Virtualization in the ARmv7 Architecture
Lecture for the Embedded Systems Course
CSD, University of Crete (May 19, 2015)

- Manolis Marazakis (maraz@ics.forth.gr)

Institute of Computer Science (ICS)
Foundation for Research and Technology – Hellas (FORTH)
Virtualization Benefits in Embedded Systems

- Workload consolidation
  - E.g. Applications + Baseband sharing a multicore SoC
- Flexible resource provisioning
- License barrier
- Legacy software support
  - important with the multitude and variety of embedded operating systems (commercial and even home-brew)
- Reliability
- Security
Virtualization trade-off

- **Performance:**
  - Applications that used to own the whole processor must now share
  - Hypervisor adds runtime overhead & increases memory footprint
    - Real-time properties?
  - Full virtualization without hardware support means software emulation

- **Complexity:**
  - Old scenario: two software stacks + two hardware systems
  - New scenario: two software stacks + one hardware system + one host kernel
  - More abstraction layers → more bugs ...

- **Security & reliability:**
  - Increased size of Trusted Computing Base (TCB)
  - Increased impact of hardware failure
  - I/O: emulation vs (para)virtual vs direct access
Essentials of a hypervisor

- Parent partition (minimum-footprint OS) + Hypervisor
- **Hypervisor**: Thin layer of software running on the hardware
  - Supports creation of *partitions (virtual machines)*
    - Each partition has one or more *virtual processors*
    - Partitions can own or share hardware resources
- **Enforces memory access rules**
- **Enforces policy for CPU usage**
  - Virtual processors are scheduled on real processors
- **Enforces ownership of other devices**
- **Provides inter-partition messaging**
  - Messages appear as interrupts
- **Exposes simple programmatic interface: “hypercalls”**
Virtualization extensions to the ARMv7-A architecture

- Virtualization extensions to the ARMv7-A architecture:
  - Available in Cortex A-15 / A-7 CPUs
  - Hyp - New privilege level (for hypervisor)
    - GuestOS: SVC privilege level, Applications: USR privilege level
  - 2-stage address translation (for OS and hypervisor levels)
  - Complementary to TrustZone security extensions

- Mechanisms to minimize hypervisor intervention for “routine” GuestOS tasks:
  - Page table management
  - Interrupt masking & Communication with the interrupt controller (GIC)
  - Device drivers (hypervisor memory relocation)
  - Emulation of Load/Store accesses and trapped instructions
    - Hypervisor Syndrome Register: Hype mode entry reason (syndrome)
  - Traps into Hyp mode for ID register accesses & idling (WFI/WFE)
  - System instructions to read/write key registers
Privilege levels

- Guest OS: same kernel/user privilege structure
- HYP mode: higher privilege than OS kernel level
  - hvc instruction (hypercall)
  - VMM controls wide range of OS accesses
- Hardware maintains TZ security (4th privilege level)
  - User Mode (Non-privileged)
  - Supervisor Mode (Privileged)
  - Hyp Mode (More Privileged)
  - Secure State
  - TrustZone Secure Monitor (Highest Privilege)
Virtual Memory (1-stage translation)

- Without virtualisation, the OS owns the memory
  - Allocates areas of memory to the different applications
  - Virtual Memory commonly used in “rich” operating systems
Virtual Memory (2-stage translation)

Stage 1 translation owned by each Guest OS

Stage 2 translation owned by the VMM

Hardware has 2-stage memory translation

Tables from Guest OS translate VA to IPA

Second set of tables from VMM translate IPA to PA

Allows aborts to be routed to appropriate software layer

Virtual address (VA) map of each App on each Guest OS

“Intermediate Physical” address map of each Guest OS (IPA)

Physical Address (PA) Map
Virtualization of interrupts

- An interrupt might need to be routed to one of:
  - Current or different GuestOS
  - Hypervisor
  - OS/RTOS running in the secure TrustZone environment

