

# Intel® JTAG Debugger 2014 Installation Guide and Release Notes

---

Installation Guide and Release Notes for Windows\* host

Document number: 329490-002US

28 January 2014

## Contents

1	Introduction .....	5
2	Technical Support and Documentation .....	6
2.1	Product Contents .....	6
3	What's New.....	7
3.1	General Updates.....	7
3.2	Target Platform Support.....	9
4	System Requirements.....	9
4.1	Host Software Requirements .....	9
4.2	Target Software Requirements .....	10
4.3	Hardware Requirements .....	10
4.4	Ordering required JTAG Device .....	10
4.4.1	Intel® ITP-XDP3 .....	10
4.4.2	Macraigor* usb2Demon* .....	10
5	Installation Notes .....	11
5.1	Pre-Installation Steps.....	11
5.1.1	Notes on installing a 32bit JRE on a 64bit Windows* system .....	11
5.1.2	Note on Microsoft* Visual Studio generated UEFI debug .....	12
5.2	Product Installation (Online Installer) .....	12
5.3	Product Installation (Full Product) .....	12
5.4	Silent Install .....	13
5.5	Installing Intel® XDP3 JTAG Probe.....	13
5.6	Installing Macraigor Systems* usb2Demon* Support .....	13
5.7	Uninstalling the Product .....	14

6	Features .....	15
6.1	Execution Trace using Intel® Processor Trace (Intel® PT) and Real-Time Instruction Trace (RTIT).....	15
6.2	Agent-based debugging of UEFI (EDK2) BIOS .....	15
6.3	OS Awareness / Kernel Module Debugging .....	15
6.4	Scripting Language .....	16
6.5	Page Translation Table .....	16
6.6	Unload of Symbol Information .....	16
6.7	Power Events Handling.....	16
6.8	NAND Flashing support on Intel® Atom™ Processor CExxxx.....	16
6.9	Supported Flash Types.....	16
7	Agent Based UEFI Debugging .....	18
7.1	Pre-requisites.....	18
7.1.1	Using Ajasys* USB debug device .....	18
7.2	Setting up the Target .....	19
7.2.1	Hardware Setup.....	19
7.2.2	BIOS Setup.....	19
7.3	Connection Setup .....	20
7.4	Starting an Agent Based Debug Session .....	21
7.4.1	Handling the module watchlist .....	21
7.5	Troubleshooting .....	23
7.6	Known issues.....	24
7.6.1	Debugging a module on the watchlist .....	24
7.6.2	Using breakpoints .....	25
7.7	UEFI Debug Appendix .....	26
7.7.1	Using an Emulator for EDK2 debugging.....	26
7.7.2	Setting up the emulator.....	27
7.7.3	Starting a debug session .....	27
8	JTAG Debugger Usage Notes.....	27
8.1	Starting the Debugger.....	27
8.1.1	Using startup scripts in installation directory .....	27
8.1.2	Using Windows* start menu .....	28
8.2	Burning blank flash on Intel® Atom™ Processor CE5300 based platforms.....	28

8.3	eMMC Flash Recovery on Intel® Puma6™ Media Gateway .....	29
8.4	eMMC Flash on Intel® Atom™ Processor CE42xx and CE53xx .....	29
8.4.1	Partitions.....	29
8.4.2	Addressing.....	29
8.4.3	Erasing .....	30
8.5	FTL Lite NAND Flash Programming on the Intel® Atom™ Processor CE4100 .....	30
8.5.1	Addressing.....	30
8.5.2	Backup.....	31
8.6	Enabling Run-Time Loaded Kernel Module Debugging.....	31
8.7	Unload of Symbol Information .....	31
9	Issues and Limitations.....	31
9.1	Target-specific issues: .....	31
9.1.1	Release of Linux* hosted 4 <sup>th</sup> generation Intel® Core™ processor support delayed 31	
9.1.2	Functional differences of 60-pin vs. 10-pin JTAG .....	32
9.1.3	Platform reset policy may inhibit debugger operation .....	32
9.1.4	Platform security policy may inhibit debugger operation.....	32
9.1.5	Target power management and platform power policy on tablet systems may inhibit debugger operation.....	32
9.1.6	Platform reset implementation may limit debugger-initiated reset:.....	33
9.1.7	Platform reset implementation may limit ability to halt at reset vector:.....	33
9.2	Intel® Atom™ Processor Z3xxx and E3xxx specific issues .....	33
9.2.1	Platform power management policy may limit debugger control of the target: .....	33
9.2.2	Launching the debugger when the target is off or in a low-power state may cause unexpected behavior.....	33
9.2.3	Hardware Threads window may show no threads/partial threads/disabled threads 34	
9.2.4	User-initiated Halt may occasionally return errors, especially if the target is in a low-power state.....	34
9.2.5	Kernel module load configurations may be unreliable .....	34
9.3	Installation related issues.....	34
9.3.1	Intel® JTAG Debugger is incompatible with the Intel® ITP II and/or PDT products 34	
9.3.2	Sharedinfo.txt not writable.....	35

9.3.3	Network access for MasterFrame.HostApplication.exe blocked by windows firewall	35
9.3.4	MSDIA DLL not registered .....	35
9.4	General Feature Limitations .....	35
9.4.1	Support for Intel® Atom™ processor bitfield editor register views .....	35
9.4.2	Failure of initial debugger re-connection can lead to debugger hang .....	35
9.4.3	Locals Window updates can be slow .....	36
9.4.4	Kernel Threads Window Population Slow .....	36
9.4.5	Debugger puts a Windows file system lock on symbol files.....	36
9.4.6	Flash programming with empty flash parts (CE4200).....	36
9.4.7	Use of Macraigor* usb2Demon to debug Intel® Atom™ processor CE4xxx based platforms may require board changes .....	36
9.4.8	Older Macraigor* usb2Demon 60pin connector headers.....	36
9.4.9	Backup of large flash partitions may fail.....	37
9.4.10	Verification of flash content on Intel® Atom™ processor E6xx based platform.....	37
9.4.11	Target doesn't boot when Macraigor* usb2Demon* JTAG device is not initialized	37
9.4.12	Memory Reads with Un-Initialized Memory .....	37
9.4.13	Memory Writes to Un-Initialized Memory.....	37
9.4.14	Flash Writer disables pre-existing Breakpoints.....	37
9.4.15	Master Flash Header Read/Write not supported for Intel® Atom™ Processor CE4200	37
9.4.16	Assembly Window Instruction Pointer Look-Ahead on Un-Initialized Memory .....	38
10	Attributions .....	38
11	Disclaimer and Legal Information .....	39

# 1 Introduction

This Intel® JTAG Debugger 2014 release provides a Windows\* 7 or 8 hosted cross-debug solution for software developers to debug kernel sources and dynamically loaded drivers and kernel modules on devices based on the Intel® architecture. It does so using the In-Target Probe eXtended Debug Port (ITP-XDP) on these platforms

As JTAG communication device the Intel® ITP-XDP3 is used. Beyond this the debugger also offers convenient and in-depth access to underlying hardware properties through a powerful graphical user interface (GUI). This makes it an ideal assistant for initial platform bringup, firmware, BIOS, OS and device driver debugging. A set of features providing in-depth access to the development platform complete the offering for system developers:

- Execution trace support using LBR (last branch record), RTIT (real-time instruction tracing and Intel® PT (Intel® Processor Trace) for identifying incorrect execution paths or memory accesses
- Support for UEFI firmware debug via JTAG as well as EDKII (EFI Development Kit) debug agent
- Graphical representation of the page translation table with full access of the descriptor tables
- Flashing Support for select processors

These debugger capabilities minimize the time it takes to isolate and correct platform and system level problems.

This document provides system requirements, installation instructions, issues and limitations, and legal information.

## 2 Technical Support and Documentation

The default installation directory is

```
C:\Program Files(x86)\Intel\System Studio 2014.0.xxx\
```

The directory <INSTALL-DIR>\Documentation\en\_US\debugger\xdb includes these release and installation notes `jtag_release_install.pdf`.

The directory

<INSTALL-DIR>\Documentation\en\_US\debugger\xdb\first\_use\ contains the QuickStart Guide `xdb_quickstart_win.pdf` and debugger Usage Guide `xdb_usage.pdf`

In addition, the Intel® JTAG Debugger user manual and “Intel® Debugger Online Help” can be accessed at

```
<INSTALL-DIR>\Documentation\en_US\debugger\xdb\cl\index.htm.
```

If you did not register your compiler during installation, please do so at the [Intel® Software Development Products Registration Center](#). Registration entitles you to free technical support, product updates and upgrades for the duration of the support term.

To submit issues related to this product please visit the [Intel Premier Support](#) webpage and submit issues under the product **Intel(R) System Studio**.

