*HUNT ENGINEERING*
**Chestnut Court, Burton Row,**
**Brent Knoll, Somerset, TA9 4BP, UK**
**Tel: (+44) (0)1278 760188,**
**Fax: (+44) (0)1278 760199,**
**Email: sales@hunteng.co.uk**
**http://www.hunteng.co.uk**
**http://www.hunt-dsp.com**

# *HUNT ENGINEERING*

# *API for LINUX*

# *Installation and User Manual*

***Document Rev C***
***API software Rev 1.9.10***
***P.Warnes / J.Thie 07-12-05***

## COPYRIGHT



## WARRANTIES LIABILITY and INDEMNITIES

HUNT ENGINEERING warrants the hardware to be free from defects in the material and workmanship for 12 months from the date of purchase. Product returned under the terms of the warranty must be returned carriage paid to the main offices of HUNT ENGINEERING situated at BRENT KNOLL Somerset UK, the product will be repaired or replaced at the discretion of HUNT ENGINEERING.

Exclusions - If HUNT ENGINEERING decides that there is any evidence of electrical or mechanical abuse to the hardware, then the customer shall have no recourse to HUNT ENGINEERING or its agents. In such circumstances HUNT ENGINEERING may at its discretion offer to repair the hardware and charge for that repair.

Limitations of Liability - HUNT ENGINEERING makes no warranty as to the fitness of the product for any particular purpose. In no event shall HUNT ENGINEERING'S liability related to the product exceed the purchase fee actually paid by you for the product. Neither HUNT ENGINEERING nor its suppliers shall in any event be liable for any indirect, consequential or financial damages caused by the delivery, use or performance of this product.

Because some states do not allow the exclusion or limitation of incidental or consequential damages or limitation on how long an implied warranty lasts, the above limitations may not apply to you.

## TECHNICAL SUPPORT

Technical support for HUNT ENGINEERING products should first be obtained from the comprehensive Support section http://www.hunteng.co.uk/support/index.htm on the HUNT ENGINEERING web site. This includes FAQs, latest product, software and documentation updates etc. Or contact your local supplier - if you are unsure of details please refer to http://www.hunteng.co.uk for the list of current re-sellers.

HUNT ENGINEERING technical support can be contacted by emailing support@hunteng.demon.co.uk, calling the direct support telephone number +44 (0)1278 760775, or by calling the general number +44 (0)1278 760188 and choosing the technical support option.

## Revision History

**API 1.9.10**

Support for HERON-BASE2 added.

**API 1.9.8**

Rename of "Parent" #define to "WDMParent" in heapi.h.

**API 1.9.7**

HeGetDeviceInfo function added.
Support for RedHat 9.0 added.

**API 1.9.5**

Support for HEPC9 added.
HEPC8 driver updated.

**API 1.6.4**

Support for RedHat 7.1 added.

**API 1.5.0**

First API for Linux released.
Tried & tested on RedHat 6.1.

# TABLE OF CONTENTS

The current version of the HUNT ENGINEERING API supports RedHat versions 6.1, 7.1 and 9.0, and Fedora Core 2.0. The table below shows what RedHat support has been added to what HUNT ENGINEERING API version.

**API 1.9.7** RedHat 9.0.

**API 1.6.4** RedHat 7.1

**API 1.5.0** RedHat 6.1.

Insert the Hunt Engineering CD. In the `\software\api\linux` directory there is a tar file with the name "`apixxx.tar`" (where 'xxx' is the latest version; at the moment of writing this is '195', so you would untar "`api195.tar`") This contains the drivers (modules), API interface shared library, include files and confidence test programs (which also serve as examples). Copy the tar file into a directory where you wish to have the API examples installed. Then un-tar the file in the chosen directory as follows:

```
tar xvf apixxx.tar
```

In the un-tarred distribution you will find an install script ("`installme`"). You have to be the root user in order to run the script. The script will attempt to do the following:

- Copy the include files (`heapi.h`, `hecodes.h`, `heioctl.h`) to `/usr/local/include`.
- Copy the API library (`libhel.so`) to `/usr/local/lib`.
- Copy 4 utilities (`apireset, apijtagreset, heartconf, and hrn_fpga`) to `/usr/local/bin`.
- "Make" the device driver.
- Copy the device driver (`hep9a.o or hep8a.o or hep3b.o or hep2e.o`) to `/lib/modules/*/misc`
- Make device nodes.
- Start the device driver (using `insmod`).

LINUX modules have to be started each time the machine is rebooted. There are 3 ways of doing this, described in the following sections.

### HEPC9 Manual Insertion and Removal of a module

To insert a module by hand, just type:

```
insmod hep9a hep9a_major=63
```

You have to be the root user to do this. To check the result of `insmod`, view the last few lines as reported by `dmesg`. A typical entry for the HEPC9 would be:

```
(hep9a, MM enabled): PCI_LATENCY_TIMER is 32
Hunt Engineering hep9a0 Rev 0
Board ID 0, IRQ=9, PCI address 0xdffff000, mapped to 0xc4855000
hep9a: devfs: character device has major number 63
```

If you have more than 1 HEPC9 installed, you'll see multiple "Board ID" lines. The Board ID corresponds with the red board switch on your HEPC9. This switch can be selected from 0 to 15. The IRQ number shows what IRQ the board will use. The HEPC9 uses a memory mapped interface; the physical address is shown as well as the memory mapped address. All accesses by the module proceed via this mapped address (virtual address).

The character device major number is used when setting up the device files. This device major number is assigned automatically if you haven't passed the hep9a_major argument.

To pass your own major number to the module, use the following (e.g. for a HEPC9):

```
insmod hep9a hep9a_major=63
```

In the case of the "installme" script, 63 is used for the HEPC9. For the API itself it doesn't matter what number you use, but you must ensure that the character device major number is the same as used when creating the device files (`/dev/hep9a`).

To remove a module from the system, type:

```
rmmod hep9a
```

The advantage of this method is that it allows finest control over module options. The disadvantage is that you need to be a root user. A better method might be "automagic" insertion, which is described in the section after the next.

### HEPC8 Manual Insertion and Removal of a module

To insert a module by hand, just type:

```
insmod hep8a hep8a_major=62
```

You have to be the root user to do this. To check the result of `insmod`, view the last few lines as reported by `dmesg`. A typical entry for the HEPC8 would be:

```
(hep8a, MM enabled): PCI_LATENCY_TIMER is 0
Hunt Engineering hep8a0 Rev 0
Board ID 0, IRQ=10, PCI address 0xf0000000, mapped to 0xc2823000
hep8a: character device has major number 62
```

If you have more than 1 HEPC8 installed, you'll see multiple "Board ID" lines. The Board ID corresponds with the red board switch on your HEPC8. This switch can be selected from 0 to 15. The IRQ number shows what IRQ the board will use. The HEPC8 uses a memory mapped interface; the physical address is shown as well as the memory mapped address. All accesses by the module proceed via this mapped address (virtual address).

The character device major number is used when setting up the device files. This device major number is assigned automatically if you haven't passed the hep8a_major argument. To pass your own major number to the module, use the following (e.g. for a HEPC8):

```
insmod hep8a hep8a_major=62
```

In the case of the "installme" script, 62 is used for the HEPC8. For the API itself it doesn't matter what number you use, but you must ensure that the character device major number is the same as used when creating the device files (`/dev/hep8a`).

To remove a module from the system, type:

```
rmmod hep8a
```

The advantage of this method is that it allows finest control over module options. The disadvantage is that you need to be a root user. A better method might be "automagic" insertion, which is described in the next section.

### Automagic Insertion of a module

The second method of using the HUNT API drivers and library is as follows. The kernel runs a special process, the kernel daemon, which can perform the actions described in the previous section "automagically". Each time a process wants to access a device file, the kernel checks if it can support the device with the corresponding device major number. If not, it asks the kernel daemon to find and insert the driver. A short while after the last

process that used the driver has terminated, the kernel daemon will remove the module. Everything is transparent to the user.

The kernel daemon needs the following to do its job:

- the module file (e.g. `hep9a.o` or `hep8a.o`) must be in directory `/lib/modules/<version>/misc`
- the file `etc/conf.modules` or `etc/modules.conf` must contain information which links the the devices's major number to the module file, and you must specify any options that should be passed to the module. Assuming you use the device major number 63, the entries in the file should look like this:

```
alias char-major-63 hep9a
options hep9a hep9a_major=63
```

To check the success of the kernel daemon, use `dmesg` or have a look in the `/proc` directory (e.g. in the case of a HEPC9, look at the `/proc/hep9a` text file).

**Compilation into the kernel**

The third method of using the HUNT API is by compiling the API drivers and library into your LINUX kernel. The drivers will always be started up, as long as the (Hunt Engineering) board is installed in the PC. To compile the HUNT API drivers and library into the LINUX kernel, please refer to the documentation that came with your LINUX distribution. Or, if you started "from scratch", on the Internet there are lots of sources that tell you how to include modules into your kernel for your particular configuration.

**Creating Device Files**

The Linux kernel identifies drivers by the device major numbers of the device files. The minor numbers are used to identify the ports on each board. To create the device files, enter as root the following:

```
mknod /dev/<name> c <major number> <minor number>
```

You are not free to choose any name. In fact, the "`installme`" script has already initialized a number of device files for you. The name is built up as follows:

HEPC9: `hep9axy`

Where `x` is the hexadecimal board number (0…f) and `y` the device id (0 = fifo A, 1 = fifo B, 2 = JTAG, 3 = HSB (serial bus), 6 = fifo C, 7 = fifo D, 8 = fifo E, 9 = fifo F).

HEPC8: `hep8axy`

Where `x` is the hexadecimal board number (0…f) and `y` the device id (0 = fifo A, 2 = JTAG, 3 = HSB (serial bus)).

You are also not free to choose the major number. You have to use the number as shown by running `dmesg`, in other words, you must use the device major number reported by the kernel as it installed the module. The default major number is 63 for the HEPC9 driver and is 62 for the HEPC8 driver.

You are also not free to choose the minor number. The minor number is calculated in the following way: board number times 16 + device id. For example, FIFO A of a HEPC9 number 0 has minor number 0, and the jtag of this board has minor number 2. For HEPC9 number 1, the minor number of FIFO A is 16, and the minor number of the jtag is 18. For your reference, have a look at the "installme" script, a number of device files are initialized here (for board switches 0 to 2).