- Physical interrupts are taken initially in the Hypervisor
  - If the Interrupt should go to a GuestOS:
    - Hypervisor maps a “virtual” interrupt for that GuestOS
Virtual Interrupt Controller

- ISR of GuestOS interacts with the virtual controller
  - Pending and Active interrupt lists for each GuestOS
  - Interact with the physical GIC in hardware
  - Creates Virtual Interrupts only when priority indicates it is necessary

- GuestOS ISRs therefore do not need calls for:
  - Determining interrupt to take [Read of Interrupt Acknowledge]
  - Marking the end of an interrupt [Sending EOI]
  - Changing CPU Interrupt Priority Mask [Current Priority]

- GIC has separate sets of internal registers:
  - Physical registers and virtual registers
    - Non-virtualized system and hypervisor access the physical registers
    - Virtual machines access the virtual registers
    - Guest OS functionality does not change when accessing the vGIC

- Hypervisor remaps virtual registers for use by GuestOS’es
  - Interrupts generate a hypervisor trap
Virtual interrupt sequence

- External IRQ (configured as virtual by the hypervisor) arrives at the GIC
- GIC Distributor signals a Physical IRQ to the CPU
- CPU takes HYP trap, and Hypervisor reads the interrupt status from the Physical CPU Interface
- Hypervisor makes an entry in register list in the GIC
- GIC Distributor signals a Virtual IRQ to the CPU
- CPU takes an IRQ exception, and Guest OS running on the virtual machine reads the interrupt status from the Virtual CPU Interface
Virtual I/O devices

- Memory-mapped devices
  - Read/write accesses to device registers have specific side-effects
- Virtual devices $\rightarrow$ emulation
  - Typically, read/write accesses have to trap to Hypervisor
    - Fetch & interpretation of emulated load/stores is performance-intensive
  - Syndrome: key information about an instruction
    - Source/destination register, Size of data transfer, ...
    - Available for some loads/stores (on abort)
      - If not available, then it is required to fetch the instruction for full emulation
- System MMU: 2$^{nd}$-stage address translation for devices
  - Allows devices to be programmed into Guest’s VA space
System MMU (IO-MMU)
ARM TrustZone (Secure System Partitioning)

- ARM TrustZone extensions introduce:
  - new processor mode: monitor
    - similar to VT-x root mode
    - banked registers (PC, LR)
    - can run unmodified guest OS binary in non-monitor kernel mode
  - new privileged instruction: SMI
    - enters monitor mode
  - new processor status: secure
  - partitioning of resources
    - memory and devices marked secure or insecure
    - in secure mode, processor has access to all resources
    - in insecure mode, processor has access to insecure resources only
  - monitor switches world (secure ↔ insecure)
  - really only supports one virtual machine (guest in insecure mode)
    - need another hypervisor and para-virtualization for multiple guests
Propagation of System Security Mode

NS: Not Secure - treated like an address line

Virtualization in the ARMv7 Architecture
Boot sequence with Hypervisor

1. Power on Reset
2. Software flow
   - boot code
   - or Startup code
3. Enter monitor mode
4. Monitor software
5. ARM Secure Monitor mode
6. ARM Secure SVC mode

Non-Secure
- ARM Non-Secure User mode
- Guest OS1 Apps
- Guest OS2 Apps
- ARM Non-Secure Privileged modes (SVC, IRQ, ...)
- ARM Non-Secure Hyp mode

Secure
- Hypervisor
Sources

- David Brash, Extensions to the ARMv7-A Architecture, HotChips 2010
- John Goodacre, Hardware accelerated Virtualization in the ARM Cortex Processors, XenSummit Asia 2011
- Roberto Mijat and Andy Nightingale, Virtualization is Coming to a Platform Near You: The ARM Architecture Virtualization Extensions and the importance of System MMU for virtualized solutions and beyond, ARM White paper, 2011