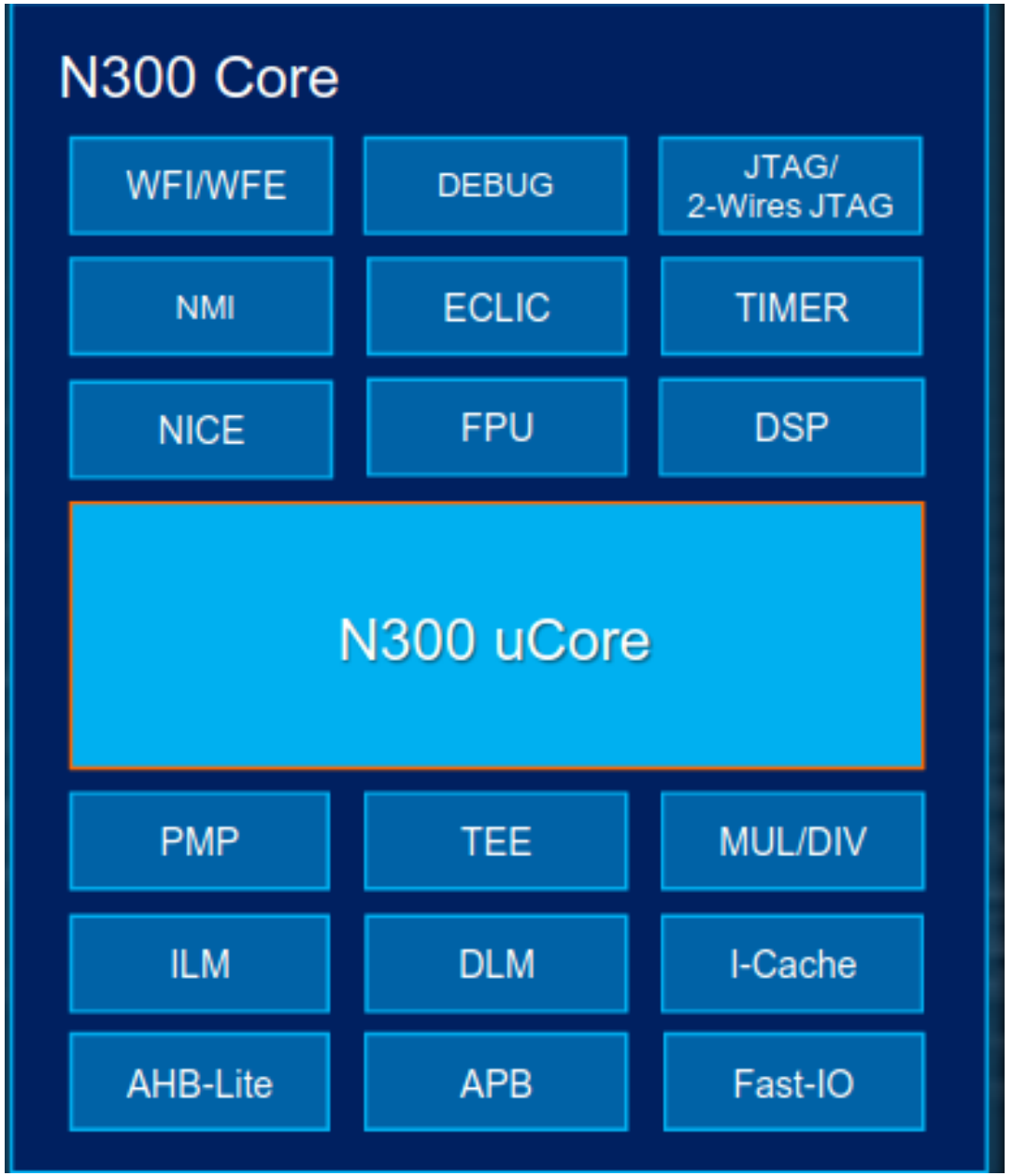
ISCAS 中国科学院软件研究所 Institute of Software Chinese Academy of Sciences | 智能软件研究中心 Intelligent Software Research Center | 芯来科技 NUCLEI

Support Nuclei SoC in QEMU RISC-V Emulation

Gao Zhiyuan, Seoul National University
Wang Jungqiang, PLCT lab, ISRC-CAS

Nuclei Products and Offerings

- We chose Nuclei HummingBird (N307),
  - GPIO, UART0, UART1
  - QSPI0, QSPI1, QSPI2
  - PWM0, PWM1, PWM2
  - I2C, No MMU
- An alternative for ARM Cortex-M3/M4/M4F/M33.



