The High-Performance Alternative

UEIPAC Software Development Kit User Manual 2.1

February 2010 Edition

Table of contents

United Electronic Industries

The High-Performance Alternative

The High-Performance Alternative

The High-Performance Alternative

1. Introduction

The UEIPAC extends the capability of the PowerDNA and PowerDNR distributed data acquisition systems. With the UEIPAC, you can create programs that will execute directly on the PowerDNA or PowerDNR hardware. You can create standalone applications that don't require any host PC to control and monitor your hardware.

A Linux kernel replaces the standard "DAQBIOS" firmware in flash memory and uses a SD-Card as its local file system. This file system contains the other components of the operating system such as libraries, utilities, init script and daemons.

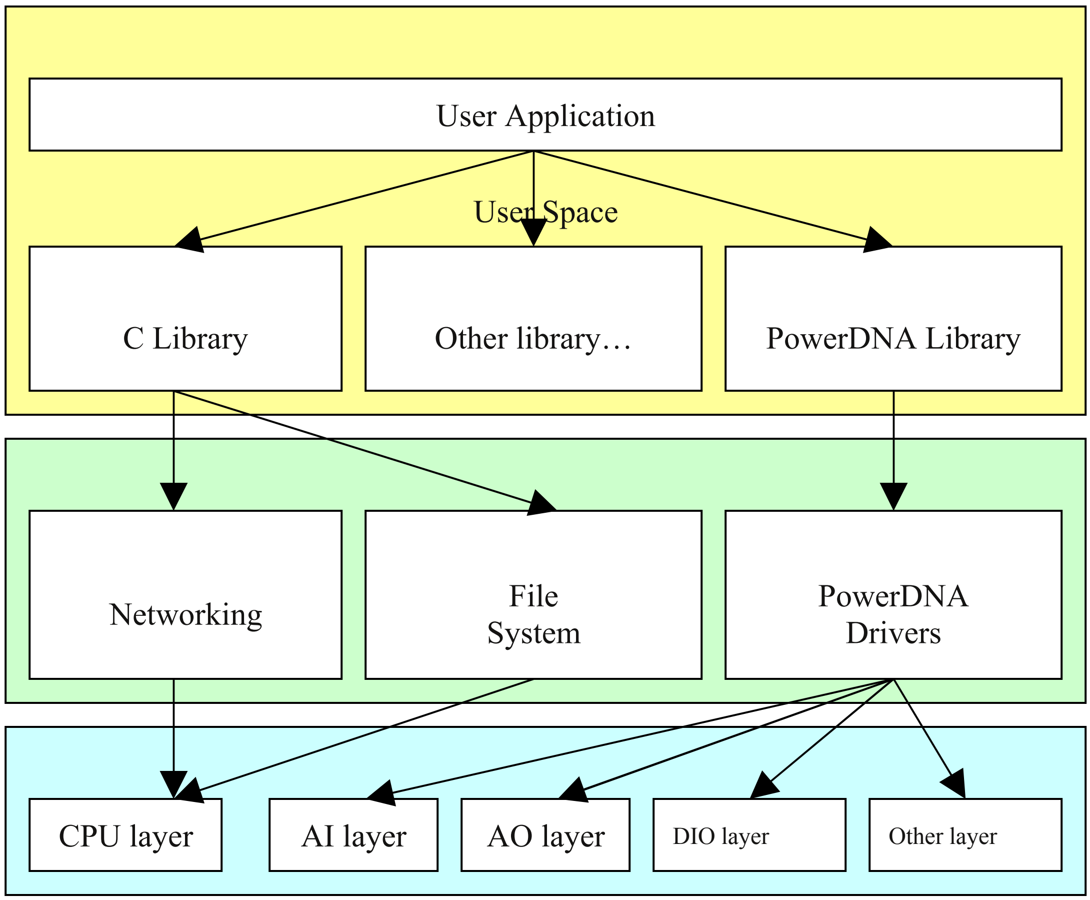
After power-up you have a ready to go Linux operating system with FTP and web servers as well as a command line shell accessible from either the serial port or telnet and SSH over the network.

You can also configure the UEIPAC to execute your application after booting-up.

Your application runs as a regular Linux process giving you access to the standard POSIX API provided by the GNU C runtime library (glibc) as well as any other library that can be compiled for Linux (for example: libxml, libaudiofile…).

The UEIPAC SDK comes with a library dedicated to communicate with the UEIPAC I/O layers.
It provides a subset of the hosted PowerDNA API; allowing you to reuse existing programs that were designed to run on a host PC and communicate with PowerDNA over the network (see section 7.4 for more informations).
You can port those programs to run directly on the UEIPAC with few modifications.

The High-Performance Alternative



2. Setting up a development system

A development system is composed of the software tools necessary to create an embedded application targeting Linux on a PowerPC processor.

The development tools can run on a Linux PC or on a Windows PC using the Cygwin environment.

It contains the following:

- GCC cross-compiler targeting the UEIPAC PPC processor.
- GNU toolchain tools such as make.
- Standard Linux libraries such as glibc.

