



Hardware Assisted Debug with Embedix Linux

- Introduction
- Types Of Interface Device
- Recommended Units
- What Targets Does the Abatron BDI2000 Support?
- How To Use the Abatron BDI2000 On:
 1. MPC8260

► Introduction

- What do you do when you have a problem to debug and all the board will do is crash ??
- What do you do when you have newly designed board you want to bring-up and you don't know whether the RAM is working ??
- What do you do when you want to debug a kernel module, and you don't have kgdb support on your platform ???

Answer: Use a hardware assisted debugging method

Many CPU architectures have some kind of low-level debug port. Mostly they are some variant of a JTAG based interface, which you can think of as a special kind of serial port that will let you get direct access to the internal registers of the CPU, without requiring any functional software to be available on the CPU. This is not a standard serial port, and requires "special" hardware to interface to it. The interface both from an electrical point of view and a protocol point of view is different between CPUs.

There are many vendors of interface devices who produce products compatible with Linux. As with most things, you get what you pay for. The Abatron BDI2000 provides the best performance to cost ratio of any other JTAG interface on the market and unlike most others, it features MMU translation.

In the past, Linux hosted debug devices have not been well supported. Abatron seems to have been the first company that actively supported Linux hosting. However, recently, MacGraigor has made some drivers available (see <http://www.ocdemon.com/>) Which work with their RAVEN devices (supported for Flash initialisation in the MPC860FADS and MPC8260ADS BSPs).

► Types Of Interface Device

One of the recurring question people have is what changes are needed on the host to support one of the hardware assisted devices. The answer is that this depends on the type of device. Broadly speaking, there are 2 types of interface device:

- A device that is connected between your host's parallel port and the target
- A device that is connected between the network and the target.

In the case of a parallel based device, you **must** have a modified gdb and usually also a kernel device driver that "knows" how to "talk" to the interface device. This can be problematic on Linux, as often Linux based device drivers are not available for the interface unit.

In the case of the network based device, all you need is a host to cross target the gdb debugger. This is always supplied with Lineo's SDK (both in the toolchain area, and often in source form in the BSP area).

NOTE: You can use a graphical wrapper like Data Display Debugger (DDD) as your debug interface, this is purely a clever cosmetic layer around gdb. In addition, companies such as Viosoft and Ada Core Technology, provide robust graphical user interfaces that are worth a look.

► Recommended Units

The remainder of this document will be focused on the Abatron BDI2000. Detailed instructions for specific boards will be provided on additional pages linked at the end of this document.

NOTE: Always refer to the Abatron BDI2000 documentation, as that will cover critical material which we don't want to duplicate here.

► What Targets Does the Abatron BDI2000 Support?

Currently the BDI2000 supports the following platforms:

- CPU32/32+
- ColdFire
- MPC5xx/8xx
- PPC4xx
- PPC6xx/7xx/82xx
- ARM7/9TDMI (MMU-less and MMU-full support)
- MIPS32/64
- X-Scale PXA, IOP, IXA, 80200
- TI OMAP1510

NOTE: Please check with <http://www.abatron.ch/> for an up-to-date list.

▶ How To Use The Abatron BDI2000 On:

MPC8260

[MPC8260ADS Hardware Assisted Debugging](#)

Mini How-to For Abatron BDI 2000 and Embedix SDK Support on the MPC8260ADS Development Platform

1. Introduction
2. Installation on a Linux host
3. Getting set up for the first time
 1. Connecting up the cables
 2. Installing the software utilities (root access required)
 3. Checking the connection and configuring the BDI2000
 4. Connecting to the BDI2000
4. Using the BDI2000 to symbolically debug the Linux kernel
 1. Method 1: Downloading a kernel, booting with the Flash filesystem
 1. Setting up the correct Abatron config file
5. Using the BDI2000 to symbolically debug kernel loadable modules

▶ Introduction

If you haven't already read it, please refer to [Hardware Assisted Debug](#)

This document describes the steps necessary to provide Abatron BDI2000 hardware assist debug capability, via Lineo's Embedix SDK, for MPC8260ADS development targets.

▶ Installation on a Linux host

Before you start, read the manual that comes with the BDI2000. It is a much more comprehensive reference than this guide.

