Intel® Software Development
Tools for Intel XScale®
Microarchitecture

Uli Dumschat
Intel Corporation
Software and Solutions Group
July 19, 2005
## Contents

**Executive Summary** .................................................. 3  
**The Problem : Developer Tools Requirements** ........... 3  
**Tools Overview** .......................................................... 3  
**Supported Processors** .............................................. 4  
**Supported Operating Systems** ................................. 4  
**Products** .................................................................. 5  
  * Intel® C++ Compiler 2.0 for Windows* CE, Professional ................................................ 6  
  * Intel® Debugging Extensions ........................................... 6  
  * Intel® C++ Software Development Tool Suite 2.0 for Intel XScale® Microarchitecture, Professional ........................................... 7  
**Compiler System** ....................................................... 7  
  * Language ............................................................... 7  
  * Compiler Switches .................................................. 8  
  * Optimization Switches .......................................... 8  
  * Compiler support for Intel® Wireless MMX™ Technology ........................................... 8  
  * Assembler ............................................................. 8  
  * Intrinsic Functions .................................................. 9  
  * Vectorization .......................................................... 9  
  * Object Format Compatibility .................................. 10  
**Debuggers** .................................................................. 11  
  * Simulator Debugger ............................................. 11  
  * JTAG Debugger ..................................................... 11  
  * Flash Memory Support ............................................ 12  
  * OS Awareness ....................................................... 12  
  * Common Debugger Features ................................. 12  
  * Debugger Script Language .................................... 13  
  * Full C/C++ Debugging Support ............................. 13  
  * Hardware Access ................................................... 13  
  * Execution Trace Support ...................................... 13  
  * Local Variable Window ....................................... 13  
**Conclusion** .................................................................. 14  
**Links to Additional Information** .............................. 14
Executive Summary

Intel's leadership in silicon design gives us unique expertise in creating software development tools that take full advantage of processor features. Today, expectations for system solutions are not based solely on silicon. Customers such as Original Equipment Manufacturers (OEMs) and Independent Software Vendors (ISVs) look for complete platform solutions that integrate the entire hardware and software stacks to provide optimal performance and flexibility. As the creator of Intel XScale® microarchitecture for phones and other handheld devices, Intel's deep knowledge provides a clear advantage in creating software-development tools. Keeping this knowledge clearly in mind, Intel has developed software tools for the silicon features of Intel XScale technology, as shown in Figure 1.

The Problem: Developer Tools Requirements

Individual on-chip components drive customer requirements for software development tools. A compiler system must support the microarchitecture (pipeline behavior), including intrinsic functions that address on-chip features (for example, Intel® Wireless MMX™ technology). Debugging systems must support all on-chip features, such as execution trace support, JTAG debugging, and control of coprocessors and peripheral devices. Developers need fully integrated tools to support their wireless-application development projects.

Tools Overview

Intel® C++ Software Development Tool Suite for Intel XScale Microarchitecture performs three functions:

- Code generation (compiler system)
- Code analysis (debugger systems)
- Integrated Development Environment (IDE) support

Figure 1. Requirements Driving Development of the Intel XScale® Microarchitecture Software Development Tool Suite
Figure 2 shows the components of these functions in the context of Intel XScale microarchitecture.

**Supported Processors**

Intel Software Development Tools support the latest processors based on Intel XScale technology for wireless devices such as PDAs, handheld devices, and cellular phone solutions.

