PSIC: Priority-Strict Multi-Core IRQ Processing

ISORC’22

Malte Bargholz, Christian Dietrich, Daniel Lohmann

May 18, 2022
Prioritized Interrupt Delivery

External Events (Interrupts)
- Device requests interruption
- Controller dispatches IRQ
- ISRs process and complete request
Prioritized Multi-Core Interrupt Delivery

- **External Events (Interrupts)**
  - Device requests interruption
  - Controller dispatches IRQ
  - ISRs process and complete request

- **Better Hardware**
  - \( m > 1 \): Embedded multi-cores
  - \( n > m \) ⇒ Still not enough CPUs
  - Priority-aware processing order?
**Partitioned or Global Delivery?**

### Partitioned Delivery

![Diagram of Partitioned Delivery]

- Simple in hardware
- High Cache locality
  - Latency despite resources
  - Less flexibility

---

**More complex hardware**

- More cache interference
- Use all available resources
- Load global event distribution

**Global and Semi-Partitioned scheduling require (some)**

- System-wide event distribution → Overheads in real-world system software.
**Partitioned or Global Delivery?**

**Partitioned Delivery**

- Simple in hardware
- High Cache locality
  - Latency despite resources
  - Less flexibility

**Global Delivery**

- More complex hardware
- More cache interference
- Use all available resources
- Offload global event distribution

Notes:

- **CPU 0** Prio ∞
- **CPU 1** Prio 0
- **CPU 2** Prio 0
- **IRQ 0** Prio 10
- **IRQ 1** Prio 15
- **IRQ 2** Prio 3
- **IRQ Controller**

**IRFs**

- **topm(I₀, . . . , Iₙ)**
- **ISR / 10**
Partitioned or Global Delivery?

**Partitioned Delivery**

- Simple in hardware
- High Cache locality
  - Latency despite resources
  - Less flexibility

---

**Global Delivery**

- More complex hardware
- More cache interference
- Use all available resources
- Offload global event distribution

Global and Semi-Partitioned scheduling require (some) **system-wide event distribution** → Overheads in real-world system software.
Partitioned or Global Delivery?

**Partitioned Delivery**

- CPU 0: Prio ∞
- CPU 1: Prio 0
- CPU 2: Prio 0
- IRQ 0: Prio 10
- IRQ 1: Prio 15
- IRQ 2: Prio 3
- Controller: /Lightning

**Global Delivery**

- CPU 0: Prio ∞
- CPU 1: Prio 0
- CPU 2: Prio 0
- IRQ 0: Prio 10
- IRQ 1: Prio 15
- IRQ 2: Prio 3
- Controller: /Lightning

### Global Priority-Strict Interrupt Processing

A system currently processes IRQs **priority strict**, iff its $m$ CPUs execute the top-$m$ ISRs.

Only temporary and **bounded** priority-strictness violations are tolerable for a real-time system, yet they are **undesirable**.

- **Less flexibility**
- **Offload global event distribution**

Global and Semi-Partitioned scheduling require (some) **system-wide event distribution** → Overheads in real-world system software.
■ Motivation

■ Problems with Current Interrupt Controllers

■ The PSIC Interrupt Controller

■ Evaluation

■ Summary
Motivation

Problems with Current Interrupt Controllers

The PSIC Interrupt Controller

Evaluation

Summary
Global Delivery: Any IRQ on any CPU

- **Infineon Aurix**: Statically configured IRQ-CPU pairs
  ⇒ No global interrupt dispatching

- **NXP MPC5676 (Power)**: Both CPUs are notified
  ⇒ Software must synchronize ISR execution.
Global Delivery: Any IRQ on any CPU

- **Infineon Aurix**: Statically configured IRQ-CPU pairs
  ⇒ No global interrupt dispatching

- **NXP MPC5676 (Power)**: Both CPUs are notified
  ⇒ Software must synchronize ISR execution.