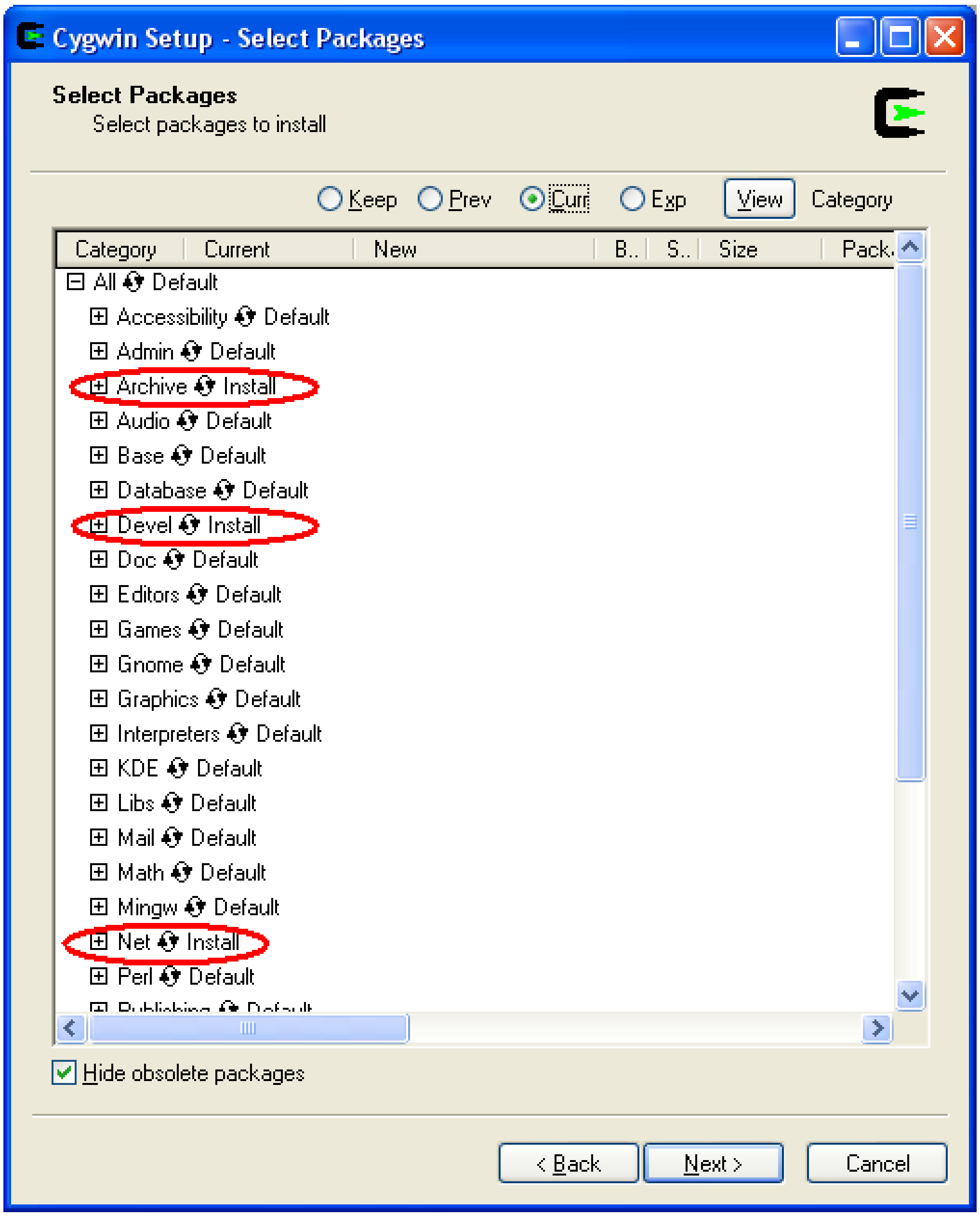
PowerDNA library to access the various PowerDNA data acquisition devices

2.1. Windows Host

If you don't have Cygwin already installed, download and run the installer "setup.exe" from http://www.cygwin.com.
Running setup.exe will install or update Cygwin. We need the packages from the following categories:

- Archive: tools to create and read archives files such as zip, bx2 and tar.
- Devel: Development tools such as make and gcc.
- Net: Network utilities such as ftp, tftp and telnet.

Click on the "Default" word next to each category you want to install. The displayed string will change to "Install".



Insert the "UEIPAC SDK" CDROM in your CD drive. Then open a cygwin command line shell.

Go to the CD's root directory (the example below assumes that the CD-ROM is the D: drive):

```
cd /cygdrive/d
```

The High-Performance Alternative

```
./install.sh
```

2.2. Linux Host

Insert the “UEIPAC SDK” CDROM in your CD drive. You might need to mount it if your Linux distribution doesn’t detect the CDROM automatically.
To mount it, type:

```
mount /dev/cdrom /mnt/cdrom
cd /mnt/cdrom
bash install.sh
```

2.3. SDK directory layout

- *powerpc-604-linux-gnu*: the GCC cross compiler
- *doc*: the manuals in PDF and HTML format
- *kernel*: the kernel source code and binary image
- *rfs.tgz*: archive containing the root file system installed on the SD card
- *uImage*: the kernel image stored in your UEIPAC flash memory
- *sdk*: the UEIPAC software development kit

The High-Performance Alternative

3. Configuring the UEIPAC

Your PowerDNA/PowerDNR hardware must be pre-configured to run Linux:

- A Linux kernel is loaded in flash memory.
- An SD card containing the root file system is inserted.

Contact UEI to convert your PowerDNA/PowerDNR hardware to a UEIPAC if it is configured with the standard "DAQBIOS" firmware.

3.1. Connecting through the serial port

Note that the serial port on the CPU layer is used as a console by default. However you can free that serial port and use it as a general purpose serial port (see section …).

Connect the serial cable to the serial port on the UEIPAC and the serial port on your PC.

You will need a serial communication program:

- Windows: ucon, MTTTY, PuTTY or HyperTerminal.
- Linux: minicom, kermit or cu (part of the uucp package).

The UEIPAC uses the serial port settings: 57600 bits/s, 8 data bits, 1 stop bit and no parity.
Run your serial terminal program and configure the serial communication settings accordingly.

Connect the DC output of the power supply (24VDC) to the "Power In" connector on the UEIPAC and connect the AC input on the power supply to an AC power source.

You should see the following message on your screen:

```
U-Boot 1.1.4 (Jan 10 2006 - 19:20:03)

CPU:   MPC5200 v1.2 at 396 MHz
       Bus 132 MHz, IPB 66 MHz, PCI 33 MHz

Board: UEI PowerDNA MPC5200 Layer
I2C:   85 kHz, ready
DRAM:  128 MB
Reserving 349k for U-Boot at: 07fa8000
FLASH:  4 MB
In:    serial
Out:   serial
Err:   serial
Net:   FEC ETHERNET
```

```
Type "run flash_nfs" to mount root filesystem over NFS

Hit any key to stop autoboot:  5
```

This message is coming from the cube's boot loader U-Boot. It waits 2 seconds to give the user a chance to alter its configuration if necessary.
After the count-down ends, U-Boot loads the Linux kernel from flash, un-compresses it, and starts it:

```
U-Boot 1.1.4 PowerDNA 3.2.1 (Dec 18 2006 - 10:41:01)

CPU:   MPC5200 v1.2 at 396 MHz
       Bus 132 MHz, IPB 66 MHz, PCI 33 MHz

Board: UEI PowerDNA MPC5200 Layer
I2C:   85 kHz, ready
DRAM:  . . ...........128 MB
FLASH:  4 MB
In:    serial
Out:   serial
Err:   serial
Net:   FEC ETHERNET

Type "run flash_nfs" to mount root filesystem over NFS

Hit any key to stop autoboot:  0
## Booting image at ffd80000 ...
   Image Name:   Linux-2.6.28.5-ueipac5200
   Created:      2009-05-01  14:31:47 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    1442840 Bytes =  1.4 MB
   Load Address: 00400000
   Entry Point:  004005e0
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
Using ueipac5200 machine description
Linux version 2.6.28.5-ueipac5200 (frederic@frederic-ubuntu64) (gcc
version 4.0.2) #1 PREEMPT Fri May 1 10:31:32 EDT 2009
Zone PFN ranges:
  DMA      0x00000000 -> 0x00008000
  Normal   0x00008000 -> 0x00008000
  HighMem  0x00008000 -> 0x00008000
Movable zone start PFN for each node
early_node_map[1] active PFN ranges
    0: 0x00000000 -> 0x00008000
Built 1 zonelists in Zone order, mobility grouping on.  Total pages:
32512
```