Additionally you may submit questions and browse issues in the [Intel® System Studio User Forum](#).

For information about how to find Technical Support, product documentation and samples, please visit <http://software.intel.com/en-us/intel-system-studio>.

**Note:** If your distributor provides technical support for this product, please contact them for support rather than Intel.

### 2.1 Product Contents

- Intel® JTAG Debugger 2014 [Build 81.x.xxx]

## 3 What's New

Below are some of the new features in the Intel® JTAG Debugger

### 3.1 General Updates

- **Execution Trace using Intel® Processor Trace (Intel® PT) and Real-Time Instruction Trace (RTIT)**

The Intel JTAG Debugger now supports Execution Trace using two new hardware capabilities, Intel PT and RTIT. These new hardware capabilities give the user increased trace depth over previous technologies, with little to no impact on target execution speed.

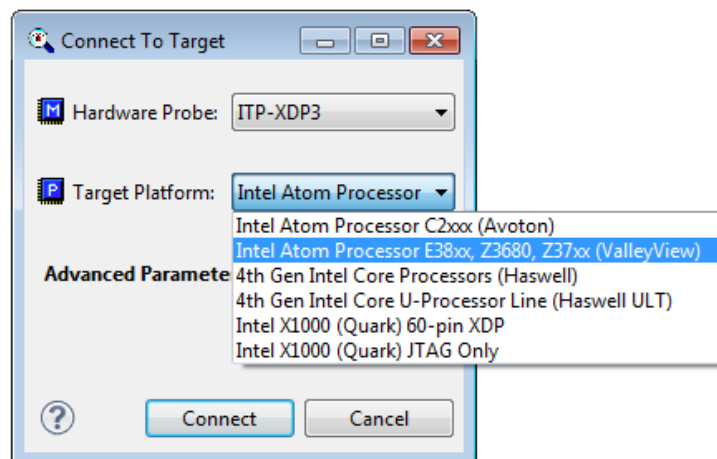
Support for these capabilities is limited to specific processors and/or platforms, for more information please contact customer support.

- **New Connection Dialog and Startup Scripts**

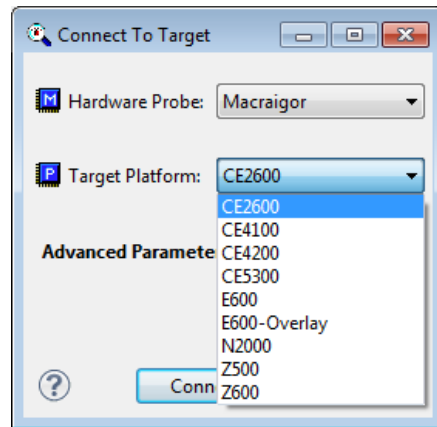
The Intel JTAG Debugger provides a graphical connection dialog for connecting/disconnecting the target, this reduces the number of startup scripts needed and moves the selection of a specific target type to the debugger GUI.

The startup scripts are now as follows:

- Start\_xdb\_2014\_products.bat - this script allows the user to debug recent processors/platforms using the Intel® ITP-XDP3 device, such as:
  - Intel® Atom™ Processor C2xxx (Avoton, Rangeley)
  - Intel® Atom™ Processor E38xx, Z3680, Z37xx (ValleyView)
  - 4th Gen Intel® Core™ Processors (Haswell)
  - 4th Gen Intel® Core™ U-Processor Line (Haswell ULT)
  - Intel® Quark™ Processor X1000 60-pin XDP
  - Intel® Quark™ Processor X1000 JTAG Only



- Start\_xdb\_legacy\_products - this script allows the user to debug older processors using either Intel® ITP-XDP3 or Macraigor\* usb2Demon\*, such as:
  - Intel® Atom™ Processor E600, N2000, Z500, Z600, Z2400, Z2500, Z2700, CE4100, CE4200, CE5300
  - Intel® Puma6™ Media Gateway



- Start\_xdb\_UEFI\_agent - this script allows the user to debug UEFI BIOS over serial or USB ports
- Start\_xdb\_firmware\_recovery - this script is for the sole purpose of recovering flash firmware on some embedded processors

- **Agent-based debugging of UEFI (EDK2) BIOS**

The Intel® JTAG Debugger now supports non-JTAG based debug of UEFI BIOS, this requires the use of a target-side debug agent. For more information see chapter 8 on [Agent-based UEFI Debugging](#).

- **Extensible Firmware Interface (EFI) Support:**

When the target is booting through an EFI BIOS (PEI or DXE phase of the boot process) which has been built with debug information, the debugger can parse the PE/COFF headers and locate an address in source.

Use the command

```
xdb> efi "loadthis <address>"
```

Providing an address is optional. If no address is passed on the current instruction pointer is used.

In addition the EFI plugin is able to automatically search and find the system table pointer. Once found many of the key aspects of the EFI environment including loaded services and modules can be listed.



```
xdb> efi "showstabs"  
xdb> efi "showmodules"
```

Please see the output of the

```
xdb> efi "showhelp"
```

command for more details.

## 3.2 Target Platform Support

- **Support for platforms based on the processors listed below using the Intel® ITP-XDP3 device**
  - Intel® Atom™ Processor C2xxx (Avoton, Rangeley)
  - Intel® Atom™ Processor E38xx, Z3680, Z37xx (ValleyView)
  - 4th Gen Intel® Core™ Processors (Haswell)
  - 4th Gen Intel® Core™ U-Processor Line (Haswell ULT)
  - Intel® Quark™ Processor X1000 60-pin XDP
  - Intel® Quark™ Processor X1000 JTAG Only

## 4 System Requirements

### 4.1 Host Software Requirements

1. Microsoft\* Windows\* 7, 8 32bit or 64bit.
2. 32bit Java runtime environment (JRE) 1.5 or newer to use the Eclipse\* framework. In a web browser, access [www.java.com](http://www.java.com) , and download and install JRE 1.6. Ensure that the Java runtime environment is 32bit even if you are installing on a 64bit system. (Go to <http://www.java.com/en/download/manual.jsp> and select "Windows 7, XP Offline (32-bit)").

#### **32-bit host system:**

3. Microsoft .NET Framework 4 (dotNetFx40\_Full\_x86.exe)
4. Microsoft .NET Framework 3.5 SP1 runtime (pre-installed by default on Microsoft\* Windows\* 7)
5. Microsoft Visual C++ 2008 Redistributable Package (vcredist9\_x86.exe)
6. Microsoft Visual C++ 2010 SP1 Redistributable Package (vcredist10\_x86.exe)

#### **64-bit host system:**

3. Microsoft .NET Framework 4 (dotNetFx40\_Full\_x86\_x64.exe)

4. Microsoft .NET Framework 3.5 SP1 runtime (pre-installed by default on Microsoft\* Windows\* 7)
5. Microsoft Visual C++ 2008 Redistributable Package (vc redistrib9\_x86\_x64.exe)
6. Microsoft Visual C++ 2010 SP1 Redistributable Package (vc redistrib10\_x86\_x64.exe)

## 4.2 Target Software Requirements

The target platform should be based on one of the following environments:

- Bare metal, BIOS, firmware environment
- CE Linux\* OS PR32 for Intel® Atom™ Processor CE4xxx, CE53xx and Intel® Puma6™ Media Gateway
- Yocto Project\* 1.x and Wind River Linux\* 4,5,6 on
  - Intel® Atom™ Processor Z5xx, E6xx, N2xxx, D2xxx, E3xxx, C2xxx
  - Intel® Quark Processor X1xxx
  - 4th Gen Intel® Core™ Processors
- Android\* 4.0.x -4.4.x on
  - Intel® Atom™ Processor Z2400, Z2500, Z2700, Z3680, Z37xx
  - 4th Gen Intel® Core™ U-Processor Line
- Wind River\* Linux\* 4, 5, 6 on
  - Intel® Atom™ Processor Z5xx, E6xx, N2xxx, D2xxx, E3xxx, C2xxx
  - 4th Gen Intel® Core™ Processors
- Tizen\* IVI 2.x, 3.x on
  - Intel® Atom™ Processor E3xxx

## 4.3 Hardware Requirements

- Second generation Intel® Core™ i5 processor or Intel® Core™ i7 processor.
- 2GB RAM
- 10GB free disk space for all product features and all architectures
- USB 2.0 host interface
- In-Target Probe eXtended Debug Port on target platform
- Intel's ITP-XDP3 JTAG Hardware Adapter or alternatively Macraigor Systems\* usb2Demon\* JTAG Hardware Adapter

## 4.4 Ordering required JTAG Device

### 4.4.1 Intel® ITP-XDP3

To order the Intel® ITP-XDP3 device, please contact the Hibbert Group\* at [Intelvtg@hibbertgroup.com](mailto:Intelvtg@hibbertgroup.com) and request the VTG order form.

