



白皮书

Robert Müller-Albrecht

开发人员产品部门

移动互联网设备上的设备驱动程序调试

文件编号: 319429-001US



简介

为移动互联网设备（Mobile Internet Device）开发基于定制设计的平台或者嵌入式应用程序的中心层面是增强和定制底层OS的功能集，对于封闭系统更是如此。此外，您可能希望连接到专用硬件平台扩展并为此编写自己的驱动程序。一些多媒体编解码优化也可以很好地应用于设备驱动程序水平。

针对移动互联网设备的Intel® C++ JTAG Debugger for Linux* OS提供的功能集可以帮助进行OS配适和驱动程序开发，并能为缺陷修复、验证和质量保证提供帮助。在本白皮书中，我们主要讨论调试Linux OS增强、Linux核心组件以及运行时加载的内核模块的方法。

概述

包含于支持移动互联网设备的Intel® C++ Software Development Tool Suite for Linux* OS中包含的支持移动互联网设备的Intel® C++ JTAG Debugger for Linux* OS是一个全图形用户界面的系统调试器。使用它需要目标设备有一个 Intel eXtended Debug Port(XDP)和一个 ITP-XDP3Intel IN-Target-Probe。

Intel® C++ JTAG Debugger的目标用户是初始设备制造商和初始设计制造商(OxM)，他们需要具备独立开发设备驱动程序和采用低水平OS内核层平台的能力。

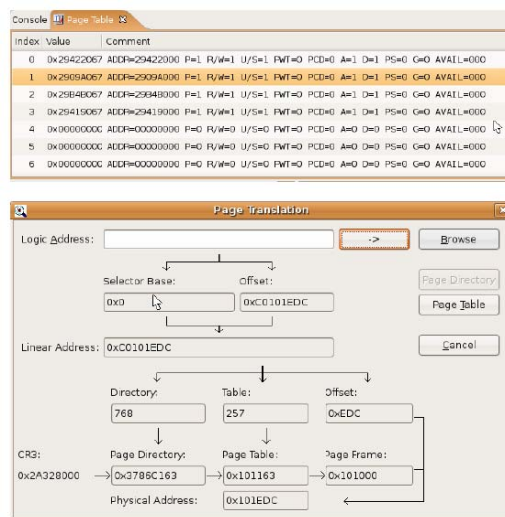
这类软件开发人员需要深入了解嵌入式OS运行的硬件。同时，开发人员不必放弃易于使用的图形用户界面以及他们习惯的高级语言支持调试功能。

完整的Intel® Atom™构架支持提供处理器技术的完整视图。可以轻松访问大部分特定于Si的功能，

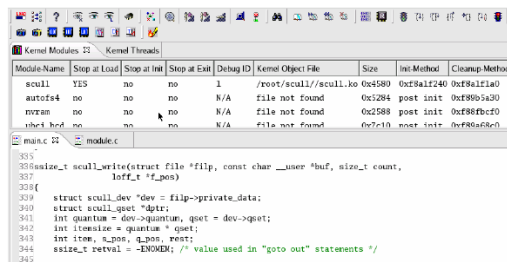
包括构架寄存器，Intel® SSE3 和图形配套芯片寄存器。Bitfield Editors提供深入全面的文档型便捷访问，可以在其中查看和修改寄存器。

Bitfield Editors不仅适用于标准寄存器，也适用于描述符表条目。

- 不仅可以很方便的进行查看和修改描述符表以进行调试，而且可以很方便地访问Page Translation Table和实时显示活动内存映射。



- 执行追踪支持增强了对执行程序流的理解。这非常有助于防止内存泄漏，数据结构校准和执行流问题。显示系统调试执行追踪提高了调试效率。
- Linux OS Awareness功能能够在任何时间全面掌握系统的行为。显示所有相关的内核信息，活动内核线程和加载的内核模块，并在OS环境下对它们进行调试。



