



White Paper

Robert Müller-Albrecht
Developer Products Division

Intel® Software Development Tool Suites for Intel® Atom™ Processor

Document Number: 319332-002US

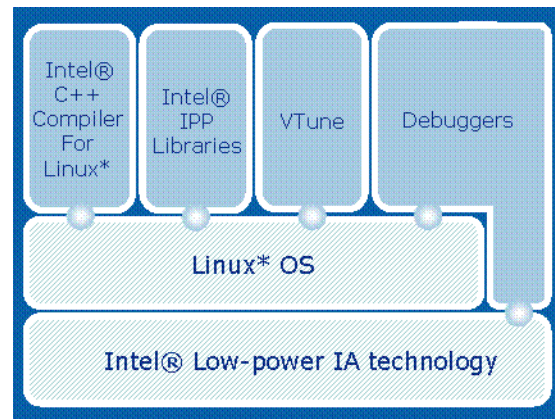


Introduction

This white paper gives a high-level overview of the Intel® Application Software Development Tool Suite and Intel® Embedded Software Development Tool Suite for Intel® Atom™ Processor and is targeted primarily towards technical decision makers. It provides a concise description of the available features and benefits, while examining key features in more detail.

Overview

The Intel® Application Software Development Tool Suite for Intel® Atom™ Processor and the Intel® Embedded Software Development Tool Suite for Intel® Atom™ Processor are solution sets covering all phases of development from coding and development to analysis, debugging, performance optimization and profiling. This Intel® Application Software Development Tool Suite consists of four different components. It includes the Intel® C++ Compiler for Linux*, the Intel® Integrated Performance Primitives, the Intel® VTune™ Performance Analyzer and the Intel® Application Debugger for Intel® Atom™ Processor. The Intel® Embedded Software Development Tool Suite for Intel® Atom™ Processor additionally offers an Intel® JTAG Debugger. A system debug solution for operating system bringup and driver development. All of these tool suite components have been specifically optimized and enhanced in their features to enable you to get the best performance and the best time-to-market out of your Intel® Atom™ processor targeted software development. These modifications and enhancements and how they specifically help with development, optimization and validation on Intel® Atom™ processor based devices is the focus of this whitepaper.



Rationale

Developing software applications and modifying and customizing device drivers for small form factor embedded systems running Linux requires a slightly different development paradigm than writing software for the native desktop PC or workstation or server centric usage models. The main difference is two-fold. Most likely the target device you are developing for does not have a keyboard and has a small touch screen based user interface. This is very end-user friendly, but not very practical for development on the device itself. Since your target device most likely is a closed system the Linux* OS image running on it may not even contain the GNU* GCC software development tools. These two factors mean that despite the platform similarities between a standard desktop PC and an Intel® Atom™ Processor based device usually the most appropriate and efficient way of developing for Intel® Atom™ Processor based devices is to do cross-development.

For Moblin* centric development there are two distinct methods of setting up this type of cross-development. Either the developer sets up a small isolated protected build environment on the development host. Usually this is referred to as a chroot environment in the Linux* world. The Moblin* Image Creator designed to configure, build and deploy customized Moblin* compliant OS images offers such an environment. Applications and build components can be build inside the Moblin* Image Creator build environment. The Intel® C++ Compiler for Linux* and the Intel® Integrated Performance Primitives can be installed directly into the Moblin* Image Creator to assist with this. Additionally you would probably like to have a test environment that allows you to not only develop your application on the host, but also do functional and user interface testing on the host before deploying the application onto the target device. At Moblin.org there is a ready to go Moblin* developer image available that could be loaded into a KVM* virtual machine. You could build your application inside this virtual machine and you could use the Intel® Application Debugger to find and correct issues in your application not only if it is running on real target hardware, but also if it is running inside a KVM* Virtual machine. For further details on this topic please consult the Moblin* Integration Guide included in the tool suite distribution and other Intel® Software Network articles focused on Moblin* targeted environment.

The Intel® Software Development Tool Suites for Intel® Atom™ Processor are a collection of development and analysis tools and utilities designed to help get the best performance out of your Intel® Atom™ processor targeted application while interfacing seamlessly with the development concept described above.