### 4.4.2 Macraigor\* usb2Demon\*

Go to <http://www.macraigor.com/usbDemon.htm> and select the Intel Atom™ Processor target with the appropriate 24, 31 or 60 pin connector for your target device.

## 5 Installation Notes

### 5.1 Pre-Installation Steps

Install the latest Java JRE. In a web browser, access [www.java.com](http://www.java.com), and download and install JRE 1.6. Ensure that the Java runtime environment is 32bit even if you are installing on a 64bit system. (Go to <http://www.java.com/en/download/manual.jsp> and select “Windows 7, XP Offline (32-bit)”.

#### 5.1.1 Notes on installing a 32bit JRE on a 64bit Windows\* system

##### **Check your %PATH% environment variable**

Please check your %PATH% environment variable to make sure it is pointing to the 32-bit JRE installation. (Generally the one in “C:\Program Files (x86)” as opposed to the one in “C:\Program Files”)

##### **Rename “java.exe” and “javaw.exe” in “C:\Windows\System32”**

Also check your “C:\Windows\System32” directory. There appears to be local copies of the java.exe and javaw.exe in this folder. These are usually from the original Java\* installation and most likely the 64-bit versions. Since “C:\Windows\System32” is always in your %PATH% these copies of the java executables will be used before the JRE directory added to your system %PATH%. Please delete the copies of “java.exe” and “javaw.exe” in “C:\Windows\System32” so that the Java directory in your PATH variable pointing to the 32-bit JREs will be used.

##### **Verify the correct Java\* Runtime Environment is used**

To verify that the correct JRE is being used open a command prompt and change directory to the bin folder in the XDB install directory.

```
> cd <INSTALL-DIR>\bin
```

```
> java CheckJRE
```

```
x86
```

If the output of this call is x86 and not x64 or amd64 than you are ready to launch XDB.

### 5.1.2 Note on Microsoft\* Visual Studio generated UEFI debug

The debugger is dependent on a shared library provided by Microsoft\* for access to debug information generated with the Microsoft\* compiler. In some cases this library will not be correctly registered on the host system, which will lead to an error message when trying to load symbols for modules (e.g. EFI modules) that are compiled using the Microsoft\* toolchain. The user can resolve this issue by manually registering the correct library using an administrator command prompt.

From a command line window go to:

```
C:\Program Files (x86)\Common Files\microsoft shared\VC
```

Run:

```
regsvr32 msdia90.dll
```

if the msdia90.dll file is not found there, reinstall the Microsoft\* Visual C++ 2008 Redistributable Package after downloading it from Microsoft's website

## 5.2 Product Installation (Online Installer)

The Intel® JTAG Debugger 2014 on Windows\* host component of the Intel® System Studio 2014 is available as part of a downloadable online installer. Of you only intend to install the Intel® JTAG Debugger you can thus reduce the package size that is downloaded for the actual install. Using the online installer requires to be connected to the internet and that https protocol based component downloads are permitted by your firewall.

Double-click on the executable file (w\_cemdb\_online\_p\_2014.0.xxx.exe) to begin installation. And follow the installer guidance.

The default installation directory is

```
C:\Program Files (x86)\Intel\System Studio 2014.0.xxx\
```

## 5.3 Product Installation (Full Product)

The Intel® JTAG Debugger 2014 on Windows\* is part of the Intel® System Studio 2014 downloadable installer. Double-click on the executable file (w\_cemdb\_p\_2014.0.xxx.exe) to begin installation. And follow the installer guidance.

The default installation directory is

C:\Program Files (x86)\Intel\System Studio 2014.0.xxx\

#### 5.4 Silent Install

For information on automated or “silent” install capability, please see <http://intel.ly/ngVHY8>.

Please note that the Intel® ITP-XDP3 JTAG device driver installation does not support silent install. If you choose silent installation for the Intel® JTAG Debugger, this device driver will need to be installed separately afterwards

#### 5.5 Installing Intel® XDP3 JTAG Probe

The Intel® ITP-XDP3 driver is automatically installed as part of the Intel® JTAG Debugger installation process.

#### 5.6 Installing Macraigor Systems\* usb2Demon\* Support

The Macraigor Systems\* usb2Demon\* device can be ordered at <http://www.macraigor.com/usbDemon.htm>.

To enable support for the Macraigor Systems\* usb2Demon\* device for debugging Intel® Atom™ processor based platforms with the Intel® JTAG Debugger it is necessary to install the Windows\* drivers for the Macraigor Systems\* usb2Demon\* device. The driver can be found at [http://www.macraigor.com/full\\_gnu.htm](http://www.macraigor.com/full_gnu.htm).

You can install either the [IA-32](#) or the [Intel®64](#) version of the Hardware Support Package to provide the device driver support

The Intel® JTAG Debugger has been validated for use with the Macraigor Systems\* usb2Demon\* device and OCDRemote\* 13.0-0. We recommend using OCDRemote\* 13.0-0 of the Macraigor\* Systems\* driver for all intended target platforms.

For further details on how to configure the OCDRemote\* driver set from Macraigor\* Systems, please refer to the full installation instructions at [http://www.macraigor.com/full\\_gnu.htm](http://www.macraigor.com/full_gnu.htm) .

## 5.7 Uninstalling the Product

In the Windows\* Control Panel, select Add/Remove Application and choose the Intel® System Studio 2014

To also uninstall the Intel® ITP-XDP3 driver please select

- Windows Driver Package – Intel Corporation XDP USB Device
- Usermode Driver for XDP Hardware

as well.

To also uninstall the Macraigor\* Systems usb2Demon device driver please select

- OCD Commander

as well.

## 6 Features

### 6.1 Execution Trace using Intel® Processor Trace (Intel® PT) and Real-Time Instruction Trace (RTIT)

In addition to Last Branch Record (LBR) The Intel JTAG Debugger supports Execution Trace using two new hardware capabilities, Intel® PT and RTIT. These new hardware capabilities give the user increased trace depth over previous technologies, with little to no impact on target execution speed.

Support for these capabilities is limited to specific processors and/or platforms, for more information please contact customer support.

### 6.2 Agent-based debugging of UEFI (EDK2) BIOS

The Intel® JTAG Debugger now supports non-JTAG based debug of UEFI BIOS, this requires the use of a target-side debug agent. For more information see chapter 8 on [Agent-based UEFI Debugging](#).

### 6.3 OS Awareness / Kernel Module Debugging

The Linux\* OS awareness pulldown menu allows visibility of all currently active kernel threads. It also provides the ability to view a list of all currently loaded kernel modules with status information and memory location of initialization methods and cleanup methods. Setting whether to stop the target and commence debugging a kernel module on module load, initialization or cleanup/exit allows to start debugging a kernel module and loading its symbolic information. You can then set your breakpoints at the function entry points of the kernel module you want to debug, release the target using the *run* command and trigger an event that will cause the breakpoint to be hit to start your actual debug session.

You do not need to select kernel modules that are already loaded, but can add additional kernel module names to the list of kernel modules that are monitored and have the debugger stop at load, initialization or cleanup just as it would with the kernel modules that are already populated in the OS awareness pulldown menu as they were loaded during the Linux\* OS boot process.

To debug kernel modules the following steps additional to selecting or adding a kernel module in the module list are necessary.

In a debugger script or in the debugger console window enter the following commands:

```
SET DIRECTORY "<kernel module path>"
```

This path setting is necessary to enable the automatic source and symbol info mapping upon kernel module load as described above.

To use this feature for runtime loaded kernel module debugging you will need to have the kernel module **xdbntf.ko** running and installed on the target device. The folder <INSTALL-DIR>\kernel-modules\xdbntf contains code to generate a Linux\* kernel module that enables kernel module debugging with the Intel system debugger.

For generation simply transfer these files to your target system and invoke make. This will generate the kernel object **xdbntf.ko**.

To enable module debugging this object has to be loaded prior to starting the debugger via the command **insmod xdbntf.ko**. After finishing the debug session, the module can be unloaded with **rmmod xdbntf**.

## 6.4 Scripting Language

Create a batch file based on a rich set of powerful debugging script language commands and execute it in non-interactive mode. Results can be logged and analyzed afterwards.

## 6.5 Page Translation Table

Instant and simple resolution and translation between physical and virtual address space.

## 6.6 Unload of Symbol Information

To unload a symbol file, open the Load dialog. Click the “Unload Symbol File” tab and select the symbol file to be unloaded. This will remove the symbol information from the debug session. Removing symbol information is useful in order to load different or new symbol information.

## 6.7 Power Events Handling

The debugger can properly handle externally controlled power events without needing to close the debugger. If a target is reset or powered-off the debugger will identify the “Target power loss.” Once power is restored the debugger will attempt to halt the target at the reset vector.