What are needed to support Nuclei

- RISC-V CPU implementations: target/riscv/cpu.c overwriting configs for spec
- Extended customized CSR registers
- ECLIC: Enhanced Core Local Interrupt Controller
- Timer
- UART
- GPIO
- NMI: non-maskable interrupt
- …

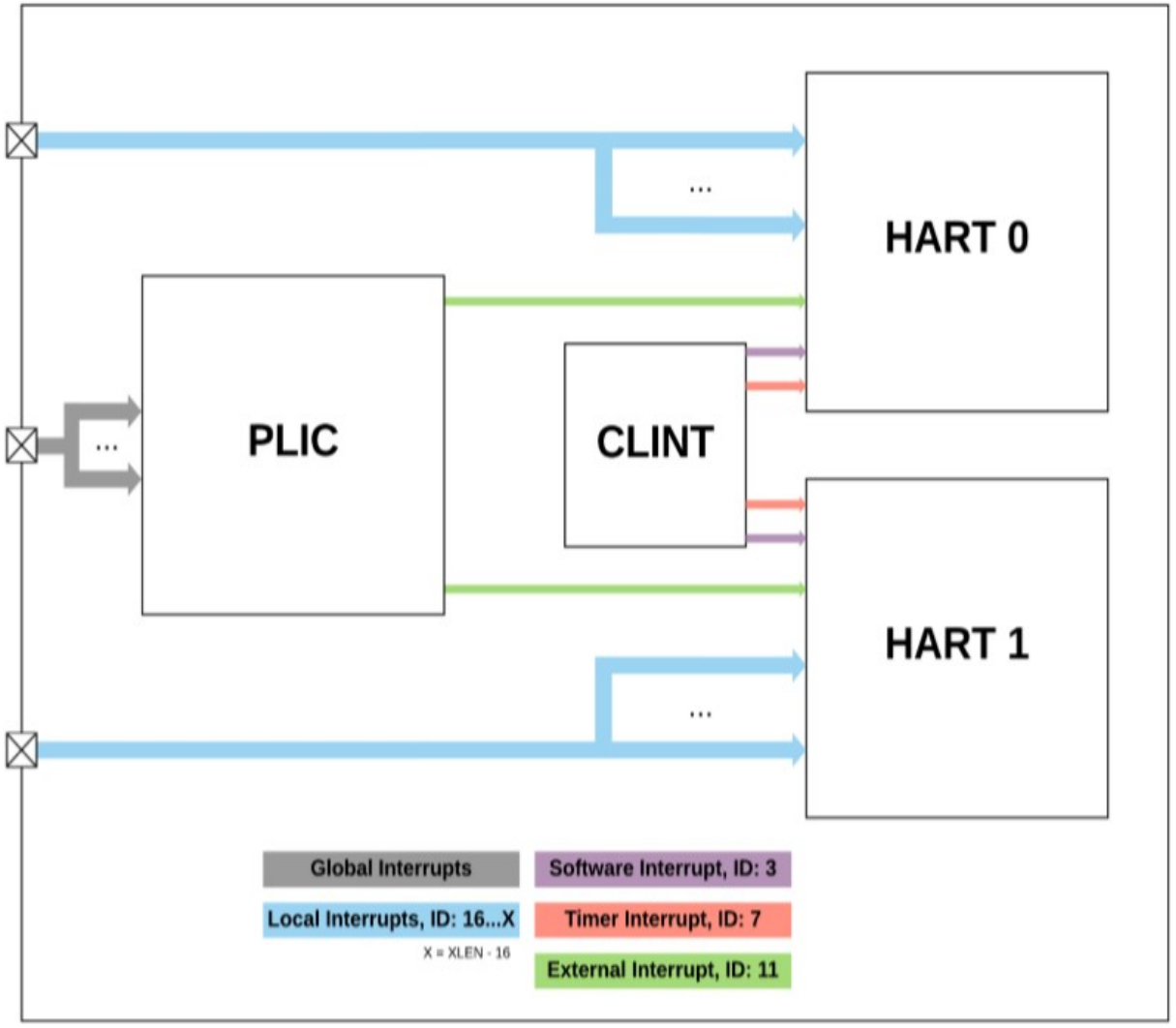