Intel® C++ Compiler for Linux*

The Intel® C++ Compiler for Linux* is a highly optimized compiler that plugs into your existing GNU* GCC installation reusing some of it's libraries as well as it's linker for maximum compatibility. Providing leading compiler technology from the silicon vendor who knows the underlying processor technology best, the Intel® C++ Compiler is the first compiler to specifically target Intel's new generation of Intel® Atom™ Processors with its optimizations. The idea is to give your application the additional performance edge needed for the best possible end-user experience.

The new Intel® Atom™ processors are in-order machines and do not provide an integrated out-of-order scheduler. The Intel® C++ Compiler models the Intel® Atom™ Processor pipeline and execution flow, thus enabling it to produce code with the optimum instruction execution sequence for low-power IA.

All the user needs to do to enable the in-order instruction pipeline modeling is to use the option switch **-xSSE3_ATOM** along with all the other compiler options they may be using to optimize their code. This allows for reordering the generated machine instructions and thus minimizing pipeline stalls

Please be advised that using **-xSSE3_ATOM** implicitly also enables aggressive loop-unrolling and execution streamlining using **Intel® SSE3** single instruction multiple data (SIMD) instructions and 128 bit registers for faster data throughput and parallelization. Consequently, if your host development environment does not support **Intel® SSE3** instruction you may want to do your functional and code-correctness testing on the host without using the **-xSSE3_ATOM** option and only enable compilation with **-xSSE3_ATOM** for the application before deploying it to the Intel® Atom™ Processor based target device. If the Intel® processor used for

your host development environment already supports SSE3 instructions this is nothing to be worried about and you can run your fully low-power IA optimized application in your platform emulation environment on your host as is.

Moblin* Integration

Installing the Compiler into KVM*

Whether you intend to do most of your development on a live Moblin* image running inside a KVM* virtual machine or whether you intend to do your development in a protected jailroot/chroot environment, there are tools components that should be installed on your development host system and should be independent of your Moblin* based Intel(R) Atom(TM) Processor target.

It is recommended to register for the Intel(R) Application Software Development Tool Suite 2.0 for Intel(R) Atom(TM) Processor at <http://software.intel.com/en-us/intel-compilers/> and download it at <https://registrationcenter.intel.com>. It is also recommended to install at least the Intel(R) VTune(TM) Performance Analyzer and the Intel(R) Debugger following the steps outlined in the product installation guide Install_All.htm. This will take care of installing the host components to take advantage of the Intel(R) Application Software Development Tool Suite's performance analysis, tuning and debug capabilities.

To install the Intel(R) C++ Compiler and the Intel(R) Integrated Performance Primitives as well as the idbserver debug agent on an existing live Moblin* image please follow the steps below:

1. Open a shell on the Moblin* image, enable root access and change to the /etc/yum.repos.d directory. There create a *.repo repository file with the following contents:

```
[inteltools]
name=inteltools
baseurl=http://downloads.moblinzone.com/development-tool-suite-2
enabled=1
gpgcheck=0
```

A template repository file can also be found at <http://software.intel.com/en-us/articles/installing-compiler-into-kvm-atom/>.

If your network setup requires the use of a proxy server you may want to add a line with the following contents

```
proxy=http://myProxyIP:myProxyPort/
```

where myProxyIP and myProxyPort reflect the actual proxy configuration in your network environment

2. Still at the Moblin* Linux* prompt enter the following command:

```
> yum groupinstall "Compiler" "IPP" "Server agent for Application Debugger"
```

and follow the installation messages.

This will install the Intel(R) C++ Compiler, the Intel(R) Integrated Performance Primitives (Intel(R) IPP) and the Intel(R) Application Debugger idbserver debug agent on the Moblin* image to be used locally.

Installing Tool Suite Components into Moblin* Image Creator

To install the Intel(R) C++ Compiler and the Intel(R) Integrated Performance Primitives into the Moblin* Image Creator jailroot environment and prepare the Moblin* Image Creator produced target image to also include the Intel(R) Application Debugger debug agent (ldbserver) and the Intel(R) VTune(TM) Analyzer Sampling Collector (SEP) please follow the steps below:

1. Go to <http://www.moblin.org> and download Moblin* Image Creator 2. Detailed Moblin* Image Creator documentation can be found at <http://moblin.org/documentation/moblin-image-creator-2/using-moblin-image-creator> and the Moblin* Image Creator itself can be downloaded at <http://git.moblin.org/cgi/moblin-image-creator-2>.
2. Follow the instructions you find at moblin.org for installing the Moblin* Image Creator.
3. Go to <http://software.intel.com/en-us/articles/moblin-integration-software-development-tool-suite-atom/> and download the two scripts **netbook-core-**

developer_JAILROOT.ks and **netbook-core-developer_TARGET.ks** .