该调试器扩展的硬件访问和OS感知功能还意味着，它能够通过JTAG远程调试动态加载的内核模块（即设备驱动程序），只需在目标上启动工具套件提供的内核模块即可。这个专用的内核模块可以导出所有模块加载事件和内存位置，因此JTAG调试器及其OS感知插件可以利用这些信息，轻松便捷地进行设备驱动程序调试。更多细节请阅读版本说明和调试器文档。

基本原理

JTAG不支持目标软件，这给通过它进行设备驱动程序和系统服务调试增添了困难。由于设备驱动程序通常实现为运行时加载的动态内核模块，所以需要调试器实施一个导出内核加载和卸载事件的机制。内核水平符号导出及事件通知的目的是使调试器监视OS内核符号，这些符号包含内核模块的内存加载地址信息，以及有关内核模块状态地址、加载和初始化方法的信息。

达到这种目的的方法有很多，传统的方法需要调整要调试的内核模块。这是一个激进的方法，需要通过定义或删除内核模块内的调试排列以释放代码块。这种方法使错误或运行时问题的分析变得很复杂，有可能只有在最终用户收到设备驱动程序后才能显示出来。

另一种方法需要一个OS内核的内核补丁，用来触发Linux*OS导出所有必需信息。这种操作实际只需要内存几千字节的系统开销。当没有连接调试器来查询导出符号时，这些补丁对运行时没有任何影响。但是值得注意的是，这种方法会产生调试钩子，这可能让竞争对手可以很容易地分析代码库。但这至少避免了一种情况，即不必在每次调试的时候都要构建一次设备驱动程序代码。

最好的解决方案既不需要内核模块调整也不需要内核补丁，而是内核模块本身来实施内核模块加载信息导出功能。这样，用户可以选取已有的任何目标平台而不必重新编译内核或者要调试设备驱动程序的调试对象。

本白皮书提供如何使用支持移动互联网设备的Intel®C++ JTAG Debugger for Linux* OS 执行调试的简明教程和使用场景。

构建内核模块以导出加载信息

运行时加载的内核模块调试功能是 Intel® C++ JTAG Debugger中Linux* OS感知插件的一部分，用户需要在目标设备上安装并运行内核模块

idbntf.ko后才能使用。文件夹

`/opt/intel/xdp/1.0.xxx/kernel-modules/idbntf`

包含生成Linux*内核模块的编码，可以启用内核模块调试功能。使用

`/opt/intel/xdp/1.0.xxx/kernelmodules/idbntf/Makefile`

在此文件夹中运行**make**即可创建**idbntf.ko**内核模块。

要生成，只需将这些文件转移到用户目标系统，然后调用**make**。这将生成内核对象 **idbntf.ko**。

要启用模块调试，此对象必须在通过命令**insmod idbntf.ko**启动调试器之前加载。然后，它将导出。初始化方法和**cleanip**方法将信息从目标加载到调试器。完成调试对话后，可以使用**rmmod idbntf**卸载模块。

启动调试对话

1. 首先，在shell环境下，确保XDP Debugger启动脚本位于

```
/opt/intel/xdp/1.0.xxx/xdp.sh
```

设置如下

```
#!/bin/bash
```

```
# Intel(R) JTAG Debugger for MIDs
```

```
# Copyright (C) 2000-2008 Intel
Corporation. All rights reserved.
INSTALLDIR=/opt/intel/xdp/1.0.008
```

```
export
```

```
LD_LIBRARY_PATH="$INSTALLDIR/lib:$INSTALLDIR/gui:$INSTALLDIR/plugin/ia/linux:$INSTALLDIR/plugin/ia/trace:$INSTALLDIR/plugin/ia/flash":$LD_LIBRARY_PATH
```

```
export
```

```
PATH="$INSTALLDIR/gui:$INSTALLDIR/lib":$PATH
```

```
export IDB_GUI_DEBUGGER="../lib/XDB"
dbggui -tgttype 'JTAG IA' -IUDGmode -
plg
'libguiialin.so,libiatrace.so,libplgflash
w.
so' -core 1 -target
device='XDP3'
scanchain='TargetPlatform' hotdebug
"$@"
```