Interrupt Controllers

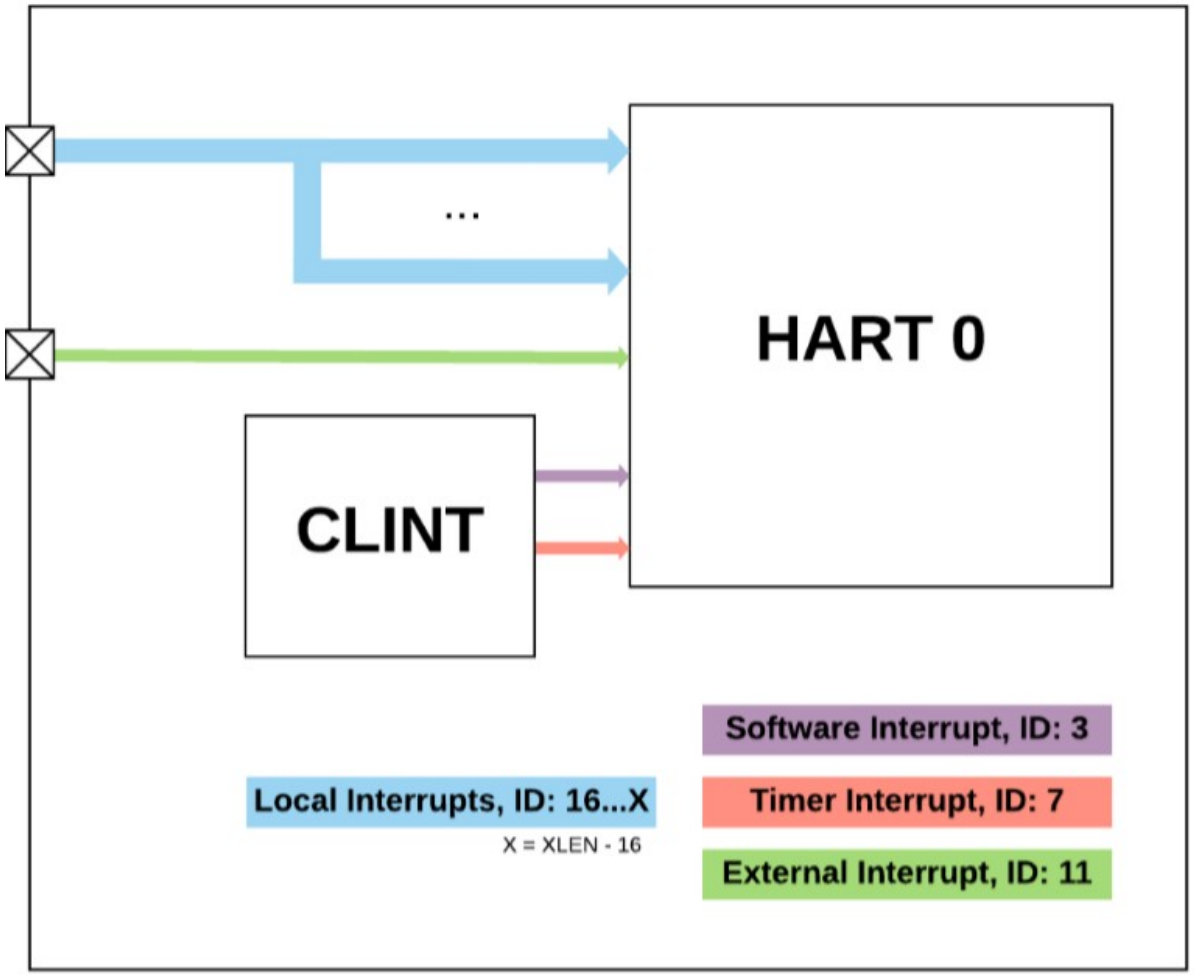
- CLIC: Core Local Interrupt Controller (SiFive)
- CLINT: Core Local Interrupter (SiFive)
- PLIC: Platform Level Interrupt Controller (SiFive)
- ECLIC: Enhanced Core Local Interrupt Controller (Nuclei)