- **Application Processors:**
  - Intel® PXA25x processor family
  - Intel PXA26x processor family
  - Intel PXA27x processor family

- **Communication Processors:**
  - Intel PXA800F

- **Technology Support:**
  - Intel® Wireless MMX™ technology
  - Intel Wireless MMX 2 technology

**Supported Operating Systems**

The leading operating systems in the wireless industry currently include the following:

- Symbian OS*
- Microsoft Windows* CE / Windows Mobile Software for Pocket PC and Smartphone*
- Linux*
- Palm OS*

---

**Figure 2: Components of the Intel® C++ Software Development Tool Suite for Intel XScale Microarchitecture, Professional**

The development tools fully support Intel Wireless MMX technology and the new Intel Wireless MMX 2 technology, which are mainly used for multimedia applications that require easy management and processing of data streams. This technology represents a next-generation update for the wireless space to the solid, mature foundation provided by the Intel MMX instruction set.
Products

The Compiler/Debugger systems Intel provides can be differentiated into two categories:

- Compiler/Debugger Tool Suite for development based on:
  - Symbian OS
  - Palm OS
  - Nucleus* OS
  - Linux OS (Debugger only)
  - OS independent

- Compiler/Debugging Extensions for development based on:
  - Windows CE
  - Windows Mobile Software for Pocket PC and Smartphone

OS-independent development, in this context, means first software/hardware integration, board bring-up, and low-level driver development, where no OS is involved at that stage.

- Symbian OS v9.1, 9.0, 8.1b (EABI)
- Palm OS v5.x,
- Nucleus* OS v1.14
- Linux* MV CEE 3.x

Table 1 shows an overview of the products and components within the tool set.

**Table 1: Compiler/Debugger Tools Overview**

<table>
<thead>
<tr>
<th>Intel® Tools Components</th>
<th>Intel® C++ Compiler 2.0 For Windows* CE, Professional</th>
<th>Intel® C++ Compiler 2.0 For Windows CE, Standard</th>
<th>Intel® C++ Software Development Tool Suite 2.0 For Intel XScale® Microarchitecture, Professional</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDE Support</td>
<td>Platform Builder</td>
<td>eVC++, Visual Studio* 2005</td>
<td>CodeWarrior*</td>
</tr>
<tr>
<td>OS Support</td>
<td>Windows CE 4.2, 5.0, WMS 2003, 5.0</td>
<td>Windows CE 4.2, 5.0, WMS 2003, 5.0</td>
<td>Symbian OS v9.1, 9.0, 8.1b (EABI) Palm OS v5.x, Nucleus* OS v1.14 Linux* MV CEE 3.x (Debugger only)</td>
</tr>
<tr>
<td>Compiler System</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Debugging Extensions</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>XDB Browser</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>eXDJTAG Driver</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>SIM, JTAG, APPS Debugger</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>OS Awareness plug-ins</td>
<td>Symbian OS* v9.1, 9.0, 8.1b (EABI), Palm OS* v5.x, Nucleus* OS v1.14 Linux* MV CEE 3.x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Distribution and Support</td>
<td>Intel, EPI, Sophia</td>
<td>Intel, EPI, Sophia</td>
<td>Intel, EPI, Sophia</td>
</tr>
</tbody>
</table>
**Intel® C++ Compiler 2.0 for Windows* CE, Professional**

This compiler provides the OEM solution for the Platform Builder used to create new Microsoft Windows CE operating-system configurations for handheld devices. The Intel® C++ Compiler is a plug-in solution for Microsoft development environments. After installing the tools, developers can choose between the Microsoft compiler and the Intel compiler, but the build environment remains the same. Due to its direct tailoring to Intel XScale microarchitecture, software developers can expect highly optimized code from the Intel C++ Compiler.

With the Intel C++ Compiler 2.0 for Windows CE, Professional, the Intel® Debugging Extensions are part of the package. The entire OEM package includes an installation for eVC++ and Visual Studio* 2005. With the “Pro” solution, OEMs obtain an application development system as well.

**Intel Debugging Extensions**

A typical problem that developers face when designing new hardware devices involves not having a tool available that allows access to all the hardware components on a chip. Intel Debugging Extensions provide this capability. For new software drivers, developers need all the detailed information of the peripheral registers that are used to configure the on-chip devices (for example, USB, LCD driver, UARTs, timer, digital I/O, and so on). The Intel XScale microarchitecture provides more than 800 registers. Access to this register information is crucial; otherwise, development of low-level software drivers becomes a time-consuming process.

