ex-Arm Ltd., Kernel Team

No NMI? No Problem! - Implementing Arm64 Pseudo-NMI

Julien Thierry <jthierry@redhat.com>

September 24, 2019

About Myself & the talk

- Joined Arm in June 2017
  - Linux Kernel team
  - Cambridge, UK
- First kernel related job
- Left in August 2019
- Work presented here done while employed by Arm

What are IRQs?

- Manifestation of event in the system: device or program triggered
- Interrupts active flow of execution
- Managed with some primitives:
  - local_irq_disable/enable()
  - local_irq_save/restore()

What are NMIs?

- Non-Maskable Interrupt
- Interrupts that can happen while interrupts are disabled
- Generally cannot be disabled locally
- Used for reporting hardware status, timer expiration, profiling, ...
- x86: Dedicated exception entry for NMIs
- SPARC: Interrupts with highest possible priority

NMI Handlers

- Dedicated context:
  - nmi_enter()
  - nmi_exit()
- Inform system wide features (e.g. printk, ftrace, rcu, ...)
- Restrictions:
  - No NMI nesting
  - No preemption
  - Keeps NMI handlers simpler

NMI Handlers and Locks

```
void inc_counter(void)
{
  unsigned long flags;

  flags = raw_spin_lock_irqsave(&count_lock);
  global_counter += 1;
  raw_spin_unlock_irqrestore(&count_lock, flags);
}
```

NMI Handlers and Locks

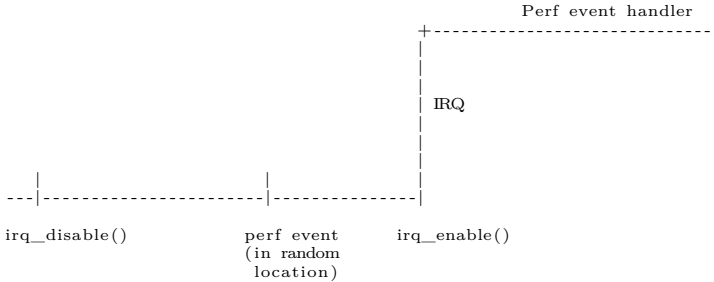
```
void inc_counter(void)
{
  unsigned long flags;

  flags = raw_spin_lock_irqsave(&count_lock);
  global_counter += 1;
  raw_spin_unlock_irqrestore(&count_lock, flags);
}
```

- Lock cannot protect against NMI
- Can protect from concurrency (SMP) $\rightarrow$ mutex

The Perf Case

Demo

The Perf Case



arm

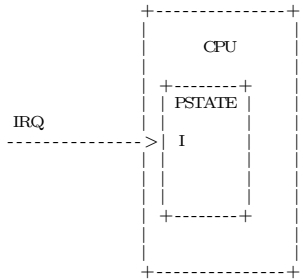
Pseudo-NMI for Arm64

- No architected NMI

Pseudo-NMI for Arm64

- No architected NMI
- But has IRQ priorities and priority masking
- Emulate NMI's behaviour using an IRQ
  - Distinguish NMI from normal IRQ: two distinct priorities
  - Block IRQs while allowing NMIs: mask priority of normal IRQs

Arm64 CPU Interrupt Control



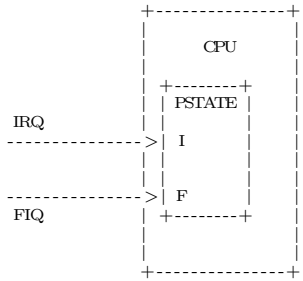
arm

Arm64 CPU Interrupt Control

PSTATE.I bit

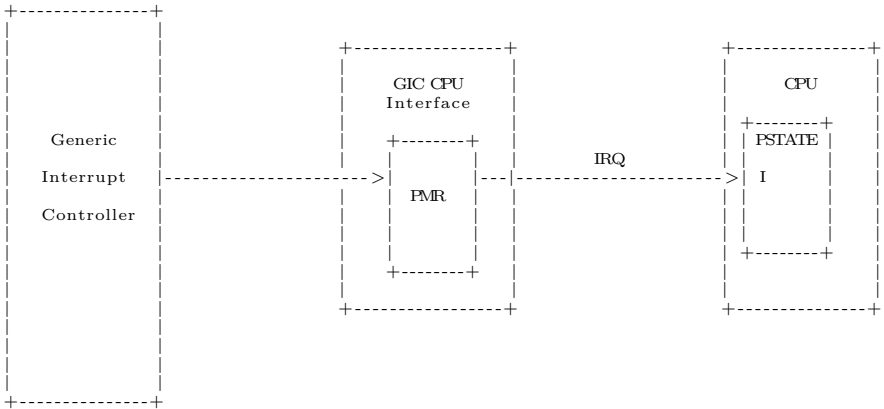
- Toggled by local_irq_enable/disable()
- Prevents CPU to jump to interrupt vector
- Single bit $\longrightarrow$ only binary state

