



# *Hot-Debug for Intel<sup>®</sup> XScale<sup>™</sup> Core Debug*

White Paper

---

*May 2002*





Information in this document is provided in connection with Intel® products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The Intel® XScale™ core may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at <http://www.intel.com>.

Copyright© Intel Corporation, 2002

AlertVIEW, i960, AnyPoint, AppChoice, BoardWatch, BunnyPeople, CablePort, Celeron, Chips, Commerce Cart, CT Connect, CT Media, Dialogic, DM3, EtherExpress, ETOX, FlashFile, GatherRound, i386, i486, iCat, iCOMP, Insight960, InstantIP, Intel, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel ChatPad, Intel Create&Share, Intel Dot.Station, Intel GigaBlade, Intel InBusiness, Intel Inside, Intel Inside logo, Intel NetBurst, Intel NetStructure, Intel Play, Intel Play logo, Intel Pocket Concert, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel TeamStation, Intel WebOutfitter, Intel Xeon, Intel XScale, Itanium, JobAnalyst, LANDesk, LanRover, MCS, MMX, MMX logo, NetPort, NetportExpress, Optimizer logo, OverDrive, Paragon, PC Dads, PC Parents, Pentium, Pentium II Xeon, Pentium III Xeon, Performance at Your Command, ProShare, RemoteExpress, Screamlane, Shiva, SmartDie, Solutions960, Sound Mark, StorageExpress, The Computer Inside, The Journey Inside, This Way In, TokenExpress, Trillium, Vivonic, and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

\*Other names and brands may be claimed as the property of others.

\*ARM and StrongARM are registered trademarks of ARM, Ltd.

# Contents

---

1.0	Background.....	5
2.0	Solution .....	6
3.0	Detecting Hot-Debug Support by Target.....	7
4.0	Software Actions Out of Reset.....	8
5.0	Initiating a Hot-Debug Debug Session .....	9
6.0	Hot-Debug Code .....	10
7.0	Requirements and Restriction .....	12

## Figures

No Figures Used At This Time

## Tables

3-1	DCSR.moe Encodings.....	7
5-2	Debugger/Debug Handler Actions During Download .....	9



## Revision History

Date	Revision	Description
May 2002	003	Corrected code in Section 6.0.
October 2001	002	Updated Section 6.0, "Hot-Debug Code". Revised first bullet in Section 7.0, "Requirements and Restriction".
May 2001	001	Initial release.

## 1.0 Background

In general, customers expect to be able to debug through a debug connector to an Intel® XScale™ core (ARM\* architecture compliant) solution and invoke an Intel® XScale™ core debugger at any time. This includes during boot-up and any time the system is running.

The Intel® XScale™ core's debug architecture requires a debug handler to be resident in the cache or external memory, before the debugger can be invoked. An on-chip mini-icache (mini-IC) is provided to hold the debug handler. In order to load the debug handler into the mini-IC, the debugger needs to hold the processor in reset. The mini-IC and the main icache share data paths. Thus, holding the processor in reset, resolves any potential conflicts between outstanding icache line fills and the JTAG unit pushing instructions into the mini-IC.

For an embedded/hand-held application where the Intel® XScale™ core is used as the system processor, this is workable though not ideal. However, for an I/O processor application, this is not acceptable since re-initializing the Intel® XScale™ core may cause problems for other parts of the system. For example, reinitializing the Intel® XScale™ core while an outstanding I/O is pending could cause a lock-up condition.

This document provides the details of a software only solution for hot-debug in the Intel® XScale™ core's debug architecture.

## 2.0 Solution

The Intel® XScale™ core's debug architecture hot-debug requires linking in some code with the user's application.

At a high level, the application runs with debug enabled. When a user wants to initiate a hot-debug session, the debugger first checks whether hot-debug is supported by the target system.

When hot-debug is supported, the debugger uses an external debug break through JTAG to cause a debug exception on the core. The core re-directs execution to a debug handler stub that is linked into the application's reset handler. The stub sends a message to the debugger through JTAG, indicating when it is safe to download the debug handler code into the mini-IC.