如果出现以下情况：它不包含上述设置；In-Target Probe eXtended Debug Port设置没有指定XDP3；指定目标不是预期的处理器或者Linux OS感知的共享对象插件之一；Flash Writing 丢失，那么请在xdb.sh脚本上修改设置

2. 检查XDP3 JTAG探测器是否已经与目标板上标有JTAG的eXtended Debug Port稳固连接。同时检查探测器与USB电缆和实验电脑上USB端口的连接。

3. 更改为调试器目录

```
chdir /opt/intel/xdb/1.0.xxx
```


4. 启动XDB Debugger

```
./xdb.sh
```

5. 如果连接失败请检查Linux Console Window

调试Linux内核

用户与JTAG接口连接后，即产生控制转移。如果用户试着将鼠标移动到目标设备上或者在命令shell上输入命令，它将不会反应——不管是否在重置后直接停止或者已经完全启动了操作系统。

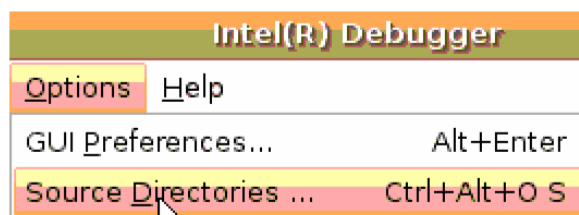
现在，可以单击调试器菜单栏的“load”按钮并导航到Linux内核源代码树中的 vmlinux kernel ELF/Dwarf2文件，它与运行在MID目标的Linux图像匹配。

也可以选择在控制台窗口中输入

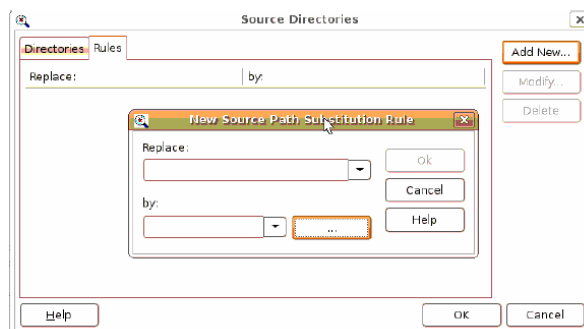
```
LOAD /SEGMENT /DEBUG /NOLOAD /GLOBAL  
OF  
"/usr/linux-2.6.22.i686/vmlinux"
```

调试器将加载和解释这个大型OS内核文件，之前可能会出现一段无反应的时间。

要正确映射与位于调试主系统上的符号信息相关的源位置路径，需要通知调试器源树的顶级目录位于调试主系统上的位置。用户可以导航到Options下拉菜单并选择Source Directories。



选择Rules标签并单击Add New...后，可得到下面的对话框，用户可以输入正确的源映射。



也可选择在控制台窗口中输入下面的命令，这和上文基于GUI的源映射方法的效果相同。

```
SET DIRECTORY /SUBSTITUTE = """/usr/linux-  
2.6.22.i686/"
```


现在。我们已经确保对Linux* OS内核具有完全的符号调试能力，可以开始调试这个内核了。

出于对本白皮书目的的考虑，让我们从最开始着手，从Linux内核及入口点开始。


我们首先要将目标处理器设置成初始状态，可以在调试器控制台窗口内发出restart命令来实现。接下来，我们需要解压OS内核并开始实际运行系统启动过程。

在Linux*上，这个进入点通常称为start_kernel()。然后我们在Linux内核启动上设置一个硬件断点。注意：必须是一个硬件断点，因为在启动过程的早期没有完全配置RAM内存。在存储端，当通过启动编码出现单步执行时，我们应该更进一步，并通知调试器暂时将所有断点作为硬件断点处理。可以通过在调试器控制台窗口输入