Arm64 CPU Interrupt Control



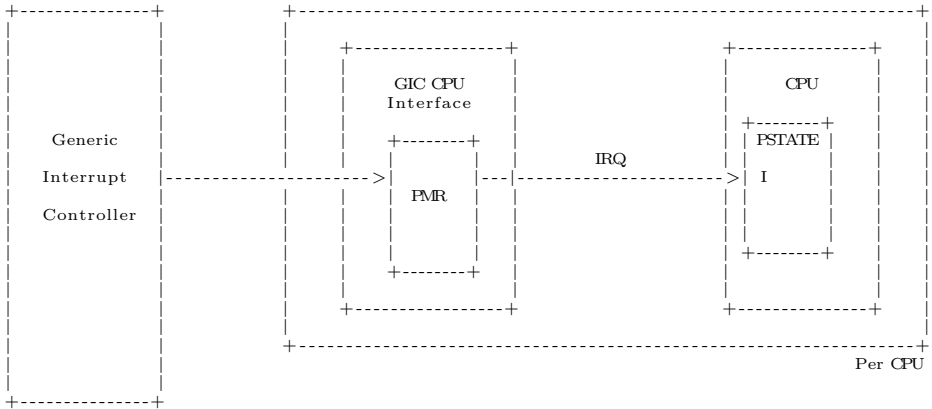
arm

More control with the GIC



arm

More control with the GIC



arm

Fitting priorities in linux

- Stop touching PSTATE.I bit
- Mask IRQs using PMR in local_irq_*()
- Configure NMIs with high enough priority in GIC

Fitting priorities in linux

- Stop touching PSTATE.I bit
- Mask IRQs using PMR in local_irq_*()
- Configure NMIs with high enough priority in GIC

But:

- Upon IRQ, interrupts are automatically disabled (PSTATE.I)
- Some contexts (idle, KVM guests, ...) cannot use PMR for IRQ disabling
- Need to take care to have consistent state when reaching generic code

Using the NMI

- Requester needs to set up target IRQ priority through irqchip
- Initial proposal: introduce new NMI state in $\mathrm{enum\ irqchip\_irq\_state}$

Using the NMI

- Requester needs to set up target IRQ priority through irqchip
- Initial proposal: introduce new NMI state in enum irqchip_irq_state
- Not so popular:

> Adding NMI delivery support at low level architecture irq chip level is perfectly fine, but the exposure of that needs to be restricted very much. Adding it to the generic interrupt control interfaces is not going to happen. That's doomed to begin with and a complete abuse of the interface as the handler can not ever be used for that.
>
> Thanks,
>
> tglx

NMI API

- include/linux/interrupt.h
- Provide NMI API similar to the IRQ one:
  - request_nmi() / free_nmi()
  - enable_nmi() / disable_nmi()
- irq_chip changes
  - Flag to advertise NMI support
  - chip->irq_nmi_setup() / chip->irq_nmi_teardown()
- Not exported to modules
- (Per-cpu NMI variants available)

Results on Perf

Demo

Timeline

- Work initiated as by Daniel Thompson from Linaro:

Timeline

- Work initiated as by Daniel Thompson from Linaro:
- 2015-03-18: RFC: Pseudo-NMI for arm64 using ICC_PMR_EL1 (GICv3)
- 2016-08-19: RFC v3, v4.8-rc2
  - 7 patches
  - Uses PMR for IRQ enabling/disabling
  - Hardcoded NMI priority for backtrace IPI
  - ”The code works-for-me (tm) and is more ”real” than the last time I shared these patches.”

Timeline

- Work initiated as by Daniel Thompson from Linaro:
- 2015-03-18: RFC: Pseudo-NMI for arm64 using ICC_PMR_EL1 (GICv3)
- 2016-08-19: RFC v3, v4.8-rc2
  - 7 patches
  - Uses PMR for IRQ enabling/disabling
  - Hardcoded NMI priority for backtrace IPI
  - ”The code works-for-me (tm) and is more ”real” than the last time I shared these patches.”
- 2017-07 or 2017-08: I start working on it