Check the access flags of the device files. If you want to grant access to them for other users, you usually will have to change them using `chmod`, or use the "--mode" option of the `mknod` command (this is used in the "installme" script).

| *Board Number* | *Device* | *Minor Number* | *Device File Name* |
|---|---|---|---|
| **0** | Fifo A | 0 | hep9a00 |
| | Fifo B | 1 | hep9a01 |
| | JTAG | 2 | hep9a02 |
| | HSB | 3 | hep9a03 |
| | Fifo C | 6 | hep9a06 |
| **1** | Fifo A | 16 | hep9a10 |
| | Fifo B | 17 | hep9a11 |
| | JTAG | 18 | hep9a12 |
| | HSB | 19 | hep9a13 |
| | Fifo C | 22 | hep9a16 |
| **x** | Fifo A | 16*x | hep9ax0 |
| | Fifo B | 16*x+1 | hep9ax1 |
| | JTAG | 16*x+2 | hep9ax2 |
| | HSB | 16*x+3 | hep9ax3 |
| | Fifo C | 16*x+6 | hep9ax6 |

Device File Names for an HEPC9 board.

| *Board Number* | *Device* | *Minor Number* | *Device File Name* |
|---|---|---|---|
| **0** | Fifo A | 0 | hep8a00 |
| | JTAG | 2 | hep8a02 |
| | HSB | 3 | hep8a03 |
| **1** | Fifo A | 16 | hep8a10 |
| | JTAG | 18 | hep8a12 |
| | HSB | 19 | hep8a13 |
| **x** | Fifo A | 16*x | hep8ax0 |
| | JTAG | 16*x+2 | hep8ax2 |
| | HSB | 16*x+3 | hep8ax3 |

Device File Names for an HEPC8 board.

**Reading Status Information**

If you have support for the `proc` files system compiled into your kernel, you can get some

more specific information about the status of your driver and board by typing:

```
cat /proc/hep9a        (for HEPC9 boards)
```

A typical entry will be as follows, for a HEPC9 board:

```
Hunt Engineering HEPC9 HERON motherboard
1 board(s), device major number is 63
hep9a0:  Rev.  0,  IRQ=9,  PhysAdr  0xdffff000,  VirtAdr
0xc4855000
```

Or by typing:

```
cat /proc/hep8a        (for HEPC8 boards)
```

A typical entry will be as follows, for a HEPC8 board:

```
Hunt Engineering HEPC8 HERON motherboard
1 board(s), device major number is 62
hep8a0:  Rev.  0,  IRQ=10,  PhysAdr  0xf0000000,  VirtAdr
0xc2823000
```

**Insmod parameters**

| HEPC9 | What does it do? |
|---|---|
| `hep9a_major` | Sets character device major no. |
| `ticks (16)` | Wait period when ints disabled. |
| `int0 (1)` | Enable/disable board 0 intrpt |
| `int1 (1)` | Enable/disable board 1 intrpt |
| `int2 (1)` | Enable/disable board 2 intrpt |
| `nmm0 (1)` | Disable/enable brd 0 mastermode |
| `nmm1 (1)` | Disable/enable brd 1 mastermode |
| `nmm2 (1)` | Disable/enable brd 2 mastermode |

Note: MasterMode: enabled by nmm0=0, disabled by nmm0=1 (nmm=no master mode).

| HEPC8 | What does it do? |
|---|---|
| `hep8a_major` | Sets character device major no. |
| `ticks (16)` | Wait period when ints disabled. |
| `int0 (1)` | Enable/disable board 0 intrpt |
| `int1 (1)` | Enable/disable board 1 intrpt |
| `int2 (1)` | Enable/disable board 2 intrpt |
| `nmm0 (1)` | Disable/enable brd 0 mastermode |
| `nmm1 (1)` | Disable/enable brd 1 mastermode |
| `nmm2 (1)` | Disable/enable brd 2 mastermode |

Note: MasterMode: enabled by nmm0=0, disabled by nmm0=1 (nmm=no master mode).

**Support for boards with a boardswitch other than 0, 1 or 2**

The 'installme' script installs the drivers (HEPC9 or HEPC8) allowing for 3 possible board switch settings: 0, 1 or 2. If you want to use other board settings, add device nodes for those board numbers (use and change the 'installme' script or read the device node section above, then use 'mknod'). You can also add insmod parameters by making slight changes to the device driver code in module/hepc9 or module/hepc8.

## Confidence testing for Linux

In your un-tarred HUNT ENGINEERING API directory you will find confidence tests in `etc/c6x/examples`, where there are 3 sub-directories: `testint`, `reads` and `writes`. Change directory to `testint/linux`, compile and execute `testint`, for example for an HEPC9:

```
cd etc/c6x/examples/testint/linux
make
testint hep9a 0
```

The `testint` program should respond:

```
Interrupts work fine
```

If you disabled interrupts, this is reported and you do not need to worry:

```
Interrupts disabled.
```

If `testint` reports anything else, there's a problem with interrupts. Please refer to the Troubleshooting section in this manual for advice on how to solve such problems.

To execute the next confidence test, change directory to the `writes` directory, and execute `writes`. For example, for a HEPC9:

```
cd ../../writes
make
writes hep9a 0 a 10000 10000 100000
```

A system that is working properly returns something similar to:

```
Start at 10000, inc 10000, end at 100000, BlockSize=250 on hep9a (0: Comporta)
Resetting...
Serial bus: slot 1: HERON4-C6701, rom version 5.
Resetting...
Booting /home/johan/api/etc/c6x/examples/writes/bios/writes4.out...
Testing...
Writes Transfer size 1000 DWORDS in 1 ticks, Speed: 390.63 KBytes/sec
Writes Transfer size 2000 DWORDS in 1 ticks, Speed: 781.25 KBytes/sec
Writes Transfer size 3000 DWORDS in 1 ticks, Speed: 1171.88 KBytes/sec
Writes Transfer size 4000 DWORDS in 1 ticks, Speed: 1562.50 KBytes/sec
Writes Transfer size 5000 DWORDS in 1 ticks, Speed: 1953.14 KBytes/sec
Writes Transfer size 6000 DWORDS in 1 ticks, Speed: 2343.75 KBytes/sec
Writes Transfer size 7000 DWORDS in 1 ticks, Speed: 2734.38 KBytes/sec
Writes Transfer size 8000 DWORDS in 1 ticks, Speed: 3125.00 KBytes/sec
Writes Transfer size 9000 DWORDS in 1 ticks, Speed: 3515.63 KBytes/sec
Writes Transfer size 10000 DWORDS in 1 ticks, Speed: 3906.30 KBytes/sec
Check whether any interrupts were used: read 0, write 0, master mode 1.
```

Please ignore the reports on Speed; the (standard) confidence test the time() function which ticks every second on my test machine. Use larger blocks to get tick counts above 1 to get a bit more meaningful speed results. If any problems appear, make sure that the module is started (e.g. you must have executed `insmod hep9a hep9a_major=63` as root, or see the "Troubleshooting" section).

To execute the final confidence test, change directory to the `reads` directory, and execute `reads`. For example, for a HEPC9:

```
cd ../../reads
make
reads hep9a 0 a 10000 10000 100000
```

A system that is working properly returns something similar to:

```
Start at 10000, inc 10000, end at 100000, BlockSize=250 on hep9a (0: Comporta)
Resetting...
Serial bus: slot 1: HERON4-C6701, rom version 5.
Resetting...
Booting /home/johan/api/etc/c6x/examples/reads/bios/reads4.out...
Testing...
Reads Transfer size 1000 DWORDS in 1 ticks, Speed: 390.63 KBytes/sec
Reads Transfer size 2000 DWORDS in 1 ticks, Speed: 781.25 KBytes/sec
Reads Transfer size 3000 DWORDS in 1 ticks, Speed: 1171.88 KBytes/sec
Reads Transfer size 4000 DWORDS in 1 ticks, Speed: 1562.50 KBytes/sec
Reads Transfer size 5000 DWORDS in 1 ticks, Speed: 1953.13 KBytes/sec
Reads Transfer size 6000 DWORDS in 1 ticks, Speed: 2343.75 KBytes/sec
Reads Transfer size 7000 DWORDS in 1 ticks, Speed: 2734.38 KBytes/sec
Reads Transfer size 8000 DWORDS in 1 ticks, Speed: 3125.00 KBytes/sec
Reads Transfer size 9000 DWORDS in 1 ticks, Speed: 3515.63 KBytes/sec
Reads Transfer size 10000 DWORDS in 1 ticks, Speed: 3906.25 KBytes/sec
Check whether any interrupts were used: read 0, write 0, master mode 1.
```

Please ignore the reports on Speed; the (standard) confidence test the time() function which ticks every second on my test machine. Use larger blocks to get tick counts above 1 to get a bit more meaningful speed results. If any problems appear, make sure that the module is started (e.g. do an `insmod hep9a hep9a_major=62` as root, or see the "Troubleshooting" section).

Once all three confidence tests run successfully, the HUNT ENGINEERING API has been installed successfully. You can now proceed to develop your application (perhaps with the confidence tests here as an example) or proceed to install the Server/Loader.

If you experienced problems, please refer to the Troubleshooting section below for advice on how to correct problems.

There are two common things that one easily forgets when using modules. First, to have started up the module (using `insmod`), the other is to have the device files use a different character major number than the modules. First, let's have a look at module installation.

To check whether the module was installed properly, the first thing you need to do is to use `dmesg` to view if the driver has reported anything, and if so, if there are error messages or that installation was fine. If it was installed successfully, you will see a message somewhere that describes the board and the resources it consumes.

e.g. for a HEPC9 board you'll see with `dmesg`:

```
(hep9a, MM enabled): PCI_LATENCY_TIMER is 32
Hunt Engineering hep9a0 Rev 0
Board ID 0, IRQ=9, PCI address 0xdffff000, mapped to
0xc4855000
hep9a: character device has major number 63
```

or for a HEPC8 board:

```
(hep8a, MM enabled): PCI_LATENCY_TIMER is 0
Hunt Engineering hep8a0 Rev 0
Board ID 0, IRQ=10, PCI address 0xf0000000, mapped to
0xc2823000
hep8a: character device has major number 62
```

If the driver wasn't installed successfully, you will see an error message. An example of an error message is lack of kernel memory (part of the driver initialisation allocates kernel memory). Possible messages are. for example:

```
hep9a0: no HEPC9 found
hep9a0: can't get major number 63
hep9a0: could not get irq line 9
hep9a0: can't allocate memory
hep9a0: can't map memory at 0xdffff000
```

If the driver wasn't installed at all, no message can be seen. In that case, run `insmod` to install the driver, and then use `dmesg` to verify the result. Use `insmod` as follows:

```
insmod hep9a hep9a_major=63
```

for an HEPC9, or for an HEPC8 use:

```
insmod hep8a hep8a_major=62
```

Please note that these device major numbers may already be in use on your system. In that case use a different number – or none at all. (If you select no device major number with `insmod`, the system will assign a device major number to the module.) Be sure to also verify and/or alter your device files, as described below.

The second thing you need to check is that the device files have been created. First, verify that you have device files: `/dev/hep9a*` for HEPC9 and `/dev/hep8a*` for HEPC8. In case you find no such files, please create them, following this procedure.

First, using `dmesg`, find out what device major number is used by the driver. E.g. lines like:

```
hep9a: character device has major number 63
hep8a: character device has major number 62
```

tells you that the HEPC9 driver uses device major number 63 and the HEPC8 driver uses device major number 62. Using this number, now create device files, using `mknod`, as follows:

```
mknod --mod=666 hep9a00 c 63 0
mknod --mod=666 hep9a02 c 63 2
```

for an HEPC9, or for an HEPC8 use:

```
mknod --mod=666 hep9a00 c 62 0
mknod --mod=666 hep9a02 c 62 2
```

Please make sure that you replace 63 and 62 by the device major number that is reported for your board by `dmesg`.