## 6.8 NAND Flashing support on Intel® Atom™ Processor CExxxx

The Intel® JTAG Debugger supports flashing eMMC NAND partitions on Intel® Atom™ Processor CE4xxx, CE5xxx and Intel® Puma6™ Media Gateway based reference platforms.

## 6.9 Supported Flash Types

**Below is a brief description of the supported flash types for the specific target platforms:**

### **Intel® Atom™ procesor E6xx**

- SPI-FLASH

### **Intel® Atom™ procesor N2xx/D2xxx**

- SPI-FLASH



**Intel® Atom™ Processor CE4xxx,CE53xx**

- NOR Flash
- eMMC NAND Flash
- eMMC Boot Partitions
- eMMC User Partition

**Intel® Puma6™ Media Gateway:**

- SPI-Flash
- eMMC NAND Flash
- eMMC Boot Partitions
- eMMC User Partition

## 7 Agent Based UEFI Debugging

### 7.1 Pre-requisites

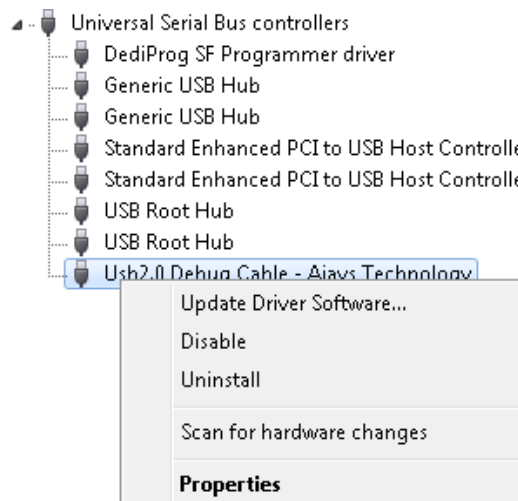
- [Ajays\\* NET20DC USB debug device](#) or a serial cable
- Debug target with EDK II based UEFI BIOS and enabled USB debug port or a serial port.

#### 7.1.1 Using Ajaysys\* USB debug device

If a serial connection is for debugging, there is no need to install Ajays\* drivers.

On initial USB debug connection use, the Intel® drivers for the Ajays\* NET20DC USB debug device need to be installed.

- Plug the Ajays\* device into a USB port on the host PC, open device manager and right click on the device.
- If the device isn't visible in the device manager list, unplug the device and connect the other end to the host PC.
- Select **Update Driver Software** and browse to the XDB installation folder. The drivers can be found in the following directory  
C:\Program Files (x86)\Intel\System Studio 2014.0.xxx\debugger\xdb\drivers\ajays-usb-dbg
- Select the 32 or 64 bit drivers depending on your system setup.
- If the device is in the device manager list, but “Update Driver Software” option is not available right after plugging in the debug device, wait until the option is available.



## 7.2 Setting up the Target

### 7.2.1 Hardware Setup

If you want to use the USB debug connection, the debug target must have an enabled USB debug port available. Check the target schematics to find the correct physical USB port. The target is connected via Ajays\* NET20DC USB debug device to the PC running XDB. The debug device provides a host-to-host connection between the PC and the target.

If you are using serial debug connection, the debug target must have a connected serial port.

### 7.2.2 BIOS Setup

Debug agent can only be used with EDK II based BIOS. To debug the boot phase, the BIOS for the intended target needs to be rebuilt. The debug agent must be included into the BIOS image and the target USB or serial driver has to be configured to support a debug channel.

1. Download the BIOS sources for your target.
2. Add or replace the latest SourceLevelDebugPkg from <https://edk2.svn.sourceforge.net/svnroot/edk2/trunk/edk2> to your BIOS build.
3. Modify BIOS configuration files to include the debug module and correct communication libraries.

For USB, set

```
SOURCE_DEBUG_ENABLE = TRUE  
  
SOURCE_DEBUG_USE_USB = TRUE
```

For serial, set

```
SOURCE_DEBUG_ENABLE = TRUE  
  
SOURCE_DEBUG_USE_USB = FALSE
```

4. In edk2 reference implementation, configuration files OvmfPkgIa32.dsc and OvmfPkgX64.dsc use these flags to include needed modules. If your BIOS sources don't have these configuration flags, see [Intel UEFI Development Kit Debugger Tool User Manual chapter 2](#) for instructions how to add the debug configuration.
5. If you are using serial connection, it is safest to disable flow control. Set the flag gEfiMdeModulePkgTokenSpaceGuid.PcdSerialUseHardwareFlowControl to FALSE.
6. Build and flash the images.

### 7.3 Connection Setup

If you are using a serial debug connection,

1. Connect the serial cable from the debug target to the debug host.
2. Modify the file `softdebugger.ini` to enable the serial connection.
3. Comment out the USB connection settings and uncomment serial connection settings. Make sure the Port is set to the correct COM port in you system.


```
[Debug Port]
;Channel = USB
;Port = \\?\USB2DBG
Channel = Serial
Port = COM1
FlowControl = 1
BaudRate = 115200
Server =
```

If you are using USB debug connection,

1. the Ajays\* NET20DC debug device must be connected with correct orientation or the connection may be unstable.
2. The debug target must provide the power to the device.
3. To check the connection orientation, open windows device manager and connect Ajays\* NET20DC debug device to the host PC.
  - If device manager screen flickers (is updated), the Ajays\* debug device is connected the wrong way.
  - Turn the device around and plug it in again.
  - The device may now not recognized by the host PC. Plug the device into a powered on target, and the device gets powered on and recognized.

After the Ajays\* NET20DC debug device is connected with correct orientation, disconnect the device, plug it first into the host and after that to the target. If you need to disconnect the device, make sure to always connect the host PC side first or the connection may not work.

## 7.4 Starting an Agent Based Debug Session

- Connect the host PC to the target with Ajays\* NET20DC debug device and launch XDB using the `start_xdb_UEFI_agent.bat` file.
- Power on the target and click on the  “connect” icon at the upper left corner of the XDB UI. If the setup is correct, XDB will connect and show disassembly on the screen.
- Add module names to the UEFI module watchlist to break at the load of the module.
- Let the target run. The execution will stop when a module with a matching name on the watchlist is loaded on the target.
  - Symbols for that module are automatically loaded to the debugger, if the symbol file is found at the same path on the host PC as read from BIOS. This is true if the PC used for debugging has the symbol files in the same path as used for building the BIOS. If symbol path doesn't match, you must browse to the symbol file location and load the symbol file manually to be able to debug the source code.
- After execution is stopped by a module load, the symbol file list on the left side of screen is updated.
- Browse to the entry point of the loaded module, set a breakpoint there and run target again.
- The module can now be debugged normally.

### 7.4.1 Handling the module watchlist

#### 7.4.1.1 EFI "WATCHLIST"

Prints the names and break information of modules on EFI modules watchlist.

#### 7.4.1.2 EFI "WATCHLIST ADD %s [INITBREAK=(ON/OFF)]"

Adds an EFI module to the module watchlist. Initbreak controls whether debugger breaks when a matching module is loaded or not. Initbreak parameter is not mandatory and is on by default.

Examples;

```
efi "watchlist add mymodule"
```

```
efi "watchlist add mymodule initbreak=off"
```

#### 7.4.1.3 EFI "WATCHLIST MODIFY %s INITBREAK=(ON/OFF)"

Modifies the watchlist entry initbreak information for an EFI module.

Example;

```
efi "watchlist modify mymodule initbreak=off"
```

#### 7.4.1.4 EFI "WATCHLIST REMOVE ALL"

Removes all EFI modules from the module watchlist.

#### **7.4.1.5 EFI "WATCHLIST REMOVE %s"**

Removes the selected EFI module from the watchlist.