Cores

- CLIC only adopted by SiFive E2
- series and S2 series
- Other SiFive cores support
- CLINT and PLIC interrupt controller
- ECLIC: only in Nuclei Cores

	E Cores	S Cores
	32-bit embedded cores MCU, edge computing, AI, IoT	**64-bit embedded cores** Storage, AR/VR, machine learning
3/5 Series	**E3 Series**	**S5 Series**
Efficient performance: 5–6-stage, single-issue pipeline	**E34** E31 features + single-precision floating point	**S54** S51 features + double-precision floating point
	E31 Balanced performance and efficiency	**S51** Low-power 64-bit MCU core
2 Series	**E2 Series**	**S2 Series**
Power & area optimized: 2–3-stage, single-issue pipeline, as small as 13.5k gates	**E24** E21 features + single-precision floating point	**S21** Area optimized 64-bit processor
	E21 E20 features + User Mode, Atomics, Multiply, TIM	
	E20 Our smallest, most efficient core	


HART 0
HART 1
PLIC
CLINT
...
Global Interrupts
Local Interrupts, ID: 16...X
X = XLEN - 16
Software Interrupt, ID: 3
Timer Interrupt, ID: 7
External Interrupt, ID: 11


HART 0
CLINT
...
Local Interrupts, ID: 16...X
X = XLEN - 16
Software Interrupt, ID: 3
Timer Interrupt, ID: 7
External Interrupt, ID: 11

CLIC vs. CLINT vs. ECLIC

- CLINT:
  - Fixed priority scheme based on source id
  - Only software and timer interrupts. Other interrupts are wired directly to the CPU.
- CLIC:
  - Programmable interrupt levels, priorities
  - Support nested interrupts (preemption)
- ECLIC:
  - 4096 interrupt sources
  - Configurable priority, enable/disable, level-triggered or edge-triggered
  - Nested Interrupts and tail-chaining

ECLIC Implementation

- ECLIC class init
- Nuclei device create a ECLIC device
  - register properties (e.g. num_sources, size)
  - connect to sysbus
- ECLIC device realize
  - Allocate space for registers per input source (e.g. clicintattr, clicintie, clicintip)
  - connect eclic with GPIO/UART/Timer for related interrupts
- When an interrupt happens,
  - IRQ is added to pending array of irqs
  - Update the array of IRQs, deal with nested or tail chaining cases

Registers in ECLIC

- Registers per interrupt source
  - clicintip: the interrupt polarity, deciding whether IRQs are triggered together with clicintattr
  - clicintie: enables or disables the interrupt source
  - clicintattr: defines a level-triggered, rising-edge triggered, or falling-edge triggered IRQ
  - clicintctl: contains priority and level values

- Registers in ECLIC
  - cliccfg: configures effective bits within clicintctl for interrupt priority calculation
  - clicinfo: read-only register defining hardware version, # of sources, effective bits in clicintctl
  - mth: defining the level threshold, where lower-leveled interrupts are disabled

```c
static void nuclei_eclic_update_intip(NucLeiECLICState *eclic, int irq, int new_intip)
{
    eclic->clicintip[irq] = !!new_intip;
    nuclei_eclic_next_interrupt(eclic);
}
```