If the device files were already there, please verify that the device files present do indeed use the correct device major number. One way to do this is to delete the entries currently there (`hep9a*,hep8a*`), and then re-create then, as outlined above.

If `dmesg` reports a `can't get major number` error, then you need to uninstall the API, then re-install with a different character device major number until you find a number that works. To uninstall, run the "`uninstall`" script. To re-install, edit the "`installme`" script and change the character device major number of the board you try to install.

If `dmesg` reports a `can't allocate memory` there's not much we can do. You have to ensure there's enough kernel memory, so try to free up any, if you can, then try to install again (run `insmod` as described above).

Other possible causes for the confidence checks are the boot jumpers on the HERON module and the board switch. Verify the following:

- There must be a HERON processor module in slot 1 of the board.
- The jumpers of the HERON module in slot 1 must be set to FIFO 0 (HEPC8 only).
- Check that the board switch is set to zero.

Once you have checked these basic issues, but the confidence checks still fail, it's time to have a look at interrupts and board addresses. First let's look at interrupts.

If, when running `dmesg`, you found error messages such as: `could not get irq line 9`, `IRQ not set for this board`, or `IRQ x not supported. Interrupts disabled`, then there certainly is an IRQ problem. But there may still be an IRQ problem even if `dmesg` reports no errors whatsoever. To check if IRQs are the problem, install the module with interrupts disabled:

```
rmmod  hep9a
insmod hep9a hep9a_major=63 int0=0
```

for an HEPC9, or for an HEPC8:

```
rmmod  hep8a
insmod hep8a hep8a_major=62 int0=0
```

Now run the Confidence Tests again. If this time they run fine, obviously there is a problem with interrupts. Reasons for IRQ problems are: all interrupts are in use, there's none free to be used by the Hunt Engineering board, or one interrupt is shared between the Hunt Engineering board and other boards, and the drivers don't work together properly. Your choice is: remove the other board, and use the Hunt Engineering board with interrupts enabled. Or you can have the other board also installed, but use the Hunt Engineering board with interrupts disabled. There's no problem disabling interrupts; but the driver will

be somewhat slower and it consumes more processing power (as it has to do polling at regular intervals).

Finally, in some PCs the CMOS allows you to allocate certain interrupts to either PCI or ISA only. It might be that your interrupt problem was caused by the interrupt being not available to the PCI bus. The CMOS configuration program can usually be started by pressing DEL, F2, or some such key immediately after reset. Typically there will be a short message on the screen telling you when the DEL key is "active".

If you have a "PnP OS" or "Windows 9x support" setting in your CMOS, make sure that this field is set to off or disabled.

Once you verified that IRQs are not the problem (e.g. the confidence check also fails when interrupts are disabled), have a look at the board's address. Addressing problems may also have been indicated by `dmesg` error messages such as `can't map memory at 0xf0000000` or `can't get io at 0x200`. Most likely other boards that have already reserved that space for themselves cause them.

On some motherboards, resources are allocated per PCI slot. Try a different PCI slot, and things may now work fine.

Leave the HERON-BASE2 disconnected for now, first install the software. Insert the Hunt Engineering CD. In the `\software\api\linux` directory there is a gzipped tar file with the name "`apixxx.tar.gz`" (where 'xxx' is the latest version; at the moment of writing this is '1910', so you would unzip and then untar "`api1910.tar.gz`"). This contains the drivers (modules), API interface shared library, include files and confidence test programs (which also serve as examples). Copy the tar file into a directory where you wish to have the API installed. Then un-tar the file in the chosen directory as follows:

```
tar xvf apixxx.tar
```

In the un-tarred distribution you will find an install script ("`installme`"). You have to be the root user in order to run the script. The script will attempt to do the following:

- Copy the include files (`heapi.h`, `hecodes.h`, `heioctl.h`) to `/usr/local/include`.
- Copy the API library (`libhel.so`) to `/usr/local/lib`.
- Copy 4 utilities (`apireset, apijtagreset, heartconf, and hrn_fpga`) to `/usr/local/bin`.
- "Make" the device driver.
- Copy the device driver (`heb2a.o`) to `/lib/modules/*/misc`
- Make device nodes.
- Copy the hotplug files

If you use RedHat 9 or earlier, and use the hotplug package that came with the installation, you must edit (append) a file used for hotplugging (for an explanation, see further below). You should append 'usb/heb2aldr.usermap' to 'usb.usermap':

```
cd /etc/hotplug
cat usb.usermap usb/heb2aldr.usermap >> usermap
mv  usb.usermap usb.usermap.prv
mv  usermap     usb.usermap
```

For Fedora Core 2.0 and higher, the hotplug package recognizes 'usb/heb2aldr.usermap' and no such append action is needed.

Next, connect the HERON-BASE2 USB connector to the PC. After 4, 5 seconds you should see the reset LED light up briefly. This is the firmware initialising.

**HERON-BASE2 firmware**

After the HERON-BASE2 has been powered on, the board needs to have the firmware downloaded. The firmware is located in directory module/heb2aldr, and is made up of the file "heb2aldr.hex". The firmware can be downloaded using a utility called "fxload", which is standard with recent Linux distributions. A batchfile called "ldr" in the module/heb2aldr directory shows how "fxload" can be used with the HERON-BASE2.

The HERON-BASE2 employs a USB chip (fx2) from Cypress, and uses the renumeration feature of this USB chip. Before the firmware is downloaded, the HERON-BASE2 is seen

by the system as a USB device with vendor ID 0x1700, product ID 0x0E00. But after the firmware has been downloaded, the USB chip disconnects from USB, and then re-connects, but now with vendor ID 0x1700, product ID 0x0E2F. It is only after the firmware has been downloaded that the 'real' HERON-BASE2 driver will start to run.

### Hotplugging

What would be really nice is that if you connect the HERON-BASE2, it automatically loads the firmware onto it (if necessary), and starts the HERON-BASE2 driver. The Linux hotplug feature does this for you, if configured correctly.

The module/heb2aldr directory has two files used for hotplugging: heb2aldr and heb2aldr.usermap. Both files need to be copied to /etc/hotplug/usb. Note that the install script will already have done this for you.

For RedHat Linux versions, you need to do one more step. The usb.usermap file in /etc/hotplug should be concatenated with usb/heb2aldr.usermap, as explained in the installation section (previous page).

You can see hotplugging in action as follows. Disconnect the HERON-BASE2, then power cycle the board (so that the firmware isn't loaded yet). When the HERON-BASE2 is powered up, the reset LED will light up for a short while. Wait until the LED has gone off. Then reconnect the HERON-BASE2. After 2 to 4 or so seconds you should see the reset LED light up briefly. This is the firmware initialising.

With dmesg you can now check that the HERON-BASE2 driver has been installed. If the HERON-BASE2 driver has been installed (using insmod, the install script will already have done this) then Linux sould have started up the HERON-BASE2 driver ('heb2a.o'). A typical dmesg (/var/log/messages) output would look like:

```
lx kernel: hub.c: new USB device 00:09.2-4, assigned address 3
lx kernel: usb.c: USB device 3 (vend/prod 0x1700/0xe00) is not claimed by
any active driver.
lx /etc/hotplug/usb.agent: Setup heb2aldr for USB product 1700/e00/1
lx /etc/hotplug/usb.agent: Module setup heb2aldr for USB product 1700/e00/1
lx /etc/hotplug/usb/heb2aldr: load /usr/share/usb/heb2aldr.hex for
1700/e00/1 to /proc/bus/usb/005/003
lx kernel: usb.c: USB disconnect on device 00:09.2-4 address 3
lx devlabel: devlabel service started/restarted
lx devlabel: devlabel service started/restarted
lx kernel: hub.c: new USB device 00:09.2-4, assigned address 4
lx kernel: HERON-BASE2: b2a_probe: board switch 0 [c4:1]
lx kernel: HERON-BASE2: b2a_probe: character device has major number 64
lx kernel: HERON-BASE2: b2a_probe: Hunt Engineering heb2a0, vendor id
0x1700, device id 0xe2f
lx kernel: HERON-BASE2: b2a_probe: packetsize 512, read queue 64, write
queue 64
lx /etc/hotplug/usb.agent: Setup heb2a for USB product 1700/e2f/0
```

Note that with dmesg you only see the non-italic output. When viewing /var/log/messages all output as shown above can be seen.

For more information on hotplugging, see http://linux-hotplug.sourceforge.net/

### HERON-BASE2 Manual Insertion and Removal of a module

Once the firmware has been downloaded to the HERON-BASE2, you can manually stop

and start the driver with 'rmmod' and 'insmod', respectively. Note that the install script will already have started the HERON-BASE2 driver with 'insmod'. And in subsequent sessions, the hotplugging feature will already have started the HERON-BASE2 driver.

To remove a module from the system, type:

```
rmmod heb2a
```

To insert a module, type:

```
insmod heb2a heb2a_major=64
```

You have to be the root user to do this. To check the result of `insmod`, view the last few lines as reported by `dmesg`. A typical entry for the HERON-BASE2 would be:

```
lx kernel: HERON-BASE2: b2a_probe: board switch 0 [c4:1]
lx kernel: HERON-BASE2: b2a_probe: character device has major number 64
lx kernel: HERON-BASE2: b2a_probe: Hunt Engineering heb2a0, vendor id
0x1700, device id 0xe2f
lx kernel: HERON-BASE2: b2a_probe: packetsize 512, read queue 64, write
queue 64
```

If you have more than 1 HERON-BASE2 installed, you'll see multiple "board switch" lines. The board switch corresponds with the red board switch on your HERON-BASE2. This switch can be selected from 0 to 15. The packetsize will tell you whether the board is running in full speed mode (64, USB 1.1) or high-speed mode (512, USB 2.0).

The character device major number is used when setting up the device files. This device major number is assigned automatically if you haven't passed the heb2a_major argument. To pass your own major number to the module, use the following:

```
insmod heb2a heb2a_major=64
```

In the case of the "installme" script, 64 is used for the HERON-BASE2. For the API itself it doesn't matter what number you use, but you must ensure that the character device major number is the same as used when creating the device files (`/dev/heb2a`).

**Creating Device Files**

The Linux kernel identifies drivers by the device major numbers of the device files. The minor numbers are used to identify the ports on each board. To create the device files, enter as root the following:

```
mknod /dev/<name> c <major number> <minor number>
```

You are not free to choose any name. In fact, the "`installme`" script has already initialized a number of device files for you. The name is built up as follows:

HERON-BASE2: `heb2axy`

Where `x` is the hexadecimal board number (0…f) and `y` the device id (0 = fifo A, 1 = fifo B, 2 = JTAG, 3 = HSB (serial bus), 5 = info).

You are also not free to choose the major number. You have to use the number as shown by running `dmesg`, in other words, you must use the device major number reported by the kernel as it installed the module. The default major number is 64 for the HERON-BASE2 driver.

You are also not free to choose the minor number. The minor number is calculated in the following way: board number times 16 + device id. For example, FIFO A of a HERON-BASE2 number 0 has minor number 0, and the jtag of this board has minor number 2. For

HERON-BASE2 number 1, the minor number of FIFO A is 16, and the minor number of the jtag is 18. For your reference, have a look at the "installme" script, a number of device files are initialized here (for board switches 0 to 2).

