RISC-V Supervisor Counter Delegation Specification (Smcdeleg/Ssccfg)

Version v1.0.0, 2024-04-01: Ratified

Table of Contents

Preamble

This document is in the Ratified state

No changes are allowed. Any desired or needed changes can be the subject of a follow-on new extension. Ratified extensions are never revised.

Copyright and license information

Contributors

This RISC-V specification has been contributed to directly or indirectly by:

- Beeman Strong <beeman@rivosinc.com>
- Atish Patra <atishp@rivosinc.com>
- Allen Baum <allen.baum@esperantotech.com>
- Greg Favor <gfavor@ventanamicro.com>
- John Hauser <jh.riscv@jhauser.us>

Chapter 1. Introduction

For a Machine-level environment, extension **Smcdeleg** (‘Sm’ for Privileged architecture and Machine-level extension, ‘cdeleg’ for Counter Delegation) encompasses all added CSRs and all behavior modifications for a hart, over all privilege levels. For a Supervisor-level environment, extension **Ssccfg** (‘Ss’ for Privileged architecture and Supervisor-level extension, ‘ccfg’ for Counter Configuration) provides access to delegated counters, and to new supervisor-level state. These extensions depend on the Zicntr and/or Zihpm extensions, and on the Sscsrind extension.

1.1. Motivation

The Zicntr extension defines a set of fixed-event counters (`cycle`, `time`, `instret`), while the Zihpm extension defines programmable counters (`hpmcounteri` and `hpmeventi`). The mcounteren CSR provides a means to make select counter CSRs readable in supervisor (S) mode, while the scounteren CSR provides a means for S-mode to further expose those selected counter CSRs as readable in user (U) mode. Counters and event selector CSRs can only be written in machine (M) mode.

In modern “Rich OS” environments, hardware performance monitoring resources are managed by the kernel, kernel driver, and/or hypervisor. Counters may be configured with differing scopes, in some cases counting events system-wide, while in others counting events on behalf of a single virtual machine or application. In such environments, the latency of counter writes has a direct impact on overall profiling overhead as a result of frequent counter writes during:

1. Sample collection, to clear overflow indication, and reload overflowed counter(s)
2. Context switch, between processes, threads, containers, or virtual machines

This extension provides a means for M-mode to allow writing select counters and event selectors from S/HS-mode. The purpose is to avert transitions to and from M-mode that add latency to these performance critical supervisor/hypervisor code sections. This extension also defines one new CSR, scountinhibit.

Indirect vs direct access to counters and event selectors was discussed at length. While a direct access method (e.g., new `shpmcounteri` CSRs) has the potential to reduce latency for performance-sensitive operations such as context switch and counter overflow handling, by avoiding the need to write an index CSR for each counter access, in practice the benefits are difficult to reap. Because the CSR number is embedded in the immediate of CSR access instructions, functions to access individual counters by index have to utilize a switch statement to jump to the instruction that accesses the chosen counter. Counters are typically accessed infrequently (say every sample, or every context switch), so this switch statement is likely to incur a branch mispredict, which will undermine the performance benefits intended by avoiding an indirect access mechanism. A static routine that accesses all counters could be crafted without branches, but with Linux perf only counters associated with active perf events are accessed.

With indirect access, branching can be avoided for all cases, with the counter index *simply written to the index register, and a static flow to read/write the associated alias register. While strong ordering between the index write and the alias register access is required, it is believed that pipelining of CSR accesses can ensure that the costs associated with this ordering are less than the cost associated with the mispredictions that result from the direct method.*

Chapter 2. Counter Delegation Enable (`menvcfg`.CDE)

Bit 60 of `menvcfg` (bit 28 of `menvcfgh`) is Counter Delegation Enable (CDE). When CDE=0, the Smcdeleg and Ssccfg extensions appear to be not implemented. When CDE=1, the behavior described in the sections below is enabled. If Smcdeleg is not implemented, CDE is read-only zero.

Chapter 3. Counter Delegation

The `mcounteren` register allows M-mode to provide the next-lower privilege mode with read access to select counters. When the Smcdeleg extension is enabled, it further allows M-mode to delegate select counters to S-mode.

The `siselect` (and `vsiselect`) index range 0x40-0x5F is reserved for delegated counter access. When a counter *i* is delegated (`mcounteren`[*i*]=1 and `menvcfg`.CDE=1), the register state associated with counter *i* can be read or written via `sireg*`, while `siselect` holds 0x40+*i*. The counter state accessible via alias CSRs is shown in Table 1 below.