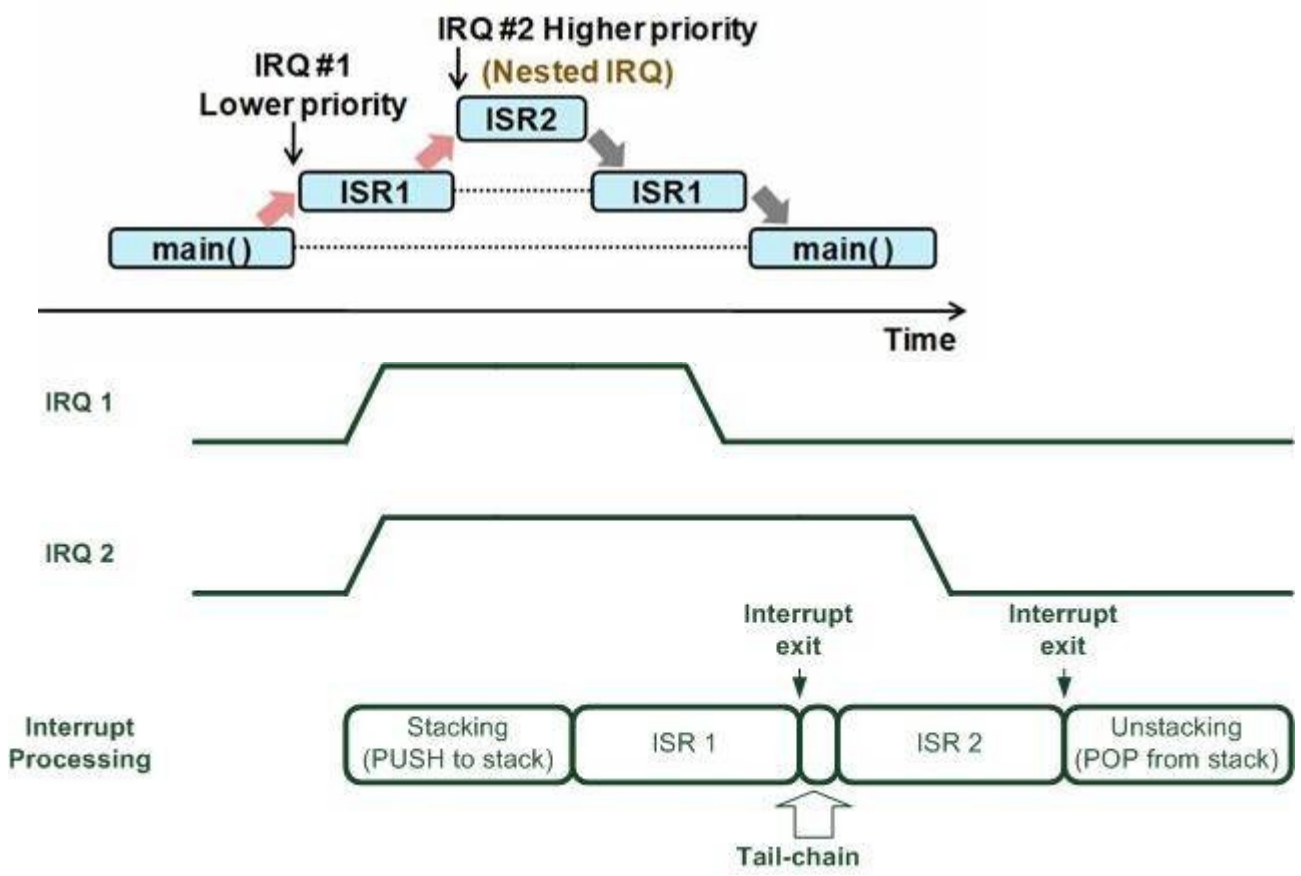
```c
static void nuclei_eclic_next_interrupt(NucLeiECLICState *eclic)
{
    RISCVCPU *cpu = RISCV_CPU(qemu_get_cpu(0));

    ECLICActiveInterrupt *active = eclic->active_list;
    size_t active_count = eclic->active_count;
    int level = 0, priority = 0;

    while (active_count) {
        clicintctl_decode(eclic, active->clicintctl, &level, &priority);
        if (eclic->clicintip[active->irq]) {
            riscv_cpu_eclic_interrupt(cpu, active->irq  | level<<12);
            return;
        }
        /* check next enabled interrupt */
        active_count--;
        active++;
    }

    /* clear pending interrupt for this hart */
    riscv_cpu_eclic_interrupt(cpu, -1);
}
```

Nested Interrupt and Tail-chaining



- **Nested Interrupt: newly arrived interrupt (IRQ#2) has a higher priority. ISR restores execution after IRQ #1 finishes.**
- **Tail-chaining: IRQ #2 has a lower priority. IRQ #1 passes the context to IRQ #2, so the context saving/restoring procedure can be saved between IRQ #1 and IRQ #2**