▶ Getting set up for the first time

Connecting up the cables

- Turn off the power on your target board
- Connect the 16 pin ribbon cable between "Target B" on the BDI2000 and "P5" on the mpc8260ads target board.
- Connect the supplied serial cable between the host and the RS232 connector on the BDI2000 (make sure you know which tty device this is)
- Connect a network cable between the BDI2000 and the network
- Connect the BDI2000 power supply to the BDI2000 to power it up
- Power on your target board

► Installing the software utilities (root access required)

- Create a directory to hold the software and copy it from the floppy:

```
mkdir -p /opt/bdi2000/ppc6xx_7xx_82xx_74xx
mount /mnt/floppy
cp -a /mnt/floppy/* /opt/bdi2000/ppc6xx_7xx_82xx_74xx
```

- Build the configuration tool (you need a compiler and linker)

```
cd /opt/bdi2000/ppc6xx_7xx_82xx_74xx
mkdir bdisetup_src
cd bdisetup_src
unzip ../bdisetup.zip
make
cp bdisetup ..
cd ..
```

► Checking the connection and configuring the BDI2000

- Check the serial connection to the BDI2000 by running (here's ours):

```
./bdisetup -v
BDI Type : BDI2000 Rev.C (SN: 93197120)
Loader   : V1.05
Firmware : V1.09 bdiGDB for PPC6xx/PPC7xx
Logic    : V1.02 PPC6xx/PPC7xx
MAC      : 00-40-49-fa-19-71
IP Addr  : 192.168.0.11
Subnet   : 255.255.255.255
Gateway  : 255.255.255.255
Host IP  : 192.168.0.6
Config   : vads8260.cnf
```

Note: that this assumes port ttyS0, if you are using another, you'll have to pass the -p<port_device> to the bdisetup program, e.g -p /dev/ttyS1 for com2

- Load(or update) the BDI firmware

```
./bdisetup -u -aGDB -tPPC700
```

- Copy the default configuration file (or a custom one) to /tftpboot

```
cp vads8260.cnf /tftpboot
```

- Allocate a fixed IP address for your BDI2000
- Configure your BDI2000 with the chosen parameters

```
./bdisetup -c -i<bdi_ipaddr> -h<host_ipaddr> -fvads8260.cnf
```

- Check and exit configuration mode. When in config mode, the red mode LED on the BDI2000 flashes, after this command it should stop flashing.

```
./bdisetup -v -s
```

▶ Connecting to the BDI2000

Now that everything has been set-up, you should be able to telnet into the BDI2000

```
telnet <bdi_ipaddr>
```

This does not require any passwords. After the initial help summary, you'll see:

```
- TARGET: waiting for target Vcc
```

When you power up your board, you'll see this is detected by the BDI2000 in the telnet session window. The BDI2000 will then run the sections in your config file (usually it sets up some registers and then downloads a file for debugging).

▶ Using the BDI2000 to symbolically debug the Linux kernel

Here's an example of how we have used the BDI2000. This does not mean there are not other valid methods. One method I intend to try is to attach to a running kernel (without the need to download), this is supported in the latest versions of the BDI2000 firmware.

NOTE: This assumes you have already built an SDK-2.0.1 mpc8260ads project.

- ▶ Method 1: Downloading a kernel, booting with the Flash filesystem

Setting up the correct Abatron config file

- Download and copy this [vads8260.cnf](#) file to your /ftpboot directory.
- Modify the target's Linux kernel
 - edit <project>/src/linux/arch/ppc/kernel/m8260_setup.c, change (line 269) from:

```
        strcpy(cmd_line, (char *) (r6+KERNELBASE));  
    }
```

to:

```
        strcpy(cmd_line, (char *) (r6+KERNELBASE));  
    } else {  
        strcpy(cmd_line, "root=1f00 init=/rcinit");  
    }
```

- Edit <project>/src/linux/Makefile, change (line 41): from:

```
CFLAGS_KERNEL =
```

to:

```
CFLAGS_KERNEL = -g
```

- Modify the kernel lbc file

```
cp <project>/config-data/buildcontrol/board/kernel.lbc  
<project>/config-data/buildcontrol/local
```

- Edit the copied lbc file, and make the following changes (to build a vimage instead of a bzImage)

```
@@ -22,7 +22,8 @@  
    arch/ppc/coffboot/vmlinux.gz
```

```
%bld_targ
-zImage
+vimage
```

- Rebuild the kernel using twcl (or tw) and copy to the download area

```
twcl --pkg kernel
cp <project>/src/linux/vimage /tftpboot
```

- Boot the system under test

telnet to the BDI2000 and then type:

```
MPC8260>reset
- TARGET: processing user reset request
- TARGET: Target PVR is 0x00810101
- TARGET: resetting target passed
- TARGET: processing target init list ....
- TARGET: processing target init list passed
- TARGET: loading program file (BIN) ....
- TARGET: loading program file passed
MPC8260>go
```

Connect to the target using minicom in another xterm (115200 N81) and observe the board boots normally.

- Setup the BDI2000 MMU mapping

On the target look in <project>/src/linux/System.map and locate the symbol swapper_pg_dir and record the address (mine looks like: c0119000 D swapper_pg_dir).

In the BDI2000 telnet session type (substitute for the address you found):

```
MPC8260>halt
Target CPU      : MPC8260/MPC8240 Rev.1.x
Target state    : debug mode
Debug entry cause : COP halt
Current PC      : 0xc0005640
Current CR      : 0x24444028
Current MSR     : 0x00009032
Current LR      : 0xc0005654
MPC8260>mm 0xf0 0xc0119000
MPC8260>go
```

- Start a debugger session and connect to the BDI2000/target.