- **ARM GiC and RISC-V PLIC** give weak guarantees

- Multiple CPUs with **sufficiently low** priority can be interrupted

  ⇒ Software must synchronize ISR execution
Global Delivery: Any IRQ on any CPU

- **Infineon Aurix**: Statically configured IRQ-CPU pairs ⇒ No global interrupt dispatching
- **NXP MPC5676 (Power)**: Both CPUs are notified ⇒ Software must synchronize ISR execution.
- **ARM GiC and RISC-V PLIC** give weak guarantees

Multiple CPUs with *sufficiently low* priority can be interrupted
⇒ Software must synchronize ISR execution
Lowest-Priority IRQ Delivery

**Do we only interrupt the lowest-priority CPU?**

- **ARM GiC and RISC-V PLIC**
  - Multiple CPUs can be interrupted and suffer from the interference
  - Inter-CPU synchronization through atomic claim operation
Lowest-Priority IRQ Delivery

Do we only interrupt the lowest-priority CPU?

- **ARM GiC and RISC-V PLIC**
  - Multiple CPUs can be interrupted and suffer from the interference
  - Inter-CPU synchronization through atomic claim operation

- **Intel (IO)APIC supports Lowest-Prio Delivery Mode**

  - Prioritized delivery and priority-check is decoupled
  - Delivered IRQ stuck in local APIC ⇒ Priority Inversion
Lowest-Priority IRQ Delivery

Do we only interrupt the lowest-priority CPU?

- ARM GiC and RISC-V PLIC
  - Multiple CPUs can be interrupted and suffer from the interference
  - Inter-CPU synchronization through atomic claim operation
- Intel (IO)APIC supports Lowest-Prio Delivery Mode

Prioritized delivery and priority-check is decoupled
Delivered IRQ stuck in local APIC ⇒ Priority Inversion
Lowest-Priority IRQ Delivery

Do we only interrupt the lowest-priority CPU?

- **ARM GiC and RISC-V PLIC**
  - Multiple CPUs can be interrupted and suffer from the interference
  - Inter-CPU synchronization through atomic claim operation

- **Intel (IO)APIC supports Lowest-Prio Delivery Mode**

  Prioritized delivery and priority-check is decoupled
  Delivered IRQ stuck in local APIC ⇒ Priority Inversion
Do we **always service** the highest-priority IRQs?

- Priority strictness requires **Nested Interruptions**
  - Interrupted ISRs stick to the CPU, buried in the stack
  - Systematic priority-strictness violation
Do we always service the highest-priority IRQs?

- Priority strictness requires Nested Interruptions
  - Interrupted ISRs stick to the CPU, buried in the stack
  - Systematic priority-strictness violation
Priority strictness requires **Nested Interruptions**
- Interrupted ISRs stick to the CPU, buried in the stack
- Systematic priority-strictness violation
- ⇒ **ISR Migration is mandatory**
Do we always service the highest-priority IRQs?

Priority strictness requires **Nested Interruptions**
- Interrupted ISRs stick to the CPU, buried in the stack
- Systematic priority-strictness violation
- $\Rightarrow$ ISR Migration is mandatory

Controller must decouple IRQ delivery and completion
- **ARM GiC**: IRQ completion and drop of priority are atomic
- **MCP5676/(IO) APIC**: Unmigratable hardware state
## State-of-the-Art Interrupt Controllers

<table>
<thead>
<tr>
<th></th>
<th>ARM GIC</th>
<th>Aurix IR</th>
<th>MPC5676</th>
<th>I/O APIC</th>
<th>PLIC</th>
<th>PSIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global IRQ Delivery</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Lowest-Priority Delivery</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IRQ Migration</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**TABLE I: Feature Matrix of available interrupt controllers**
Motivation

Problems with Current Interrupt Controllers

The PSIC Interrupt Controller

Evaluation

Summary
**PSIC: Priority-Strict Interrupt Controller**

- Multi-Core Interrupt Controller for real-time systems
  - **Global Delivery** of the highest-priority IRQ to the lowest-priority CPU
  - PSIC reconsiders decisions if CPU-priorities change
  - Simple **ISR migration**: No CPU-local hardware state

