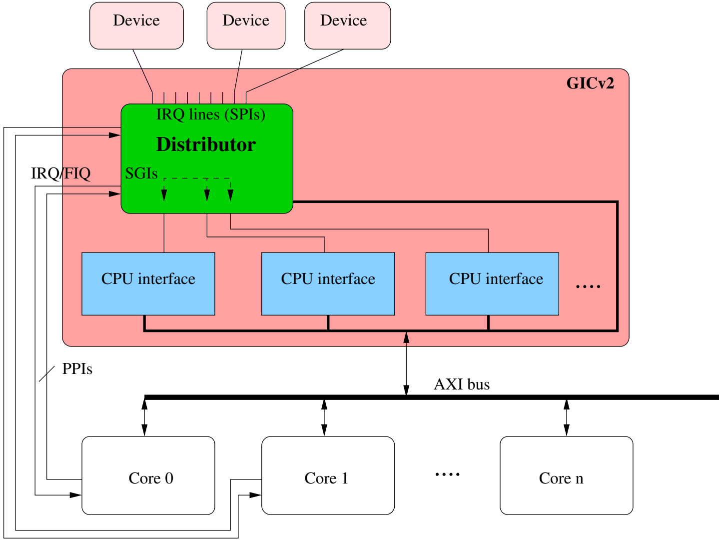
ARM® Interrupt Virtualization

Andre Przywara <andre.przywara@arm.com>

ARM interrrupt virtualization agenda

- GICv2 and virtualization overview
- KVM VGIC implementation
- GICv3 architecture
- GICv3 induced code changes in KVM
- VGIC evolution and future plans

GICv2 architecture

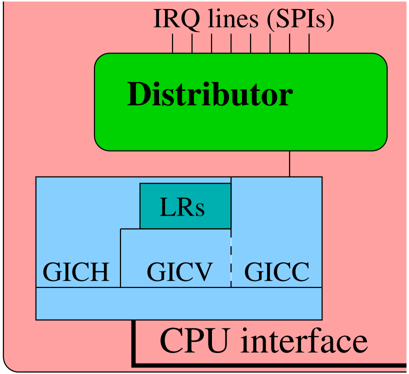


ARM GICv2 at a glance

- Programmed via MMIO accesses
  - Some registers are banked per CPU (at the same memory address)
- "distributor" is the central component
  - has an input pin for each wired interrupt (*SPI*s)
  - connects to a separate CPU interface (one per core)
  - connects to the IRQ pins on each core
  - has per-core input pins for private interrupts (*PPI*s)
  - handles inter-processor interrupts internally (*SGI*s)

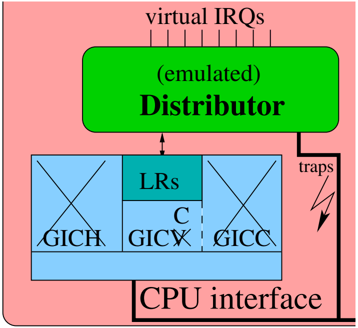
Virtualization support in GICv2

- *Virtual* CPU interface allows IRQ ACKs and EOIs without exiting the guest
- Hypervisor sets up virtual IRQs in List Registers
- In the guest the (virtual) CPU interface relates to these
- Allows connecting a physical interrupt to a virtual one
  - Physical IRQs gets EOIed at the same time
  - No need to trap or monitor EOI in this case anymore



KVM VGIC implementation

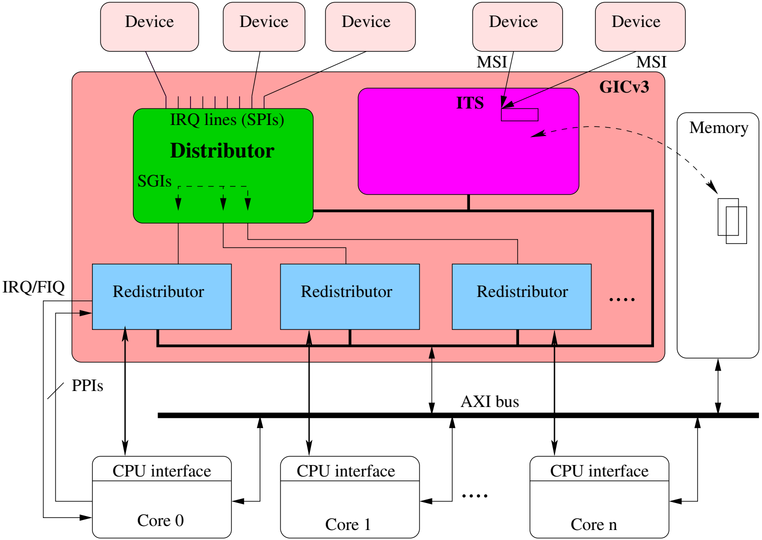
- Lives in virt/kvm/arm (to be shared between arm and arm64)
- Presented as an in-kernel IRQ controller to userland
  - Userland needs to setup addresses for the MMIO mapping
- Distributor is emulated (in vgic.c and vgic-v2-emul.c)
- Pending interrupts are written into the list registers (LRs)
- Interrupt acknowledgment and EOI is handled without exiting the guest (by the GIC hardware)
- After guest run distributor emulation syncs back from the LRs



