Interrupts

Krste Asanović
UC Berkeley & SiFive Inc.
krste@berkeley.edu

4th RISC-V Workshop
MIT CSAIL, Cambridge, MA
July 12, 2016
Interrupt Uses in Different Applications

- High-performance Unix-like systems
  - Interrupt handling small fraction of processing time
    - Fast cores, smart devices
  - Minimal interrupt handler
  - Scheduling in software

- Low/mid embedded systems
  - Interrupt handling significant fraction of processor time
    - Slow cores, dumb devices
  - Significant fraction of code in handlers
  - Interrupt controller acts as task scheduler

- High-performance real-time systems
  - Can’t waste time on interrupt overhead
  - Handlers poll I/O devices with regular heartbeat

- And everything inbetween
RISC-V Interrupt Design Goals

- Simplicity
- Support all kinds of platforms from microcontrollers to virtualized servers
- Enable tradeoffs between performance and implementation cost
- Flexibility to support specialized needs
Categorizing Sources of RISC-V Interrupts

- **Local Interrupts**
  - Directly connected to one hart
  - No arbitration between harts to service
  - Determine source directly through xcause CSR
  - Only two standard local interrupts (software, timer)

- **Global (External) Interrupts**
  - Routed via Platform-Level Interrupt Controller (PLIC)
  - PLIC arbitrates between multiple harts claiming interrupt
  - Read of memory-mapped register returns source
### Machine Interrupt Pending CSR (\texttt{mip})

*(Add Non-Standard Local Interrupts Here)*

<table>
<thead>
<tr>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIRI</td>
<td>MEIP</td>
<td>HEIP</td>
<td>SEIP</td>
<td>UEIP</td>
<td>MTIP</td>
<td>HTIP</td>
<td>STIP</td>
<td>UTIP</td>
<td>MSIP</td>
<td>HSIP</td>
<td>SSIP</td>
<td>USIP</td>
</tr>
</tbody>
</table>

- \texttt{mip} reflects pending status of interrupts for hart
- Separate interrupts for each supported privilege level (M/H/S/U)
- User-level interrupt handling (“N”) optional feature when U-mode present (discussed later)
Platform-Level Interrupt Controller (PLIC)
Software Interrupts

- **MSIP**
  - Only writeable in machine-mode via memory-mapped control register (mapping is platform-specific)
  - One hart can write to different hart’s MSIP register
  - Mechanism for inter-hart interrupts

- **HSIP, SSIP, USIP**
  - Hart can only write bit xSIP in own *mip* register when running at privilege mode x or greater

- App/OS/Hypervisor can only perform inter-hart interrupts via ABI/SBI/HBI calls
  - Destination virtual hart might be descheduled
  - Interrupts virtualized by M-mode software using MSIP
Timer Interrupts

- **MTIP**
  - Single 64-bit real-time hardware timer and comparator in M-mode
  - MTIP set when $\text{mtime} \geq \text{mtimecmp}$
  - MTIP cleared by writing new $\text{mtimecmp}$ value

- **HTIP, STIP, UTIP**
  - M-mode multiplexes single hardware timer and comparator for lower-privilege modes on same hart
  - ABI/SBI/HBI calls to set up timer
  - M-mode software writes/clears HTIP/STIP/UTIP

- Most systems will also have other hardware timers attached via PLIC etc.
## Machine Interrupt Enable CSR (mie)

<table>
<thead>
<tr>
<th>XLEN-1</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>WPRI</td>
<td>MEIE</td>
<td>HEIE</td>
<td>SEIE</td>
<td>UEIE</td>
<td>MTIE</td>
<td>HTIE</td>
<td>STIE</td>
<td>UTIE</td>
<td>MSIE</td>
<td>HSIE</td>
<td>SSIE</td>
<td>USIE</td>
<td></td>
</tr>
<tr>
<td>XLEN-12</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **mie** mirrors layout of **mip**
- Provides per-interrupt enables

![Figure 3.11: Machine interrupt-enable register (mie).](image-url)

HTIP bits may be written by M-mode software to deliver timer interrupts to lower privilege levels. User, supervisor and hypervisor software may clear the UTIP, STIP and HTIP bits with calls to the AEE, SEE, or HEE, respectively.