4. To create a KVM* image or a Live-CD of the target Moblin* system, please use netbook-core-developer_TARGET.ks:


```
> moblin-image-creator -c
<path_to_ks>/netbook-core-developer_TARGET.ks -f raw --
cache=/tmp/mycache
```

The image created will contain The Intel(R) Application Debugger ildbserver and the Intel(R) VTune(TM) Analyzer Sampling Collector for target system located as a tarball in the /tmp directory. The Parameter '-f raw' of the moblin-image-creator command instructs the Moblin* Image Creator to create an image which can be loaded into a KVM* virtual machine. For more information on Moblin* Image Creator usage please consult moblin.org.

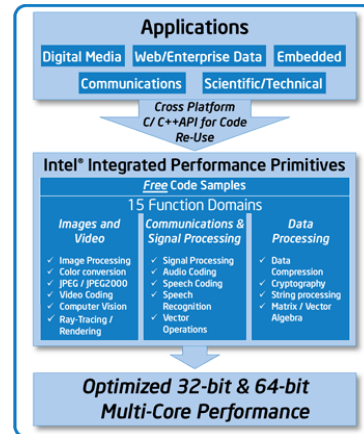
5. To create an image with the Intel(R) C++ Compiler & the Intel(R) Integrated Performance Primitives that you can then use to do builds in a protected jailroot/chroot environment before creating the target image please use netbook-core-developer_JAILROOT.ks:


```
> moblin-image-creator -c
<path_to_ks>/netbook-core-developer_JAILROOT.ks -f loop --
cache=/tmp/mycache
```

The image will contain the Intel(R) C++ Compiler and the Intel(R) IPP at /opt/intel/Compiler. The command parameter '-f loop' of moblin-image-creator tells it to create a loop image. You can mount this image and then chroot into it for your protected builds using the following commands:

- ```
> mount -o loop moblin-netbook-core-developer_JAIRoot.img /mnt/loop
> chroot /mnt/loop su
```

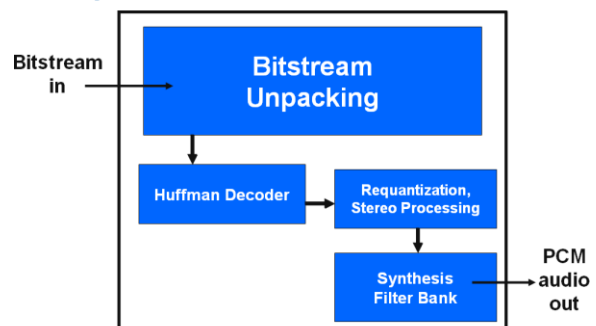
With Intel® Integrated Performance Primitives (Intel® IPP) application developers can concentrate on feature implementation rather than optimization of application code. Intel® IPP provides highly-optimized mathematical, signal and image processing functions for performance critical applications.



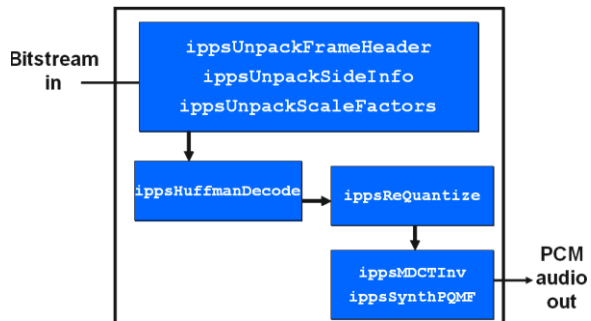
The Intel Integrated Performance Primitives supply a broad range of functions for multimedia, audio codecs, video codecs, image compression, image processing, signal processing, speech compression, computer vision and math support routines.