Implementation details and challenges

- Banked MMIO accesses deny usage of KVM I/O bus framework (missing vCPU)
- Required setup (address base) of irqchip before use has funny effects
- Code layout is designed around MMIO handling
  - Requires extra work on interrupt injection / sync back
- State is held in bitmaps and bytemaps (like the hardware)
  - Works fine with limited, contigious IRQ numbers
- Handling of level vs. edge triggered interrupts
  - Lots of case distinctions necessary - hard to read
- Saving state (for migration) proves to be annoying
  - Requires syncing the *virtualized* CPU interface as well

GICv3 architecture



GICv3 changes

GICv2 compatibility mode would simplify things, but it is optional :-(

- System register access to CPU interface (drops banked MMIO)
- IRQ routing allows millions of cores
  - Lifts the 8-CPU limit of GICv2
  - Uses MPIDR based values to specify one target core per IRQ
- Splits distributor to separate private and shared IRQs
- New class of interrupts (LPI) via an Interrupt Translation Service (ITS)
  - Allows MSI/MSI-X support
  - Supports indirections for target cores (via collections)
  - Introduces device ID sampled from the bus
  - New IRQ class with possibly thousands of LPIs and probably sparse allocation
  - Tables are held in physical memory

GICv3 KVM implications

GICv2 compatibility support simplifies things, but it optional.

- No banked MMIO accesses anymore!
  - But now we have to support both cases in one code base :-(
- Distributor / redistributor split
  - Similar, but not identical $\rightarrow$ code refactoring
  - Introducing *more than one* MMIO region
- Potentially large, sparsely allocated LPIs spoil VGICv2 bitmaps
  - Leads to LPIs being hold in separate data structures
- ITS data structures are held in *guest* physical memory
  - Expensive to access from KVM kernel code
  - Fortunately caching is common in hardware too
  - Wasting precious, but here unneeded guest memory

KVM challenges

KVM code in general is architected to match x86. (*No offense!*)

- GSI IRQ routing not a real fit
  - Technically not needed for ARM, but no IRQFDs without it
  - Requires pointless identity (or offsetted) mapping for SPIs
  - LPI numbers are purely internal
- ITS MSIs are identified by doorbell/device-ID/payload triple
  - Common usage is one doorbell and payload=0 (per-device IRQ number)
  - Hardware samples device ID from the bus upon doorbell access
  - Requires addition of device ID to KVM MSI structures
  - Payload is not a global interrupt number
  - Guest can change payload (as device ID provides isolation)

VGIC evolution

Going from:

- one hardware device / one emulation model
- with max. 8 CPUs and
- a contigious, limited number of wired IRQs

to:

- multiple hardware devices / multiple emulation models
- with potentially 2^{32} CPUs and
- non-contigious, large number of wired IRQs and MSIs

asks for some code changes and refactoring ...

VGIC refactoring

- Explicit VGIC setup and initialization (*done*)
- Use proper KVM I/O bus MMIO handlers (*done*)
- Support multiple hardware models (*done*)
- Support multiple emulation models (with KVM_CREATE_DEVICE) (*done*)
- Utilize connection of physical and virtual IRQs (*WIP*)
- Re-architect VGIC code to focus on used IRQs instead of MMIO accesses (*to be done*)
  - Probably split support for different IRQ classes (SGIs, PPIs, SPIs, LPIs)
- Incorporate more virtualization features for GICv4 (*to be done*)
  - GICv4 provides virtual LPIs being directly injected into a guest
  - Holds tables mapping vCPUs to physical CPUs

Thank You