After the debugger completes the download, it signals for the debug handler stub to continue by sending the start address of the downloaded debug handler. The stub then branches to the beginning of the debug handler and the starts a debug session.

The solution has several pieces of code that need to be linked into the application to make it work:

- Code to determine whether the reset handler was entered do to a reset or some debug exception.
- Code to enable debug in the DCSR, following every reset.
- Code to setup a special value in the DCSR as an indicator to a debugger that the software hot-debug solution is supported by the current system.
- Debug handler stub code that performs the necessary actions to allow a debugger to download the debug handler on the fly.

Once the above pieces of code are linked into the application's reset handler, a debugger can detect whether the current system supports the software hot-debug solution. If supported, the debugger can initiate a debug session at any time. The following sections provide some details on how hot-debug works. The actual code to be linked into the application is also provided, as well as special requirements and restrictions to ensure proper hot-debug operation.

Refer to the software Debug chapter of the product specific user's manual for details on the Intel® XScale™ core's debug architecture. The following sections describe specific details on how the Intel® XScale™ core hot-debug solution works. The actual code can be found in [Section 6.0](#).

### 3.0 Detecting Hot-Debug Support by Target

The debugger needs some indication of whether the target contains the software necessary to support hot-debug. Without this indication, a debugger may try to plug into a system that does not support hot-debug and cause undesirable behavior.

To detect that the hot-debug solution is present, a debugger must read the DCSR through JTAG and examine the DCSR.moe bits (DCSR[4:2]). [Table 3-1](#) shows the meaning of the bit encodings:

**Table 3-1. DCSR.moe Encodings**

Bit Encoding	Meaning
000b	Indicates hot-debug is not supported. The processor must be held in reset while downloading the debug handler into the mini-IC.
001b - 110b	Indicates a monitor is actively running. Hot-debug may or may not be supported; the debugger must wait for the debug monitor to exit to determine whether hot-debug is supported
111b	Indicates hot-debug is supported, and a debug monitor is not actively running.

The hot-debug solution relies on the DCSR.moe bits properly being set by software. There are a couple situations that require software for correct set up of the MOE bits. When the MOE bits are not correctly set for these situations, the hot-debug capability may not work as expected.

The DCSR.moe bits are always cleared by a processor reset. When hot-debug is supported, the initialization code must set the DCSR.moe bits to 111b. This should be done as soon as the reset handler starts execution, in order to avoid the debugger reading the incorrect value in the MOE bits.

When exiting a debug session, the debugger must correctly restore the DCSR.moe bits. Following a debug exception, the MOE bits are updated by the processor to indicate the type of debug exception that occurred. Prior to exiting a debug session, the debugger must restore the MOE bits with 000b or 111b, depending on whether hot-debug is supported or not. This allows a debugger to subsequently detect whether hot-debug is supported.

## 4.0 Software Actions Out of Reset

Part of the Intel® XScale™ core hot-debug solution requires code to be placed at the beginning of the reset handler. The code serves several purposes, as described below.

First the reset handler must determine the reason for entry into the handler. In Halt mode, the debug exception vector maps to address 0, which is also the location of the reset vector. The reset handler must determine whether the handler was entered due to a debug exception or a normal processor reset. This is important because different actions must occur depending on the cause for entry into the handler.

The reset handler examines the CPSR to determine the whether a debug exception or a reset event occurred.

When the processor is in SVC mode (CPSR[4:0]=0x13), then a normal reset event caused entry into the reset handler. Following a reset, the processor clears the debug enable bit (DCSR[31]) and the MOE bits (DCSR[4:2]) in the DCSR. To support hot-debug, software must set the debug enable bit to enable debug exceptions, and setup the MOE bits with 111b. The reset handler can then continue on with its normal initialization actions.

When the processor is in DBG mode (CPSR[4:0]=0x15), then a debug exception caused entry into the reset handler. In this case, the reset handler branches to the linked debug handler stub, which allows the debugger to download the full debug handler into the mini-IC. Once the download is complete, the stub branches to the downloaded debug handler.