set option /hard=on
进行此项操作。

可以通过使用断点对话框  或者在控制台窗口发出下面的命令设置断点。

set break at start_kernel hard

要运行到内核入口点，可以单击  按钮或者在调试器控制台窗口输入

Run

在运行BIOS和OS引导程序后，目标执行将在函数入口点start_kernel()暂停。这时，用户可以看到源窗口并能准确地追踪所处的Linux OS位置。

在调试器控制台窗口发出命令

```
run until sched_init
```

可以执行OS的调度程序初始化。

```
run until mwait_idle
```

可以到达中心OS空闲循环

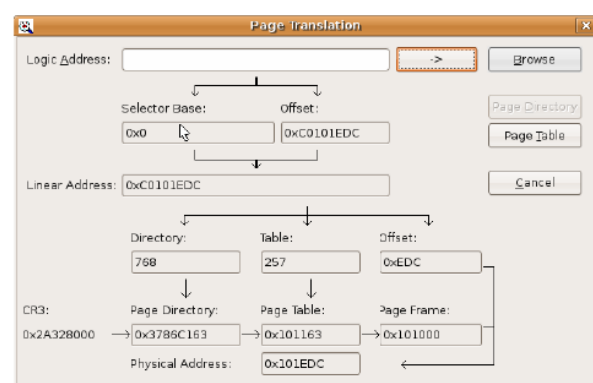
要理解OS内存配置，可近距离观看描述符表




，也可以通过查看页面转化表



（见下图）直接观察物理内存向虚拟内存转化的图形表示形式。




要得到当前程序计数器位置的页面转化，检查EIP

寄存器  的值并将其复制粘贴到Logic

Address区域。单击  按钮将得到页面转化的完整表示形式。

如果要检查实际的页面表条目，可以单击

，这将出现下表

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	0x2984B067	ADDR=2984B000 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

单击任何一个检索条目都会弹出一个比特组编辑器，它可以提供个别页面表配置的详细比特视图和当前设置的相关解释。

调试设备驱动程序

假设idbntf.ko模块已经加载到目标中且内核模块加载信息将导出到主机的调试器上。让我们用以以太网驱动程序为例说明如何在OS启动过程中对自动加载的内核模块进行调试。首先，要确保构建了内核模块并启用了符号信息生成功能，这样，一旦调试器知道加载内存加载地址，调试器便有机会对源进行映射。在调试器自身内，我们需要告知Linux OS感知的插件如何发现内核模块源，可以通过下面的命令进行此项操作：


```
set directory "/usr/linux-2.6.22.i686/drivers/net/"
```

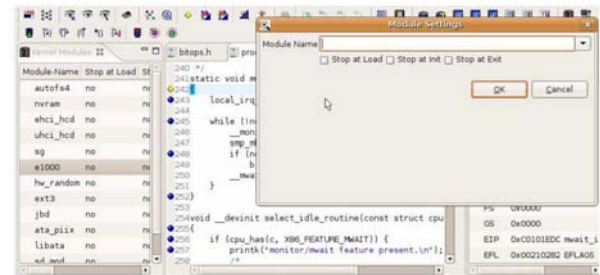
```
OS "setdir \"/usr/linux-2.6.22.i686/drivers/net/\""
```

当运行的系统加载了以太网驱动程序且建立了以太网连接之后，我们将控制调试器并中止引导进程。这需要对OS引导顺序进行修改，使idbntf.ko成为第一个加载的内核模块，至少应该在要调试的设备驱动程序之前加载。接下来，我们只需在调试器中通知Linux kernel OS感知的插件停止驱动程序初始化方法。

可以通过在调试器控制台窗口输入下面的命令来进行此项操作：



```
OS "stopinit e1000 /on"
```

也可以选择通过内核模块窗口进行此项操作。右键单击该窗口中的e1000内核模块，将会弹出一个环境菜单，可以通过这个菜单设置调试器，以停止初始化方法。



在以太网驱动程序初始化之后，我们将停止init_method并加载符号信息，所以我们即可看到源代码。

向下拉动源代码并在特定函数上（即IP地址获取函数）设置一个断点。

点击Restart按钮并运行按钮。等待OS重新启动，可以看到它如何在预定路线上停止以太网驱动程序加载。

启动后加载的内核模块调试过程完成之后，对设备驱动程序或引导过程完成后加载的内核模块与我们讨论的很相似。任意选取一个名为scull.ko的内核模块为例

在调试器中，通过调试器控制台窗口，再次通知Linux OS感知的插件内核模块源的位置：

```
set directory "/home/lab/scull/"
```

```
OS "setdir \" /home/lab/scull/\""
```