Example;

```
efi "watchlist remove mymodule"
```

## 7.5 Troubleshooting

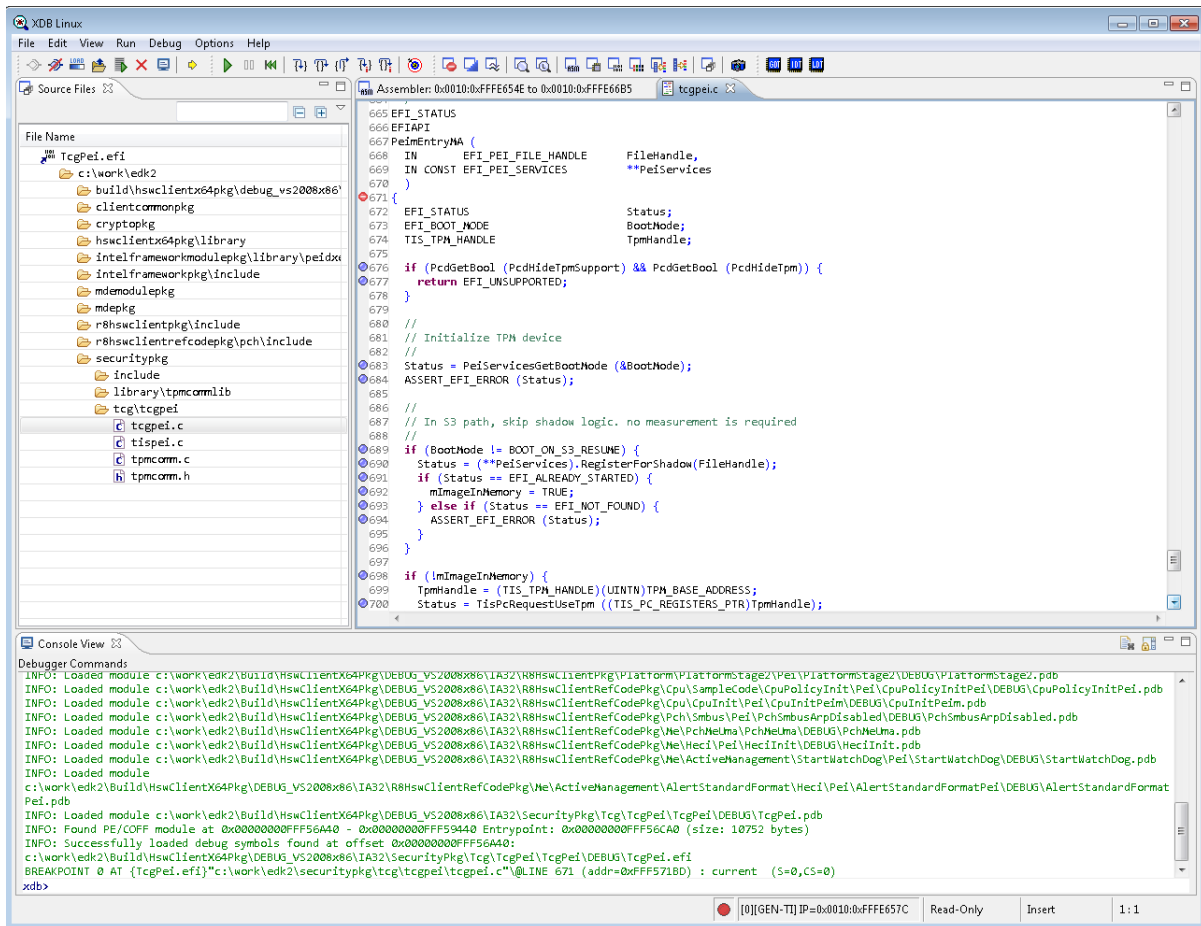
Problem	Possible root cause	Resolution
<p>USB connection to target doesn't work.</p>	<p>Target BIOS is not compiled with USB debug connection configuration</p> <p>Ajays* NET20DC debug device needs to be reset.</p> <p>Ajays* NET20DC debug device has been connected the wrong way</p>	<p>Make sure the BIOS image is compiled with configuration flags</p> <pre>SOURCE_DEBUG_ENABLE = TRUE</pre> <pre>SOURCE_DEBUG_USE_USB = TRUE</pre> <p>Close XDB and power down the target. Disconnect the Ajays* NET20DC debug device from both ends and plug it back in with the host side first.</p> <p>Disconnect the Ajays* NET20DC debug device from both host and target. Plug it back in only to the host PC and make sure device manager screen doesn't flicker (is not updated) when you plug it in Connect the debug device to the target.</p>

## 7.6 Known issues

### 7.6.1 Debugging a module on the watchlist

The debugger will stop the target when a module on the watchlist gets loaded. Target will stop at the debug agent code and load the symbols for the module on the watchlist. To debug the module you need to browse to the module source files and set a breakpoint somewhere in the code.

1. Add your module into the watchlist.
2. Wait for the module to get loaded and target to stop.
3. Open the “source files” window. The window is visible at the left side of the screen.
4. Browse to module source code and set a breakpoint.
5. Run the target. Execution stops at the breakpoint and you can debug the module.

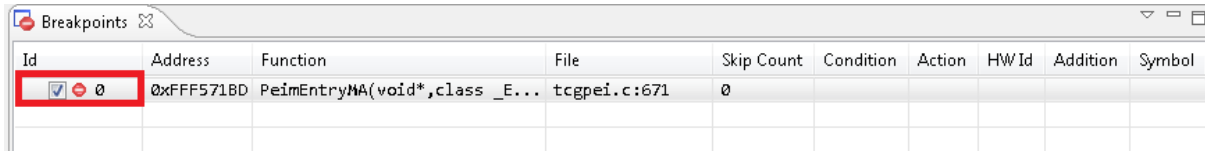




## 7.6.2 Using breakpoints

After reset, the breakpoints will get disabled in target even though they look enabled in the GUI. After target stops at a watchlist module load, the breakpoints need to be disabled and re-enabled for them to work.

1. Open the breakpoints window.
2. Disable breakpoint by unchecking the enable box.
3. Re-enable breakpoint by checking the enable box.
4. Run the target.



Id	Address	Function	File	Skip Count	Condition	Action	HWId	Addition	Symbol
<input checked="" type="checkbox"/> 0	0xFFFF571BD	PeimEntryMA(void*,class _E...	tcgpei.c:671	0					

## 7.7 UEFI Debug Appendix

### 7.7.1 Using an Emulator for EDK2 debugging

It is possible to use an emulator to run and debug edk2 reference BIOS. The debugger is connected to the emulator over a serial port emulation.

#### 7.7.1.1 Emulator BIOS Setup

The reference BIOS is built using VS2008 tool chain. You must have VS2008 installed on your computer before building the reference BIOS.

1. Check out the EDK2 sources from <https://edk2.svn.sourceforge.net/svnroot/edk2/trunk/edk2>.
2. Open command prompt to the root of downloaded EDK2 sources and run edksetup.bat
3. Give build command;  
**build -a IA32 -a X64 -p OvmfPkg\OvmfPkgIa32X64.dsc -v -D SOURCE\_DEBUG\_ENABLE -t VS2008x86**
4. The image will be created to `..\Build\Ovmf3264\DEBUG_VS2008x86\FV\OVMF.fd`
5. Create a new directory for running the BIOS you built.
6. Copy the image file into this directory and rename it to “bios.bin”.

#### 7.7.1.2 Serial emulation for emulator

1. Download and install “com0com” from <http://sourceforge.net/projects/com0com/>. This will emulate a virtual com port pair to connect XDB and target emulation. Windows must be in test mode for the installation to succeed. Test mode allows you to install unsigned drivers. See com0com readme for instructions how to enable test mode.
2. Create a virtual com port pair.
3. Open the `..\bin\softdebugger.ini` file in the XDB install folder.
4. Comment out USB configuration settings (Channel and Port) and uncomment serial configuration settings at the beginning of the file.

```
[Debug Port]
;Channel = USB
;Port = \\.\?\\USB2DBG
Channel = Serial
Port = COM1
FlowControl = 1
BaudRate = 115200
Server =
```

5. Change the setting “Channel” in `softdebugger.ini` file to one of the virtual COM port pair you created.

### 7.7.2 Setting up the emulator

1. Download Qemu\* version 0.13.0 from <http://homepage3.nifty.com/takeda-toshiya/qemu/qemu-0.13.0-windows.zip>. Using an unsupported Qemu\* version may cause problems with the serial connection.
2. Create a new folder for Qemu\* in the same location as BIOS run directory created earlier.
3. Open a command prompt in the new run folder where you copied the bios.bin file from edk2 build. Start Qemu\* with the command;  
`..\qemu-0.13.0-windows\qemu-system-x86_64.exe -L . -serial COMX`  
where COMX is the other end of the virtual com pair.
4. Qemu\* will prompt for connection settings. Select 115200 and hardware flow control.

### 7.7.3 Starting a debug session

1. Start XDB using the start\_xdb\_sd.bat file.
2. Add interesting modules to the watchlist.
3. Start Qemu\* with the command described in previous section step 3.