Check the access flags of the device files. If you want to grant access to them for other users, you usually will have to change them using `chmod`, or use the "--mode" option of the `mknod` command (this is used in the "installme" script).

| ***Board Number*** | ***Device*** | ***Minor Number*** | ***Device File Name*** |
|---|---|---|---|
| **0** | Fifo A | 0 | heb2a00 |
| | Fifo B | 1 | heb2a01 |
| | JTAG | 2 | heb2a02 |
| | HSB | 3 | heb2a03 |
| | Info | 5 | heb2a05 |
| **1** | Fifo A | 16 | heb2a10 |
| | Fifo B | 17 | heb2a11 |
| | JTAG | 18 | heb2a12 |
| | HSB | 19 | heb2a13 |
| | Info | 21 | heb2a15 |
| **X** | Fifo A | 16*x | heb2ax0 |
| | Fifo B | 16*x+1 | heb2ax1 |
| | JTAG | 16*x+2 | heb2ax2 |
| | HSB | 16*x+3 | heb2ax3 |
| | Info | 16*x+5 | heb2ax5 |

Device File Names for an HERON-BASE2 board.

**Insmod parameters**

| HERON-BASE2 | What does it do? |
|---|---|
| `heb2a_major` | Sets character device major no. |

**Support for boards with a boardswitch other than 0, 1 or 2**

The 'installme' script installs the drivers (HERON-BASE2) allowing for 3 possible board switch settings: 0, 1 or 2. If you want to use other board settings, add device nodes for those board numbers (use and change the 'installme' script or read the device node section above, then use 'mknod'). You can also add insmod parameters by making slight changes to the device driver code in module/heb2a.

## Confidence testing for Linux

In your un-tarred HUNT ENGINEERING API directory you will find confidence tests in `etc/c6x/confchk`, where there are six sub-directories: `testint`, `heart`, `em2`,

`fpga`, `reads` and `writes`. If you have a DSP module in slot 1, change directory to `writes/linux`, compile and execute `writes`, for example for an HERON-BASE2:

```
cd etc/c6x/confchk/writes/linux
make
writes heb2a 0 a 10000 10000 100000
```

A system that is working properly returns something similar to:

```
Start at 10000, inc 10000, end at 100000, BlockSize=250 on hep9a (0: Comporta)
Resetting...
Serial bus: slot 1: HERON4-C6701, rom version 5.
Resetting...
Booting /home/johan/api/etc/c6x/confchk/writes/bios/writes4.out...
Testing...
Writes Transfer size 1000 DWORDS in 1 ticks, Speed: 390.63 KBytes/sec
Writes Transfer size 2000 DWORDS in 1 ticks, Speed: 781.25 KBytes/sec
Writes Transfer size 3000 DWORDS in 1 ticks, Speed: 1171.88 KBytes/sec
Writes Transfer size 4000 DWORDS in 1 ticks, Speed: 1562.50 KBytes/sec
Writes Transfer size 5000 DWORDS in 1 ticks, Speed: 1953.14 KBytes/sec
Writes Transfer size 6000 DWORDS in 1 ticks, Speed: 2343.75 KBytes/sec
Writes Transfer size 7000 DWORDS in 1 ticks, Speed: 2734.38 KBytes/sec
Writes Transfer size 8000 DWORDS in 1 ticks, Speed: 3125.00 KBytes/sec
Writes Transfer size 9000 DWORDS in 1 ticks, Speed: 3515.63 KBytes/sec
Writes Transfer size 10000 DWORDS in 1 ticks, Speed: 3906.30 KBytes/sec
Check whether any interrupts were used: read 0, write 0, master mode 1.
```

Please ignore the reports on Speed; the (standard) confidence test uses the time() function which ticks every second on my test machine. Use larger blocks to get tick counts above 1 to get a bit more meaningful speed results. If any problems appear, make sure that the module is started (e.g. the firmware must have been downloaded and you may have had to execute `insmod heb2a heb2a_major=64` as root; see also the "Troubleshooting" section).

To execute another confidence test with a DSP module, change directory to the `reads` directory, and execute `reads`. For example, for a HERON-BASE2:

```
cd ../../reads
make
reads heb2a 0 a 10000 10000 100000
```

A system that is working properly returns something similar to:

```
Start at 10000, inc 10000, end at 100000, BlockSize=250 on heb29a (0: Comporta)
Resetting...
Serial bus: slot 1: HERON4-C6701, rom version 5.
Resetting...
Booting /home/johan/api/etc/c6x/confchk/reads/bios/reads4.out...
Testing...
Reads Transfer size 1000 DWORDS in 1 ticks, Speed: 390.63 KBytes/sec
Reads Transfer size 2000 DWORDS in 1 ticks, Speed: 781.25 KBytes/sec
Reads Transfer size 3000 DWORDS in 1 ticks, Speed: 1171.88 KBytes/sec
Reads Transfer size 4000 DWORDS in 1 ticks, Speed: 1562.50 KBytes/sec
Reads Transfer size 5000 DWORDS in 1 ticks, Speed: 1953.13 KBytes/sec
Reads Transfer size 6000 DWORDS in 1 ticks, Speed: 2343.75 KBytes/sec
Reads Transfer size 7000 DWORDS in 1 ticks, Speed: 2734.38 KBytes/sec
Reads Transfer size 8000 DWORDS in 1 ticks, Speed: 3125.00 KBytes/sec
Reads Transfer size 9000 DWORDS in 1 ticks, Speed: 3515.63 KBytes/sec
Reads Transfer size 10000 DWORDS in 1 ticks, Speed: 3906.25 KBytes/sec
Check whether any interrupts were used: read 0, write 0, master mode 1.
```

Please ignore the reports on Speed; the (standard) confidence test uses the time() function which ticks every second on my test machine. Use larger blocks to get tick counts above 1

to get a bit more meaningful speed results. If any problems appear, make sure that the module is started (e.g. the firmware must have been downloaded and you may have had to execute `insmod heb2a heb2a_major=64` as root; see also the "Troubleshooting" section).

If you don't have a DSP module, but only an FPGA module (HERON-FPGA or HERON-IO) then run the fpga confidence test:

```
cd etc/c6x/confchk/fpga/linux
make
fpgatst heb2a 0 1 100000 100000 1000000
```

if the module is in slot 1. Use '2' instead of '1' if the module is in slot 2. A system that is working properly returns something similar to:

```
Testing board "heb2a 0", from 100000 to 1000000 in steps of 100000.

FPGA test on board heb2a 0, fifo A.
Found device: 2vp7ff672.

Checking for HERON-FPGA and HERON-IO module in slot 0x1 (via 'heb2a 0')
The module in slot 0x1 is a HERON-FPGA9 version 1.
The module supports loading of compressed bitstreams.
The FPGA on this module is a Virtex-2 Pro device with 1 Power PC.
The FPGA speed grade is -5.
Checking header section of the file
</usr/local/share/fpga/fpga9v1/example1/2vp7ff672.hcb>.
Sending in compressed format.

% Done: 100
The bitstream has been downloaded.
Configuration was successful.
Hrn_fpga completed successfully.

HSB FIFO configuration succeeded (fifo 0).
Transfer size 100000 DWORDS in 0 ticks,  Speed: inf KBytes/sec
Transfer size 200000 DWORDS in 1 ticks,  Speed: 1562.50 KBytes/sec
Transfer size 300000 DWORDS in 0 ticks,  Speed: inf KBytes/sec
Transfer size 400000 DWORDS in 0 ticks,  Speed: inf KBytes/sec
Transfer size 500000 DWORDS in 1 ticks,  Speed: 3906.25 KBytes/sec
Transfer size 600000 DWORDS in 2 ticks,  Speed: 2343.75 KBytes/sec
Transfer size 700000 DWORDS in 1 ticks,  Speed: 5468.75 KBytes/sec
Transfer size 800000 DWORDS in 2 ticks,  Speed: 3125.00 KBytes/sec
Transfer size 900000 DWORDS in 1 ticks,  Speed: 7031.25 KBytes/sec
Transfer size 1000000 DWORDS in 2 ticks,  Speed: 3906.25 KBytes/sec
```

Please ignore the reports on Speed; the (standard) confidence test uses the time() function which ticks every second on my test machine. Use larger blocks to get tick counts above 1 to get a bit more meaningful speed results. If any problems appear, make sure that the module is started (e.g. the firmware must have been downloaded and you may have had to execute `insmod heb2a heb2a_major=64` as root; see also the "Troubleshooting" section).

Once the confidence tests run successfully, the HUNT ENGINEERING API has been installed successfully. You can now proceed to develop your application (perhaps with the confidence tests here as an example) or proceed to install the Server/Loader.

If you experienced problems, please refer to the Troubleshooting section below for advice on how to correct problems.

When you have problems with hotplugging, most likely you should have appended the usb/heb2aldr.usermap file to the usb.usermap file:

```
cd /etc/hotplug
cat usb.usermap usb/heb2aldr.usermap >> usermap
mv  usb.usermap usb.usermap.prv
mv  usermap     usb.usermap
```

Verify that the last line of the new usb.usermap file looks like this:

```
cat usb.usermap
…
heb2aldr  0x0003  0x1700  0xe00  0x0000  0x0000  0x00  0x00  0x00
0x00 0x00 0x00 0x00000000
```

Also make sure that files `heb2aldr.usermap` and `heb2aldr` are in /etc/hotplug/usb.

To test hotplugging, disconnect the HERON-BASE2, and then power it off. Power the board up again, and wait until the reset LED goes off. Then re-connect the USB cable of the HERON-BASE2. After 4 to 5 seconds or so you should see the reset LED light up. This is the firmware initialising the board. The HERON-BASE2 renumerates and will now be seen by Linux as a USB device with vendor id 0x1700 and product id 0x0e2f.

By viewing the system logs or by viewing the /var/log/messages file directly you can see hotplugging activity. A proper sequence should show something like:

```
lx kernel: hub.c: new USB device 00:09.2-4, assigned address 3
lx kernel: usb.c: USB device 3 (vend/prod 0x1700/0xe00) is not claimed by
any active driver.
lx /etc/hotplug/usb.agent: Setup heb2aldr for USB product 1700/e00/1
lx /etc/hotplug/usb.agent: Module setup heb2aldr for USB product 1700/e00/1
lx /etc/hotplug/usb/heb2aldr: load /usr/share/usb/heb2aldr.hex for
1700/e00/1 to /proc/bus/usb/005/003
lx kernel: usb.c: USB disconnect on device 00:09.2-4 address 3
```

Once the firmware has been downloaded, verify that the HERON-BASE2 driver has been started. Usually this will have been done by the hotplug system. In your system logs or with dmesg you can then see entries like:

```
lx kernel: HERON-BASE2: b2a_probe: board switch 0 [c4:1]
lx kernel: HERON-BASE2: b2a_probe: character device has major number 64
lx kernel: HERON-BASE2: b2a_probe: Hunt Engineering heb2a0, vendor id
0x1700, device id 0xe2f
lx kernel: HERON-BASE2: b2a_probe: packetsize 512, read queue 64, write
queue 64
```

If you don't see such entries, you can start the HERON-BASE2 manually:

```
insmod heb2a heb2a_major=63
```

You can also use the 're' or 'ra' scripts in the `module/heb2a` directory. With dmesg or by viewing the system logs verify that you see `HERON-BASE2:` entries as in the above. If the driver wasn't installed successfully, the HERON-BASE2 driver will log an error message. An example of an error message is lack of kernel memory (part of the driver initialisation allocates kernel memory). Possible messages are for example:

```
[002] … b2a_probe: expected interface number 0, found 1
```

```
[004] … b2a_probe: Wrong attributes (1) for address 2
[017] … b2a_probe: Not all endpoints found, mask 3f
[018] HERON-BASE2: b2a_probe: unable to read board switch
[020] HERON-BASE2: b2a_probe: can't get major number 64
```

If you see error messages, disconnect the HERON-BASE2, power cycle it, and reconnect. Then run dmesg again or view the systems logs and see if the error is still there.