There is a separate timer interrupt-enable bit, named MTIE, HTIE, STIE, and UTIE for M-mode, H-mode, S-mode, and U-mode timer interrupts respectively.

Each lower privilege level has a separate software interrupt-pending bit (HSIP, SSIP, USIP), which can be both read and written by CSR accesses from code running on the local hart at the associated or any higher privilege level. The machine-level MSIP bits are written by accesses to memory-mapped control registers, which are used by remote harts to provide machine-mode interprocessor interrupts. Interprocessor interrupts for lower privilege levels are implemented through ABI, SBI or HBI calls to the AEE, SEE or HEE respectively, which might ultimately result in a machine-mode write to the receiving hart's MSIP bit. A hart can write its own MSIP bit using the same memory-mapped control register.

We only allow a hart to directly write its own HSIP, SSIP, or USIP bits when running in appropriate mode, as other harts might be virtualized and possibly descheduled by higher privilege levels. We rely on ABI, SBI, and HBI calls to provide interprocessor interrupts for this reason.

Machine-mode harts are not virtualized and can directly interrupt other harts by setting their MSIP bits, typically using uncached I/O writes to memory-mapped control registers depending on the platform specification.

The MEIP, HEIP, SEIP, UEIP bits correspond to external interrupt-pending bits for machine, hypervisor, supervisor, and user external interrupts, respectively. These bits are read-only and are set and cleared by a platform-specific interrupt controller, such as the standard platform-level interrupt controller specified in Chapter 6. There is a separate external interrupt-enable bit, named MEIE, HEIE, SEIE, and UEIE for M-mode, H-mode, S-mode, and U-mode external interrupts respectively.

The non-maskable interrupt is not made visible via the **mip** register as its presence is implicitly known when executing the NMI trap handler.

For all the various interrupt types (software, timer, and external), if a privilege level is not supported, the associated pending and interrupt-enable bits are hardwired to zero in the **mip** and **mie** registers respectively. Hence, these are all effect WARL fields.

Implementations can add additional platform-specific machine-level interrupt sources to the high bits of these registers, though the expectation is that most external interrupts will be routed through the platform interrupt controller and be delivered via MEIP.

An interrupt \( i \) will be taken if bit \( i \) is set in both **mip** and **mie**, and if interrupts are globally enabled.

By default, M-mode interrupts are globally enabled if the hart’s current privilege mode is less than M, or if the current privilege mode is M and the MIE bit in the **mstatus** register is set.
Interrupts in \textit{mstatus}

- Only take a pending interrupt for privilege mode $x$ if $x\text{IE}=1$ and running in mode $x$ or greater
- Interrupts always disabled for privileges less than current level
All interrupts trap to M-mode by default

- **mcause** CSR indicates which interrupt occurred
- M-mode can redirect to other privilege level by:
  - set up target interrupt and privilege stack
  - copy mepc to hepc/sepc/uepc respectively
  - copy mcause to hcause/scause/ucause
  - set mepc to target trap vector
  - set MPP to target privilege level, MPIE to false
  - execute mret

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Exception Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>User software interrupt</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Supervisor software interrupt</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Hypervisor software interrupt</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>Machine software interrupt</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>User timer interrupt</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>Supervisor timer interrupt</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>Hypervisor timer interrupt</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>Machine timer interrupt</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>User external interrupt</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>Supervisor external interrupt</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>Hypervisor external interrupt</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>Machine external interrupt</td>
</tr>
<tr>
<td>1</td>
<td>≥12</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
Optional Interrupt Handler Delegation