The High-Performance Alternative

```
Kernel command line: console=ttyPSC0,57600 root=62:1 rw
MPC52xx PIC is up and running!
PID hash table entries: 512 (order: 9, 2048 bytes)
clocksource: timebase mult[79364d9] shift[22] registered
I-pipe 2.4-04: pipeline enabled.
Console: colour dummy device 80x25
console [ttyPSC0] enabled
Dentry cache hash table entries: 16384 (order: 4, 65536 bytes)
Inode-cache hash table entries: 8192 (order: 3, 32768 bytes)
Memory: 126376k/131072k available (2808k kernel code, 4548k reserved,
116k data, 436k bss, 152k init)
Calibrating delay loop... 65.53 BogoMIPS (lpj=32768)
Mount-cache hash table entries: 512
net_namespace: 292 bytes
NET: Registered protocol family 16
DMA: MPC52xx BestComm driver
DMA: MPC52xx BestComm engine @f0001200 ok !
NET: Registered protocol family 2
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 4096 (order: 3, 32768 bytes)
TCP bind hash table entries: 4096 (order: 2, 16384 bytes)
TCP: Hash tables configured (established 4096 bind 4096)
TCP reno registered
NET: Registered protocol family 1
audit: initializing netlink socket (disabled)
type=2000 audit(0.208:1): initialized
I-pipe: Domain Xenomai registered.
Xenomai: hal/powerpc started.
Xenomai: real-time nucleus v2.4.7 (Andalusia) loaded.
Xenomai: starting native API services.
Xenomai: starting POSIX services.
Xenomai: starting RTDM services.
VFS: Disk quotas dquot_6.5.1
Dquot-cache hash table entries: 1024 (order 0, 4096 bytes)
msgmni has been set to 247
io scheduler noop registered
io scheduler anticipatory registered (default)
io scheduler deadline registered
io scheduler cfq registered
Generic RTC Driver v1.07
Serial: MPC52xx PSC UART driver
f0002000.serial: ttyPSC0 at MMIO 0xf0002000 (irq = 129) is a MPC52xx
PSC
brd: module loaded
loop: module loaded
net eth0: Fixed speed MII link: 100FD
MPC52xx SPI interface probed at 0xf0000f00, irq0=141, irq1=142
mpc52xx_spi_init_mmc: SDCard is now ready
mpc52xx_mmc0: p1
mice: PS/2 mouse device common for all mice
```

The High-Performance Alternative

```
TCP cubic registered
NET: Registered protocol family 17
EXT2-fs warning: mounting unchecked fs, running e2fsck is recommended
VFS: Mounted root (ext2 filesystem).
Freeing unused kernel memory: 152k init
init started: BusyBox v1.13.3 (2009-04-13 15:41:06 EDT)
loading modules
    pdnabus
    pdnadev
Starting Network...
Checking Network Configuration:                           [  OK  ]
Loading Static Network Interface:                         [  OK  ]
Checking Network Connection:                              [  OK  ]
Starting inetd...                                         [  OK  ]
Starting local script...
PowerDNA Driver, version 2.1.0

Address     Irq  Model Option  Phy/Virt  S/N     Pri  LogicVer
--------------------------------------------------------------
0xc9080000   7    201   100    phys    0027153     0  02.09.03
0xc9090000   7    308     1    phys    0028647     0  02.0e.00
0xc90a0000   7    207     1    phys    0030353     0  02.0c.05
0xc90b0000   7    205     1    phys    0023120     0  02.09.03
0xc90c0000   7    403     1    phys    0034744     0  02.0e.00
0xc90d0000   7    503     1    phys    0025808     0  02.09.03
--------------------------------------------------------------
                                                          [  OK  ]


BusyBox v1.13.3 (2009-04-29 09:50:58 EDT) built-in shell (ash)
Enter 'help' for a list of built-in commands.

~ #
```

You can now navigate the file system and enter standard Linux commands such as ls, ps, cd…

3.2. Root file system

Booting from the SD card

The UEIPAC ships with the root file system entirely located on the SD card. It uses the EXT2 format.

It is recommended to type the command "halt" before powering down the UEIPAC and the command "reboot" to restart the UEIPAC.

The High-Performance Alternative

If you power down abruptly the UEIPAC, the following message will appear at boot time:

```
EXT2-fs warning: mounting unchecked fs, running e2fsck is recommended
```

You must check the file system for errors with the following commands:

```
# mount -o remount,ro /
# e2fsck /dev/sdcard1
e2fsck 1.38 (30-Jun-2005)
/dev/sdcard: clean, 702/124160 files, 6632/247872 blocks
# reboot
```

Booting from an NFS share
It is also possible to use an NFS network share to hold the root file system instead of the SD card.

Refer to appendix D for instructions.

3.3. Configuring the Network

Configuring a static IP address
Your UEIPAC is configured at the factory with the static IP address 192.168.100.2 to be part of a private network.

You can change the IP address using the following command:

```
setip <new IP address>
```

The IP address change takes effect immediately and is stored in the configuration file /etc/network.conf

Configuring dynamic IP address (using a DHCP server)
If you have DHCP server available, you can configure the UEIPAC to automatically fetch an IP address when it boots up.
Edit the file /etc/network.conf and change the line:

```
DHCP=no
```

To:

```
DHCP=yes
```

You must restart the network to activate the change:

```
/etc/init.d/network restart
```

Name resolution
If your UEIPAC uses a static address, you need to edit the file /etc/resolv.conf to add the IP address of your DNS server.

If your UEIPAC uses DHCP, the /etc/resolv.conf file is automatically populated and name resolution will work right away.

Connecting through Telnet
Once the IP address is configured, you shouldn't need the serial port anymore. You can use telnet to access the exact same command line interface.

Type the following command on your host PC, then login as "root". The password is "root".

```
telnet <UEIPAC IP address>
```

Type the command "exit" to logout.

Connecting through SSH
Type the following command on your host PC. The password is "root".

```
ssh root@<UEIPAC IP address>
```

Type the command "exit" to logout.

3.4. Configuring Date and Time

Changing the date
The UEIPAC is equipped with a real-time clock chip that preserves the date and time settings when the UEIPAC is not powered.
By default, the date is set to the current data and time in the UTC (GMT) time zone.

To print the current date and time, use the following command:

```
date
```

To change the current date and time use:

```
date MMDDhhmm
```

For example "date 06021405" will set the new date to June second, 2:05 PM.

To make this change permanent upon reboot, save the date to the RTC chip with the following command:

```
hwclock -w
```

Changing the time zone
To set the time zone you need to set the environment variable TZ.

For example if you type the command:
```
export TZ=EST5EDT,M3.2.0,M11.1.0
```

It will set the time zone to eastern time with daylight saving time starting on the Sunday(0) of the second week(2) of March(3) and ending on Sunday(0) of the first week(1) of November(11).

To make this change permanent upon reboot, add the command to the file /etc/profile

You can find a detailed explanation on the syntax of TZ at:
http://www.gnu.org/software/libtool/manual/libc/TZ-Variable.html

Connecting to a NTP server
The "rdate" utility can be used to retrieve the time from a NTP server.