## 5.0 Initiating a Hot-Debug Debug Session

A debugger must first determine whether hot-debug is supported by the target. This is described in Section 3.0, “Detecting Hot-Debug Support by Target” on page 7.

Once the debugger detects that hot-debug is supported, it can initiate a debug session by causing an external debug break via the DCSR JTAG data register. The external debug break should be generated using the following steps to ensure correct behavior:

1. Scan a value into the DCSR JTAG data register to set the Halt mode bit.
2. Scan a value into the DCSR JTAG data register to set the external debug break bit (and keeping the Halt mode bit set).

Since debug was already globally enabled by the reset handler, the external debug break causes a debug exception shortly after the DCSR JTAG data register is updated. The processor re-directs execution to the debug/reset vector and enters the reset handler.

At this point, the reset handler detects that the processor is in DBG mode and branches to the debug handler stub. Table 5-2 shows the actions taken by the debug handler stub and the debugger to ensure the debug handler is correctly downloaded into the mini-IC.

**Table 5-2. Debugger/Debug Handler Actions During Download**

Debug Handler Stub Actions	Debugger Actions
Executes synchronization routine to ensure all outstanding instruction fetches have completed. Invalidate the BTB.	Polls the DBG_TX JTAG data register for a 'ready-for-download' message indicating it is safe to begin the download.
Following sync routine, writes the 'ready-for-download' message to the TX register. 'ready for download' = 0x00B00000	
Spins in a loop, polling RX register, waiting for debugger to indicate the download is complete.	Detects the write to TX and reads the value through the DBG_TX JTAG data register. Sees the 'ready-for-download' message and begins the download.
	Downloads vector table and debug handler code into mini-IC through the LDIC JTAG data register <sup>1</sup> . After completion of the download, waits ~50 TCKs w/ LDIC JTAG instruction in the JTAG IR before continuing.
	Writes the debug handler start address to the DBG_RX JTAG data register. This signals the download is complete and also provides the debug handler stub with the start address of for the full debug handler.
Detects valid data in RX and comes out of its polling loop. Reads start address from RX and branches to the debug handler.	Polls DGB_TX for debug handler entry message

1. Each cache line written to must first be invalidated through JTAG.

Subsequent debug breaks during the current debug session are intercepted by the debug/reset vector downloaded into the mini-IC. The processor then branches directly to the downloaded debug handler. However, for each new debug session, the debug handler code should be re-downloaded to ensure correct behavior. See Section 7.0, “Requirements and Restriction” on page 12 for the actions a debugger should take before exiting a debug session.