## 8 JTAG Debugger Usage Notes

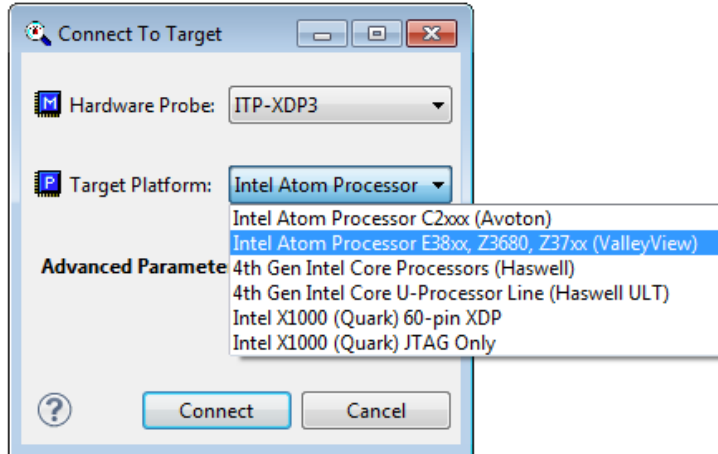
### 8.1 Starting the Debugger

#### 8.1.1 Using startup scripts in installation directory

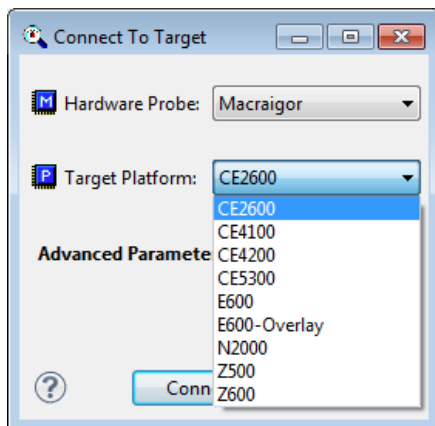
To start the Intel® JTAG Debugger for Intel® Atom™ Processor change into the C:\Program Files (x86)\Intel\System Studio 2014.0.xxx\ installation directory.

From there run the debugger launch shell script that best fits your host-target setup.

- Start\_xdb\_2014\_products.bat - this script allows the user to debug recent processors/platforms using the Intel® ITP-XDP3 device, such as:
  - Intel® Atom™ Processor C2xxx (Avoton, Rangeley)
  - Intel® Atom™ Processor E38xx, Z3680, Z37xx (ValleyView)
  - 4th Gen Intel® Core™ Processors (Haswell)
  - 4th Gen Intel® Core™ U-Processor Line (Haswell ULT)
  - Intel® Quark™ Processor X1000 60-pin XDP
  - Intel® Quark™ Processor X1000 JTAG Only



- Start\_xdb\_legacy\_products.bat - this script allows the user to debug older processors using either Intel® ITP-XDP3 or Macraigor\* usb2Demon\*, such as:
  - Intel® Atom™ Processor E600, N2000, Z500, Z600, Z2400, Z2500, Z2700, CE4100, CE4200, CE5300
  - Intel® Puma6™ Media Gateway



- Start\_xdb\_UEFI\_agent.bat - this script allows the user to debug UEFI BIOS over serial or USB ports
- Start\_xdb\_firmware\_recovery.bat - this script is for the sole purpose of recovering flash firmware on some embedded processors

### 8.1.2 Using Windows\* start menu

In the Windows\* start menu select

All Programs>Intel® System Studio 2014>Intel® JTAG Debugger Startup

Use the launch script entry for that best fits your setup.

## 8.2 Burning blank flash on Intel® Atom™ Processor CE5300 based platforms

Burning blank flash, requires that the SPI boot path is enabled and that the board is not configured to boot from eMMC.

If the board is configured to boot from eMMC and the eMMC flash is empty than the target is held in reset. This is indicated by an active yellow LED on the board. There are switches on the board that are used to select the boot path, on the platform code-named "Mt. Carmel" Fab B, these are switches SW3F1 switch 6 and switch 7. For SPI boot, switch 6 should be OFF and switch 7 should be on. Please ensure that SPI boot is enabled if the a blank flash is to be written.

### 8.3 eMMC Flash Recovery on Intel® Puma6™ Media Gateway

For flashing a blank or corrupted eMMC flash on a platform based on the Intel® Puma6™ Media Gateway we provide a special flash recovery script in conjunction with the Intel® JTAG Debugger GUI Flash Memory Tool plugin.

Please follow the steps below to recover corrupted eMMC flash or write to blank eMMC flash memory on a platform based on the Intel® Puma6™ Media Gateway:

1. Power cycle the board
2. Launch `start_xdb_firmware_recovery.bat` in a command line window from the `C:\Program Files (x86)\Intel\System Studio 2014.0.xxx\` directory.
3. Wait for the "Flash Recovery completed successfully!" message on the
4. Launch `start_xdb_legacy_products.bat` in `/opt/intel/system_studio_2014.0.xxx/debugger/xdb` from a command line window.
5. Close assembler window
6. `C:\Program Files (x86)\Intel\System Studio 2014.0.xxx\`
7. from a command line window Close assembler window
8. Open flash plugin
  - a. Select CE2600 in Board dropdown and eMMC-user in the Flash dropdown box
9. Set Data file to your CEFDK binary (`ce2600.bin`), set Offset to `0x80800` and select burn
10. Restart target and it should boot to CEFDK.

### 8.4 eMMC Flash on Intel® Atom™ Processor CE42xx and CE53xx

#### 8.4.1 Partitions

There are 3 partitions defined on eMMC. Our flash tool is capable of programming each of the 3 partitions: boot1, boot2 and the user partition.

#### 8.4.2 Addressing

Addressing space for each partition is independent. Therefore you should address the beginning of each partition as address 0. Currently 32KB is the minimum write or read size that we support.

### 8.4.3 Erasing

Block erasing is not implemented for this flash. eMMC flash can be written to without erasing first. Therefore the only erase we support is on the entire partition.

Note- Currently “Verification” is not supported on eMMC flash. Please do a backup and manual binary diff the two files.

## 8.5 FTL Lite NAND Flash Programming on the Intel® Atom™ Processor CE4100

**Note-** Please see the Platform Support Guide that came with your development kit for more detailed information about programming the FTL Lite partition in the Intel® Atom™ Processor CE4100 NAND flash.

The first 66 blocks of the flash memory is reserved as the FTL Lite partition. In this space each block is written redundantly 8 times per DHG’s spec. In order to minimize user complexity, our implementation of the FTL Lite partition accounts for these extra copies. However, by doing so, this requires a minimum read or write size to be 1 block (128 KB).

### 8.5.1 Addressing

The conversion from FTL Lite partition block number to address is simple however not straightforward. Since the redundant blocks are not addressable you must divide the block number by 8 (number of redundant copies) than multiply by block size (128KB). Below is a formula and table that converts some frequently used block locations. (As described in the Platform Support Guide)

$$\text{address} = \text{block \#} / 8 * 0x20000$$

<u>Data</u>	<u>Block #</u>	<u>Address</u>
MBH and BBT	0	0x 0000 0000
CEFDK	8	0x 0002 0000
Redboot	40	0x 000A 0000
Platform parameters	48	0x 000C 0000

### 8.5.2 Backup

When doing a backup use the same addressing method described above. For the length parameter, values must be a multiple of block size. Therefore 128KB (0x20000) is the minimum read size. Also keep in mind that only 1 block will be read from the first 8 blocks of redundant data. Our algorithm automatically accounts for the redundant block and will return only 1 of the 8 copies. Therefore reading 2 blocks at offset 0 will return block 0 and block 8.

**Warning-** Reading back a recently erased block of data will sometimes crash the target. ECC error logic isn't in sync with erased blocks and therefore can cause the target to hang.

## 8.6 Enabling Run-Time Loaded Kernel Module Debugging

To use this feature for runtime loaded kernel module debugging you will need to have the kernel module **xdbntf.ko** running and installed on the target device. The folder **/kernel-modules/xdbntf** contains code to generate a Linux kernel module that enables kernel module debugging with the Intel system debugger.

For generation simply transfer these files to your target system and invoke make. This will generate the kernel object **xdbntf.ko**.

The xdbntf Makefile has been modified to allow building within the CE Linux\* environment, but these modifications are not guaranteed to work with all CE Linux\* flavors. Please see the "read.me" in the xdbntf source folder for the latest info.

To enable module debugging this object has to be loaded prior to starting the debugger via the command **insmod xdbntf.ko**. After finishing the debug session, the module can be unloaded with **rmmod xdbntf**.

## 8.7 Unload of Symbol Information

To unload a symbol file, open the Load dialog. Click the "Unload Symbol File" tab and select the symbol file to be unloaded. This will remove the symbol information from the debug session. Removing symbol information is useful in order to load different or new symbol information.

# 9 Issues and Limitations

## 9.1 Target-specific issues:

**9.1.1 Release of Linux\* hosted 4<sup>th</sup> generation Intel® Core™ processor support delayed**  
Availability of Linux\* hosted support for debugging system software on the 4<sup>th</sup> generation Intel® Core™ processor (code-named "Haswell") via JTAG, has been delayed. This capability will be part of a future update release.

Please use the Windows\* hosted Intel® JTAG Debugger instead.

For information about early access to Linux\* hosted Intel® Core™ processor JTAG debugger support please contact [IntelSystemStudio@intel.com](mailto:IntelSystemStudio@intel.com).