- PSIC is (near) drop-in replacement for RISC-V’s PLIC
  - Compatible memory-mapped IO interface
  - PSIC manages CPU and IRQ priorities
  - CPU-local interface: trigger, claim, redeliver, complete
- CPU-local message boxes
- Deliver one IRQ per cycle
- SW takes responsibility after claim

- Priority changes invalidate mailboxes
- Retract invalidated decisions
- Global IRQ-Source blocking
PSIC: System-Software Integration

- ISR migration requires CPU-local (software-)state migration
  - ISRs become light-weight threads, trampoline performs switch
  - PSIC handles ISR ownership, no global synchronization
  - Software induces only bounded priority-strictness violations

```c
context_t ctx[MAX_IRQ];
irq_t active[MAX_CPU];

void PSIC_Trampoline() {
    cpu_t id = getCPU();

    // Save PREV Context
    irq_t PREV = active[id];
    save(&ctx[PREV]);

    // Claim NEXT and Redeliver PREV
    irq_t NEXT = PSIC.claim();
    if (prev != 0)
        PSIC.redeliver(prev);

    // Load new Context
    active[id] = NEXT;
    load(&context[NEXT]);
}

void ISR_3() {
    enable_int();
    // ... workload ...
    disable_int();

    setup(&ctx[3], &ISR_3);
    PSIC.complete(3);
    PSIC.setCPUPrior(0);
}
```
Motivation

Problems with Current Interrupt Controllers

The PSIC Interrupt Controller

Evaluation

Summary
Prototype and Hardware Cost

- Integrated Prototype with Rocket RISC-V Generator
  - Configurable: IRQ sources, CPU interfaces, priority width
  - Synthesized for Xilinx Zynq7000 FPGA (XC7X020), 100Mhz

(a) By #CPU Interfaces (#IRQ=32)

(b) By #IRQ Sources (#CPU=8)
Prototype and Hardware Cost

- Integrated Prototype with Rocket RISC-V Generator
  - Configurable: IRQ sources, CPU interfaces, priority width
  - Synthesized for Xilinx Zynq7000 FPGA (XC7X020), 100Mhz

Maximum Clock Frequency

(c) Maximum clock frequency by #IRQs after routing for a XC7X020 FPGA. (#CPU=4)
Parallel cyclic Test with periodic ($t_i$) IRQ

<table>
<thead>
<tr>
<th></th>
<th>CPU1</th>
<th>CPU2</th>
<th>CPU3</th>
<th>CPU4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_i = 200$</td>
<td>PLIC</td>
<td>5489</td>
<td>5489</td>
<td>5487</td>
</tr>
<tr>
<td></td>
<td>PSIC</td>
<td>5475</td>
<td>5243</td>
<td>5243</td>
</tr>
<tr>
<td>$t_i = 50$</td>
<td>PLIC</td>
<td>6267</td>
<td>6261</td>
<td>6253</td>
</tr>
<tr>
<td></td>
<td>PSIC</td>
<td>6228</td>
<td>5247</td>
<td>5247</td>
</tr>
</tbody>
</table>

TABLE III: Time taken ($\mu$s) for one loop with $C = 65535$
Motivation

Problems with Current Interrupt Controllers

The PSIC Interrupt Controller

Evaluation

Summary
Summary

- **Global/Semi-partitioned scheduling requires** **Global Event Delivery**
  - Current multi-core controllers **do not support** priority-strict delivery
  - Global delivery, Lowest-priority delivery, ISR migration

- **PSIC: Priority-strict IRQ delivery onto multiple cores**
  - Deliver one IRQ per cycle to a CPU-local mailbox
  - Retract and redeliver IRQs on CPU-priority change
  - Replacement for RISC-V’s platform-Level interrupt controller

- **Prototype Implementation** for Rocket Chip Generator
  - Complexity and critical path driver: Max-IRQ selector
  - Reasonable overheads for real-world–sized systems