Intel® SW Development Tools for Intel® XScale Microarchitecture – Debug Tools FAQ

Target OS: OS Independent + all other Operating Systems excluding Windows CE*.

Version 1.0.1

Disclaimer

Table of Contents

Disclaimer 2

Topic: General (XDB and Debugging For OS Independent Applications) . 4

Topic: XDB General 4

- How do I dump the complete register contents into a text file? 4

Topic: XDB Simulator 5

Topic: XDB JTAG Debugger 5

- Why does XDB keep taking my breakpoints as HW breakpoints, even when they are deselected? 5
- Why does the processor not recognize the WMMX instructions? 5
- How do I set up a page table and enable caching using an XDB Debugger script instead of doing it in bootcode: 6
- When bringing up my OS or bootcode inside the XDB JTAG Debugger I get stuck when the first interrupt handler gets called. 6

Problem Report Submission 9

Topic: General (XDB and Debugging For OS Independent Applications)

Topic: XDB General

- ***How do I dump the complete register contents into a text file?***

There are two methods:

```
1.  SET PROTFILE /OVERWRITE "C:/Program
Files/Intel/SDT2.0/xdb/test.prot"
     show sysreg /mode
     show sysreg /copro
     show sysreg /device

     CLOSE PROTFILE
```

This will produce a complete register listing in your ASCII format protocol file.

2. The other option would be the system configuration SAVE/RESTORE inside of XDB from the FILE pull down menu, but this does generate a nice ASCII listing.

See Also: N/A

Product/Version: All

Topic: XDB Simulator

N/A

Topic: XDB JTAG Debugger

- ***Why does XDB keep taking my breakpoints as HW breakpoints, even when they are deselected?***

Software breakpoints only work if the address at which you set the breakpoint has initialized SRAM or SDRAM memory. Hence if at least one of the following conditions is true:

- o MMU or the memory control registers or the GPIO settings for accessing your RAM are not complete
- o you are setting your breakpoint at a ROM or Flash address
- o the memory at which you try to set your breakpoint has limitations and doesn't fully allow 8bit or 32bit reads and writes

The debugger will revert to using hardware breakpoints, since the response from the memory is not consistent with initialized RAM memory.

See Also: N/A

Product/Version: All

- ***Why does the processor not recognize the WMMX instructions?***

You must enable the CP0, CP1 in the CP15 Co-processor access register. This must be done by SW even when using XDB scripts etc. to update the CP15 register in the debugger. If not, the code can be debugged but may NOT run stand alone on a target. All customers should be told that they need to make sure the CP_ACCESS register are set properly by some SW. Only if someone wants to test pure user application should you consider using XDB scripts setting this register.

Note: You need to make sure that both CP0 [has the 64 bit wr0-15 registers] and CP1 [which has the control registers] are enabled.

See Also: N/A

Product/Version: All

- ***How do I set up a page table and enable caching using an XDB Debugger script instead of doing it in bootcode:***

This can be very helpful for small test applications, when you do not want to set up memory mapping in your test code, but want to take advantage of the better memory I/O performance with MMU and caching enabled anyway.

The debugger macro definition below allows you to use the command "SetPageTable" to set up MMU and page tables in one brief debugger command.

```
define macro /overwrite SetPageTable "\
 set sysreg arm_control=0x78;\
 asm /addr=asm(@1+0x00) \"mov r0, #0xc00\";\
 asm /addr=asm(@1+0x04) \"orr r0, r0, #@2+2\";\
 asm /addr=asm(@1+0x08) \"LDR R1, [PC, #0xc]\";\
 asm /addr=asm(@1+0x0c) \"str r0, [r1], #4\";\
 asm /addr=asm(@1+0x10) \"adds r0, r0, #0x100000\";\
 asm /addr=asm(@1+0x14) \"bcc @1+0x0c\";\
 asm /addr=asm(@1+0x18) \"b @1+0x18\";\
 set val /size=long @1+0x1c=@3;\
 set reg pc = @1+0x00;\
 run until @1+0x18;\
 set sysreg trans_table_base=@3;\
 set sysreg arm_control=0x187F;\
 set sysreg tlb_inv_i_d=0xffffffff;\
 set sysreg cache_inv_i_d_btb=0xffffffff;\
 set sysreg domain_acc_control = 0xffffffff;\
 \n"

! write back caching:
@SetPageTable(0xa0003000, 0xc, 0xa0004000)
```

See Also: N/A

Product/Version: All

- ***When bringing up my OS or bootcode inside the XDB JTAG Debugger I get stuck when the first interrupt handler gets called.***

On Intel XScale® Microarchitecture based processors of the PXA25x, PXA27x, PXA800, PXA900 as well as IOP3xx and IXP4xx generation JTAG communication is handled through a debug handler that is downloaded to a dedicated mini instruction cache.
The address space used by this debug handler is overlaid on top of regular memory address space at an address specified during debugger connect.
Whenever a debug event (e.g. breakpoint hit, exception trapped, single step completed) is encountered the debug handler is being accessed through an overlay

at 0x0-0x20 or 0xffff0000 - 0xffff0020, where the reset vector of the code is overlaid with a reset vector that points to the debug handler entry point.

Since bootcode, applications and operating systems that use paging at some point during the boot process will enable the MMU turn on a mapped memory layout and relocate and replace the exception vectors (branch instructions to the interrupt handlers that are either located at 0x0 or 0xffff0000) we run into the problem that, although the code at these addresses has been modified by the bootcode in regular memory, it has not been modified in the mini-icache overlay at these addresses.

The solution to this problem during OS bootup inside the XDB JTAG Debugger is to have the code boot up until the paging setup is complete and the MMU is enabled. Then you would synchronize the exception vectors between regular memory and the mini-icache overlay and continue running.

We provide examples of this approach in the C:\Program Files\Intel\SDT2.0.1\Example\ocdxs\batch directory.

The easiest and usually successful approach is to execute an XDB script which executes a RUN until the first interrupt occurs, then synchronizes the vectors and leaves it up to the user to start debugging from there or continue running.

The contents of this script file would look something like

```
set exc irq
run
custom "syncvec"
del exc irq
```

In this case the IRQ exception is being trapped. Once an IRQ interrupt occurs the debugger halts execution of code on the target. The exception vectors are being synchronized and we disable trapping of the IRQ exception. In most cases either an IRQ or a SWI exception are the first hit by bootcode.

See Also: N/A

Product/Version: All

Problem Report Submission

Your feedback is very important to us. To receive technical support for the tools provided in this product and technical information visit Intel® Premier Support at https://premier.intel.com.

For general support information please visit http://www.intel.com/software/products/support/.