## 6.0 Hot-Debug Code

```
reset_handler_start:

    ## reset handler should first check whether this is a debug exception
    ## or a real RESET event.

    ## NOTE: r13 is only safe register to use.
    ## - For RESET, don't really care about which register is used
    ## - For debug exception, r13=DBG_r13, prevents application registers
    ## - from being corrupted, before debug handler can save.

    mrs    r13, cpsr
    and    r13, r13, #0x1f
    cmp    r13, #0x15                # are we in DBG mode?
    beq    dbg_handler_stub         # if so, go to the dbg handler stub

    mov    r13, #0x8000001c         # otherwise, enable debug, set MOE bits
    mcr    p14, 0, r13, c10, c0, 0 # and continue with the reset handler

    ## normal reset handler initialization follows code here,
    ## or branch to the reset handler.

    .align 5    ## align code to a cache line boundary.

dbg_handler_stub:

    ## First save the state of the IC enable/disable bit in DBG_LR[0].
    mrc    p15, 0, r13, c1, c0, 0
    and    r13, r13, #0x1000
    orr    r14, r14, r13, lsr #12

    ## Next, enable the IC.
    mrc    p15, 0, r13, c1, c0, 0
    orr    r13, r13, #0x1000
    mcr    p15, 0, r13, c1, c0, 0

    ## do a sync operation to ensure all outstanding instr fetches have
    ## completed before continuing. The invalidate cache line function
    ## serves as a synchronization operation, that's why it is used
    ## here. The target line is some scratch address in memory.
    adr    r13, line2
    mcr    p15, 0, r13, c7, c5, 1

    ## invalidate BTB. make sure downloaded vector table does not hit one of
    ## the application's branches cached in the BTB, branch to the wrong place
    mcr    p15, 0, r13, c7, c5, 6

    ## Now, send 'ready for download' message to debugger, indicating debugger
    ## can begin the download. 'ready for download' = 0x00B00000.
TXloop:
    mrc    p14, 0, r15, c14, c0, 0 # first make sure TX reg. is available
    bvs    TXloop
    mov    r13, #0x00B00000
    mcr    p14, 0, r13, c8, c0, 0  # now write to TX

    ## Wait for debugger to indicate that the download is complete.
RXloop:
    mrc    p14, 0, r15, c14, c0, 0 # spin in loop waiting for data from the
    bpl    RXloop                 # debugger in RX.

    ## before reading the RX register to get the address to branch to, restore
    ## the state of the IC (saved in DBG_r14[0]) to the value it have at the
```

```
## start of the debug handler stub. Also, note it must be restored before
## reading the RX register because of limited scratch registers (r13)
mrc    p15, 0, r13, c1, c0, 0

### First, check DBG_LR[0] to see if the IC was enabled or disabled
tst    r14, #0x1

### Then, if it was previously disabled, then disable it now, otherwise,
### there's no need to change the state, because its already enabled.
biceq  r13, r13, #0x1000
mrc    p15, 0, r13, c1, c0, 0

## Now r13 can be used to read RX and get the target address to branch to.
mcr    p14, 0, r13, c9, c0, 0    # Read RX and
mov    pc, r13                  # branch to downloaded address.

## scratch memory space used by the invalidate IC line function above.
.align 5                        # make sure it starts at a cache line
                                    # boundary, so nothing else is affected

line2:
.word 0
.word 0
.word 0
.word 0
.word 0
.word 0
.word 0
.word 0
```

## 7.0 Requirements and Restriction

The following are restrictions of the reset handler and linked in hot-debug code that must be met to ensure proper hot-debug functionality.

- The code must be in a cacheable region of memory. The debug handler stub will temporarily enable the IC allowing the polling loops used to communicate with the debugger to get cached. If the code is non-cacheable, the polling loop is small enough such that it can execute out of the fill buffers. However, since these same fill buffers are used when loading code into the mini-IC through JTAG, there is a conflict for these resources.
- Use of R13 in DBG mode is very restrictive and must be handled carefully. The code provided above has been tested for correct behavior. Any changes to the code may cause improper behavior during a hot-debug.

Following are restrictions and requirements on the debugger to ensure correct hot-debug behavior:

- When the code is being downloaded into the mini-IC, each line must first be invalidated through JTAG. Then the code for that cache line can be downloaded. This ensures that an address is not valid in multiple places in the instruction cache (main or mini).
- After the complete debug handler and vector table have been downloaded into the mini-IC, the debugger must wait 50 TCKs before removing the LDIC JTAG instruction from the JTAG IR. This allows the last line of the debug handler is correctly update in the mini-IC.
- The debugger must leave the system in a safe state before exiting. Before ending the debug session, the debugger must do the following:
  - Clear the Halt Mode bit. The problem occurs for a processor reset that occurs when a debug session is not in progress. The reset clears the global debug enable bit in the DCSR. One of the first things the reset handler does is re-set this bit. However, when the Halt Mode bit is set, then software cannot write to the DCSR (unless the processor is in Special Debug State). So the global debug enable bit does not really get set by the reset handler, and the debugger is then not able to use the external debug break to initiate a debug session.
  - Invalidate the BTB. This can only be done via the debug handler. Invalidating the BTB ensures that the BTB does not contain cached branch targets that were there for debug but no are no longer valid. This mainly applies to the vector table, since this is probably the only debug code that overlaps the applications code.
  - Invalidate the mini-IC. This can be done through JTAG. This makes sure the application does not hit code in the mini-IC, in particular the exception vectors that were downloaded with the debug handler.
  - Set the DCSR.moe bits to 111b. This allows a debugger to detect that hot-debug is supported when attempting a subsequent debug sessions.