Intel Debugging Extensions provide users with full access to all the peripheral and coprocessor registers of Intel XScale microarchitecture. Using the peripheral register browser (XDB Browser) and clicking on a specific register invokes the Bitfield Editor, which displays all flags and bits of a peripheral register. All register details for the expected device behavior are documented within the Intel Debugging Extensions, reducing the inconvenience of referring to manuals.

Providing an extra tool within the Microsoft Platform Builder, these host-side extensions communicate through either TCP/IP or a JTAG connection with the target system. For JTAG communication, the Intel Debugging Extension installation package contains a specific eXDI driver (eXDI JTAG driver). Once this driver is installed, the Microsoft Platform Builder can communicate with the target system through a JTAG interface. Device manufacturers, in particular, use this solution to debug and observe code segments such as the startup code within the OEM_Init() layer of Microsoft Windows CE.

The Intel XScale microarchitecture contains an on-chip trace mechanism, which records code as it executes. In addition to standard debugging methods, with the Intel Debugging Extensions on the host side, developers can easily read and analyze the trace buffer. Looking back into the program's execution history is easy.

The bitfield editor helps to verify and change register contents, with just a mouse-click.

**Figure 3: The Bitfield Editor**
The Intel tools provide Independent Software Vendors (ISVs) with an Intel C++ Compiler solution that provides an equally high level of optimization. Developers can install this product into the Microsoft Embedded Visual C++* or Microsoft Visual Studio* 2005 application development environment. The major difference between the OEM and ISV solutions is that the OEM package contains additional eXDI JTAG driver support.

Intel® C++ Software Development Tool Suite 2.0 for Intel XScale® Microarchitecture, Professional

This product line contains support for different operating systems that are used in handheld devices, as well as support for OS-independent software development. Board developers argue the merits of OS-independent support, since basic start-up routines (such as board-bringup code) are OS-independent. But which parts of the development tools are OS dependent? Operating system dependence refers mainly to the awareness plug-ins for the debugger, while the compiler performs more generically in terms of binary output format, debug symbol information, and other similar information.

The new Application Binary Interface for the ARM* Architecture (known as EABI) helps and supports the interoperability between different compiler and debugger products. If an OS and the appropriate libraries are ABI-compliant, that would mean another compiler could be used, without the need to recompile the ABI compliant libraries.

A compiler system is shipped with the tool suite, as well as a bundle of different debuggers (including Simulator, JTAG and an OS-aware debugger), for all phases of software development. In terms of IDE compatibility, the tool suite fits into Metrowerks CodeWarrior* IDE.

Compiler System

The compiler system forms the heart of a software development tool environment. User expectations are high for compiler systems, and justifiably so. Compiler performance poses one of the most important considerations driving customers to choose a new compiler system. Nevertheless, compatibility with other components and other compiler systems are also key factors.

Language

The Intel compiler is ANSI C/C++ compliant. Legacy code can be compiled without major obstacles. In terms of specific language extensions and intrinsic functions (see the Intrinsic Functions section of this document), the syntax is also compatible with that of other development systems, such as Microsoft development tools.

With regard to the assembler tool, which is often considered to be a part of the compiler system, certain requirements exist. The mnemonic code often orients with GNU or ARM style. Optimally, the compiler supports both styles through a selectable option switch.
Compiler Switches
Normally, a compiler remains a hidden product that lacks a GUI. Often, an Integrated Development Environment (IDE) or Make file drives a compiler. IDEs, in particular, provide helpful tools such as editors, configuration tools, a build manager, and version control. Developers expect a new compiler that replaces an old one to provide the same functionality in terms of compiler directives or compiler option switches and to be compatible with the previous system. The Intel C++ Compiler’s modular structure allows the Intel compiler team to adapt the core compiler technology to new environments by modifying the appropriate configuration files. Supported IDEs are Microsoft eMbedded Visual C++, Platform Builder for Microsoft Windows CE, Visual Studio 2005, and Metrowerks CodeWarrior. If required, the compilers can be command-line driven as well.