You may also get error `[020]` as shown above. Please note that the device major number for the HERON-BASE2 may already be in use on your system. In that case use a different number – or none at all. (If you select no device major number with `insmod`, the system will assign a device major number to the module.) Be sure to also verify and/or alter your device files, as described below.

The second thing you need to check is that the device files have been created. First, verify that you have device files: `/dev/heb2a*` for HERON-BASE2. In case you find no such files, please create them, following this procedure.

First, using `dmesg`, find out what device major number is used by the driver. E.g. lines like:

```
heb2a: character device has major number 64
```

tells you that the HERON-BASE2 driver uses device major number 64. Using this number, now create device files, using `mknod`, as follows:

```
mknod --mod=666 heb2a00 c 64 0
mknod --mod=666 heb2a01 c 64 1
etc.
```

for a HERON-BASE2. Please make sure that you replace 64 by the device major number that is reported for your board by `dmesg`.

If the device files were already there, please verify that the device files present do indeed use the correct device major number. One way to do this is to delete the entries currently there and then re-create then, as outlined above.

If `dmesg` reports a `can't get major number` error, then you need to uninstall the API, then re-install with a different character device major number until you find a number that works. To uninstall, run the "`uninstall`" script. To re-install, edit the "`installme`" script and change the character device major number of the board you try to install.

If `dmesg` reports a `can't allocate memory` there's not much we can do. You have to ensure there's enough kernel memory, so try to free up any, if you can, then try to install again (run `insmod` as described above).

If the confidence checks still fail, make sure that you have at least one DSP or FPGA module in slot 1 or slot 2. Ensure that you use the right switches when using slot 2, and that you use only the reads and writes test with a DSP module, and the fpga test with a FPGA module.

Finally verify the 4 LEDs at the side of the HERON-BASE2. The +12V, -12V, 3.3V and 2.5V LEDs must all be on.

If you have problems with performance, make sure that you use a USB 2.0 port. If the port is a USB 1.1 port, performance will be markably slower. If you have an older USB package it may be that that doesn't recognise USB 2.0 ports, and uses them as USB 1.1 ports.

Insert the Hunt Engineering CD. In the `/software/api/linux` directory there is a tar file with the name "`apixxx.tar`" (where 'xxx' is the latest version; at the moment of writing this is '195', so you would untar "`api195.tar`"). This contains the drivers (modules), API interface shared library, include files and confidence test programs (which also serve as examples). Copy the tar file into a directory where you wish to have the API examples installed. Then un-tar the file in the chosen directory as follows:

```
tar xvf apixxx.tar
```

In the un-tarred distribution you will find an install script ("`installme`"). You have to be the root user in order to run the script. The script will attempt to do the following:

- Copy the include files (`heapi.h`, `hecodes.h`, `heioctl.h`) to `/usr/local/include`.
- Copy the API library (`libhel.so`) to `/usr/local/lib`.
- Copy 2 utilities (`apireset`, `apijtagreset`) to `/usr/local/bin`.
- "Make" the device driver.
- Copy the device driver (`hep9a.o` or `hep8a.o` or `hep3b.o` or `hep2e.o`) to `/lib/modules/*/misc`
- Start the device driver (using `insmod`).

LINUX modules have to be started each time the machine is rebooted. There are 3 ways of doing this, described in the following sections.

### HEPC3/4/PCI1 Manual Insertion and Removal of a module

(for a HEPC3, HEPC4, HECPCI1:)

```
insmod hep3b hep3b_major=61
```

You have to be the root user to do this. To check the result of `insmod`, view the last few lines as reported by `dmesg`. A typical entry in the `dmesg` log for the HEPC3, HEPC4, HECPCI1 would be:

```
hep3b0: PCI_LATENCY_TIMER is 32
Hunt Engineering hep3b0 Rev 0
Board ID 0, IRQ=11, PCI addr 0x6000 0x6100 0x6200 0x6300 0x6400
hep3b: character device has major number 61
```

If you have more than 1 HEPC3, HEPC4 or HECPCI1 installed, you'll see multiple "Board ID" lines. The IRQ number shows what IRQ the board will use. The HEPC3, HEPC4 and HECPCI1 use I/O ports as shown by "PCI addr".

The character device major number is used when setting up the device files. This device major number is assigned automatically. To pass your own major number to the module, use the following (e.g. for a HEPC3):

```
insmod hep3 hep8a_major=62
```

In the case of the "installme" script, 61 is used for HEPC3, HEPC4 or HECPCI1. For the API itself it doesn't matter what number you use, but you must ensure that the character device major number is the same as used when creating the device files (`/dev/hep3b*`).

To remove a module from the system, type (e.g. for a HEPC3):

```
rmmod hep3b
```

The advantage of this method is that it allows finest control over module options. The disadvantage is that you need to be a root user. A better method might be "automagic" insertion, which is described in the next section.

**Automagic Insertion of a module**

The kernel runs a special process, the kernel daemon, which can perform the actions described in the previous section "automagically". Each time a process wants to access a device file, the kernel checks if it can support the device with the corresponding device major number. If not, it asks the kernel daemon to find and insert the driver. A short while after the last process that used the driver has terminated, the kernel daemon will remove the module. Everything is transparent to the user.

The kernel daemon needs the following to do its job:

- the module file (e.g. `hep3b.o`) must be in directory `/lib/modules/<version>/misc`
- the file `etc/conf.modules` or `etc/modules.conf` must contain information which links the the devices's major number to the module file, and you must specify any options that should be passed to the module. Assuming you use the device major numbers 61, the entries in the file should look like this:

```
alias char-major-61 hep3b
options hep3b hep3b_major=61
```

To check the success of the kernel daemon, use `dmesg` or have a look in the `/proc` directory (e.g. in the case of a HEPC3, look at the `/proc/hep3b` text file).

**Compilation into the kernel**

This is best described in the documentation that came with your LINUX distribution. Or if you started "from scratch", on the Internet there are lots of sources that tell you how to include (Hunt Engineering) modules into your kernel for your particular configuration.

**Creating Device Files**

The Linux kernel identifies drivers by the device major numbers of the device files. The minor numbers are used to identify the ports on each board. To create the device files, enter as root the following:

```
mknod /dev/<name> c <major number> <minor number>
```

You are not free to choose any name. In fact, the "`installme`" script has already initialized a number of device files for you. The name is built up as follows:

HEPC3: `hep3bxy`

Where `x` is the hexadecimal board number (0…f) and `y` the device id (0 = comport A or fifo A, 1 = comport B or fifo B, 2 = JTAG, 3 = comport C or fifo C).

You are also not free to choose the major number. You have to use the number as shown by running `dmesg`, in other words, you must use the device major number reported by the kernel as it installed the module. The default major number is 61 for the HEPC3, HEPC4 and HECPCI1 driver.

You are also not free to choose the minor number. The minor number is calculated in the

following way: board number times four + device id. For your reference, have a look at the "`installme`" script, a number of device files are initialized here.

Check the access flags of the device files. If you want to grant access to them for other users, you usually will have to change them using `chmod`, or use the "--mode" option of the `mknod` command (this is used in the "`installme`" script).

| ***Board Number*** | ***Device*** | ***Minor Number*** | ***Device File Name*** |
|---|---|---|---|
| **0** | Fifo A | 0 | hep3b00 |
| | Fifo B | 1 | hep3b01 |
| | JTAG | 2 | hep3b02 |
| **1** | Fifo A | 4 | hep3b10 |
| | Fifo B | 5 | hep3b11 |
| | JTAG | 6 | hep3b12 |
| **x** | Fifo A | 4*x | hep3bx0 |
| | Fifo B | 4*x+1 | hep3bx1 |
| | JTAG | 4*x+2 | hep3bx2 |

Device File Names for a HEPC3 board.

**Reading Status Information**

If you have support for the `proc` files system compiled into your kernel, you can get some more specific information about the status of your driver and board by typing:

```
cat /proc/hep3b          (for HEP3B boards)
```

**Insmod parameters**

| HEPC3 | What does it do? |
|---|---|
| `hep3b_major` | Sets character device major no. |
| `Ticks (16)` | Wait period when ints disabled. |
| `int0 (1)` | Enable/disable board 0 intrpt |
| `int1 (1)` | Enable/disable board 1 intrpt |
| `int2 (1)` | Enable/disable board 2 intrpt |
| `jtag0 (1)` | Enable/disable board 0 JTAG |
| `jtag1 (1)` | Enable/disable board 1 JTAG |
| `jtag2 (1)` | Enable/disable board 2 JTAG |
| `buf_size` | Buffer size used for MasterMode |
| `latency 32` | PCI latency, used by AMCC chip |
| `nmm0 (0)` | Disable/enable brd 0 mastermode |
| `nmm1 (0)` | Disable/enable brd 1 mastermode |
| `nmm2 (0)` | Disable/enable brd 2 mastermode |

Note 1: The value in brackets is the default value. Default for buf_size is 4096.
Note 2: MasterMode: enabled by nmm0=0, disabled by nmm0=1 (nmm=no master mode).

## Confidence testing for Linux

In your un-tarred HUNT ENGINEERING API directory you will find confidence tests in `etc/c4x/examples/linux`, which has 3 sub-directories: `example`, `reads` and `writes`. Change directory to the example directory, then execute the example (e.g. for a HEPC3, HEPC4 or HECPCI1:)

```
cd etc/c4x/examples/linux/example
example hep3b 0 a identify.out
```

For example:

```
Manufacturers ID = 01       HUNT ENGINEERING

CPU ID = 00               Texas 320C40
CPU Clock rate = 31nS, CLKIN = 40MHz, H1 freq = 20Mhz
Module Function = 0003      HET40SX SRAM TIM
Revision Level = b
Global 0 memory starts at        80000000 hex
Global 1 memory starts at        ffffffff hex
Local 0 memory starts at         00300000 hex
Local 1 memory starts at         ffffffff hex
Global   0   memory   has   40000   hex   words   which   is
1048576 bytes
Global 1 memory has 0 hex words which is    0 bytes
Local 0 memory has 40000 hex words which is 1048576 bytes
Local 1 memory has 0 hex words which is     0 bytes
```

Please be aware that in your case the precise contents are different from the above. An

IDROM holds module specific information such as memory sizes, and your TIM-40 module is surely different from the one used to produce the above example.