The following command just prints the time returned by the NTP server:
```
rdate -p <NTP server IP address>
```

The following command changes the current date and time to match the ones returned by the NTP server:
```
rdate -s <NTP server IP address>
```

To make this change permanent upon reboot, save the date to the RTC chip with the following command:
```
hwclock -w
```

3.5. Changing the password

Type the following command and enter your new password two times:
```
passwd
```

You can now logout and login with your new password.

3.6. Configuring the web server

The UEIPAC comes with a simple web server enabled. Copy your html pages in the folder **/www** to make them accessible from a remote web browser.

The High-Performance Alternative

4. Transferring files

You can use either NFS, FTP, SSH or TFTP to transfer files between your host PC and the UEIPAC.

4.1. NFS

If you have a NFS server running on your development machine, you can mount a shared directory on the UEIPAC. This will make the shared directory available on the UEIPAC local file system.

To mount a shared directory (for example /shared located on host at 192.168.100.1 mounted on /mnt):

```
mount -o nolock -t nfs 192.168.100.1:/shared /mnt/nfs_share
```

After typing this command, all files present in the host PC directory /shared will also be accessible on the UEIPAC's /mnt/nfs_share directory.

4.2. FTP Client

To connect to an external FTP server from the UEIPAC, use the commands "ftpput" and "ftpget".

To retrieve a file from an FTP server:

```
ftpget –u <username> -p <password> <FTP server IP address> <local
file name> <remote file name>
```

To send a file to an FTP server:

```
ftpput –u <username> -p <password> <FTP server IP address> <remote
file name> <local file name>
```

4.3. FTP Server

The UEIPAC comes with the vsftpd FTP server. The server is active by default.

You can login as "root" with password "root". You get read and write access to the entire file system.

4.4. SSH

The UEIPAC also comes with the SSH server "dropbear" preinstalled.

Use the command scp to transfer a file between your PC and the UEIPAC.
To send a file to the UEIPAC:

```
scp <source file path on PC> root@192.168.100.2:<destination path on
UEIPAC>
```

To receive a file from the UEIPAC:

```
scp root@192.168.100.2:<source file path on UEIPAC> <destination path
on PC>
```

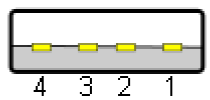
4.5. TFTP Client

To retrieve a file from a TFTP server, use the following command:

```
tftp -g -r <remote file name> <TFTP server IP address>
```

5. Connecting USB devices

You can only connect USB devices to PowerDNA cubes or PowerDNR racks equipped with a USB type A connector.



Type A

The Linux kernel supports most USB devices but the UEIPAC only comes with drivers for USB mass storage devices to save space on the SD card.

Please contact UEI if you plan to use any other USB device.

5.1. USB Mass Storage

USB mass storage devices use multiple form factors. It goes from the smallest USB flash drive to enclosures used to connect ATA or SATA hard-drives.

The UEIPAC supports all of those devices as long as they comply with the USB mass storage device class and are formatted with one of the following formats: FAT, EXT2.

After connecting a mass storage device to the UEIPAC, the following kernel messages will appear on the serial console (if you are connected using telnet or SSH, use the command “dmesg” to view kernel messages):

```
usb 1-1: new high speed USB device using fsl-ehci and address 2
usb 1-1: configuration #1 chosen from 1 choice
scsi0 : SCSI emulation for USB Mass Storage devices
usb 1-1: New USB device found, idVendor=08ec, idProduct=0011
usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 1-1: Product: USB Drive
usb 1-1: Manufacturer: Fujifilm
usb 1-1: SerialNumber: 0713B317290025CC
scsi 0:0:0:0: Direct-Access     Fujifilm USB Drive        4.20 PQ: 0
ANSI: 0 CCS
sd 0:0:0:0: [sda] 499712 512-byte hardware sectors: (255 MB/244 MiB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through
sd 0:0:0:0: [sda] 499712 512-byte hardware sectors: (255 MB/244 MiB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through
 sda: sda1
sd 0:0:0:0: [sda] Attached SCSI removable disk
```

Note the device node name assigned to this USB device, it uses the format “sdxn”:

- x is **a** for the first drive, **b** for the second and so on.
- n is the partition number

In the kernel message above, we see that the USB mass storage device’s first partition is using the device node **sda1**

You can mount the file system located on this device with the command:

```
mount /dev/sda1 /mnt
```

The files are now accessible under the directory **/mnt**

You must un-mount the file system before un-plugging the device to avoid file corruption:

```
umount /mnt
```

5.2. Wifi network interface

The UEIPAC comes with drivers for Wifi network usb interfaces that use the following chipsets:

- Realtek RTL8187
- Ralink RT2570, RT2571

Load kernel modules

At the command line prompt type one of the following commands depending on your wifi chipset:

```
modprobe rtl8187
modprobe rt200xusb
modprobe rt2500usb
modprobe rt73usb
```

Wifi network interface are names **wlan0**, **wlan1** etc…

The **iwconfig** utility is used to configure wifi communication parameters.

You can verify that your interface was properlt detected by typing the command **iwconfig**. A new entry **wlan0** should appear:

```
lo        no wireless extensions.

eth0      no wireless extensions.
```

United Electronic Industries

The High-Performance Alternative

```
eth1      no wireless extensions.

wmaster0  no wireless extensions.

wlan0     IEEE 802.11bg  ESSID:""
          Mode:Managed  Frequency:2.412 GHz  Access Point: Not-
Associated
          Tx-Power=0 dBm
          Retry min limit:7   RTS thr:off   Fragment thr=2352 B
          Encryption key:off
          Power Management:off
          Link Quality:0  Signal level:0  Noise level:0
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

Connection to an open access point

Specify that you want to connect as a client to a network with an access point:

```
iwconfig wlan0 mode managed
```

Set the ESSID of the access point:

```
iwconfig wlan0 essid <name of your access point>
```

Bring up wifi interface:

```
ifconfig wlan0 up
```

You can now scan the access points accessible by your wifi interface:

```
iwlist wlan0 scan
```

If there is a DHCP server on your network, get an IP address for your wifi interface:

```
udhcpc -i wlan0 -s /etd/udhcpc/default.script
```

Otherwise, assign a static IP address to your wifi interface:

```
ifconfig wlan0 192.168.100.3 netmask 255.255.255.0
route add default gateway 192.168.100.1
```

Connection to an access point with WEP security

The procedure is almost identical to connecting to an open access point. In addition you need to specify your WEP key:

```
iwconfig wlan0 key <WEP key in hexadecimal>
```

128 bit WEP use 26 hex characters, 64 bit WEP uses 10

The High-Performance Alternative

Connection to an access point with WPA/WPA2 security

Generate the pre-shared key using the access point's password