Optimization Switches
Table 2 shows some of the optimization switches that developers can apply to software.

<table>
<thead>
<tr>
<th>Optimization Switch</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/O{1</td>
<td>2</td>
</tr>
<tr>
<td>/Qip</td>
<td>Enable single-file interprocedural optimizations</td>
</tr>
<tr>
<td>/Qipo</td>
<td>Enable multi-file interprocedural optimizations</td>
</tr>
<tr>
<td>/Qunrollx</td>
<td>Set maximum number of times to unroll loops to x</td>
</tr>
<tr>
<td>/Qx</td>
<td>Enable maximum optimizations</td>
</tr>
<tr>
<td>/Og</td>
<td>Enable global optimizations</td>
</tr>
</tbody>
</table>

While an optimization switch optimizes on a module level, pragmas are a useful technique for optimizing application code fragments (see the following text box). The developer specifies the parts that must be optimized within his application. Remember that optimization is not always the answer. Optimizing can potentially have negative side effects because highly optimized code often cannot be easily debugged; it makes sense to decrease the level of optimization in some cases.

Pragma syntax e.g.
#pragma optimize( “[optlist]”, {on | off} )

Compiler support for Intel® Wireless MMX™ Technology
The new Intel compiler includes features that accommodate the new Intel PXA27x processor family. The Intel Wireless MMX technology is based on SIMD technology. SIMD stands for Single Instruction Multiple Data, which means that multiple data (eight pixels) can be simultaneously processed with one instruction. Within the Intel C++ Compiler, a switch associated with the Intel XScale technology helps to identify potential high-level C language constructs (for example, FOR loops) that can be turned into Intel Wireless MMX instructions, making the application run faster.

The Intel C++ Compiler supports development of highly optimized multimedia applications using Intel Wireless MMX technology at three levels:

- Assembler support
- Intrinsic support
- Vectorizer option-switch support

Assembler
The Intel Wireless MMX technology includes 43 instructions that provide equivalent functionality to Intel MMX technology and the integer part of SSE technology. The assembler tool directly supports these basic instructions in addition to the Intel XScale instruction set. Therefore, using the mnemonic code to create Intel Wireless MMX technology-optimized routines is accomplished using standard methods. The compiler system can assemble and link Intel Wireless MMX instructions that are passed via inline assembly code (from C language) or via user-written assembly code.

tmcrr wR0, R0, R1
waddbus wR0,wR0,wR1
tmrcc R0, R1, wR0
Intrinsic Functions

Intrinsic functions can be thought of as “macros.” These can be used from C/C++ levels with all the comfort of using functions (for example, passing parameters) without the overhead of real functions. The compiler replaces the intrinsic functions with the real Intel Wireless MMX instructions or instruction sequences. The compiler takes care of detailed instruction scheduling and register allocation.

The next example shows how wireless MMX instructions can be used in C/C++ applications; “__m64” is a new data type that is available for 64-bit operations.

```c
#include <mmintrin.h>
__m64 fun_add(__m64 m1, __m64 m2)
{
    __m64 m3;
    m3 = _mm_add_pi8(m1, m2);
    return m3;
}
```

The following code is generated for the above example:

```
.global fun_add
fun_add:
    waddb  wR0, wR0, wR1 @ 7.18
    bx     r14 @ 9.8
@ mark_end;
.data
@ -- End  fun_add
```

Vectorization

The vectorizer is a code-optimization method that the compiler uses. Instead of executing a FOR loop in a scalar fashion, the vectorizer intelligently turns the algorithm into the appropriate vector operations. The performance of the vectorizer, however, depends on the quality of the written C code. The better the developer structures loops to allow the vectorizer to identify data parallelism opportunities and translate the code into Intel Wireless MMX instructions, the better the performance results.