The Intel® IPP library supports both dynamic and static applications. The applications can be built for one specific cpu variant, or can include code that automatically chooses the best code for any cpu at runtime.

#### Example: MP3 Decoder



## Intel® Integrated Performance Libraries

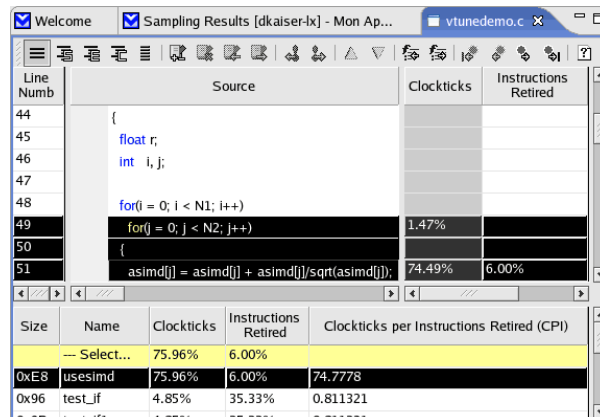


A typical example of how simple and straightforward the use of the Intel® Integrated Performance Primitives can be is the implementation of a regular MP3 decoder. Using a sequence of 7 Intel® IPP library functions it is possible to model the entire decode process and get a highly optimized decoder that can easily be recompiled and ported to any Intel® architecture based platform.

## Intel® VTune Performance Analyzer

The Intel® VTune™ Performance Analyzer makes it fast and easy to find performance bottlenecks with a list of the most active functions. Click on a function name to display the source and show the most time-consuming source statements. Furthermore Event Based Sampling support for the Intel® Atom™ Processor permits determining causes for execution stalls that impact performance.

Source and assembly views show you exactly which lines of code are taking the most time. For the VTune™ Performance Analyzer there is no recompile required. Unlike traditional instrumented profilers that make you recompile or modify your build script, just use your production executables.



To do performance sampling on the Mobile Internet Device you would either use a precompiled sampling collector that you copy over or you would unpack the sampling collector package locally on the target device using the following set of commands:

```

$ gunzip vtune91_target.tar.gz
$ tar -xf vtune91_target.tar
$ cd vtune91_target/

```

Assuming you have libstdc++.so.5 and the correct Linux headers installed you would then install the Intel® Vtune Analyzer Sampling Collector with the following script:

```

./install-vtune-sep.sh

```

### Running the Vtune™ Performance Analyzer Sampling Collector

1) Ensure that the sampling driver is loaded in the kernel and that the current user can access the driver:

```

$ cd /path/to/vdk[[BR]]
$ grep ^DRIVER_GROUP insmod-vtune.sh
$./insmod-vtune

```

2) To perform sampling, use the sampling collector

```

$ cd /tmp
$ sep -start -d 20 -out myData

```

3) To view resulting data use the command line sampling viewer tool:

```

$ cd /tmp
$ sfdump5 myData.tb5 -processes
display results by Process View
$ sfdump5 myData.tb5 -modules
display results by Module View
$ sfdump5 myData.tb5 -hf -mn vmlinux

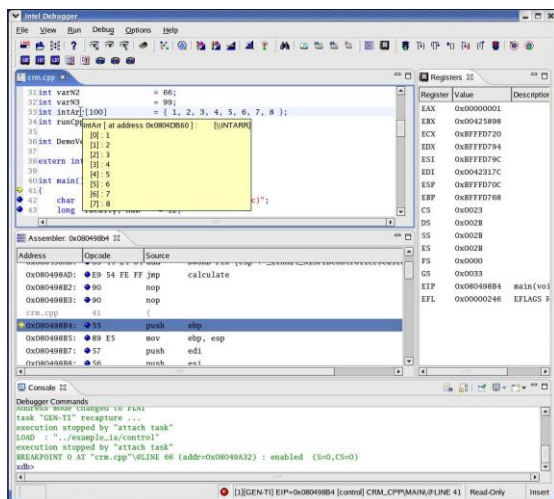
```

```
display all kernel functions that
have samples associated with them.
```

Alternatively you can of course also copy the \*.tb5 sampling results file over to your host system and use the full featured Intel® Vtune™ Performance Analyzer for graphical user interface supported analysis of your gathered performance data.