On my system I have placed an entry for the BDI2000's IP address in my /etc/hosts file so I can refer to it as bdi2000. To start a ddd session I do:

```
ddd --debugger /home/seh/project/emb-bin/gdb
/home/seh/project/8260/src/linux/vmlinux
```

In the command line window (bottom window) within ddd, type:

```
(gdb) target remote bdi2000:2001
Remote debugging using bdi2000:2001
0xc0005644 in idled () at idle.c:75
```

- Set a test breakpoint and continue

To check that everything is working, you can try setting a breakpoint. For example, in the ddd command line window type:

```
(gdb) b schedule
Breakpoint 1 at 0xc000e5d4: file sched.c, line 518.
(gdb) c
Continuing.
```

```
Breakpoint 1, schedule () at sched.c:518
(gdb)
```

You may now set other breakpoints and debug your Linux kernel in the same way as you would for a user space process.

- Clearing breakpoints and resuming

To get ready for the next section (kernel module debugging), we'll delete all breakpoints and continue. In the ddd command line window type:

```
(gdb) d
Delete all breakpoints? (y or n) y
(gdb) c
Continuing.
```

Note: If you want to quit debugging and leave the system running, you can use 'File/Exit' in the ddd menu. If you use detach, the target will be reset by the BDI2000 and the test file re-loaded.

► Using the BDI2000 to symbolically debug kernel loadable modules

- First, follow the sets above in 'Using the BDI2000 to symbolically debug the Linux kernel'
- Build the kernel module of interest with debugging enabled (-g)
- Deploy the module to the target

I find it easiest to mount the whole <project>/src area on the target. To do this you need to export the project source area. For example, I added the following entry to my /etc/exports file:

```
/home/seh/project/8260/src (rw,insecure,no_root_squash,no_all_squash)
After adding an entry to /etc/exports, you need to run exportfs -r as root, or re-start the NFS server.
```

Once exported, on the target you need to mount the exported source area. The mpc8260ads BSP allows this by typing:

```
mount /mnt/src
as it sets up entries in /etc/hosts and /etc/fstab on the target. The long form of this command would be:
mount <hostip>:/home/seh/project/src /mnt/src
```

- Load the module under test and record the section addresses

In this example, I'm going to be debugging the vxlinux.o module, here's what I do on the target after loading any dependent modules:

```
# cd /mnt/src/embedix_mdk_src-1.0/mdk4vxworks
# insmod vxlinux.o
# cat /proc/ksyms | grep insmod_vxlinux
c2098d00 __insmod_vxlinux_S.text_L73876 [vxlinux]
```

```
c20aad94 __insmod_vxlinux_S.data_L22304 [vxlinux]
c2095000 __insmod_vxlinux_O/mnt/src/embedix_mdk_src-
1.0/mdk4vxworks/vxlinux.o_M]
c20bd318 __insmod_vxlinux_S.bss_L768 [vxlinux]
c2095060 __insmod_vxlinux_S.rodata_L15520 [vxlinux]
```

When you look at a modules symbols, you'll always find a .text entry which you should note. In addition, you may also see a .bss and a .data section, if these are present, make a note of them.

- Add the symbol file in the ddd session

In the ddd session you set up for kernel debugging, type <Ctrl>+c and then use the add-symbol-file command to setup the symbols for your module. In the example presented here, this is what I'd type:

```
^C
Program received signal SIGSTOP, Stopped (signal).
idled () at idle.c:75
(gdb) add-sym /home/seh/project/8260/src/embedix_mdk_src-
1.0/mdk4vxworks/vxlinux.o
0xc2098d00 -s .data 0xc20aad94 -s .bss 0xc20bd318
add symbol table from file
"/home/seh/project/8260/src/embedix_mdk_src-1.0/mdk4vxworks/vxlinux.o"
at
    .text_addr = 0xc2098d00
    .data_addr = 0xc20aad94
    .bss_addr = 0xc20bd318
(y or n) y
Reading symbols from
/home/seh/project/8260/src/embedix_mdk_src-
1.0/mdk4vxworks/vxlinux.o...done.
(gdb)
```

You may now set breakpoints, step, next and continue as normal.

NOTE: Currently it is not possible to set a breakpoint at the first instruction of the module (init_module), as the module must be inserted to get the symbol table addresses needed by gdb. This means that if your module immediately crashes, you have to put in some code to 'hold it off' running until you set a flag in the debugger.

This document was published with the combined efforts of Lineo and the support team at Ultimate Solutions, Inc. For additional information, please contact Ultimate Solutions.



Ultimate Solutions, Inc.

10 Clever Lane

Tewksbury, MA 01876

Toll Free: 866.455.3383 Phone: 978.455.3383

Fax: 978.926.3091 Email: info@ultsol.com

Web: <http://www.ultsol.com>