The following example shows how the vectorizer tries to convert a FOR loop:

```c
void multi (short *a, short c){
    int i;

    #pragma vector aligned
    for (i = 0; i < 100; i++){
        a[i] = a[i] * c;
    }
}
```

The “vector aligned” qualifier in the example is a request for the loop to be vectorized with the aligned instructions.

The generated assembly code for this example when the vectorizer is enabled follows:

```
@ mark_begin;
.L1.1:                          @ Preds .L1.0
    .align 2
    .global multi
    multi:
        mov    r2, r1 @ 3.1
        mov    r1, #0 @ 8.1
        tbcsth wR0, r2 @ 10.19
        sub    r0, r0, #4 @ 10.19
    .L1.2:                          @ Preds .L1.2
    .L1.1
        ldr    r12, [r0, #4] @ 10.12
@ mark_end;
.data
@ -- End  multi
```

The generated assembly code for this example when the vectorizer is enabled follows:
The Intel compiler systems produce object output in the following formats:

- ELF with DWARF® 2.0
- COFF with CodeView* debug symbol information

These formats satisfy the most common binary output available from compilers today.

Figure 5 shows compatibility with other tools’ components. The compiler system shown on the left side of the figure creates different output formats to interact with third-party debugger systems, while the Intel debuggers can ‘understand’ third-party compiler output formats as well. The debugger system handles two different files – the debug symbol information and the executable image file. The debug symbol file comprises a table that translates between the source code on the debugger host side and the addresses on the target side where the application code executes. The actual image file is the application, which developers can either download through the debuggers, or they can boot the image from FLASH memory on the target device. The communication between the debugger and the target system can be via serial connection, USB, TCP/IP, or JTAG.

Object Format Compatibility

Figure 5 shows a high-level view of how Intel achieves compatibility between its Software Development Tools and other third-party-vendor tools.

Object Format: ELF, COFF
Debug Information: Dwarf 2.0, STABS+, Codeview

Image downloaded through debugger or loading from FLASH

Figure 5: The Build Process Happens with a Mouse Click
Debuggers

Intel provides a set of Simulator, JTAG, and OS-aware debuggers designed for all phases of software development (see Figure 6).

These debuggers are Intel XScale microarchitecture-centric and used for handheld devices and phones, meaning that they are developed to support the hardware architecture in detail. Depending on the stage of system development, designers can use different debuggers.

**Simulator Debugger**

The XDB Simulator is an instruction-set simulator that runs on PC platforms and simulates the Intel XScale technology-based devices and peripherals. The second logical part of the product is the debugger on top of the simulator. This allows low-level register debugging, as well as high-level C++ application debugging. One of the key features of the XDB Simulator Debugger is the simulation of on-chip devices for the Intel XScale microarchitecture, such as an LCD display, UARTs, DMAs, OS timer, and an Interrupt Controller. With only a mouse-click within the XDB Simulator Debugger, developers can access, inspect, validate, and modify all peripheral registers used in configuring on-chip devices.

Furthermore, developers can inspect internal registers within the Intel XScale microarchitecture. Viewing these registers is extremely helpful in determining the internal status of a peripheral device – information that cannot be viewed on real silicon.

**JTAG Debugger**

System integrators have clear requirements for their debugging tools. OEMs who integrate OS layers with their hardware devices need non-intrusive debugging tools that allow developers to inspect hardware without adding extra software solutions to a target system. JTAG provides a common method used for on-chip level debugging. Once the debugger (on the host PC side) connects to the silicon through an appropriate JTAG device, developers can access registers and ensure that hardware runs properly. In this state, debugging becomes possible without the aid of any loaded software clients. The XDB JTAG Debugger GUI supports various third-party JTAG vendor solutions, which are available from EPI, Sophia, and Macraigor.