然后打开Linux Modules窗口，单击右键并在下拉菜单中选择“add”。可以看到所有在启动过程中加载的其他内核模块。也会发现，大部分内核模块都没有状态文件。对于带有符号信息的内核模块的位置，在弹出的对话框中输入“scull”并选择“Stop at Exit”现在，可以通过运行命令或者按钮释放目标。

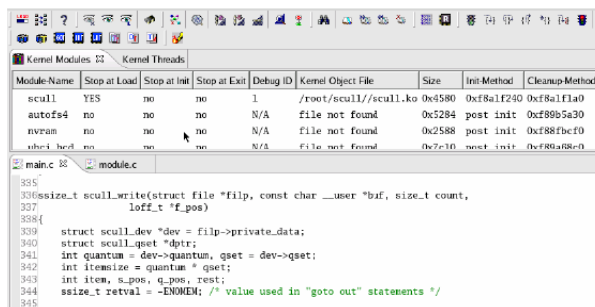
PuTTY/SSH或Telnet 到释放的和运行的目标上。在转变到滚动目录后，通过输入下面的内容对内核模块进行初始化

```
./scull.init start
```

调试器在scull init方法处如期停止

在scull_read()函数上设置一个断点并再次释放目标

然后对/dev/scull0设备发送回显。调试器暂停，可以步进该函数。



结束语

支持移动互联网设备（MID）的Intel® C++ Software Development Tool Suite for Linux* OS是

一个完整的工具解决方案集合，可以满足MID软件性能要求，可以提高基于Linux的MID系统和应用程序开发流程的生产力，并增强了体验。

该工具套件涵盖了整个软件开发流程，包括编写代码、编译、调试和分析性能。包含的所有工具都可以用于Linux并且与GNU工具兼容。

因此，它是您GNU GCC开发环境的理想补充，能够有效地优化移动互联网设备的应用程序开发和部署。

从何处获取

支持移动互联网设备（MID）的Intel® C++ Software Development Tool Suite for Linux* OS可以免费下载，对软件的支持可以从Intel® Software Development Products网页

（<http://www.intel.com/software/products/>）购买。客户可以通过以下网页访问产品更新和产品支持：

- Intel Support and Downloads:
<http://www.intel.com/support/>
- Intel Software Development Products Support:
<http://www.intel.com/software/products/support/>
- Intel Software Development Products Self Help:
<http://www.intel.com/support/performance/tools/index.htm>
- Intel® Software Network Discussion Forums:
<http://softwareforums.intel.com/ids>
- Intel Premier Support:
<http://premier.intel.com/>

有关产品和购买信息，请访问：

www.intel.com/software/products

Intel、Intel徽标、Intel. Leap ahead和Intel. Leap ahead徽标、Pentium、Intel Core和Itanium是Intel Corporation及其子公司在美国和其他国家或地区的商标或注册商标。

*其他名称和商标可能是其他公司的资产。

本文档中提供的信息专门针对Intel产品提供。本文档未以禁止反言或其他方式授予任何知识产权的许可，无论是明示的还是暗示的。除非有有关该产品的其他Intel销售条款规定，否则Intel不承担由此导致的任何责任，也不承认任何有关该产品销售权与/或者产品使用权的明示或暗示的授权，其中包括以特殊目的、以营利为目的的授权，或者对专利权、版权、或其他知识产权的侵害。Intel产品不用于医疗、救生、生命维持应用程序。Intel可能随时更改规范或产品说明，恕不另行通知。

Copyright © 2006, Intel Corporation。保留所有权利

0506/DAM/ITF/PP/500 312568-001

1 <http://developer.intel.com/software/products/compilers/clin/docs/manuals.htm>

2 <http://developer.intel.com/software/products/compilers/cwin/docs/manuals.htm>

3 <http://www.streamline-computing.com/>

4 <http://www.etnus.com/>

白皮书 <标题>

文件编号: 319429-001US