THE ONE SITUATION IN WHICH THIS PROGRAM WILL NOT SERVE AS A CONFIDENCE TEST IS WHEN THE ROOT MODULE OF YOUR SYSTEM DOES NOT HAVE A SERIAL IDROM. The only HUNT ENGINEERING TIM-40 module to which this applies is the HEQUAD.

To execute the next confidence test, change directory to writes, and execute "writes". For example, for an HEPC3, HEPC4 or HECPCI1:

```
cd ../writes
writes hep3b 0 a 10000 10000 100000
```

A properly working system returns something similar to:

```
Start at 10000, inc 10000, end at 100000, Blocksize=250 on hep…
Resetting…
Booting writes.out…
Testing…
Writes Transfer size 10000 DWORDS in 10000 ticks, Speed …
…
Writes Transfer size 100000 DWORDS in 10000 ticks, Speed …
```

Please ignore the reports on Speed; the (standard) confidence test uses `clock()`, and this always seems to return 0 on Linux, so that the timing is always calculated as 10000 ticks. If any problems appear, make sure that the module is started (e.g. do an `insmod hep3b hep3b_major=61` as root, or see the "Troubleshooting" section).

To execute the final confidence test, change directory to `reads`, and execute `reads`. For example, for an HEPC3, HEPC4 or HECPCI1:

```
cd ../reads
reads hep3b 0 a 10000 10000 100000
```

Please ignore the reports on Speed; the (standard) confidence test uses `clock()`, and this always seems to return 0 on Linux, so that the timing is always calculated as 10000 ticks. If any problems appear, (1) make sure that the module is started (e.g. do an `insmod hep3b hep3b_major=61` as root, or see the "Troubleshooting" section).

Once all three confidence tests run successfully, the HUNT ENGINEERING API has been installed successfully. You can now proceed to develop your application (perhaps with the confidence tests here as an example) or proceed to install the Server/Loader.

If you experienced problems, please refer to the Troubleshooting section for advice on how to correct problems.

There are two common things that one easily forgets when using modules. First, to have started up the module (using `insmod`), the other is to have the device files use a different character major number than the modules. First, let's have a look at module installation.

To check whether the module was installed properly, the first thing you need to do is to use `dmesg` to view if the driver has reported anything, and if so, if there are error messages or that installation was fine. If it was installed successfully, you will see a message somewhere that describes the board and the resources it consumes.

e.g. for a HEPC3, HEPC4 or HECPCI1 board you'll see with `dmesg`:

```
hep3b0: PCI_LATENCY_TIMER is 32
Hunt Engineering hep3b0 Rev 0
Board ID 0, IRQ=11, PCI addr 0x6000 0x6100 0x6200 0x6300
0x6400
hep3b: character device has major number 61
```

If the driver wasn't installed successfully, you will see an error message. An example of an error message is lack of kernel memory (part of the driver initialisation allocates kernel memory). Possible messages are. for example:

```
hep3b0: no HEPC3/HEPC4/HECPCI1 found
hep3b0: can't get major number 61
hep3b0: could not get irq line 10
hep3b0: can't allocate memory (x bytes).
hep3b0: can't get io at 0x6000
```

If the driver wasn't installed at all, no message can be seen. In that case, run `insmod` to install the driver, then use `dmesg` to verify the result. Use `insmod` as follows for HEPC3/4/PCI1:

```
insmod hep3b hep3b_major=61
```

Please note that these device major numbers may already be in use on your system. In that case use a different number – or none at all. (If you select no device major number with `insmod`, the system will assign a device major number to the module.) Be sure to also verify and/or alter your device files, as described below.

The second thing you need to check is that the device files have been created. First, verify that you have device files `/dev/hep3b*`. In case you find no such files, please create them, following this procedure:

Using `dmesg`, find out what device major number is used by the driver. e.g. lines like:

```
hep3b: character device has major number 61
```

tells you that the HEPC3 / HEPC4 / HECPCI1 driver uses device major number 61. Using this number, now create device files, using `mknod`, as follows:

```
mknod --mod=666 hep3b00 c 61 0
mknod --mod=666 hep3b01 c 61 1
mknod --mod=666 hep3b02 c 61 2
```

Please make sure that you replace 61 by the device major number that is reported for your board by `dmesg`.

If the device files were already there, please verify that the device files present do indeed use

the correct device major number. One way to do this is to delete the entries currently there (`hep3b*`), and then re-create then, as outlined above.

When using a HEPC3, HEPC4 or HECPCI1, and with `dmesg` you find an error message `no HEPC3/HEPC4/HECPCI1 found`, please insert the board first, then install the module (with `insmod`). These boards are PCI boards, and the module scans the PCI bus to find any installed boards. If it finds nothing, the module cannot be installed.

If `dmesg` reports a `can't get major number` error, then you need to uninstall the API, then re-install with a different character device major number until you find a number that works. To uninstall, run the "uninstall" script. To re-install, edit the "`installme`" script and change the character device major number of the board you try to install.

If `dmesg` reports a `can't allocate memory` there's not much we can do. You have to ensure there's enough kernel memory, so try to free up any, if you can, then try to install again (run `insmod` as described above).

Once you have checked these basic issues, but the confidence checks still fail, it's time to have a look at interrupts and board addresses. First let's look at interrupts.

If, when running `dmesg`, you found error messages such as: `could not get irq line 10`, `IRQ not set for this board`, or `IRQ x not supported. Interrupts disabled`, then there certainly is an IRQ problem. But there may still be an IRQ problem even if `dmesg` reports no errors whatsoever. To check if IRQs are the problem, install the module with interrupts disabled:

HEPC3/HEPC4/HECPCI1:
```
rmmod  hep3b
insmod hep3b hep3b_major=61 int0=0
```

Now run the Confidence Tests again. If this time they run fine, obviously there is a problem with interrupts. Reasons for IRQ problems are: all interrupts are in use, there's none free to be used by the Hunt Engineering board, or one interrupt is shared between the Hunt Engineering board and other boards, and the drivers don't work together properly. Your choice is: remove the other board, and use the Hunt Engineering board with interrupts enabled. Or you can have the other board also installed, but use the Hunt Engineering board with interrupts disabled. There's no problem disabling interrupts; but the driver will be somewhat slower and it consumes more processing power (as it has to do polling at regular intervals).

Note that PCI boards (HEPC3, HEPC4, HECPCI1) use a different numbering scheme. The HEPC3, HEPC4 and HECPCI1 board is always addressed via number 0 if you have only 1 board installed, 0 or 1 if you have 2 installed, and so on.

Finally, in some PCs the CMOS allows you to allocate certain interrupts to either PCI or ISA only. It might be that your interrupt problem was caused by the interrupt being not available to the ISA bus or PCI bus. The CMOS configuration program can usually be started by pressing DEL or some such key immediately after reset. Typically there will be a short message on the screen telling you when the DEL key is "active".

Once you verified that IRQs are not the problem (e.g. the confidence check also fails when interrupts are disabled), have a look at the board's address. Addressing problems may also have been indicated by `dmesg` error messages such as `can't map memory at 0xf0000000` or `can't get io at 0x200`. Most likely other boards that have already reserved that space for themselves cause them. With the HEPC3, HEPC4 and HECPCI1 you can use the `jtag0`, `jtag1` or `jtag2` parameter with `insmod` to disable the jtag interface. This will reduce the memory or I/O area that the module will try to reserve for

itself. Install the driver as follows:

(HEPC3, 4, I1)
```
rmmod  hep3b
insmod hep3b hep3b_major=61 jtag0=0
```

Now run the Confidence Tests again.

If the Confidence Tests still don't run, find out what other board makes a claim on the same memory or I/O space. In RedHat 6.1 with KDE, you can have a look at what boards claim what memory using K→ Settings → Information → IO-Ports. If there are boards that claim the memory space or I/O space that overlaps the memory space or I/O space that the Hunt Engineering board wants to use, for PCI boards there is no other solution but to remove the other board.

Finally, for HEPC3, HEPC4, HECPCI1, there's a possibility that the mastermode feature doesn't work well on or with your PC. Sometimes selecting a different PCI slot can solve the problem. If all PCI slots result in the Confidence Tests failing, you can try to run the Tests while disabling mastermode. e.g. do an `insmod` as follows:

HEPC3, HEPC4, HECPCI1
```
rmmod  hep3b
insmod hep3b hep3b_major=61 nmm0=1
```

The `nmm` parameter stands for **n**o **m**aster **m**ode, disabling mastermode means `nmm=1`.

Insert the Hunt Engineering CD. In the `/software/api/linux` directory there is a tar file with the name "`apixxx.tar`" (where 'xx' is the latest version; at the moment of writing this is '195', so you would untar "`api195.tar`"). This contains the drivers (modules), API interface shared library, include files and confidence test programs (which also serve as examples). Copy the tar file into a directory where you wish to have the API examples installed. Then un-tar the file in the chosen directory as follows:

```
tar xvf apixxx.tar
```

In the un-tarred distribution you will find an install script ("`installme`"). You have to be the root user in order to run the script. The script will attempt to do the following:

- Copy the include files (`heapi.h`, `hecodes.h`, `heioctl.h`) to `/usr/local/include`.
- Copy the API library (`libhel.so`) to `/usr/local/lib`.
- Copy 2 utilities (`apireset`, `apijtagreset`) to `/usr/local/bin`.
- "Make" the device driver.
- Copy the device driver (`hep9a.o` or `hep8a.o` or `hep3b.o` or `hep2e.o`) to `/lib/modules/*/misc`
- Start the device driver (using `insmod`).

LINUX modules have to be started each time the machine is rebooted. There are 3 ways of doing this, described in the following sections.

**HEPC2E Manual Insertion and Removal of a module**

```
insmod hep2e hep2e_major=60 int0=12
```

You have to be the root user to do this. To check the result of `insmod`, view the last few lines as reported by `dmesg`. A typical entry for the `dmesg` log HEPC2E would be

```
Hunt Engineering hep2e0 Rev d
Board ID 0, IRQ=12, Addr 0x200, JTAG (1): 0x240, 0x600, 0xa00
Hunt Engineering hep2e0 Rev d
Board ID 1, IRQ=0, Addr 0x160, JTAG (1): 0x1a0, 0x560, 0x960
Hunt Engineering hep2e0 Rev d
Board ID 2, IRQ=0, Addr 0x300, JTAG (1): 0x340, 0x700, 0xb00
hep2e: character device has major number 60
```

The HEPC2E has 3 possible addresses, and the driver is set up to accept each of these. It doesn't know in advance what address will be used, and only at the open statement the i/o space is actually reserved by the driver. When using the API, the HEPC2E board number decides what board address is selected. Board number 0 uses address 0x200, board number 1 uses address 0x160 and board number 2 uses address 0x300.

The character device major number is used when setting up the device files. This device major number is assigned automatically. To pass your own major number to the module, use the following:

```
insmod hep2e hep2e_major=60
```

In the case of the "installme" script, 60 is used for the HEPC2E. For the API itself it

doesn't matter what number you use, but you must ensure that the character device major number is the same as used when creating the device files (`/dev/hep2e*`).