- Can delegate interrupt (and exception) handling to lower privilege level to reduce overhead
- `mideleg` has same layout as `mip`
- If a bit is set in `mideleg` then corresponding interrupt delegated to next lowest privilege level (H, S, or U)
- Can be delegated again using `hideleg/sideleg`
- Once delegated, the interrupt will not affect current privilege level (MIE setting ignored)
Platform-Level Interrupt Controller (PLIC)
PLIC Conceptual Block Diagram

Interrupt 1 Signals

Gateway
Interrupt Request

IP Priority

Interrupt 2 Signals

Gateway
Interrupt Request

IP Priority

PLIC Gateways

IE

 giám

Max Pri.
Threshold
Max ID

Interrupt Notification
Interrupt ID

To Target 0

To Target 1

PLIC Core

Interrupt ID

Interrupt Notification
PLIC Interrupt Gateways

Convert from external interrupt signal/message encoding to internal PLIC interrupt request, e.g.,
- Level-triggered gateways
- Edge-triggered gateways
- Message-signaled gateways
- XXX gateways in future

Will not forward a new request to PLIC core unless previous request’s handler has signaled completion
- Level-triggered will issue new PLIC interrupt request if level still asserted after completion signaled
- Edge-triggered/message-signaled could queue requests
PLIC Per-Interrupt ID and Priority

- Each interrupt has ID and priority
- Interrupt IDs are integers from 1...N
- ID of zero means “no interrupt”
- Priorities are integers, larger number is higher priority
- Priority zero means “never interrupt”
- Priorities can be fixed or variable
  - Degenerate case, all are fixed at “1”.
- Ties broken by ID (lower ID is higher priority)
PLIC Per-Target Information

- Each target has vector of interrupt enables
- Each target has priority threshold
- Only interrupts with priority above threshold will cause interrupt
- Set threshold to 0, has no effect
  - Minimal implementation, hardwire threshold to zero
- Set threshold to MAX_PRI, then all interrupts masked
- Interrupt notifications asserted at target if enabled interrupt is above threshold
  - Notifications can take arbitrary time to arrive at target
PLIC Conceptual Block Diagram

Interrupt 1 Signals

Gateway

Interrupt Request

IP

Priority

Interrupt 2 Signals

Gateway

Interrupt Request

IP

Priority

PLIC Gateways

PLIC Core

Interrupt Notification

Interrupt ID

Max Pri.

Threshold

Max ID

EIP

IE

>?

To Target 0

To Target 1

Max Pri.

Threshold

Max ID

EIP

IE

>?
PLIC Claim/Completion

- Interrupted hart context *claims* interrupt from PLIC with read of memory-mapped register
- PLIC returns highest priority active interrupt for that hart
- Can return 0 if no active interrupts remain
  - Other hart might have claimed interrupt earlier

- Hart signals completion to gateway after handler finishes
PLIC Core Atomic Actions

- **Write Register:** A message containing a register write request is dequeued. One of the internal registers is written, where an internal register can be a priority, an interrupt-enable (IE), or a threshold.

- **Accept Request:** If the IP bit corresponding to the interrupt source is clear, a message containing an interrupt request from a gateway is dequeued and the IP bit is set.

- **Process Claim:** An interrupt claim message is dequeued. A claim-response message is enqueued to the requester with the ID of the highest-priority active interrupt for that target, and the IP bit corresponding to this interrupt source is cleared.

Implementations can perform one action over many cycles, or many actions per cycle, provided behavior agrees with some sequence of these actions.
PLIC Interrupt Flow

Interrupt Source | Gateway | PLIC Core | Target
---|---|---|---
Interrupt Signalled | Interrupt Request | Interrupt Notification | Interrupt Claim
Interrupt Request | Claim Response | Interrupt Completion | Handler Running

Next Request
PLIC Interrupt Preemption/Nesting

- Preemption and nesting are function of the target core, not the PLIC
- Need a different hart context to receive nested interrupt
- Each standard RISC-V privilege level can provide one level of preemption/nesting
  - M-mode interrupt will preempt S-mode handler on hart
- Can add additional hart contexts to core to support nested interrupt handling, with per-cores rules on preemption/priority
PLIC Access Control