```
wpa_passphrase <name of your access point> <access point password>
```

Edit the file **/etc/wpa_supplicant.conf** and update the **ssid** and **psk** entry to match the key generated by **wpa_passphrase**

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
ap_scan=1

network={
     ssid=<put your access point ESSID here>
     proto=WPA
     key_mgmt=WPA-PSK
     pairwise=TKIP
     group=TKIP
     psk=<put your pre-shared key generated with wpa_passphrase here>
     priority=2
}
```

Specify that you want to connect as a client to a network with an access point in managed mode:

```
iwconfig wlan0 essid <name of your access point> mode managed
```

Run wpa_supplicant in daemon mode to authenticate with the access point:

```
wpa_supplicant -iwlan0 -c/etc/wpa_supplicant.conf -Dwext -B
```

Run **iwconfig** to verify that the authentication worked:

```
wlan0     IEEE 802.11bg  ESSID:"fred"
          Mode:Managed  Frequency:2.447 GHz  Access Point:
00:13:10:AA:FA:10
          Bit Rate=1 Mb/s   Tx-Power=27 dBm
          Retry min limit:7   RTS thr:off   Fragment thr=2352 B
          Encryption key:B507-40C4-9A48-806D-D664-910F-B354-6CF4-
DEBF-EA54-CE6F-B291-BD0E-593F-BFA9-405D [2]   Security mode:open
          Power Management:off
          Link Quality=80/100  Signal level:-31 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

If there is a DHCP server on your network, get an IP address for your wifi interface:

```
udhcpc -i wlan0 -s /etd/udhcpc/default.script
```

Otherwise, assign a static IP address to your wifi interface:

```
ifconfig wlan0 192.168.100.3 netmask 255.255.255.0
route add default gateway 192.168.100.1
```

The High-Performance Alternative

Direct connection to another computer in ad-hoc mode

Specify that you want to connect in ad-hoc mode:
```
iwconfig wlan0 mode ad-hoc
```

Set the ESSID of the access point:
```
iwconfig wlan0 essid <name of your access point>
```

Bring up wifi interface:
```
ifconfig wlan0 up
```

If there is a DHCP server on your network, get an IP address for your wifi interface:
```
udhcpc -i wlan0 -s /etd/udhcpc/default.script
```

Otherwise, assign a static IP address to your wifi interface:
```
ifconfig wlan0 192.168.100.3 netmask 255.255.255.0
route add default gateway 192.168.100.1
```

5.3. UMTS/GSM modem

The UEIPAC comes with drivers for Sierra Wireless modems.

The UEIPAC supports USB modems connected to the UEIPAC USB port and embedded mini pci express modems connected to a CAR-550 carrier card.

This manual focuses on using a Sierra wireless MC8790 card that offers UMTS/HSPA and quad-band GSM/GPRS/EDGE network access for roaming on high-speed networks worldwide.

Prerequisite

You need to purchase a data plan with a cell phone provider that supports UMTS and/or GSM/GPRS.
ATT and T-Mobile provide such a service in the USA.
Once you purchased a data plan you will receive a SIM card that you need to insert in the CAR-550 before being able to establish a connection.
Don't forget to activate your account as soon as you receive your SIM card (usually done over the phone or on-line).

Manual configuration

From the UEIPAC point of view, the wireless modem is seen as a serial port to which it can send Hayes AT commands as if it were an old fashion RTC modem.
UEIPAC uses the PPP software to control the modem and configure a network connection with your phone provider.

Load kernel modules

At the command line prompt type the following commands:

```
modprobe sierra
modprobe ppp
```

You should see the following messages printed on the console:

```
~ # modprobe sierra
usbcore: registered new interface driver usbserial
usbserial: USB Serial Driver core
USB Serial support registered for Sierra USB modem
sierra 1-1:1.0: Sierra USB modem converter detected
usb 1-1: Sierra USB modem converter now attached to ttyUSB0
sierra 1-1:1.1: Sierra USB modem converter detected
usb 1-1: Sierra USB modem converter now attached to ttyUSB1
sierra 1-1:1.2: Sierra USB modem converter detected
usb 1-1: Sierra USB modem converter now attached to ttyUSB2
sierra 1-1:1.3: Sierra USB modem converter detected
usb 1-1: Sierra USB modem converter now attached to ttyUSB3
sierra 1-1:1.4: Sierra USB modem converter detected
usb 1-1: Sierra USB modem converter now attached to ttyUSB4
sierra 1-1:1.5: Sierra USB modem converter detected
usb 1-1: Sierra USB modem converter now attached to ttyUSB5
sierra 1-1:1.6: Sierra USB modem converter detected
usb 1-1: Sierra USB modem converter now attached to ttyUSB6
usbcore: registered new interface driver sierra
sierra: v.1.3.2:USB Driver for Sierra Wireless USB modems
~ # modprobe ppp
PPP generic driver version 2.4.2
```

Configure provider

The system is pre-configured to connect to ATT network. If you are using a different provider, edit the file /etc/ppp/peers/gsm_chat
Look for the following line:

```
OK      'AT+CGDCONT=1,"IP","ISP.CINGULAR"'
```

Replace it with the APN (Access point name) of you provider.
For example T-mobile's APN is “epc.tmobile.com”, so the line in /etc/ppp/peers/gsm_chat becomes:

```
OK      'AT+CGDCONT=1,"IP","EPC.TMOBILE.COM"'
```

The High-Performance Alternative

Start PPP daemon

Issue the following command to start the PPP daemon and configure the network connection.

```
/etc/init.d/pppd start
```

After a few seconds, the script will return printing the message "[OK]" if it successfully configured the network connection or "[Failed]" if it did not.

```
~ # /etc/init.d/pppd start
Starting pppd...PPP BSD Compression module registered
PPP Deflate Compression module registered    [  OK  ]
```

In case of failure, type the command "dmesg" to print the log and send that information to UEI technical support.
Type the command "ifconfig" to print the network connections currently configured on your UEIPAC. There should be three connections: local, eth0 and ppp0.

```
eth0       Link encap:Ethernet  HWaddr 00:0C:94:00:C5:CB
           inet addr:192.168.100.2  Bcast:192.168.100.255
Mask:255.255.255.0
           UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
           RX packets:0 errors:0 dropped:0 overruns:0 frame:0
           TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:1000
           RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
           Base address:0x4000

lo         Link encap:Local Loopback
           inet addr:127.0.0.1  Mask:255.0.0.0
           UP LOOPBACK RUNNING  MTU:16436  Metric:1
           RX packets:0 errors:0 dropped:0 overruns:0 frame:0
           TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:0
           RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

ppp0       Link encap:Point-to-Point Protocol
           inet addr:166.203.211.199  P-t-P:10.64.64.64
Mask:255.255.255.255
           UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
           RX packets:14 errors:0 dropped:0 overruns:0 frame:0
           TX packets:15 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:3
           RX bytes:182 (182.0 B)  TX bytes:257 (257.0 B)
```

Make sure that ppp0 was assigned an IP address.

You can now connect to the internet from your UEIPAC.

The High-Performance Alternative

Automatic startup

To automatically load the kernel modules, edit the file /etc/modules and add the following lines at the end of the file:

```
sierra
ppp
```

To automatically start the ppp daemon, add a symbolic link to /etc/init.d/pppd in the directory /etc/rc.d with the following command:

```
ln -s /etc/init.d/pppd /etc/rc.d/S30pppd
```

5.4. Serial Port

The UEIPAC comes with driver for USB-serial devices based on the Prolific PL-2303 chipset.

Load kernel modules

At the command line prompt type the following:

```
modprobe pl2303
```

You will see the following messages printed on the serial console (type dmesg to see those messages when logged in via telnet or ssh):

```
usbcore: registered new interface driver usbserial
USB Serial support registered for generic
usbcore: registered new interface driver usbserial_generic
usbserial: USB Serial Driver core
USB Serial support registered for pl2303
pl2303 1-5.1:1.0: pl2303 converter detected
usb 1-5.1: pl2303 converter now attached to ttyUSB0
usbcore: registered new interface driver pl2303
pl2303: Prolific PL2303 USB to serial adaptor driver
```

Make note of the device node attached to the serial port. In the example above it is **/dev/ttyUSB0.**

You will use this device node to address the serial port. See example **SampleLinuxSerialPort** for an example showing how to program standard Linux serial port.

Automatic startup

The High-Performance Alternative

To automatically load the kernel modules, edit the file /etc/modules and add the following lines at the end of the file:

```
pl2303
```

5.5. LibUSB

The UEIPAC comes with the LibUSB library to facilitate programming of USB devices for which there is no driver.

It allows the enumeration of USB devices as well as access to USB communication pipes:

- control transfers which are typically used for command or status operations
- interrupt transfers which are initiated by a device to request some action from the host
- isochronous transfers which are used to carry data the delivery of which is time critical (such as for video and speech)
- bulk transfers which can use all available bandwidth but are not time critical.

Prerequisite

LibUSB uses usbfs whichis a filesystem specifically designed for USB devices. Once this filesystem is mounted it can be found at **/proc/bus/usb/**. It consists of information about all the USB devices that are connected to the computer.
LibUSB makes use of this filesystem to interact with the USB devices.

Mount USBFS manually

Type the following command to mount USBFS:

```
mount -t usbdevfs none /proc/bus/usb
```

Mount USBFS automatically

Add the following line to **/etc/fstab** to automatically mount USBFS at boot time:

```
none  /proc/bus/usb  usbfs  defaults  0  0
```

Write a program using libusb

The UEIPAC ships with a simple example showing how to enumerate USB devices and query information: **SampleLibUSB**
LibUSB API documentation is available at http://www.libusb.org

The High-Performance Alternative

6. Using the serial port for general purpose

Edit the file /etc/inittab and add the character '#' in front of the line:
ttyS0::respawn"-/bin/sh

Then reboot.

This will disable the serial console and let you control the serial port from your program using the POSIX termios API.

The High-Performance Alternative

7. Testing the I/O layers

7.1. devtbl

Run the command "devtbl", it will print a list of the I/O layers that were detected on this module.

```
PowerDNA Driver, version 2.1.0

Address     Irq  Model Option  Phy/Virt  S/N     Pri  LogicVer
--------------------------------------------------------------
0xc9080000   7    207    1     phys    0027887    0   02.0c.05
0xc9090000   7    403    1     phys    0030384    0   02.0c.05
0xc90a0000   7    403    1     phys    0030385    0   02.0c.05
0xc90b0000   7    501    1     phys    0029693    0   02.0c.05
0xc90c0000   7    601    1     phys    0030279    0   02.0c.05
--------------------------------------------------------------
~ #
```

7.2. Run examples

All the examples were compiled during the install process and are ready to be transferred and executed.
Compiled versions of each example are also available on the UEIPAC file system in the "/usr/local/examples" directory.
There is one example for each supported I/O layer named "SampleXXX" (where XXX is the model ID of each layer).
Go to the directory "<UEIPAC SDK directory>/sdk/DAQLib_Samples" and copy the chosen example to your UEIPAC using one of the methods described in section 4.

For example using FTP:

```
ftp <UEIPAC IP address>
bin
cd tmp
put SampleXXX
```

The example by default uses the first I/O layer (device 0). You can change the device using command line options. Here are a few of the options available:

```
-h : display help
-d n: selects the device to use (default: 0)
-f n.nn : set the rate of the DAQ operation (default: 1000 Hz)
-c "x,y,z,..." : select the channels to use (default: channel 0)
```

For example the following command run the AI-207 test program using device 2 and channels 3,5,and 7:

```
/tmp # ./Sample207 -d 2 -c "3,5,7"
```

The High-Performance Alternative

```
There are 3 channels specified: 3 5 7
 0: ch3 bdata 310dfff6 fdata 15.781501V
 0: ch5 bdata 310dfff7 fdata 15.781501V
 0: ch7 bdata 310dfff6 fdata 15.781501V

 1: ch3 bdata 310dfff6 fdata 15.781501V
 1: ch5 bdata 310dfff6 fdata 15.781501V
 1: ch7 bdata 310dfff6 fdata 15.781501V
...
```

All examples are configured to stop when they receive the SIGINT signal. You can send this signal by typing CTRL+C or with the following command if the program runs in the background of if you are logged on a different console than the one running the program:

```
killall -SIGINT Sample207
```

7.3. PowerDNA server

PowerDNA server emulates the behavior of a PowerDNA IO module running the standard DAQBIOS firmware. It emulates a subset of the DAQBIOS protocol so that the UEIPAC can be accessed from PowerDNA explorer or the UeiDaq framework in immediate mode. ACB and DMAP modes are not supported in the PowerDNA server.

To run the PowerDNA server, type the command “pdnaserver &”.

The High-Performance Alternative

8. Application development

8.1. Prerequisites

Make sure that the directory “<UEIPAC SDK directory>/powerpc-604-linux-gnu/bin” is added to your PATH environment variable. This will allow you to invoke the GCC cross compiler without having to specify its full path.
It is required to run the different Makefiles that build the PowerDNA library and the examples (this should have been done automatically by the install script).

8.2. Compiling and running Hello World

The UEIPAC SDK comes with the GNU toolchain compiled to run on your host PC and build binaries targeting the PowerPC processor that runs on your UEIPAC.
The SDK comes with all the familiar GNU tools: ar, as, gcc, ld, objdump… To avoid confusion with a different version of those tools (for example a version compiled to run and produce binaries for your host PC) , their names are prefixed with “powerpc-604-linux-gnu-“. For example the GNU C compiler is named “powerpc-604-linux-gnu”.

The following steps will guide you in writing your first program and running it on your UEIPAC.

1. Create a file called hello.c
2. Edit the file and enter the following text:

```
#include<stdio.h>

int main(int argc, char* argv[])
{
    printf("Hello World from UEIPAC\n");
    return 0;
}
```

3. Compile the file with the command:
   `powerpc-604-linux-gnu-gcc hello.c –o hello`
4. Download the compiled program “hello” to the cube:

```
ftp <UEIPAC IP address>
bin
cd tmp
put hello
```

The High-Performance Alternative

5. Login on your UEIPAC using either Telnet or the serial console and type the following commands:
```
cd /tmp
chmod +x hello
./hello
```

You should see the text “Hello World from UEIPAC” printed on the console.

8.3. Debugging Hello World

The UEIPAC SDK contains a version of the GNU debugger compiled to run on your host PC and debug binaries targeting the PowerPC processor. Its name is “powerpc-604-linux-gnu-gdb”.
It allows you to debug a program remotely from your host PC.

The following steps will guide you in debugging the “hello world” program.

1. Rebuild the hello program using the –g option. This will include debug symbols in the binary file.
```
powerpc-604-linux-gnu-gcc -g hello.c -o hello
```
2. Upload the new binary to the UEIPAC using FTP.
3. On the UEIPAC console, start the GDB server to debug the program remotely (It will communicate with the host on port 1234):
```
gdbserver :1234 hello
```
4. On the host, start GDB and connect to the target
```
powerpc-604-linux-gnu-gdb hello
target remote <UEIPAC IP address>:1234
```
5. Set the shared library search path so that GDB will find the proper library used by your program:
```
set solib-search-path <UEIPAC SDK Dir>/powerpc-604-linux-gnu/powerpc-604-linux-gnu/lib:<UEIPAC SDK Dir>/sdk/DAQLib
```

   Note that this step is only necessary if you wish to step inside the code of the shared libraries. If you don’t set this variable, GDB will print a few error messages about library mismatch but you can still go ahead and debug your program.
6. The program is now in “running” state and GDB paused its execution. Let’s put a breakpoint at the beginning of the “main” function:
```
break main
```

The High-Performance Alternative

7. We can now resume execution with the "cont" command and GDB will pause the execution again when entering the "main" function.
8. You can step in your program using the "n" command to step over each line of execution and "s" to step inside any called functions.

To avoid typing the same commands over and over when starting a debugging session, you can create a file named ".gdbinit" in your home directory. This file will contain commands that you want GDB to execute at the beginning of a session.

For example the following ".gdbinit" file automatically connect to the target and pauses the execution in the main function each time you start gdb:

```
set solib-search-path <UEIPAC Driver Dir>powerpc-604-linux-
gnu/powerpc-604-linux-gnu/lib:<UEIPAC Driver Dir>/sdk/DAQLib
target remote 192.168.100.2:1234
break main
cont
```

Read the GDB documentation at http://sourceware.org/gdb/documentation/ to learn how to fully use the GDB debugger.

8.4. PowerDNA Library

The PowerDNA library implements the API used to program the PowerDNA IO layers:

The following layers are supported by the UEIPAC SDK:

- Analog Input: AI-201, AI-205, AI-207, AI-208, AI-211, AI-225, AI-254, AI-255
- Analog Output: AO-302, AO-308, AO-332
- Digital Input/Output: DIO-401, DIO-402, DIO-403, DIO-404, DIO-405, DIO-406, DIO-416, DIO-432, DIO-433, DIO-448
- Counter/Timer: CT-601, QUAD-604
- Messaging: SL-501, SL-508, CAN-503, 429-566, 429-512

The source code is installed in "<UEIPAC SDK directory>/sdk/DAQLib".
Examples are located in "<UEIPAC SDK directory>/sdk/DAQLib_Samples".

The UEIPAC SDK uses a subset of the PowerDNA Software Suite API. It even allows you to control other IO modules that run the standard DAQBios firmware from the UEIPAC the same way you would from a host PC running Windows or Linux.

The PowerDNA API uses the IP address specified in the function DqOpenIOM() to determine whether you wish to access the layers local to the UEIPAC or "remote" layers installed in a remote PowerDNA IO module. Set the IP address to the loopback address "127.0.0.1" and the API will know that you want to access the "local" layers.

The PowerDNA API implements various modes to communicate with the I/O layers:

- Immediate: It is the easiest mode for point by point input/output on all layers. It also is the least efficient because it requires one call for each incoming and/or outgoing request. You can not achieve maximum performances with that mode
  Immediate mode examples are named "SampleXXX"
- Data Mapping (DMAP): This is the most efficient mode for point by point input/output on AI, AO, DIO and CT layers. Incoming and outgoing data from/to multiple layers are all packed in a single call.
  DMAP mode examples are named "SampleDMapXXX"
- Buffered (ACB): Allows access to AI, AO, DIO and CT layers at full speed.
  It is designed to correct communication errors that might happen on the network link.The error correction mechanisn will cause issues with real-time deadlines
  ACB mode examples are named "SampleACBXXX"
- Messaging: Allows access to messaging layers (serial, CAN, ARINC-429) at full speed. It is designed to correct communication errors that might happen on the network link.The error correction mechanisn will cause issues with real-time deadlines
  Messaging mode examples are named "SampleMsgXXX"
- Variable Size Data Mapping (VMAP): Allows access to all layers at full speed, transferring incoming and outgoing data in buffers in one call.
  VMAP mode examples are named "SampleVMapXXX"

The UEIPAC SDK only supports the immediate (also known as "point by point") DMAP and VMAP modes to control the "local" layers.
The three other modes (ACB, MSG and M3) are designed to work over ethernet and have built-in error correction which is not needed on the UEIPAC. You can, however use those modes to control "remote" layers installed in I/O modules that runs the DAQBios firmware over the network.

	Firmware running on the IO module		
I/O mode	DAQBios	UEIPAC/local layers	UEIPAC/remote layers
Immediate	Yes	Yes	Yes
ACB	Yes	No	Yes

DMAP	Yes	Yes	Yes
MSG	Yes	No	Yes
VMAP	Yes	Yes	Yes

PowerDNA API

The following section details the subset of PowerDNA APIs available when running your program on a UEIPAC.

Refer to the "PowerDNA API Reference Manual" document to get detailed information about each API.

Initialization, miscellaneous API

Those APIs are used to initialize the library, obtain a handle on the kernel driver and perform miscellaneous tasks such as translating error code to readable messages.

- DqInitDAQLib
- DqCleanUpDAQLib
- DqOpenIOM
- DqCloseIOM
- DqTranslateError
- All DqCmd*** APIs

Immediate mode API

Those APIs are used to read/write I/O layers in a software-timed fashion. They are designed to provide an easy way to access I/O layers at a non-deterministic pace.

Each I/O layer comes with its own set of immediate mode APIs. For example you will use the DqAdv201*** APIs to control an AI-201.

All DqAdvXYZ*** APIs where XYZ is the model number of a supported I/O layer are supported on the UEIPAC.

DMAP API

In DMAP mode, the UEIPAC continuously refreshes a set of channels that can span multiple layers at a specified rate paced by a hardware clock.
Values read from or written to each configured channel are stored in an area of memory called the DMAP. At each clock tick, the firmware synchronizes the DMAP values with their associated physical channels.

Supported APIs to use RTDMAP mode are DqRtDmap***.

Here is a quick tutorial on using the RTDMAP API (handling of error codes is omitted):

Initialize the DMAP to refresh at 1000 Hz:
```
DqRtDmapInit(handle, &dmapid,1000.0);
```

Add channel 0 from the first input subsystem of device 1:
```
chentry = 0;
DqRtDmapAddChannel(handle, dmapid, 1, DQ_SS0IN, &chentry, 1);
```

Add channel 1 from the first output subsystem of device 3:
```
chentry = 1;
DqRtDmapAddChannel(handle, dmapid, 3, DQ_SS0OUT, &chentry, 1);
```

Start all devices that have channels configured in the DMAP:
```
DqRtDmapStart(handle, dmapid);
```

Update the value(s) to output to device 3:
```
outdata[0] = 5.0;
DqRtDmapWriteScaledData(handle, dmapid, 3, outdata, 1);
```

Synchronize the DMAP with all devices:
```
DqRtDmapRefresh(handle, dmapid);
```

Retrieve the data acquired by device 1:
```
DqRtDmapReadScaledData(handle, dmapid, 1, indata, 1);
```

Stop the devices and free all resources:
```
DqRtDmapStop(handle, dmapid);
DqRtDmapClose(handle, dmapid);
```

Refer to Appendix A for detailed documentation of each RTDMAP function.

VMAP API

In VMAP mode, the UEIPAC continuously acquires/updates data in buffers.
Each layer is programmed to acquire/update data to/from its internal FIFO at a rate paced by its hardware clock.
The content of all the layer's FIFOs is accessed in one operation.

Supported APIs to use VMAP mode are DqRtDmap*** and DqRtVmap***.

Initialize the VMAP to acquire/generate data at 1kHz:

The High-Performance Alternative