---

**Figure 6: Simulator, JTAG and OS-aware Debuggers—Helpful Tools for the Product Development Cycle**

![Diagram showing the product development cycle with stages of hardware and software development, including Simulator, JTAG, and OS-based debugging tools.](image-url)
FLASH Memory Support
The XDB JTAG Debugger provides FLASH memory support, whereby developers can burn the basic software loader, startup routines, or complete OS images and applications. The XDB JTAG Debugger provides a wide set of FLASH programming algorithms, which make it easy to download software into the onboard memory. There is no extra tool required, other than the XDB JTAG Debugger.

OS Awareness
Deeply ingrained into the design philosophy of Intel debuggers is flexibility in integrating new debugger features. Such new feature integration is achieved through plug-ins. One of these features, called OS awareness, takes into account the behavior of an operating system when debugging a multitasking application. A new menu entry – the OS – enhances the look and feel of the debugger.

Figure 7 shows a Symbian OS-aware plug-in. With the aid of this plug-in, developers can easily inspect tasks, thread lists, chunk lists, and so on.

The kernel of the OS maintains this information. The debugger can view and modify this information at any time, independent of the executed user application software. If the debug infrastructure of the OS supports it, developers can set complex, task-specific breakpoints. Such breakpoints allow program execution to stop only when code executes inside of a specific task.

Designers can use OS awareness plug-ins in conjunction with a JTAG debugger or a ROM monitor-based debugger, whereby communication occurs through TCP/IP, USB, or serial connection.

Common Debugger Features
The XDB Debuggers provide a common look and feel for all debugger variants. Once a developer becomes familiar with the Simulator Debugger, all other variants become intuitive.

The XDB JTAG Debugger provides FLASH memory support, whereby developers can burn the basic software loader, startup routines, or complete OS images and applications. The XDB JTAG Debugger provides a wide set of FLASH programming algorithms, which make it easy to download software into the onboard memory. There is no extra tool required, other than the XDB JTAG Debugger.

Deeply ingrained into the design philosophy of Intel debuggers is flexibility in integrating new debugger features. Such new feature integration is achieved through plug-ins. One of these features, called OS awareness, takes into account the behavior of an operating system when debugging a multitasking application. A new menu entry – the OS – enhances the look and feel of the debugger.

Figure 7 shows a Symbian OS-aware plug-in. With the aid of this plug-in, developers can easily inspect tasks, thread lists, chunk lists, and so on.

The kernel of the OS maintains this information. The debugger can view and modify this information at any time, independent of the executed user application software. If the debug infrastructure of the OS supports it, developers can set complex, task-specific breakpoints. Such breakpoints allow program execution to stop only when code executes inside of a specific task.

Designers can use OS awareness plug-ins in conjunction with a JTAG debugger or a ROM monitor-based debugger, whereby communication occurs through TCP/IP, USB, or serial connection.

Common Debugger Features
The XDB Debuggers provide a common look and feel for all debugger variants. Once a developer becomes familiar with the Simulator Debugger, all other variants become intuitive.
Debugger Script Language

Behind each debugger button, a specific script-language command executes. Developers can easily “learn” these commands by “recording” them into script-language batch files while they interactively debug the application. Once recorded, they can immediately execute the recorded script-language batch file, saving time and allowing developers to easily reach and execute complex debugging scenarios.

OEMs frequently use large script-language batch files for automated debugging, testing, and validation. Script files developed for the Simulator Debugger can be applied on the JTAG version, which is connected to the real target.

Full C/C++ Debugging Support

The debuggers support all features of C++ programming, including setting breakpoints into C++ templates, resolving multiple inheritances, inspecting classes/objects and their members and methods, and overloading operators.