Timeline

- Work initiated as by Daniel Thompson from Linaro:
- 2015-03-18: RFC: Pseudo-NMI for arm64 using ICC_PMR_EL1 (GICv3)
- 2016-08-19: RFC v3, v4.8-rc2
  - 7 patches
  - Uses PMR for IRQ enabling/disabling
  - Hardcoded NMI priority for backtrace IPI
  - ”The code works-for-me (tm) and is more ”real” than the last time I shared these patches.”
- 2017-07 or 2017-08: I start working on it
- 2017-10-11: RFC: arm64: provide pseudo NMI with GICv3, v4.15-rc2
  - 7 patches
  - Testing with perf interrupt
  - dropped backtrace IPI

Timeline

- 2018-05-21: V3, v4.17-rc6
  - 6 patches

Timeline

- 2018-05-21: V3, v4.17-rc6
  - 6 patches
  - Review

> As it is, this patch is almost impossible to review. It turns the interrupt masking upside down, messes with the GIC, hacks KVM... Too many things change at once, and I find it very hard to build a mental picture of the changes just by staring at it.
>
> [...]
>
> Thanks,
>
> M.

Timeline

- 2018-05-25: V4
  - 26 patches

Timeline

- 2018-05-25: V4
  - 26 patches
- 2018-08-28: V5, v4.19-rc1
  - Depends on separate API for NMIs series (4 patches)

Timeline

- 2018-05-25: V4
  - 26 patches
- 2018-08-28: V5, v4.19-rc1
  - Depends on separate API for NMIs series (4 patches)
- 2019-01-31: V10
  - 25 patches

Timeline

- 2018-05-25: V4
  - 26 patches
- 2018-08-28: V5, v4.19-rc1
  - Depends on separate API for NMIs series (4 patches)
- 2019-01-31: V10
  - 25 patches
- 2019-02-06: Merge of v10 + v6 of NMI API in 5.1

Timeline

Timeline

- 2019-03-29: Bug: System hangs when using Ftrace graph tracer

Timeline

- 2019-03-29: Bug: System hangs when using Ftrace graph tracer
- 2019-06-21: Fixed in 5.3 (V4 of 8 patches series)

Try it, buy it

- Get Aarch64 capable platform with GICv3
- Get Linux v5.3+ sources
- Kernel Features $\rightarrow$ Support for NMI-like interrupts (CONFIG_ARM64_PSEUDO_NMI)
- Boot option irqchip.gicv3_pseudo_nmi=1
- arm_pmu patches using NMI (V4)
  - Still needs an update

arm

Next steps

- Use NMI for more features on Arm64
  - Backtrace, CPU stop IPI: requires moving IPIs to use IRQ framework
  - Hard lockup detector: need some rework of $\mathrm{arm\_pmu}$ driver

Next steps

- Use NMI for more features on Arm64
  - Backtrace, CPU stop IPI: requires moving IPIs to use IRQ framework
  - Hard lockup detector: need some rework of $\mathrm{arm\_pmu}$ driver
- Look into using NMI API for SPARC NMIs

Next steps

- Use NMI for more features on Arm64
  - Backtrace, CPU stop IPI: requires moving IPIs to use IRQ framework
  - Hard lockup detector: need some rework of $\mathrm{arm\_pmu}$ driver
- Look into using NMI API for SPARC NMIs
- No plan for GICv2 support
- Prior to GICv3, CPU interface is memory mapped

Next steps

- Use NMI for more features on Arm64
  - Backtrace, CPU stop IPI: requires moving IPIs to use IRQ framework
  - Hard lockup detector: need some rework of $\mathrm{arm\_pmu}$ driver
- Look into using NMI API for SPARC NMIs
- No plan for GICv2 support
- Prior to GICv3, CPU interface is memory mapped
  - Slower accesses than for system register

arm

Next steps

- Use NMI for more features on Arm64
  - Backtrace, CPU stop IPI: requires moving IPIs to use IRQ framework
  - Hard lockup detector: need some rework of $\mathrm{arm\_pmu}$ driver
- Look into using NMI API for SPARC NMIs
- No plan for GICv2 support
- Prior to GICv3, CPU interface is memory mapped
  - Slower accesses than for system register
  - Need access to per-cpu base address from $\mathrm{local\_irq\_}*()$

Next steps

- Use NMI for more features on Arm64
  - Backtrace, CPU stop IPI: requires moving IPIs to use IRQ framework
  - Hard lockup detector: need some rework of $\mathrm{arm\_pmu}$ driver
- Look into using NMI API for SPARC NMIs
- No plan for GICv2 support
- Prior to GICv3, CPU interface is memory mapped
  - Slower accesses than for system register
  - Need access to per-cpu base address from $\mathrm{local\_irq\_}*()$
  - Technically do-able, but not necessarily clean nor interesting/usable

Thanks!

Questions

arm