## Intel® Application Debugger

The Intel® Application Debugger for Intel® Atom™ Processor provides a full Eclipse RCP based GUI that helps to have better visibility of the application and system properties and thus have better control over the debugging process. Latest GUI technology enables developers to be more productive.



The connection to the debuggee is a remote connection using a TCP/IP interface and a remote debug server running on the target.

This module can be found at /opt/intel/atom/idb/2.x.xxx/server in the tool suite installation and simply copied over onto the target device to permit connectivity to it for application debugging on the actual Mobile Internet Device.

Once you connected the debugger to the debug server you can attach a debug session to any running process you have the sources for or you can even download an application binary onto the target through the debugger and start the debug session that way (this requires that the debug server on the target is launched with root privileges).

The Application Debugger is aware of multi threaded applications and uses a simplified method of 'stop all' debugging. Basically when any thread is hitting a breakpoint the debugger will freeze all threads at their current location. If you single step in one thread the others will enter run mode. The thread awareness of the debugger extends to providing you a graphical listing of all active threads and their properties and IDs.


Additionally, after applying a necessary kernel patch, you can use the execution trace facility to verify what instructions you actually have been executing. This is very useful if you are getting an incorrect result or you are trying to analyze the exact instruction execution flow leading up to a stack overflow or an exception.

This is done as follows:

- stop at a suitable location before you enter the interesting area. For example stop at the function call 'ShowProgress(l, range);'

- click on the trace enable button 

- continue until you passed the function or area of interest.

- click on the show trace button . The data from the target will now be downloaded which could take some time. Meanwhile you see an empty window. When the data is collected the execution path is regenerated with source code intermixed with the assembly instructions:

| Address                          | Opcodes        | Source                                  |
|----------------------------------|----------------|-----------------------------------------|
| shpr1me.cpp:FindPrimes Line 87   |                | ShowProgress(i, range);                 |
| 0x80493b8                        | E874FFFFFF     | FindPrimes+0x67: call ShowProgress      |
| 0x804940c                        | 55             | ShowProgress: pushl %ebp                |
| 0x804940d                        | 88EC           | ShowProgress+0x1: movl %esp, %ebp       |
| shpr1me.cpp:ShowProgress Line 49 |                | {                                       |
| 0x804940f                        | 83EC1c         | ShowProgress+0x3: subl \$0x1c, %esp     |
| shpr1me.cpp:ShowProgress Line 50 |                | int percentDone = 0;                    |
| 0x8049412                        | C745FC00000000 | ShowProgress+0x6: movl \$0x0, -4(%ebp)  |
| shpr1me.cpp:ShowProgress Line 53 |                | gProgress++;                            |
| 0x8049419                        | FF05C4E0508    | ShowProgress+0xd: incl 0x8054ec4        |
| 0x804941f                        | A1C4E0508      | ShowProgress+0x13: movl 0x8054ec4, %eax |
| 0x8049424                        | 8945F4         | ShowProgress+0x18: movl %eax, -12(%ebp) |
| 0x8049427                        | 0B45F4         | ShowProgress+0x1b: fldl -12(%ebp)       |
| 0x804942a                        | D95CF4         | ShowProgress+0x1e: fstps -12(%ebp)      |

## Standard Debugger

### Features

The Intel® C++ Application Debugger for Linux\* OS Supporting Mobile Internet Devices does not only provide the cross debug capabilities and the execution trace capability unique to Mobile Internet Devices, but also supports the standard capabilities most users expect from a modern symbolic source-code application debugger:

- Attaches to (and detaches from) a running process and debugs the corresponding program
- Loads a program into (and unloads a program from) the debugger, automatically creating and deleting corresponding processes as necessary
- Supports multiple-process debugging, where the processes may be associated with the same program or with multiple other programs:
  - Actively run one process at a time
  - Switch focus between processes
  - See processes and examine detailed process state
  - Set breakpoints for a specific process
- Debugs programs with shared libraries
- Provides language-specific command-expression evaluation
- Provides ability to “call” functions in a target process from within a command expression
- Displays the source listing of a program
- Sets breakpoints to stop program execution when specified sections of program code are executed