To remove a module from the system, type (e.g. for a HEPC2E):

```
rmmod hep2e
```

The advantage of this method is that it allows finest control over module options. The disadvantage is that you need to be a root user. A better method might be "automagic" insertion, which is described in the next section.

**Automagic Insertion of a module**

The kernel runs a special process, the kernel daemon, which can perform the actions described in the previous section "automagically". Each time a process wants to access a device file, the kernel checks if it can support the device with the corresponding device major number. If not, it asks the kernel daemon to find and insert the driver. A short while after the last process that used the driver has terminated, the kernel daemon will remove the module. Everything is transparent to the user.

The kernel daemon needs the following to do its job:

- the module file (e.g. `hep2e.o`) must be in directory `/lib/modules/<version>/misc`
- the file `etc/conf.modules` or `etc/modules.conf` must contain information which links the the devices's major number to the module file, and you must specify any options that should be passed to the module. Assuming you use the device major number 60, the entries in the file should look like this:

```
alias char-major-60 hep2e
options hep2e hep2e_major=62 int0=12
```

To check the success of the kernel daemon, use `dmesg` or have a look in the `/proc` directory (e.g. in the case of a HEPC2E, look at the `/proc/hep2e` text file).

**Compilation into the kernel**

This is best described in the documentation that came with your LINUX distribution. Or if you started "from scratch", on the Internet there are lots of sources that tell you how to include (Hunt Engineering) modules into your kernel for your particular configuration.

**Creating Device Files**

The Linux kernel identifies drivers by the device major numbers of the device files. The minor numbers are used to identify the ports on each board. To create the device files, enter as root the following:

```
mknod /dev/<name> c <major number> <minor number>
```

You are not free to choose any name. In fact, the "`installme`" script has already initialized a number of device files for you. The name is built up as follows:

HEP2E: `hep2exy`

Where `x` is the hexadecimal board number (0…f) and `y` the device id (0 = comport A or fifo A, 1 = comport B or fifo B, 2 = JTAG, 3 = comport C or fifo C).

You are also not free to choose the major number. You have to use the number as shown by running `dmesg`, in other words, you must use the device major number reported by the kernel as it installed the module. The default major number is 60 for the HEPC2E driver.

You are also not free to choose the minor number. The minor number is calculated in the following way: board number times four + device id. For your reference, have a look at the "installme" script, a number of device files are initialized here.

Check the access flags of the device files. If you want to grant access to them for other users, you usually will have to change them using `chmod`, or use the "--mode" option of the `mknod` command (this is used in the "installme" script).

| ***Board Number*** | ***Device*** | ***Minor Number*** | ***Device File Name*** |
|---|---|---|---|
| **0** | Fifo A | 0 | hep2e00 |
| | Fifo B | 1 | hep2e01 |
| | JTAG | 2 | hep2e02 |
| **1** | Fifo A | 4 | hep2e10 |
| | Fifo B | 5 | hep2e11 |
| | JTAG | 6 | hep2e12 |
| **2** | Fifo A | 8 | hep2e20 |
| | Fifo B | 9 | hep2e21 |
| | JTAG | 10 | hep2e22 |

Device File Names for a HEPC2E board. Note that only 3 possibilities exist, as the HEPC2E can only be accessed via ports 0x150, 0x200 and 0x300, allowing for a maximum of 3 HEPC2E's in any PC.

**Reading Status Information**

If you have support for the `proc` files system compiled into your kernel, you can get some more specific information about the status of your driver and board by typing:

```
cat /proc/hep2e        (for HEPC2E boards)
```

**Insmod parameters**

| HEP2E | What does it do? |
|---|---|
| `hep2e_major` | Sets character device major no. |
| `Ticks (16)` | Wait period when ints disabled. |
| `int0 (1)` | Enable/disable board 0 intrpt |
| `int1 (1)` | Enable/disable board 1 intrpt |
| `int2 (1)` | Enable/disable board 2 intrpt |
| `jtag0 (1)` | Enable/disable board 0 JTAG |
| `jtag1 (1)` | Enable/disable board 1 JTAG |
| `jtag2 (1)` | Enable/disable board 2 JTAG |

Note 1: Parameters int0, int1 and int2 are used on the HEPC2E to select IRQ also.
Note 2: The value in brackets is the default value. Default for buf_size is 4096.

## Confidence testing for Linux

In your un-tarred HUNT ENGINEERING API directory you will find confidence tests in `etc/c4x/examples/linux`, which has 3 sub-directories: `example`, `reads` and `writes`. Change directory to the example directory, then execute the example

```
cd etc/c4x/examples/linux/example
example hep2e 0 a identify.out
```

A successful run will dump the contents of the IDROM. For example:

```
Manufacturers ID = 01       HUNT ENGINEERING

CPU ID = 00               Texas 320C40
CPU Clock rate = 31nS, CLKIN = 40MHz, H1 freq = 20Mhz
Module Function = 0003      HET40SX SRAM TIM
Revision Level = b
Global 0 memory starts at        80000000 hex
Global 1 memory starts at        ffffffff hex
Local 0 memory starts at         00300000 hex
Local 1 memory starts at         ffffffff hex
Global 0 memory has 40000 hex words which is
1048576 bytes
Global 1 memory has 0 hex words which is    0 bytes
Local 0 memory has 40000 hex words which is 1048576 bytes
Local 1 memory has 0 hex words which is     0 bytes
```

Please be aware that in your case the precise contents are different from the above. An IDROM holds module specific information such as memory sizes, and your TIM-40 module is surely different from the one used to produce the above example.

THE ONE SITUATION IN WHICH THIS PROGRAM WILL NOT SERVE AS A CONFIDENCE TEST IS WHEN THE ROOT MODULE OF YOUR SYSTEM DOES NOT HAVE A SERIAL IDROM. The only HUNT ENGINEERING TIM-40 module to which this applies is the HEQUAD.

To execute the next confidence test, change directory to writes, and execute "writes". For example:

```
cd ../writes
writes hep2e 0 a 10000 10000 100000
```

A properly working system returns something similar to:

```
Start at 10000, inc 10000, end at 100000, Blocksize=250 on hep...
Resetting...
Booting writes.out...
Testing...
Writes Transfer size 10000 DWORDS in 10000 ticks, Speed ...
...
Writes Transfer size 100000 DWORDS in 10000 ticks, Speed ...
```

Please ignore the reports on Speed; the (standard) confidence test uses `clock()`, and this always seems to return 0 on Linux, so that the timing is always calculated as 10000 ticks. If any problems appear, make sure that the module is started (e.g. do an `insmod hep2e hep2e_major=61` as root, or see the "Troubleshooting" section).

To execute the final confidence test, change directory to `reads`, and execute `reads`. For example:

```
cd ../reads
reads hep2e 0 a 10000 10000 100000
```

A properly working system returns something similar to:

```
Start at 10000, inc 10000, end at 100000, Blocksize=250 on hep...
Resetting...
Booting reads.out...
Testing...
Reads Transfer size 10000 DWORDS in 10000 ticks,  Speed ...
...
Reads Transfer size 100000 DWORDS in 10000 ticks, Speed ...
```

Please ignore the reports on Speed; the (standard) confidence test uses `clock()`, and this always seems to return 0 on Linux, so that the timing is always calculated as 10000 ticks. If any problems appear, (1) make sure that the module is started (e.g. do an `insmod hep3b hep3b_major=61` as root, or see the "Troubleshooting" section).

Once all three confidence tests run successfully, the HUNT ENGINEERING API has been installed successfully. You can now proceed to develop your application (perhaps with the confidence tests here as an example) or proceed to install the Server/Loader.

If you experienced problems, please refer to the Troubleshooting section for advice on how to correct problems.

There are two common things that one easily forgets when using modules. First, to have started up the module (using `insmod`), the other is to have the device files use a different character major number than the modules. First, let's have a look at module installation.

To check whether the module was installed properly, the first thing you need to do is to use `dmesg` to view if the driver has reported anything, and if so, if there are error messages or that installation was fine. If it was installed successfully, you will see a message somewhere that describes the board and the resources it consumes.

e.g. for a HEPC2 board you'll see with `dmesg`:

```
Hunt Engineering hep2e0 Rev d
Board ID 0, IRQ=12, Addr 0x200, JTAG (1): 0x240, 0x600,
0xa00
Hunt Engineering hep2e0 Rev d
Board ID 1, IRQ=0, Addr 0x160, JTAG (1): 0x1a0, 0x560,
0x960
Hunt Engineering hep2e0 Rev d
Board ID 2, IRQ=0, Addr 0x300, JTAG (1): 0x340, 0x700,
0xb00
hep2e: character device has major number 60
```

If the driver wasn't installed successfully, you will see an error message. An example of an error message is lack of kernel memory (part of the driver initialisation allocates kernel memory). Possible messages are. for example:

```
hep2e0: can't get major number 60
hep2e0: IRQ not set for this board.
hep2e0: could not get irq line 12
hep2e0: can't allocate memory (x bytes)
hep2e0: board number x cannot exist, 3 is maximum
hep2e0: IRQ x not supported. Interrupts disabled
hep2e0: can't get io at 0x200
```

If the driver wasn't installed at all, no message can be seen. In that case, run `insmod` to install the driver, then use `dmesg` to verify the result. Use `insmod` as follows:

```
insmod hep2e hep2e_major=60 int0=12
```

(You may need a different "int" parameter, depending on your HEPC2E board settings.)

Please note that these device major numbers may already be in use on your system. In that case use a different number – or none at all. (If you select no device major number with `insmod`, the system will assign a device major number to the module.) Be sure to also verify and/or alter your device files, as described below.

The second thing you need to check is that the device files have been created. First, verify that you have device files: `/dev/hep2e*` (for HEPC2E). In case you find no such files, please create them, following this procedure:

Using `dmesg`, find out what device major number is used by the driver. e.g. lines like:

```
hep2e: character device has major number 60
```

tells you that the HEPC2E driver uses device major number 60. Using this number, now create device files, using `mknod`, as follows:

for HEPC2E:

```
mknod --mod=666 hep2e00 c 60 0
mknod --mod=666 hep2e01 c 60 1
mknod --mod=666 hep2e02 c 60 2
```

Please make sure that you replace 60 by the device major number that is reported for your board by `dmesg`.

If the device files were already there, please verify that the device files present do indeed use the correct device major number. One way to do this is to delete the entries currently there (`hep2e*`), and then re-create then, as outlined above.

If `dmesg` reports a `can't get major number` error, then you need to uninstall the API, then re-install with a different character device major number until you find a number that works. To uninstall, run the "uninstall" script. To re-install, edit the "`installme`" script and change the character device major number of the board you try to install.

If `dmesg` reports a `can't allocate memory` there's not much we can do. You have to ensure there's enough kernel memory, so try to free up any, if you can, then try to install again (run `insmod` as described above).

Once you have checked these basic issues, but the confidence checks still fail, it's time to have a look at interrupts and board addresses. First let's look at interrupts.

