Date: 2020/9/08
Task Group: Fast Interrupts
Chair: Krste Asanovic
Co-Chair: Kevin Chen
Number of Attendees: ~10
Current issues on github: https://github.com/riscv/riscv-fast-interrupt/issues

First, we thanked David Horner for his meticulous review and revision of the sample assembly code in the spec. David also suggested that it would improve readability to break up code into separate modules.

#98 Name of the controller: CLIC or FLIC?

It was suggested that CLIC and CLINT are too close in name and sound.

The CLINT name has propagated and is being used widely for the original scheme, despite not having a clear definition anywhere.

It is proposed we should therefore change the name of CLIC to something that is further lexically from CLINT. One proposal is FLIC for "Fast Local Interrupt Controller".

Krste: In my opinion, CLIC name has been around for a while, and I don't think it really needs changing, and that CLINT and CLIC are reasonably distinguishable.

Kevin: The main problem is that these 2 names are NOT self-explanatory. Also, having identical words and meaning, they are too confusing for millions of new readers in the future:

Core Local INTerruptor (for CLINT)
Core Local Interrupt Controller (for CLIC)

Therefore, before ratification, we should replace them with clear self-explanatory names to avoid confusion and misuse in the future.

#95 Remove N - User Mode Interrupt from document

Given that N has not been ratified, and is not being pushed at this time, it makes sense to drop references to N extension from the CLIC spec. I would not want to repurpose bits that might be used for user-level interrupt handling at this time however.

#52 Clarify interrupt selection

Yes, both clicintie[i] and mstatus.xie should be enabled. clicintie[i] is the individual control bit while mstatus.xie is a global control bit for that particular mode (only valid in the corresponding mode).
This will be clarified in the spec later.

Created the following new issues to record postings from John Hauser in last year:

#96 Proposed reformat of cliccfg

From John Hauser, email March 10, 2019:
As currently defined, the CLIC has a single configuration byte, 'cliccfg', that determines the splitting of interrupt control settings into levels and priorities. This one byte applies to all of a hart's privilege levels (M, S, and/or U) simultaneously. I strongly believe this is suboptimal. Instead, each privilege level should be given its own separate configuration. As I'm sure you know, it's common for each privilege level to be running software from different authors, and while the different levels will often interact (through environment calls and such), each privilege level should get to choose for itself how many nested interrupt levels it must accomodate on its interrupt stack, and whether to use selective hardware vectoring. Such choices should not be dictated from above, or from below.

Splitting up 'cliccfg' implies also an adjustment to the way the 'nmbits' field operates. There will now be up to three configuration registers, which here I will call 'clicmcfg', 'clicscfg', and 'clicucfg'. The new registers can have these layouts:

clicmcfg bits field
7:6 reserved (WARL 0)
5 nmbits
4:1 nlbits[3:0]
0 nvbits

clicscfg bits field
7:6 reserved (WARL 0)
5 nmbits
4:1 nlbits[3:0]
0 nvbits

clicucfg bits field
7:5 reserved (WARL 0)
4:1 nlbits[3:0]
0 nvbits

Bit 'clicmcfg.nmbits' determines whether any interrupts can be delegated to privilege levels beyond M-mode. If 'clicmcfg.nmbits' = 0, then all interrupts are taken in M-mode, and 'clicscfg' and 'clicucfg' are ignored. Likewise, when S-mode is supported, bit 'clicscfg.nmbits' determines whether interrupts can be delegated beyond S-mode to U-mode. If 'clicmcfg.nmbits' = 1 and 'clicscfg.nmbits' = 0, then no interrupts are delegated beyond S-mode, and 'clicucfg' is again ignored.

For illustration purposes, let's assume our CLIC implements all 8 bits in each 'clicintctl[]' byte. If 'clicmcfg.nmbits' = 0, then the 8 bits are all used for the M-mode interrupt priority, split up by 'clicmcfg' fields 'nlbits' and 'nvbits', same as before. On the other hand, if 'clicmcfg.nmbits' = 1, then bit 7 of each interrupt control byte decides whether an interrupt is taken in M-mode or is delegated to the next lower privilege level.

The same process then happens again recursively for 'clicscfg', except that at the supervisor level only the lower 7 bits of an interrupt control byte are involved.