- PLIC registers are memory mapped, platform-specific
- M-mode-only access to interrupt enables and priorities
- Lower privilege modes only access claim, completion, and threshold registers
  - can only signal completion for inputs for which they’re enabled
# SiFive Freedom Platform PLIC Mapping

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4000_0000</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x4000_0004</td>
<td>source 1 priority</td>
</tr>
<tr>
<td>0x4000_0008</td>
<td>source 2 priority</td>
</tr>
<tr>
<td></td>
<td>Reserves 1023 priority</td>
</tr>
<tr>
<td>0x4000_1000</td>
<td>Start of pending array (read-only)</td>
</tr>
<tr>
<td></td>
<td>End of pending array</td>
</tr>
<tr>
<td>0x4000_1800</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x4000_1FFF</td>
<td></td>
</tr>
<tr>
<td>0x4000_2000</td>
<td>target 0 enables</td>
</tr>
<tr>
<td>0x4000_2080</td>
<td>target 1 enables</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>0x401E_FF80</td>
<td>target 15871 enables</td>
</tr>
<tr>
<td>0x401F_0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>0x401F_FFFC</td>
<td></td>
</tr>
<tr>
<td>0x4020_0000</td>
<td>target 0 priority threshold</td>
</tr>
<tr>
<td>0x4020_0004</td>
<td>target 0 claim/complete</td>
</tr>
<tr>
<td>0x4020_1000</td>
<td>target 1 priority threshold</td>
</tr>
<tr>
<td>0x4020_1004</td>
<td>target 1 claim/complete</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>0x43FF_F000</td>
<td>target 15871 priority threshold</td>
</tr>
<tr>
<td>0x43FF_F004</td>
<td>target 15871 claim/complete</td>
</tr>
</tbody>
</table>

Table 5.1: SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are supported.

- **Machine-mode only**
- **Target per page to simplify protection**
Interrupt/Trap Vectors

- By default, single entry point per privilege level:
  - mtvec/htvec/stvec/utvec
- Useful in many systems where common handling code used, with bulk of work scheduled later
  - “Interrupt is data”
- Can optionally add differentiated entry points per trap type for embedded applications
  - “Interrupt is control”
### User-Level Interrupts “N”

- Natural extension of interrupt model into user permissions
- Adds user CSRs, and *uret* instruction

<table>
<thead>
<tr>
<th>Number</th>
<th>Privilege</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td>URW</td>
<td>ustatus</td>
<td>User status register.</td>
</tr>
<tr>
<td>0x004</td>
<td>URW</td>
<td>uie</td>
<td>User interrupt-enable register.</td>
</tr>
<tr>
<td>0x005</td>
<td>URW</td>
<td>utvec</td>
<td>User trap handler base address.</td>
</tr>
<tr>
<td>0x040</td>
<td>URW</td>
<td>uscratch</td>
<td>Scratch register for user trap handlers.</td>
</tr>
<tr>
<td>0x041</td>
<td>URW</td>
<td>uepc</td>
<td>User exception program counter.</td>
</tr>
<tr>
<td>0x042</td>
<td>URW</td>
<td>ucause</td>
<td>User trap cause.</td>
</tr>
<tr>
<td>0x043</td>
<td>URW</td>
<td>ubadaddr</td>
<td>User bad address.</td>
</tr>
<tr>
<td>0x044</td>
<td>URW</td>
<td>uip</td>
<td>User interrupt pending.</td>
</tr>
</tbody>
</table>
Interrupts in Secure Embedded Systems (M, U modes)

- M-mode runs secure boot and runtime monitor
- Embedded code runs in U-mode
- Physical memory protection on U-mode accesses
- Interrupt handling can be delegated to U-mode code
- Provides arbitrary number of isolated subsystems
Questions?