- Sets watchpoints to stop program execution when a specified area of memory or specified program variable is written
- Adds conditions to breakpoints and watchpoints so that program execution will only stop at the specified break or watch event when the condition is true
- Supports setting of pending breakpoints if a breakpoint location specified cannot be resolved to an address in the current debuggee at the time it is being set.
- Steps both into or over calls to routines
- Steps through the execution of a program one source line or one machine instruction at a time
- Examines the stack of currently active functions
- Examines and changes program variables and data structure values in same or in different scopes
- Examines and changes the contents of memory in various formats (including international character strings)
- Disassembles and examines machine code
- Examines and changes general purpose register values
- Supports mixed-language applications, C++ templates, C++ user-defined operators
- Provides a customizable debugging environment by using environment variables, initialization files, sourced scripts, aliases (i.e., parameterized macros), and debugger variables for commands and command sequences
- Supports in-place edit of assembly code in RAM for instant fix and execution replay.
- Regular expression searches of the symbol table
- Debugs optimized code:
  - In-lined instances of functions (show in backtrace and selectable for current focus)
  - Registerized variables
  - Semantic stepping
  - PC-to-source column mapping (for multi-statement lines)

These are solidly implemented key debugging features that allow programmers to debug at both the

source and machine levels in a single session by using a variety of interface options, which can be customized according to their preferences.

## Intel® JTAG Debugger

The Intel® JTAG Debugger is only available as part of the Intel® Embedded Software Development Tool Suite for Intel® Atom™ Processor. It provides the same powerful graphical user interface as the application debugger. Its use requires the availability of an Intel® eXtended Debug Port (XDP) on the target device as well as an ITP-XDP3 Intel® In-Target-Probe. Please contact your Intel Application Engineer if you require access to this type of device.

The target audience for the Intel® C++ JTAG Debugger are original equipment manufacturers and original design manufacturers (OxM), that require the capability of doing their own device driver development and low level OS kernel layer platform adaptations.

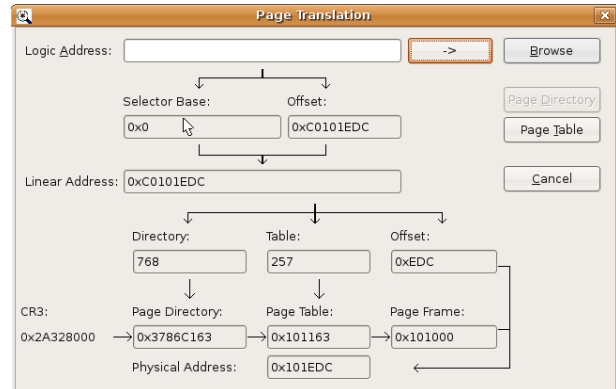
This specific class of software developers requires deep insight into the hardware the embedded OS is running on. At the same time developers should not have to give up the usability of a graphical user interface and the high level language support debug features they are used to.

Full Intel® Atom™ architecture support provides an in-depth view into the processor technology. Provides easy access to most Si specific features, including architectural registers, Intel® SSE3 and graphics companion chip registers. Registers can be viewed and modified in Bitfield Editors that provide in depth and fully documented convenient access.

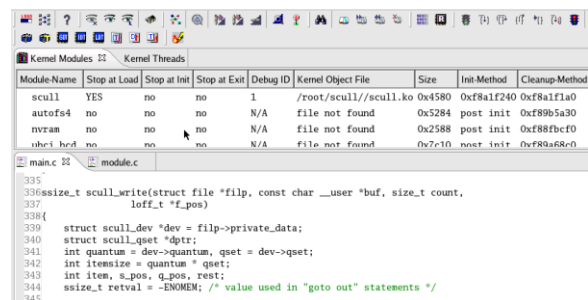
- Bitfield Editors are not only available for standard registers, but also for descriptor table entries. Not only can the descriptor tables be easily viewed and modified for debugging purposes, but it is also possible to conveniently access the Page