Adapting a table from the CLIC document:

---------- clicmcfg clicscfg
priv-modes .nmbits .nmbits clicintctl[i] Mode taking interrupt

M	0	none	xxxxxxxx	M, level+priority=xxxxxxxx
M/U	0	none	xxxxxxxx	M, level+priority=xxxxxxxx
M/U	1	none	0xxxxxxx	U, level+priority=xxxxxxx
M/U	1	none	1xxxxxxx	M, level+priority=xxxxxxx

M/S/U 0 - xxxxxxxx M, level+priority=xxxxxxxx
M/S/U 1 - 1xxxxxxx M, level+priority=xxxxxxx
M/S/U 1 0 0xxxxxxx S, level+priority=xxxxxxx
M/S/U 1 1 01xxxxxx S, level+priority=xxxxxx
M/S/U 1 1 00xxxxxx U, level+priority=xxxxxx

Just as before, each interrupt source gets configured to be
destined for a specific privilege level. For a given interrupt, the
configuration byte corresponding to its target privilege level, either
'clicmcfg', 'clicscfg', or 'clicucfg', determines how the remaining
interrupt control bits will be split into interrupt level, priority,
and SHV bit.

Concerning the implementation cost of this, it is important to realize
that the selection of the highest-priority enabled interrupt is not
affected by this splitting, but rather depends solely on the value
of the control byte in 'clicintctl[]', same as before. Only after
the highest-priority interrupt has been identified does it need to
be selectively split, which can be handled with a few small muxes of
the fields of 'clicmcfg', 'clicscfg', and 'clicucfg', once the target
privilege level is identified.

The CLIC memory-mapped control aperture for machine level (the "M-mode
CLIC region") should contain all three configuration bytes, 'clicmcfg',
'clicscfg', and 'clicucfg'. If there is a corresponding control
aperture for supervisor level, only 'clicscfg' and 'clicucfg' are
visible there. A user-level control aperture contains only 'clicucfg'.

Whether or not you agree with me that the implementation cost of this change is insignificant, I believe it's a necessity, for the reasons I gave earlier about separation of control.

#97 Proposed reformat of *cause CSRs

From John Hauser, 14 March 2019:
The overloading of the '*cause' CSRs with new fields has always felt awkward to me, even though I accept it was well motivated by a desire to keep the instruction counts to a minimum. Fortunately, I believe I've found a different arrangement, just as good, that avoids CLIC mode needing to modify the '*cause' registers.

In my new proposal, the complete set of CSRs that would be added with CLIC are these:

0xm07 *tvt Trap-handler vector table base address
0xm45 *istatus Interrupt status
0xm46 *maskil Masked interrupt level
0xm47 *nexti Interrupt handler address and enable modifier
0xm48 *scratchcsw Conditional scratch swap on priv mode change
0xm49 *scratchcswl Conditional scratch swap on level change

The '*tvt', '*nexti', '*scratchcsw', and '*scratchcswl' CSRs are all unchanged from before (aside from my adjusting the name and encoding of '*nxti'). I'm not sure about the exact encodings, but I've mostly kept them the same as before. The real changes are all in '*istatus', which coalesces fields that were previously in '*cause' and '*intstatus'.
I also went ahead and added '*maskil' for setting an interrupt mask level, something that was already on the group's agenda.

To keep it simple, I'll usually just use M mode as an example. Here is the proposed format for 'mistatus':

mistatus

XLEN-1 TT trap type: 0 = exception, 1 = interrupt

27:26 MPP alias of mstatus.MPP
25 MPIE alias of mstatus.MPIE
24 PTT previous trap type
23:16 PIL previous interrupt level
15:8 IL (read-only) current interrupt level
7:5 (reserved)
4 HV 1 = reading trap vector table (was 'inhv')
3:0 (reserved)

You'll recognize most of these fields from the earlier 'mcause'.

The new 'mmaskil' is a read-write 8-bit register. New interrupts can be taken in M mode only for an interrupt level greater than max('mistatus.IL', 'mmaskil'). There are no limitations on writing 'mmaskil' from M mode. On initial entry to an interrupt trap, it is guaranteed that 'mmaskil' < 'mistatus.IL', because otherwise the interrupt couldn't have been taken. Code that wants to raise the mask level (typically an interrupt handler, but could be anything) should usually save the old value first and then later restore it.

In CLIC mode, the hardware performs these extra actions on any trap entry in M mode:

```
mistatus.PTT = mistatus.TT
if (trap is an interrupt) {
    mistatus.TT = 1
    mistatus.PIL = mistatus.IL
    mistatus.IL = new interrupt level
} else {
    // trap is a synchronous exception
    mistatus.TT = 0
}
```

These actions are added on MRET:

```
if (mistatus.TT) {
    mistatus.IL = mistatus.PIL
}
```

mistatus.TT = mistatus.PTT
mistatus.PTT = 0

An interrupt handler that enables nested interrupts needs to save and restore 'mistatus' instead of 'mcause'. Note that on entry to a trap in M mode, the sign bits of 'mcause' and 'mistatus' will be the same (although this relationship does not maintain forever).

