

FAQ for System Navigator for AMD Geode™ GX and LX Processors

May 4, 2005

Does the FS2 System Navigator support true execution breakpoints?

Yes. There are 8 debug registers in the AMD Geode™ GX and LX processors all of which can be set as hardware execution breakpoints. Four of the debug registers cannot be corrupted by user application code (unlike the Pentium debug registers). In addition, 4 of the debug registers can be combined to form 2 "super" triggers that support breaking on a range of execution addresses. Range triggers can be particularly useful to ensure that a program does not execute outside of predefined boundaries.

Can the debug registers break on anything other than instruction execution?

Yes. They also support Memory Read (memr), Memory Read or Write (memrw), and I/O Read or Write (iorw). Example (respectively):

```
trigger 0 -mode memr -addr 0x10000L -length 4
trigger 1 -mode memrw -addr 0x20000L -length 4
trigger 2 -mode iorw -addr 0x378L -length 1
```

Typically there is a fair amount of debugging required around the transition between real and protected mode. Are there any features that can help in this area?

There are several. First off, you can actually single step through the real mode to protected mode transition, monitoring all of the register updates. The System Navigator also includes 2 opcode triggers that support breaking on a particular class of opcodes. The opcode triggers monitor 24-bits (with masking) plus 8 bits of prefix (e.g., segment override, address size or REP). This type of breakpoint can be used for example to break whenever a particular control register, such as CR0 is changed.

How do I set up to break execution on entry to protected mode?

Trigger 8 and 9 are designed specifically for opcode matching. You set an opcode trigger on any instruction that does a move to CR0. The opcode for "mov to cr0" is 0F 22 C0 through 0F 22 C3. Since the 0F is a prefix byte, it is represented as bit 0 in the MSB of the opcode for the trigger. The trigger then, is set up as:

```
trigger 8 -opcode 0x0122C000 0xFFFFFC00
```

The 01 represents the prefixes (in this case, just F0) and the remaining bytes 22 C0 are the rest of the instruction. The opcode range of C0 thru C3 is handled by the mask value of FC for that opcode byte.

We do a lot of debugging of System Management Mode code. Does the System Navigator support this type of debug work? How do you use it?

Yes. All of the System Navigator features are available in SMM. It can even single step through the transition to SMM if desired.

There are two different step commands. One command steps over interrupts, the other command steps into interrupts. "Interrupts" means any hardware interrupt, exception, or System Management Mode entry condition. Since SMM is usually an interrupt, the "step" command will step over an SMINT or other instruction that enters SMM. The debugger therefore has a unique command to step into SMM and other interrupt handlers, which is:

```
xstep
```

Note that there are other ways that the processor can enter SMM – namely I/O operations as well as a hardware interrupt. For the Geode GX processor, debugging SMM is no different than any other code since, unlike Pentium, SMM code occupies some portion of user memory space. So you can set triggers or software or hardware breakpoints in SMM mode just like any other code.

Can an opcode breakpoint help in debugging SMM code?

Yes. The SMINT instruction is typically used for entering SMM. SMINT opcodes are 0F 38 so to trigger on this opcode, use:

```
trigger 8 -opcode 0x01380000 0xFFFF0000
```

How can the System Navigator be used to debug GP Faults?

There is a specific command for this:

```
bkptcond set gpf
```

This will stop execution whenever a general-protection fault occurs. You can then use the trace or look on the stack to see what was happening just before the fault occurred.

Similarly, you can stop at a reset.

```
bkptcond set reset
```

A watchdog or shutdown due to a double-fault might cause a reset and you could look back in the trace to see why. (This only works with off-chip trace since the reset would clear the on-chip trace).

How can you find the code that is trashing its own memory space?

Set up a trigger that will break execution if an instruction ever writes to code space, as defined by an address range. Note that address ranging uses the extended debug registers (ID 4-7) and the “-length range” flag. Example:

```
trigger 4 -mode memw -addr 0x10000L -length range  
trigger 5 -mode memw -addr 0x1ffffL -length range
```

When the processor breaks from a match to this trigger, look at the last instruction in the trace for the errant code.

How can you find the cause of code “going off into the weeds”; that is, it starts executing code at an address that isn’t code space or where there is no physical memory?

Set up a trigger that will halt the processor whenever an instruction is about to be executed from a memory range or ranges that are out of bounds.

```
trigger 6 -mode execution -addr 0xf00000L -length range
trigger 7 -mode execution -addr 0xffffffffL -length range
```

Extended debug registers support two ranges (trigger ID pairs 4-5 and 6-7).

After halting execution, how is trace useful beyond what you can observe from a callstack?

Generally, the trace is more useful than a traditional stack traceback because you see not only the direct function call ancestors but also the siblings. For example, if function A calls B, then A calls C and then stops in C. A stack traceback would show A calling C. Hardware trace, however, shows A calling B then A calling C and will also show any interrupts or exceptions that occur.

How can I switch between debugging the Geode GX and CS5535 or LX and CS5536 companion device?

The `status` command indicates which chip was detected in the specified position in the JTAG chain. Assuming the CS5535 is first in the JTAG chain (TDI goes into CS5535 and TDO comes out of the Geode GX processor), to select the CS5535 the command is:

```
config JtagChain 25,X
```

To select the Geode GX processor, the command is:

```
config JtagChain X, 24
```

For more information, see Appendix A of the FS2 Getting-Started Manual for the System Navigator.

What debug commands are applicable to the CS5535 companion device?

Since the CS5535 companion device is a memory-mapped I/O device with MSR registers, it responds to console commands that access these registers, namely:

```
MSR or msr <address>
```

Set or display a single MSR (Machine Specific Register). Display always shows all 64 bits of an MSR as two 32-bit values, least significant DWORD first. When setting, the most significant DWORD is optional; if not supplied, it is written with zero.

```
byte, word, dword
IR, DR <bytes>
```

Standard memory read/write commands
These low-level commands set the JTAG IR or DR registers and return the original IR/DR contents.

For more information contact:

First Silicon Solutions, Inc.
4000 SW Kruse Way Place
Bldg 3, Suite 210
Lake Oswego, OR 97035
(503) 489-0311 x103
Email: info@fs2.com

On the web: www.fs2.com