### **9.1.2 Functional differences of 60-pin vs. 10-pin JTAG**

Some platforms (e.g. the Intel® Quark™ Processor X1000 based Galileo board) do not implement the full 60-pin debug port that is traditionally used on Intel systems, in this case the functionality of the debugger will be limited, especially in the following areas:

- Detection of reset by the debugger
- Initiation of reset by the debugger
- Halting the target at the reset vector

### **9.1.3 Platform reset policy may inhibit debugger operation**

Some platforms implement reset in such a way that the debugger may not be able to gain control of the target immediately after reset. This impacts the debugger operation in the following ways:

- The debugger may not be able to restore breakpoints after reset; the breakpoints may appear “enabled” in the GUI but will not in fact be enabled in the target.
- The debugger will not halt automatically at the reset vector; if the user wishes to debug early in the boot process there will be no way to manually initiate a halt quickly enough.

### **9.1.4 Platform security policy may inhibit debugger operation**

In some platforms the security policy may disable JTAG access to the CPU, this is intended to prevent reverse-engineering. In this case the JTAG debugger will successfully connect to the target, however it will not be able to discover any CPUs on the JTAG bus, and will warn the user that no CPUs are available. To resolve this issue please ensure that that platform firmware has enabled access to the CPUs via JTAG, this is typically done by flashing a special “debug” firmware into the target.

Also note that in some cases the CPU or CPU module may have physically disabled JTAG access, especially in production or near-production versions. In this case please work with the platform business unit to obtain JTAG-enabled hardware.

### **9.1.5 Target power management and platform power policy on tablet systems may inhibit debugger operation**

On some tablet designs the platform architecture includes aggressive power-management of the CPU, this will impact debugger operation in that a CPU in a low-power state cannot be accessed via JTAG. This will manifest in a variety of ways:

- Error messages indicating that “no threads are available”
- Error messages indicating that “target could not halt”
- No threads displayed in the hardware threads window



In general these problems will be mitigated by doing one or more of the following:

- Ensure that the CPUs are in an active state when the debugger is first started (e.g. in early boot firmware where no power management is present)
- Ensure that the OS has a workload that will inhibit low power states (e.g. play a video, run animated wallpaper, etc.)
- Disable low-power states in the platform when possible (generally a BIOS setting)

#### 9.1.6 Platform reset implementation may limit debugger-initiated reset:

Debugger-initiated reset is not an industry standard feature, it is implemented using sideband signals on the 60-pin Intel XDP port, and it is subject to the reset implementation on the target system. Some targets may not reset reliably via the debugger's reset/restart feature, this will typically result in a message such as "WARNING: target did not halt after reset, forcing a halt" being displayed in the debugger console, followed by additional error messages. In this case the user may need to manually initiate a reset on the target via buttons, debug card, etc.

#### 9.1.7 Platform reset implementation may limit ability to halt at reset vector:

Halting the CPU at the reset vector (first instruction fetched) is a CPU/platform dependent feature and may be limited due to target implementation details. The main impact to the user is:

- **inability to debug early platform boot code due to a runaway target.** In this case it may be necessary to build a special firmware with a hard-coded infinite loop early in the boot flow.
- **inability of the debugger to re-apply breakpoints after target reset.** In this case the debugger may show breakpoints as "enabled" in the GUI, but they will not be installed in the target, the user should manually halt the target, disable, re-enable breakpoints to ensure they are applied correctly.

## 9.2 Intel® Atom™ Processor Z3xxx and E3xxx specific issues

### 9.2.1 Platform power management policy may limit debugger control of the target:

The platform power-management policy may include power management of the CPUs, this may limit availability of debugger features when the threads are in a low-power state. Examples include:

### 9.2.2 Launching the debugger when the target is off or in a low-power state may cause unexpected behavior

The debugger needs to see all threads when connecting to the target in order to correctly initialize both the debugger state, and the debug resources in the target. The Debugger will report the number and type of threads observed during initialization (e.g. "INFO: Connected to Processor type: <name> (4 threads)") the user should confirm that this matches the expected configuration. Resetting the target (with the debugger running) should clear the issue and cause all expected threads to show up in the HWThreads window.

### 9.2.3 Hardware Threads window may show no threads/partial threads/disabled threads

Hardware Threads window may show no threads/partial threads/disabled threads due to the CPUs appearing/disappearing from the JTAG scanchain, and/or showing up in the JTAG scanchain as “disabled”. This condition should only happen when the target is in “run” state, when halted (e.g. from hitting a breakpoint) the debugger should correctly show all threads. **\*if the debugger halts and all threads are not shown\*** then the debugger is in an incoherent state and you may need to restart your debug session.

### 9.2.4 User-initiated Halt may occasionally return errors, especially if the target is in a low-power state.

Errors include “E-2201: Target has no active threads, this operation is not permitted.” and “E-2201: Target did not halt execution.”

This condition occurs when the debugger attempts to halt the target, but the CPUs are asleep and therefore unresponsive to debugger commands. Workaround: manually bring the CPUs out of sleep (e.g. by fiddling with the tablet) **\*or\*** try repeatedly to halt via the debugger, after 2-3 tries the target typically wakes up.

### 9.2.5 Kernel module load configurations may be unreliable

Due to limitations in silicon debug features, kernel module load notifications may not function correctly on Intel® Atom™ Processor Z3xxx and E3xxx based platforms. The following should be observed to work around this:

- **Software breakpoint in the target must be set prior to using xdbntf.** The breakpoint should be in an unused/unreachable code location, it is not necessary that the breakpoint is ever hit, its purpose is to enable the JTAG debugger redirection logic in the Silicon, which will allow the kernel module notifications to function correctly.
- **in some cases the notification will only partially work:** this will manifest as a hung system, with the JTAG debugger indicating a “running” state. In this case the user should manually halt the target, at which point the debugger will detect the notification, consume it, and resume target execution.

## 9.3 Installation related issues

### 9.3.1 Intel® JTAG Debugger is incompatible with the Intel® ITP II and/or PDT products

The Intel® JTAG debugger uses a software component that is shared with the Intel® ITPII Platform Debug Toolkit product, and only a single instance of this component can be installed on a host system. Due to this limitation it is not possible to install both the Intel® JTAG Debugger and ITPII / PDT on the same system. Note that that install may not fail, however the host system will be left in an unknown state and neither product is guaranteed to work correctly. To resolve this issue uninstall all instances of the Intel® JTAG Debugger and ITP II / PDT and then install only the product that you wish to use.

### 9.3.2 Sharedinfo.txt not writable

The debugger is dependent on writing to a file called “sharedinfo.txt” which is normally installed at “c:\ProgramData\Intel\DAL\MasterFrame\sharedinfo.txt”. Default security policy on the host system may cause this file to be read-only after Intel® System Studio installation. If this is the case then the user will observe an error message when connecting to the target. To resolve this issue either (a) run the debugger with administrator privileges or (b) change the permissions on this file (and any containing folders) to be writable.

### 9.3.3 Network access for MasterFrame.HostApplication.exe blocked by windows firewall

The debugger utilizes a server process to access the target, this “Masterframe” process will attempt to access the network when initializing, and this access may trigger a warning from the windows firewall. This is expected and the user should allow the access for the debugger to function correctly.

### 9.3.4 MSDIA DLL not registered

The debugger is dependent on a shared library provided by Microsoft\* for access to debug information generated with the Microsoft\* compiler. In some cases this library will not be correctly registered on the host system, which will lead to an error message when trying to load symbols for modules (e.g. EFI modules) that are compiled using the Microsoft\* toolchain. The user can resolve this issue by manually registering the correct library using an administrator command prompt.

From a command line window go to:

```
C:\Program Files (x86)\Common Files\microsoft shared\VC
```

Run:

```
regsvr32 msdia90.dll
```

if the msdia90.dll file is not found there, reinstall the Microsoft\* Visual C++ 2008 Redistributable Package after downloading it from Microsoft’s website

## 9.4 General Feature Limitations

### 9.4.1 Support for Intel® Atom™ processor bitfield editor register views

To receive information on how to access bitfield editor chipset register views for Intel® Atom™ Processors, please send an email to [EmbeddedDevTools@intel.com](mailto:EmbeddedDevTools@intel.com) for details.

### 9.4.2 Failure of initial debugger re-connection can lead to debugger hang

In some cases when using the “2014 products” startup script, the debugger will hang when re-connecting to the target. This is typically observed when the initial connect attempt has failed (e.g. due to the CPUs or target being powered off). Workaround: always shut down the debugger and re-launch it when re-connecting to the target.

### 9.4.3 Locals Window updates can be slow

The local window updates may be slow in many cases where it is evaluating many large structs or in scopes with many locals. If the slowness is noticed, it is recommended to close the locals window.