If, when running `dmesg`, you found error messages such as: `could not get irq line 10`, `IRQ not set for this board`, or `IRQ x not supported. Interrupts disabled`, then there certainly is an IRQ problem. But there may still be an IRQ problem even if `dmesg` reports no errors whatsoever. To check if IRQ's are the problem, install the module with interrupts disabled:

HEPC2E:

```
rmmod  hep2e
insmod hep2e hep2e_major=60
```

Now run the Confidence Tests again. If this time they run fine, obviously there is a problem with interrupts. Reasons for IRQ problems are: all interrupts are in use, there's none free to be used by the Hunt Engineering board, or one interrupt is shared between the Hunt Engineering board and other boards, and the drivers don't work together properly. Your choice is: remove the other board, and use the Hunt Engineering board with interrupts enabled. Or you can have the other board also installed, but use the Hunt Engineering board with interrupts disabled. There's no problem disabling interrupts; but the driver will be somewhat slower and it consumes more processing power (as it has to do polling at regular intervals).

For HEPC2E boards, in addition you have to check that the IRQ you specified with `insmod` matches the IRQ as set on the HEPC2E. Also, you need to match the `int` parameter with the board number. e.g. to use IRQ 10 with a HEPC2E addressed at 0x160:

```
insmod hep2e hep2e_major=60 int1=10
```

Or e.g. to use IRQ15 with a HEPC2E addressed at 0x300:

```
insmod hep2e hep2e_major=60 int2=15
```

Or e.g. to use IRQ12 with a HEPC2E addressed at 0x200:

```
insmod hep2e hep2e_major=60 int0=12
```

In other words, `int0` specifies the IRQ for the HEPC2E addressed at 0x200, `int1` specifies the IRQ for the HEPC2E addressed at 0x160, and `int2` specifies the IRQ for the

HEPC2E addressed at 0x300. Similarly, when running the Confidence Tests, you have to specify the proper board number. e.g. for a HEPC2E addressed at 0x160:

```
example hep2e 1 a identify.out
```

Or e.g. for a HEPC2E addressed at 0x300:

```
example hep2e 2 a identify.out
```

Or e.g. for a HEPC2E addressed at 0x200:

```
example hep2e 0 a identify.out
```

Finally, in some PCs the CMOS allows you to allocate certain interrupts to either PCI or ISA only. It might be that your interrupt problem was caused by the interrupt being not available to the ISA bus or PCI bus. The CMOS configuration program can usually be started by pressing DEL or some such key immediately after reset. Typically there will be a short message on the screen telling you when the DEL key is "active".

Once you verified that IRQs are not the problem (e.g. the confidence check also fails when interrupts are disabled), have a look at the board's address. Addressing problems may also have been indicated by `dmesg` error messages such as `can't map memory at 0xf0000000` or `can't get io at 0x200`. Most likely other boards that have already reserved that space for themselves cause them. With the HECPCI1 you can use the `jtag0`, `jtag1` or `jtag2` parameter with `insmod` to disable the jtag interface. This will reduce the memory or I/O area that the module will try to reserve for itself. Install the driver as follows:

(HEPC2E:)
```
rmmod  hep2e
insmod hep2e hep2e_major=60 jtag0=0 int0=12
```

As before, for the HEPC2E parameter `jtag0` is for the HEPC2E addressed at 0x200, the parameter `jtag1` is for the HEPC2E addressed at 0x160 and parameter `jtag2` is for the HEPC2E addressed at 0x300. Now run the Confidence Tests again.

If the Confidence Tests still don't run, find out what other board makes a claim on the same memory or I/O space. In RedHat 6.1 with KDE, you can have a look at what boards claim what memory using K→ Settings → Information → IO-Ports. If there are boards that claim the memory space or I/O space that overlaps the memory space or I/O space that the Hunt Engineering board wants to use, for PCI boards there is no other solution but to remove the other board. For the HEPC2E, you can select a different address: choice of 0x160, 0x200 and 0x300. Power down the machine, set the address jumper to a free address, use `insmod` to define the proper IRQ for that HEPC2E, and run the Confidence Tests again.

## When to use the API

The HUNT ENGINEERING API offers a platform independent, target independent way of accessing a HUNT ENGINEERING board. You can write your own programs that use the API to boot DSP programs onto a board. However, the Server/Loader may provide all functionality you need, and you may not have to (or want to) develop API applications.

The Server/Loader is a HUNT ENGINEERING tool that can boot a network of DSP processors. The Server/Loader uses a 'network' file that describes what processors must be booted with what file. A 'network' file is a simple and short ASCII file. ASCII files can be created with, for example, 'edit' on MS-DOS, 'notepad' on Windows, or 'vi' on Linux. In cases where you want the Server/Loader to only boot the network, and process messages from your DSP application yourself, you can use the Server/Loader library. Please refer to the Server/Loader manual for more information.

## Include file

All API applications must '`#include <heapi.h>`' in all source files that call API functions. This include file is located in '/usr/local/include'. The 'installme' script should have copied 'heapi.h' into this directory.

## Compile parameters

The HUNT ENGINEERING API covers several different operating systems. In your application you have to select for what operating system you want to compile. To select a LINUX application, set parameter '_LINUX' to 1. For example, you could do this in the makefile. For example, in the makefile:

```
CFLAGS=-O2 -Wall -D_LINUX=1
```

## Libraries

The Hunt Engineering API is a shared library ('libhel.so'). This file is located in directory '/usr/local/lib'. The 'installme' script should have copied it there. It must be linked before other (GNU) libraries. For example:

```
reads: main.o cload.o
   $(CC) $(CFLAGS) main.o cload.o -o reads
                  /usr/local/lib/libhel.so /usr/lib/librt.so
```

The HUNT ENGINEERING CD has a number of examples that show how you could use the API. The examples are located on the CD in: \software\examples\host_api_examples\c6x\examples. There will be PDF files in the example directories that explain how the examples work. Each example directory has a number of sub-directories; each sub-directory dedicated to a certain operating system. In sub-directory 'linux' you will find a PDF file that explains how to compile and build the example for LINUX.

The HUNT ENGINEERING CD has a number of examples that show how you could use the API. The examples are located on the CD in: \software\examples\host_api_examples\ c4x\examples. There will be PDF files in the example directories that explain how the examples work. Each example directory has a number of sub-directories; each sub-directory dedicated to a certain operating system. In sub-directory 'linux' you will find a PDF file that explains how to compile and build the example for LINUX.

The API Reference Manual can be found in the \manuals directory on the HUNT ENGINEERING CD. The Reference Manual explains in detail how to use the API, shows all possible return values, contains an overview of all functions in the API library plus a per-function explantion, and so on. Essentially, the platform and carrier board independent API interface is explained in this manual.

Technical support for HUNT ENGINEERING products should first be obtained from the comprehensive Support section http://www.hunteng.co.uk/support/index.htm on the HUNT ENGINEERING web site. This includes FAQs, latest product, software and documentation updates etc. Or contact your local supplier - if you are unsure of details please refer to http://www.hunteng.co.uk for the list of current re-sellers.

HUNT ENGINEERING technical support can be contacted by emailing support@hunteng.demon.co.uk, calling the direct support telephone number +44 (0)1278 760775, or by calling the general number +44 (0)1278 760188 and choosing the technical support option.

There are three main development tools that are supported by HUNT ENGINEERING which are described in the following sections. With a very few exceptions, these tools use the API, which provides two important benefits:

They can co-exist and rely upon the API to assist in preventing interference.

New module carriers are automatically supported without any additional work required from the tools provider

In addition to those DSP development tools the API can itself be used as a development tool for your system, as it provides an easy way to interface programs on your Host computer with programs on your DSPs.

## HUNT ENGINEERING API

For systems running under LINUX, the API provides a library file that links with your application to provide that application with all of the standard API interface. When your LINUX application actually runs, the Operating system automatically loads a Dynamic Link Library (‘libhel.so’), and it tries to call a the appropriate driver module for the board you have specified in the `HeOpen()` call.

If, later in the life of your system you choose to change the host board type, you simply change the “board type” identifier given to the `HeOpen()` function.

You then need to re-run the ‘installme’ script to install the module driver for that board type if you did not do so originally. If you want to access two or more different board types, run the script once for each board type.

## HUNT ENGINEERING Server/Loader

There is a LINUX console mode version of the HUNT ENGINEERING Server/Loader, which uses the LINUX version of the API.

Also the LINUX library version of the Server/Loader (which uses the LINUX API) can be linked against a LINUX application.

In either case this uses the LINUX API, which must be installed as described in the installation sections.

## Code Composer Studio

Code Composer Studio is the integrated compiler/linker, Real-Time Kernel, debugger, and program editor from Texas Instruments. It is very hard indeed developing DSP programs without this tool (but with stubborn determination it could be done). However, this software is only supported on Windows platforms.

To use Code Composer Studio, therefore, you would need to have an additional machine (or a dual booting machine) running a Windows version to develop the DSP application.

There are four main development tools that are supported by HUNT ENGINEERING which are described in the following sections. With a very few exceptions, these tools use the API, which provides two important benefits:

They can co-exist and rely upon the API to assist in preventing interference.

New module carriers are automatically supported without any additional work required from the tools provider

In addition to those DSP development tools the API can itself be used as a development tool for your system, as it provides an easy way to interface programs on your Host computer with programs on your DSPs.

## HUNT ENGINEERING API

For systems running under LINUX, the API provides a library file that links with your application to provide that application with all of the standard API interface. When your LINUX application actually runs, the Operating system automatically loads a Dynamic Link Library (‘libhel.so’), and it tries to call a the appropriate driver module for the board you have specified in the `HeOpen()` call.

If, later in the life of your system you choose to change the host board type, you simply change the “board type” identifier given to the `HeOpen()` function.

You then need to re-run the ‘installme’ script to install the module driver for that board type if you did not do so originally. If you want to access two or more different board types, run the script once for each board type.

## HUNT ENGINEERING Server/Loader

The HUNT ENGINEERING Server/Loader provides a means to load single sequential programs onto each processor in a system. The code can be generated using the TI C compiler/assembler. Access to the Host machine resources via a STDIO library is provided to the 'C4x that is connected directly to the host via a comport.

There is a LINUX console mode version of the HUNT ENGINEERING Server/Loader, which uses the LINUX version of the API. Also the LINUX library version of the Server/Loader (which uses the LINUX API) can be linked against a LINUX application.

In either case this uses the LINUX API, which must be installed as described in the installation sections.

## 3L ‘C4x Parallel C

The ‘C4x Parallel C compiler from 3L is based upon the TI C compiler for 'C4x, but provides a system that uses a kernel and loader program to support multiple communicating sequential processes on single or multi-processor systems.

The server provides access to the resources of the host machine via STDIO libraries.

3L software supports directly HUNT ENGINEERING Module carriers on several different operating systems, and hence it does not require the HUNT ENGINEERING

API or the Server/Loader. Please contact 3L on platform support and availability.

Refer to the relevant 3L technical note for instructions on its installation and use with the relevant Module carrier.

## GO DSP Code Composer

The 'C4x Code Composer product from GO DSP (a TI company) is an integrated development and debug environment for multiple 'C4x systems.

It requires the JTAG interface of the 'C4x, but as it can be used in conjunction with the TI C compiler/assembler, the HUNT ENGINEERING server/loader or 3L Parallel C, it could also require one or more of the host ports.

Code Composer is available only for Windows.