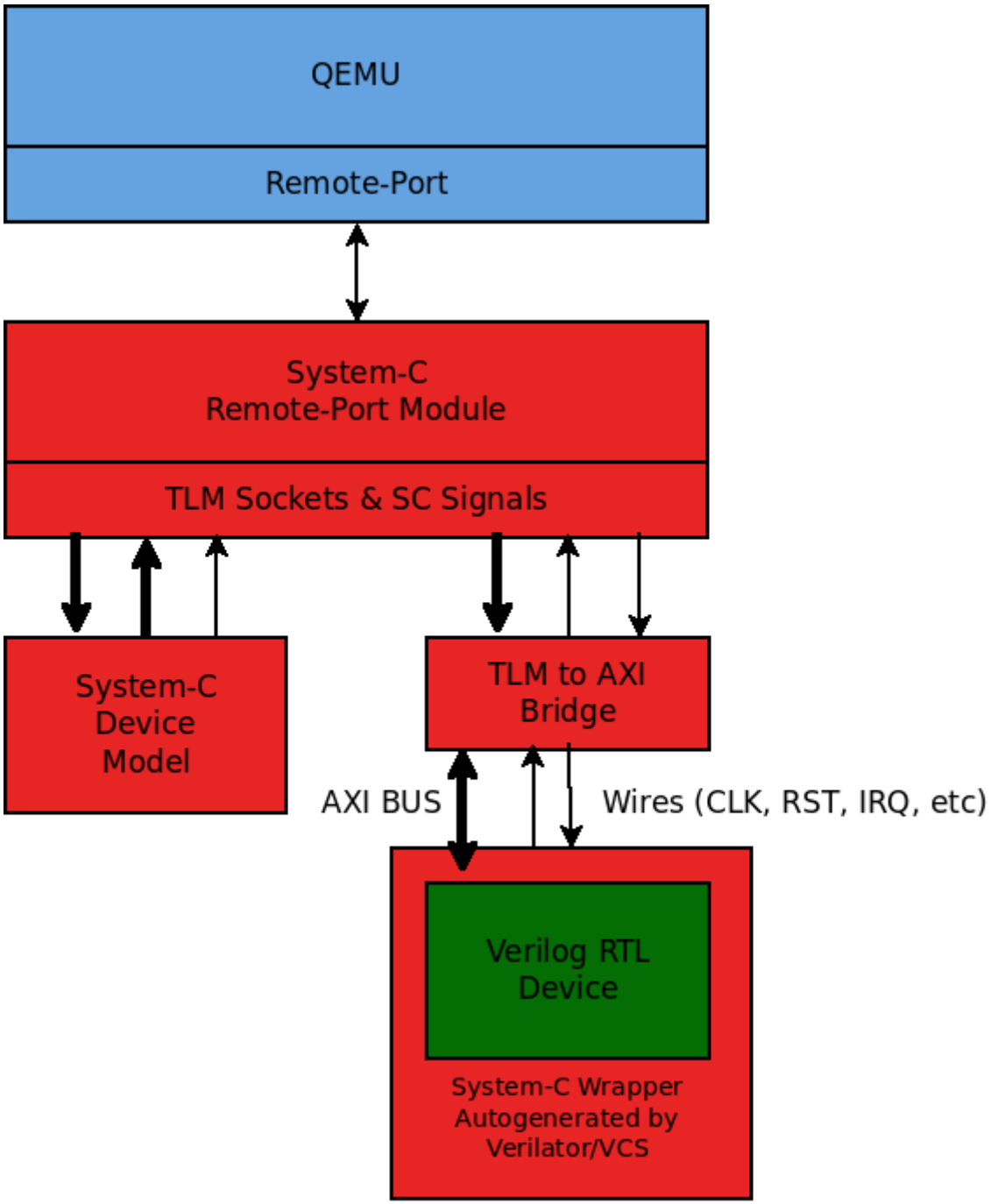
- RAMs are mapped SHARED into all instances (-mem-path)
- Bus accesses, interrupts, sync and DMA-stream are forwarded over sockets
- Atomic ops are done with IPC semaphores (translators need modifications)
- TLB flushes for inter-core code modifications not supported (could be implemented but hasn't been needed yet..).



Note: Illustration not drawn to scale.

SystemC / TLM

- SystemC, C++ class libs for simulating systems
- TLM: Inter-operability extensions for connecting models from different vendors into a larger system
- OpenSource simulator from OSCI
- Increasingly used by chip design teams
- Acts as a glue/binding between models written in different languages (e.g Verilog, VHDL, C, etc)


QEMU
Remote-Port
System-C
Remote-Port Module
TLM Sockets & SC Signals
System-C
Device
Model
TLM to AXI
Bridge
AXI BUS
Wires (CLK, RST, IRQ, etc)
Verilog RTL
Device
System-C Wrapper
Autogenerated by
Verilator/VCS

SystemC / TLM

- QEMU runs with Icount
- Timer based sync-calls stops QEMU from running to far a head of time (configurable quantum)
- Time-scaling and warping may be used to trade real-time experience vs accuracy

Verilator

- Open Source Verilog to SystemC (or C++) compiler
- Generated SystemC easily integrated into SystemC/TLM.
- It all runs with free tools QEMU, OSCIs OpenSource SystemC sim, Verilator, gtkwave.

Tools binary tracing

- Qemu-etrace
  - Binary trace emitted by QEMU
  - Exec, gpio, mem, asm
- Processed by external tool
  - Post-process trace files
  - Live processing via unix sockets

Tools etrace

- Qemu-etrace
  - Decode traces into human readable form
  - Load ELF files to provide symbol info
  - Collects code coverage statistics mapping executed addresses back into source code lines by reverse lookup of debug info
  - Emits lcov info files, lcovs genhtml can then generate nice HTML reports

Debugging

- GDB session break driven by LOG_GUEST_ERROR
- ARMv8 specific extensions
  - easily switch virtual memory view between Els
  - Access to more sysregs
  - A64/A32 Reg mapping hacks
- Issues: We have problems with the runstate control in GDB. (Run a specific core for a while etc)

Qtesting

- Qtest with TCG CPUs
- Python classes
  - QEMU.apu.bus32[0x1000] = 0xa0
  - Val = QEMU.apu.bus16[0x1000]
  - QEMU.set_irq(name, val)
- Scripted tests for embedded SW
  - Corner cases, HW errors, etc

Model autogen

- Data driven register decode/accesses
  - Common practice in RTL design
  - Auto-generate RTL, C header files, documentation from a single source of reg description
  - We extended this to generate QEMU skeletons for models
  - Identify patterns and auto-generate interrupt controllers / logic

Questions

Thanks for listening