### 9.4.4 Kernel Threads Window Population Slow

The Linux\* OS awareness plug-in for the Intel® JTAG Debugger includes a Kernel Threads Window, that displays all current kernel threads and information about their state. When the Kernel Threads Window is opened it can take several seconds before the actual content is displayed. The initial window content of “No data.” will disappear once kernel thread data is available. This can take up to 20 seconds.

### 9.4.5 Debugger puts a Windows file system lock on symbol files

Currently when a Symbol file is loaded in the debugger a file system lock is placed on this file to prevent other processes from deleting or modifying this file. If this lock is preventing you from recompiling your program, simply use the Unload feature found in the Load Dialog. Unloading a symbol file will release the file system lock and allow you to modify or delete the symbol file without exiting the debugger.

### 9.4.6 Flash programming with empty flash parts (CE4200)

In order for the flash burning algorithm to execute the target must first be restarted and sitting at the reset vector. This is normally handled by the flash plugin. However, on CE4200 boards that do not have programmed flash parts, the platform’s reset does not function. Therefore, when programming these boards, a manual reset is required.

To successfully burn flash on platforms with blank flash, a user should follow these steps:

- 1) Connect the debugger, and verify that the target is halted
- 2) Manually reset the target by pressing the reset button on the target
- 3) Verify that the EIP is not pointing at 0xFFFF0
- 4) Open the flash plugin dialog and resume normal steps to program the flash.

### 9.4.7 Use of Macraigor\* usb2Demon to debug Intel® Atom™ processor CE4xxx based platforms may require board changes

On some of the Intel reference platforms for Intel® Atom™ processor CE4100 the RefDes serial resistor R6D20 may need to be replaced with a 0 Ohm resistor. On some of the Intel reference platforms for Intel(R) Atom(TM) processor CE4200 the RefDes serial resistor R4E5 may need to be replaced with a 0 Ohm resistor. Please refer to the board schematics and the platform design guides for details.

### 9.4.8 Older Macraigor\* usb2Demon 60pin connector headers

The reference voltage and signal voltage pins on Macraigor\* usb2Demon 60pin connector headers from prior to September 2010 are connected. Since those two voltages differ from each other on the Intel® Atom™ Processor E600 reference platform it is strongly recommended to only use recently purchased Macraigor\* usb2Demon 60pin connector headers with these platforms. Not following this recommendation can damage the reference platform.

#### **9.4.9 Backup of large flash partitions may fail**

It is recommended to avoid using the flash writer plug-in to backup large flash partitions (>200Mb) from the Intel® Atom™ Processor onto the workstation.

#### **9.4.10 Verification of flash content on Intel® Atom™ processor E6xx based platform**

Verifying flash content may only be possible immediately after writing it. When the system boots, it might modify the flash content (e.g. saving changed settings). When the flash content will be compared to the original file that got written to the flash, the verification will show differences.

#### **9.4.11 Target doesn't boot when Macraigor\* usb2Demon\* JTAG device is not initialized**

If the Macraigor\* probe is connected to the host system the first time, it doesn't get initialized until a debugger is using the probe. Without initialization the probe, the JTAG pins are in an undefined state and can prevent the target system from booting. This might lead to unstable behavior on the target including failing boot sequence. After the debugger has initialized the probe the target system should boot without problems.

This does not affect the XDP3 connection. XDP3 probe is automatically initialized when it gets plugged into the host system.

#### **9.4.12 Memory Reads with Un-Initialized Memory**

If any window other than the memory window is open on uninitialized memory (e.g. Assembly Window), memory read attempts may lead to the target entering an undefined state.

#### **9.4.13 Memory Writes to Un-Initialized Memory**

Memory writes to un-initialized or read-only memory (this includes setting software breakpoints or accessing memory mapped registers) can lead to a crash of the target or a loss of the target control. The debugger will not prevent these memory accesses when requested by the user (e.g. changing instructions in the disassembly window).

#### **9.4.14 Flash Writer disables pre-existing Breakpoints**

Flashing the BIOS will disable all code breakpoints and data breakpoints you may have had set prior to using the flash writer.

#### **9.4.15 Master Flash Header Read/Write not supported for Intel® Atom™ Processor CE4200**

On the Intel® Atom™ Processor CE4200 the Master Flash Header serves as a road map for the contents of flash that are processed by security and host firmware. It contains the location and size of each element in the flash, as well as a list of host firmware images that the security processor will attempt to boot. Currently the flash writer plug-in for the Intel® JTAG Debugger does not support writing or modifying the Master Flash Header.

It is of course possible to use the terminal Master Flash Header commands `mfhlist` `mfhinfo` and `mfhinit` in conjunction with the Intel® JTAG Debugger flash writer plug-in.

mfhlist provides the location of the Master Flash Header entries and where the current platform boot configuration expects the various flash images to be put. It's output can be used as a guidance for setting the start address when using the flash writer plug-in.

For NOR Non-Trusted Boot and NAND/eMMC Non-Trusted boot can be configured such that target boot is possible even if no Master Flash Header is present on the platform.

eMMC Trusted Boot does require the presence of a Master Flash Header and requires that the actual memory layout does match its contents.

Please read the Platform User Guide closely for further details on the Master Flash Header and its usage.

#### **9.4.16 Assembly Window Instruction Pointer Look-Ahead on Un-Initialized Memory**

When the assembly window tries to center the instruction pointer it will read memory ahead of instruction pointer to do so. If memory ahead of the instruction pointer is invalid this will lead to a target crash. To avoid this behavior reading memory ahead of the instruction pointer can be turned off using the `Options >Options ...` menu entry and deselecting it in the Assembler tab.

## **10 Attributions**

Portions of this software were originally based on the following:

- software copyright (c) 1999, IBM Corporation., <http://www.ibm.com>.
- software copyright (c) 1999, Sun Microsystems., <http://www.sun.com>.
- the W3C consortium (<http://www.w3c.org>) ,
- the SAX project (<http://www.saxproject.org>)
- voluntary contributions made by Paul Eng on behalf of the Apache Software Foundation that were originally developed at iClick, Inc., software copyright (c) 1999.

This product includes updcrc macro,  
Satchell Evaluations and Chuck Forsberg.  
Copyright (C) 1986 Stephen Satchell.

This product includes software developed by the MX4J project  
(<http://mx4j.sourceforge.net>).

This product includes ICU 1.8.1 and later.  
Copyright (c) 1995-2006 International Business Machines Corporation and others.

Portions copyright (c) 1997-2007 Cypress Semiconductor Corporation.  
All rights reserved.

This product includes XORP.  
Copyright (c) 2001-2004 International Computer Science Institute

This product includes software licensed from Macraigor Systems, LLC.  
Copyright (c) 2004-2009, Macraigor Systems LLC. All rights reserved.

This product includes software from the book  
"Linux Device Drivers" by Alessandro Rubini and Jonathan Corbet,  
published by O'Reilly & Associates.

This product includes hashtab.c.  
Bob Jenkins, 1996.

## 11 Disclaimer and Legal Information

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.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

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 information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized  
Intel® JTAG Debugger Installation Guide and Release Notes



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 order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: <http://www.intel.com/design/literature.htm>

MPEG-1, MPEG-2, MPEG-4, H.261, H.263, H.264, MP3, DV, VC-1, MJPEG, AC3, AAC, G.711, G.722, G.722.1, G.722.2, AMRWB, Extended AMRWB (AMRWB+), G.167, G.168, G.169, G.723.1, G.726, G.728, G.729, G.729.1, GSM AMR, GSM FR are international standards promoted by ISO, IEC, ITU, ETSI, 3GPP and other organizations. Implementations of these standards, or the standard enabled platforms may require licenses from various entities, including Intel Corporation.

BlueMoon, BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Inside, Cilk, Core Inside, E-GOLD, Flexpipe, i960, Intel, the Intel logo, Intel AppUp, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Insider, the Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel Sponsors of Tomorrow., the Intel Sponsors of Tomorrow. logo, Intel StrataFlash, Intel vPro, Intel XScale, InTru, the InTru logo, the InTru Inside logo, InTru soundmark, Itanium, Itanium Inside, MCS, MMX, Moblin, Pentium, Pentium Inside, Puma, skool, the skool logo, SMARTi, Sound Mark, Stay With It, The Creators Project, The Journey Inside, Thunderbolt, Ultrabook, vPro Inside, VTune, Xeon, Xeon Inside, X-GOLD, XMM, X-PMU and XPOSYS are trademarks of Intel Corporation in the U.S. and/or other countries.

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

Microsoft, Windows, Visual Studio, Visual C++, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java is a registered trademark of Oracle and/or its affiliates.

Copyright (C) 2008–2014, Intel Corporation. All rights reserved.