Hardware Access

Intel Debuggers grant accessibility to Intel XScale technology and on-chip peripheral device and coprocessor registers, shortening the validation process of low-level software drivers. Developers can access all peripheral register bits and modify them without rebuilding the developed code for testing purposes. The bitfield editor window thoroughly documents the description of each bit (flag) within all peripheral registers, as shown in Figure 9.

Execution Trace Support

As previously described, the Intel XScale microarchitecture supports a hardware-tracing concept known as execution trace, where a buffer logs the execution of a program. The advantage of such execution trace is clear: while a debugger merely freezes application execution upon encountering a breakpoint and provides an actual state of variables and registers, an execution trace provides developers with detailed insight into the history of an executed program. A sample execution trace appears in Figure 10.

Figure 10a. XDB Debuggers’ Trace Capabilities Allow Historical Insight into Executed Program – Mixed Mode (Assembler and C/C++)

Figure 10b. Execution Trace in Source-code View

The developer can choose between ASM or C/C++ view. This feature aids in searching for hard-to-detect errors.

Local Variable Window

This feature displays the actual scope of executed code. Local and actual application code variables are visibly displayed, making it easy to verify valid variables within code segments.
Conclusion

The Intel® Software Development Tools for Intel XScale® microarchitecture address wireless devices based on Intel® Personal Internet Client Architecture (Intel® PCA) processors and provide a sophisticated set of compilers and debuggers designed to seamlessly guide developers through the entire system- and application-software design process. The compilers are designed to provide maximum system and application software performance for Intel PCA processor-based devices.

The Compilers are tailored to the Intel XScale microarchitecture, including full multimedia design support on Intel® Wireless MMX™ technology and the new Intel® Wireless MMX™ 2 technology (available in future silicon). The outstanding compiler performance helps to create efficient system and application software from C/C++ level. The Intel debugger technology helps to shorten the design cycle and helps to make the whole edit, compiler, and debug cycle more transparent.

The debuggers allow access to the Intel PCA processor-specific co-processor and peripheral device registers, and to modify settings just by a mouse-click. Execution Trace support, Flash burning capabilities, Intel Wireless MMX technology register access, as well as full OS awareness help developers to create new wireless devices in a shorter time.

The Intel® product offering contains two major product lines:

a) Intel® C++ Software Development Tool Suite 2.0 for Intel XScale® Microarchitecture, Professional

The Tool Suite provides a highly optimizing C++ compiler, a set of various debuggers (Simulator, JTAG, Application Debugger), and Integrated Development Environment (IDE) support that are strongly linked to the silicon features. The tools provide support for the Intel® family of wireless devices and phones based on Intel XScale microarchitecture. The supported operating systems are Symbian OS*, Palm OS*, Nucleus* OS, and Linux* (debugger support only). The Tool Suite is highly recommended for OS-independent software development (board bring-up software, etc.) as well.

b) Intel® C++ Compiler for Windows* CE, Professional & Standard

Intel also offers system and application software development products that support the Microsoft Windows* CE and Microsoft Windows Mobile Software*-based development. Developers can obtain a significant performance boost by using the Intel® C++ Compiler for Windows* CE, Professional or Standard. While the Professional solution addresses mainly OEMs and ODMs with system software development, the Standard version is highly recommended for application software development.

Both products contain the same Compiler technology, and Debugging Extensions which enhances Microsoft’s Debugging technology by providing additional browser windows that allow access to Intel PCA processor-specific hardware features (co-processor, peripheral device registers, Intel Wireless MMX technology, execution trace, etc.). The OEM version contains additional JTAG support and contains the Standard version for application development as well.

The Intel Software Development Tools for Intel XScale microarchitecture incorporates our thorough knowledge of the microarchitecture and Intel Wireless MMX technology, as well as our foresight into future innovations anticipated for these technologies. Developers can use these powerful tool solutions to implement their current and future wireless designs.

Links to Additional Information

Intel Software Products

Tools Distribution Channels:
Embedded Performance, Inc.
Sophia Systems
For product and purchase information visit: www.intel.com/software/products