Table 1. Indirect HPM State Mappings

`siselect` value	**`sireg`**	**`sireg4`**	**`sireg2`**	**`sireg5`**
0x40	`cycle`[1]	`cycleh`[1]	`cyclecfg`[14]	`cyclecfgh`[14]
0x41			*See below*	
0x42	`instret`[1]	`instreth`[1]	`instretcfg`[14]	`instretcfgh`[14]
0x43	`hpmcounter3`[2]	`hpmcounter3h`[2]	`hpmevent3`[2]	`hpmevent3h`[23]
…	…	…	…	…
0x5F	`hpmcounter31`[2]	`hpmcounter31h`[2]	`hpmevent31`[2]	`hpmevent31h`[23]

[1] Depends on Zicntr support
[2] Depends on Zihpm support
[3] Depends on Sscofpmf support
[4] Depends on Smcntrpmf support

> `hpmeventi` *represents a subset of the state accessed by the* `mhpmeventi` *register. Likewise,* `cyclecfg` *and* `instretcfg` *represent a subset of the state accessed by the* `mcyclecfg` *and* `minstretcfg` *registers, respectively. See below for subset details.*

If extension Smstateen is implemented, refer to extension Smcsrind/Sscsrind (upon which this extension depends) for how setting bit 60 of CSR mstateen0 to zero prevents access to registers `siselect`, `sireg*`, `vsiselect`, and `vsireg*` from privileged modes less privileged than M-mode, and likewise how setting bit 60 of hstateen0 to zero prevents access to `siselect` and `sireg*` (really `vsiselect` and `vsireg*`) from VS-mode.

The remaining rules of this section apply only when access to a CSR is not blocked by mstateen0[60] = 0 or hstateen0[60] = 0.

While the privilege mode is M or S and `siselect` holds a value in the range 0x40-0x5F, illegal instruction exceptions are raised for the following cases:

- attempts to access any `sireg*` when `menvcfg`.CDE = 0;
- attempts to access `sireg3` or `sireg6`;
- attempts to access `sireg4` or `sireg5` when XLEN = 64;

- attempts to access sireg* when siselect = 0x41, or when the counter selected by siselect is not delegated to S-mode (the corresponding bit in mcounteren = 0).

> *The memory-mapped mtime register is not a performance monitoring counter to be managed by supervisor software, hence the special treatment of siselect value 0x41 described above.*

For each siselect and sireg* combination defined in Table 1, the table further indicates the extensions upon which the underlying counter state depends. If any extension upon which the underlying state depends is not implemented, an attempt from M or S mode to access the given state through sireg* raises an illegal instruction exception.

If the hypervisor (H) extension is also implemented, then as specified by extension Smcsrind/Sscsrind, a virtual instruction exception is raised for attempts from VS-mode or VU-mode to directly access vsiselect or vsireg*, or attempts from VU-mode to access siselect or sireg*. Furthermore, while vsiselect holds a value in the range 0x40-0x5F:

- An attempt to access any vsireg* from M or S mode raises an illegal instruction exception.
- An attempt from VS-mode to access any sireg* (really vsireg*) raises either an illegal instruction exception if menvcfg.CDE = 0, or a virtual instruction exception if menvcfg.CDE = 1.

If Sscofpmf is implemented, sireg2 and sireg5 provide access only to a subset of the event selector registers. Specifically, event selector bit 62 (MINH) is read-only 0 when accessed through sireg*. Similarly, if Smcntrpmf is implemented, sireg2 and sireg5 provide access only to a subset of the counter configuration registers. Counter configuration register bit 62 (MINH) is read-only 0 when accessed through sireg*.

Chapter 4. Supervisor Counter Inhibit Register (`scountinhibit`)

Smcdeleg/Ssccfg defines a new `scountinhibit` register, a masked alias of `mcountinhibit`. For counters delegated to S-mode, the associated `mcountinhibit` bits can be accessed via `scountinhibit`. For counters not delegated to S-mode, the associated bits in `scountinhibit` are read-only zero.

When `menvcfg`.CDE=0, attempts to access `scountinhibit` raise an illegal instruction exception. When the Supervisor Counter Delegation extension is enabled, attempts to access `scountinhibit` from VS-mode or VU-mode raise a virtual instruction exception.

The CSR number for `scountinhibit` is 0x120.

Chapter 5. Virtualizing `scountovf`

For implementations that support Smcdeleg/Ssccfg, Sscofpmf, and the H extension, when `menvcfg`.CDE=1, attempts to access `scountovf` from VS-mode or VU-mode raise a virtual instruction exception.

Chapter 6. Virtualizing Local Counter Overflow Interrupts

For implementations that support Smcdeleg, Smcofpmf, and Smaia, the local counter overflow interrupt (LCOFI) bit (bit 13) in each of CSRs `mvip` and `mvien` is implemented and writable.

For implementations that support Smcdeleg/Ssccfg, Smcofpmf/Sscofpmf, Smaia/Ssaia, and the H extension, the LCOFI bit (bit 13) in each of `hvip` and `hvien` is implemented and writable.

By virtue of implementing `hvip`.LCOFI, it is implicit that the LCOFI bit (bit 13) in each of `vsie` and `vsip` is also implemented.