One advantage of the new scheme is that, because the '*cause' registers are left unchanged, synchronous exception handlers can be far more oblivious to CLIC mode than with the current specification. I've arranged things so that, if an exception handler wants to save state and re-enable interrupts to allow for nested traps, it can do so in the same way it would without any knowledge of CLIC mode. (Why it would want to do so is a valid question, but I see no reason to exclude the possibility gratuitously.)

I believe all the code examples in the CLIC document work the same in the new scheme, assuming 'mcause' is replaced by 'mistatus'. For the interrupt-driven C-ABI example, the sequence

```
li a0, (MMODE)<<PP|(0)<<PIL|(1)<<PIE
csrw mcause, a0 # Initialize mcause to have pp=M, pil=0, pie=1
```

must be replaced by this equivalent code:

```
li a0, (1)<<TT|(MMODE)<<PP|(1)<<PIE|(0)<<PIL
csrw mistatus, a0 # Init mistatus to have TT=1, MPP=M, MPIE=1, PIL=0
```

From Kevin Chen:
Please note that the LSB of "mcause" contains the interrupt ID (12-bit exccode) which is automatically updated by HW when a new interrupt is taken. If you don't back up mcause, this interrupt ID will be overwritten by more critical interrupts in the case of nesting interrupts. Once it is overwritten (and gone) without backup, the ISR will miss this piece of information and may not be able to complete successfully.

From John Hauser:
Except that none of the examples in the CLIC document actually looks at the bottom 12 bits of *cause for handling interrupts. When interrupts are hardware-vectored, there is usually no need for an interrupt handler to look at the interrupt number in *cause. But even when interrupts are not hardware-vectored, so long as we set up a table of interrupt handlers pointed to by *tvt, the special *nexti CSR gives us a pointer into this table without making any use of the interrupt number in *cause. If I'm not mistaken, all the examples in the CLIC document use one of these two methods, which is why I'm able to say that moving the fields out of *cause hasn't changed any of the examples, other than replacing *cause with my new *istatus.

Futhermore, let's not lose sight of the fact that the current CLIC already proposes techniques using *nexti that don't reliably maintain the interrupt number in *cause. I call your attention to the "Example Unix C ABI interrupt trampoline" in section 7.1. That code has two instances of

csrrsi a0,mnxti,MIE

which may claim a new pending interrupt, causing the interrupt number in mcause to be updated. But in each case, the updated value of mcause could be wiped out immediately afterward by a higher-level interrupt, because interrupts are enabled. So the value that was stored in mcause can't be relied on by an interrupt handler. The only reliable indication of the interrupt identity is the value written to register a0. My proposal wouldn't change that.

The same thing happens again with the example for the "interrupt-driven C-ABI model" in section 8. A higher-level interrupt that occurs immediately after the access to mnexti will wipe out the information in mcause. (Of course, it also happens again when the trampoline example from above gets expanded in sections 9.2 and 10.1.)

If interrupt handlers don't use the special methods supported by CLIC for vectoring interrupts and instead depend on the interrupt number in

*cause, they will be a little slower. But even then, it's not clear to me that my proposal changes things by much. Can you provide example code for the sort of situation you're concerned about?

From John Hauser, 19 March 2019:
I've realized I can make my proposal a bit simpler. And by "a bit" in this case I literally mean one flip-flop. The *istatus CSR in my earlier proposal has a bit "PTT" for "previous trap type". I once had a reason for needing this bit, but the reason went away; it's no longer needed. So here's the new proposed format for mistatus without PTT:

mistatus

XLEN-1 TT trap type: 0 = exception, 1 = interrupt
26:25 MPP alias of mstatus.MPP
24 MPIE alias of mstatus.MPIE
23:16 PIL previous interrupt level
15:8 IL (read-only) current interrupt level
7:5 (reserved)
4 HV 1 = reading trap vector table (was 'inhv')
3:0 (reserved)

On a trap entry to M mode, when CLIC mode is enabled:

```
if (trap is an interrupt) {
    mistatus.TT = 1
    mistatus.PIL = mistatus.IL
    mistatus.IL = new interrupt level
} else {
    // trap is a synchronous exception
    mistatus.TT = 0
}
```

On MRET:

```
if (mistatus.TT) {
    mistatus.IL = mistatus.PIL
```

}
mistatus.TT = 0

Naturally, S mode and U mode would have their own sistatus and uistatus, with equivalent behavior.

As before, if an interrupt handler wants to re-enable interrupts to allow nested higher-level interrupts, it must save and restore *istatus. However, synchronous exception handlers generally do not need to know about *istatus, and they also see no change in the *cause CSR, because my proposal leaves it the same whether CLIC mode is enabled or not.