Translation Table and have the active memory mapping displayed in real time:

| Index | Value      | Comment                                                              |
|-------|------------|----------------------------------------------------------------------|
| 0     | 0x29422067 | ADDR=29422000 P=1 R/W=1 U/S=1 PWT=0 PCD=0 A=1 D=1 PS=0 G=0 AVAIL=000 |
| 1     | 0x2909A067 | ADDR=2909A000 P=1 R/W=1 U/S=1 PWT=0 PCD=0 A=1 D=1 PS=0 G=0 AVAIL=000 |
| 2     | 0x29B4B067 | ADDR=29B4B000 P=1 R/W=1 U/S=1 PWT=0 PCD=0 A=1 D=1 PS=0 G=0 AVAIL=000 |
| 3     | 0x29419067 | ADDR=29419000 P=1 R/W=1 U/S=1 PWT=0 PCD=0 A=1 D=1 PS=0 G=0 AVAIL=000 |
| 4     | 0x0000000C | ADDR=00000000 P=0 R/W=0 U/S=0 PWT=0 PCD=0 A=0 D=0 PS=0 G=0 AVAIL=000 |
| 5     | 0x0000000C | ADDR=00000000 P=0 R/W=0 U/S=0 PWT=0 PCD=0 A=0 D=0 PS=0 G=0 AVAIL=000 |
| 6     | 0x0000000C | ADDR=00000000 P=0 R/W=0 U/S=0 PWT=0 PCD=0 A=0 D=0 PS=0 G=0 AVAIL=000 |



- Execution Trace Support enhances the understanding of the flow of an executed program. It thus helps significantly with the isolation of memory leaks, data structure alignment and execution flow issues. Displaying execution trace for system debugging enables more effective debug cycles
- Linux OS Awareness for full understanding of the system behavior at all times. Display all relevant kernel information, active kernel threads and loaded kernel modules and debug them in the OS context.



This Debugger being based on the JTAG standard it provides direct hardware access and enables

developers to access Si specific features independent of any software running on the target processor.

The extended hardware access and the extended OS awareness of this debugger also means that it is possible to debug dynamically loaded kernel modules (i.e. device drivers) remotely via JTAG, by simply launching a kernel module provided with the tool suite on that target. This dedicated kernel module exports all the module load events and memory locations so the JTAG debugger with it's OS awareness add-ons can pick this information up and allow for easy and convenient device driver debugging. Please read the release notes and debugger documentation closely for details.

Finally the Intel® C++ JTAG Debugger installation includes the Intel(R) JTAG Flash Memory Programmer that allows downloading and burning images into target-device FLASH memory (BIOS / Firmware) without removing the chip. This is available through a debugger plug-in shared object that allows you to burn images into the target-device FLASH memory from within the debugger GUI and provides a convenient means for updating the BIOS on your Mobile Internet Device. If you do not want to use the Intel® C++ JTAG Debugger graphical user interface for this task, there is also a command line version of this flash programmer available.

## Conclusion

The Intel® Software Development Tool Suites for Intel® Atom™ processor are complete tools solution sets to address Intel® Atom™ processor specific software performance requirements, and to enhance the productivity and experience of the Linux-based system and application development process.

The Tool Suite covers the entire cycle of software development: coding, compiling, debugging, and analyzing performance. All included tools are Linux hosted and compatible with GNU tools.

As such they are an ideal supplement to your GNU GCC development environment to make it capable of efficient and optimized application development and deployment for your Intel® Atom™ processor based device.

## Where to find it

The Intel® Application Software Development Tool Suite for Intel® Atom™ Processor and Intel® Embedded Software Development Tool Suite for Intel® Atom™ Processor can be purchased or a 30 day evaluation license can be obtained the the tool suites can be downloaded from the Intel® Software Development Products Web pages (<http://software.intel.com/en-us/intel-compilers/>).

Customers have access to product updates and product support through the following Web pages:

- Intel Software Development Products Support:

<http://software.intel.com/sites/support/>

- Intel® Software Network Discussion Forums:

<http://software.intel.com/en-us/forums/software-development-toolsuite-atom/>

- Intel Premier Support:

<https://premier.intel.com/>

For product and purchase information visit:

[www.intel.com/software/products](http://www.intel.com/software/products)

Intel, the Intel logo, Intel. Leap ahead. and Intel. Leap ahead. logo, Pentium, Intel Core, and Itanium 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.

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

Copyright © 2009, Intel Corporation. All Rights Reserved.

0506/DAM/ITF/PP/500 319332-001

Document Number: 319332-002US

## Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the “Intel® Compiler User and Reference Guides” under “Compiler Options.” Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101