Requiring support for the LCOFI bits listed above ensures that virtual LCOFIs can be delivered to an OS running in S-mode, and to a guest OS running in VS-mode. The behavior of these bits is described in sections 5.3 (for `mvip` and `mvien`) and 6.3.2 (for `hvip` and `hvien`) in the RISC-V Advanced Interrupt Architecture specification version 1.0. It is optional whether the LCOFI bit (bit 13) in each of `mideleg` and `hideleg`, which allows all LCOFIs to be delegated to S-mode and VS-mode, respectively, is implemented and writable.

In a production "Rich OS" environment, it is expected that M-mode firmware will delegate counters to supervisor by setting bits in `mcounteren`, with `menvcfg`.CDE=1. Delegated counters should be inhibited in M-mode by setting `mhpmevent`i.MINH, `mcyclecfg`.MINH, and `minstretcfg`.MINH, while other privilege mode inhibit bits (SINH, UINH, VSINH, VUINH) should be left cleared. It is also expected that local counter overflow interrupts (LCOFIs) will be delegated to S-mode (`mideleg`.LCOFIE=1), or delivered to S-mode via other means.

If M-mode firmware opts to reserve some counters for its own use (by not delegating them), and intends to use them for interrupt-based sampling, it will not delegate LCOFIs to S-mode. Instead, it will need to leverage Smaia in order to deliver virtual LCOFIs to S-mode when an LCOFI is the result of an overflow of a delegated counter (selective delegation).

- *"Bare Metal" Configuration*

The operating system (running in S-mode) can determine which counters have been delegated by writing all ones to `scountinhibit`, then reading back the resulting value. It can then use `siselect` and `sireg*` to program the delegated counters and their associated event selectors or counter configuration registers. Unchanged is the OS's ability to allow user code to read select counters by setting bits in scounteren.

Should the OS prefer to count events per context, it can swap the counter, event selector, and counter configuration CSRs, for each counter in use, during context switch.

For sampling usages, the OS will initialize a counter with a large positive value *suitably close to overflow, and clear the associated event selector overflow (OF) bit via sireg3/sireg4. Upon counter overflow, OF will be set and an LCOFI will be pended. The LCOFI interrupt service routine (ISR) will be invoked in S-mode, and can inhibit counting for all delegated counters by writing to scountinhibit, then can read scountovf to determine which counters have overflowed. It can then re-initialize the overflowed counter(s) by writing the counter via sireg/sireg4, and clearing the OF bit via sireg2/sireg5, for each overflowed counter. It may opt to snapshot all counters, or other hart state. Finally it can resume counting, by clearing scountinhibit, before resuming workload execution.*

- *Hypervisor Configuration*

A hypervisor may use the counters as described above, and can utilize the xINH bits in the event selectors (via sireg2/sireg5) to dictate whether the counter increments during hypervisor execution, guest execution, or both.

A guest OS or nested hypervisor running in VS-mode may attempt to access performance counter resources. This extension supports a "trap and emulate" approach to allowing guest use of counters. Guest access to counters, event selectors, and counter configuration registers (via sireg) will result in a virtual instruction exception, which will trap to the hypervisor. The hypervisor can then emulate the access, which may involve utilizing a different physical counter than the one selected by the guest. Similarly, guest access to scountinhibit or scountovf will trap to HS-mode, ensuring that the hypervisor can emulate all registers that affect, or are affected by, counter behavior. Prior to Smcdeleg/Ssccfg, VS-mode access to scountovf did not trap, which resulted in a virtualization hole for hypervisors that virtualize Zicntr/Zihpm resources, since it allowed a guest direct read access to the hpmeventX.OF bits.*

More likely, a hypervisor will not indicate support for Supervisor Counter Delegation to a guest. The hypervisor thereby requires the guest to use the SBI PMU interface. Because the SBI allows multiple CSRs to be written per call, this approach should reduce the number of traps to HS-mode, and thus reduce the virtualization overhead associated with Zicntr and Zihpm use. Virtualization overhead can be further reduced when counters are delegated to HS-mode, which allows hypervisors to directly access delegated counters on guest SBI calls, rather than requiring a nested SBI call from HS-mode to M-mode.

When a guest counter overflows and pends an LCOFI, the hypervisor has two options for delivering that interrupt to the guest:

1. *LCOFIs can be selectively delegated to the guest by the hypervisor. If hideleg.LCOFI=0, an unmasked LCOFI will trap to HS-mode, where the hypervisor can determine whether it should be handled by the guest. The hypervisor can set hvien.LCOFI=hvip.LCOFI=1 in order to deliver a virtual LCOFI to VS-mode.*
2. *LCOFIs can be wholly delegated to the guest by the hypervisor. By setting hideleg.LCOFI=1, an unmasked LCOFI will trap to VS-mode.*

In either case, when the LCOFI or virtual LCOFI traps to VS-mode, the handler will acknowledge the interrupt by clearing `sip`*.LCOFI (really* `vsip`*.LCOFI).*