```
DqRtVmapInit(handle, vmapid, 1000.0);
```

Add channels from the first input subsystem of device 0:
```
int channels = {0, 1, 2, 3 };
DqRtVmapAddChannel(handle, vmapid, 0, DQ_SS0IN, channels, flags, 1);
```

Start all devices that have channels configured in the VMAP:
```
DqRtVmapStart(handle, vmapid);
```

Specify how much input data to transfer during the next refresh.
```
DqRtVmapRqInputDataSz(handle, vmapid, 0, numScans*sizeof(uint16),
&act_size, NULL);
```

Synchronize the VMAP with all devices:
```
DqRtVmapRefresh(handle, vmapid);
```

Retrieve the data acquired by device 0:
```
DqRtVmapGetInputData(handle, vmapid, 0, numScans*sizeof(uint16),
&data_size, &avl_size, (uint8*)bdata);
```

Stop the devices and free all resources:
```
DqRtVmapStop(handle, vmapid);
DqRtVmapClose(handle, vmapid);
```

Event API

The event API only works when running your program on a UEIPAC. You can't call any event function when communicating with PowerDNA over Ethernet.

The event API allows you to get notified in your application when a hardware event occurs.
The hardware events are:

- SyncIn event: a digital edge was sensed on the syncin pin of the Sync connector.
- Timer event: occurs at each tick of a hardware timer located on the CPU layer.

Here is a quick tutorial on using the event API (handling of error codes is omitted):

Configure hardware timer to generate an event every millisecond.
```
DqEmbConfigureEvent(handle, DqEmbEventTimer, 1000);
```

Wait for the next event, if no event occur or after 2 seconds, the function returns the event "`DqEmbEventTimeout`":

United Electronic Industries

The High-Performance Alternative

```
DqEmbWaitForEvent(handle, 2000, &event);
```

Cancel the timer event:

```
DqEmbCancelEvent(handle, DqEmbEventTimer);
```

Refer to Appendix B for detailed documentation of each event API function.

Unsupported APIs
All other APIs than the one mentioned above are not supported on the UEIPAC.
This includes all the ACB (DqACB***), DMAP (DqDmap***), MSG (DqMsg***) and M3 (DqMmm***) APIs.

Building and running the examples
Change your current directory to "<UEIPAC SDK directory>/sdk/DAQLib_Samples" and type `make` to make sure that your setup can build the samples correctly.

If you get any error while building the examples, check that the path to the cross-compiler is in your PATH environment variable and that the environment variable UEIPACROOT is set to the SDK directory.

You can now transfer any of the built examples to the UEIPAC, using FTP and run it.

Each example accepts command line options to specify the following parameters:

- -d <device id>: specify the device
- -c <channel list>: specify the channel list
- -f <frequency>: specify the rate
- -n <number of Scans>: specify the number of samples per channels

For example the following command runs the Sample201example to acquire channels 0,2 and 4 from device 1:

```
Sample201 –d 1 –c "0,2,4"
```

Building your own program
The first step is to compile your program, use the **–I** option to tell the compiler where the PowerDNA API headers are:

```
powerpc-604-linux-gnu-gcc –I ${UEIPACROOT}/includes –c myprogram.c
```

Then link your program, use the **–L** option to tell the linker where the PowerDNA API library is and the **–l** option to tell the linker to link against the PowerDNA library:

```
powerpc-604-linux-gnu-gcc –L ${UEIPACROOT}/includes –lpowerdna
myprogram.o –o myprogram
```

The High-Performance Alternative

The PowerDNA API is implemented in two libraries:

- **libpowerdna.so** implements the PowerDNA API for regular Linux processes
- **libpowerdna_rt.so** implements the PowerDNA API for real-time tasks

8.5. Real-Time Programming

The UEIPAC comes with support for the Xenomai Real-time framework (see http://www.xenomai.org).
Xenomai is a real-time development framework cooperating with the Linux kernel, in order to provide hard real-time support to user-space applications, seamlessly integrated into the Linux environment.

Xenomai uses the flow of interrupts to give real-time tasks a higher priority than the Linux kernel:

- When an interrupt is asserted, it is first delivered to the real-time kernel, instead of the Linux kernel. The interrupt will be later also delivered to the Linux kernel when the real-time kernel is done.
- Upon receiving an interrupt, the real-time kernel can schedules its real-time tasks
- Only when the real-time kernel is not running anything will the interrupt be passed on to the Linux kernel.
- Upon receiving the interrupt Linux can schedule its own processes and threads.
- Xenomai's real-time kernel highest priority allows it to preempt the Linux kernel whenever a new interrupt arrives with no delay and repeat the cycle

Xenomai allows to run real-time tasks either strictly in kernel space, or within the address space of a Linux process.
A real-time task in user space still has the benefit of memory protection, but is scheduled by Xenomai directly instead of the Linux kernel. The worst case scheduling latency of such kind of task is always near the hardware limits and predictable.

Using Xenomai parlance, real-time tasks are running in the primary domain while the Linux kernel and its processes are running in secondary domain.

A real-time task always start in primary domain, however it will jump to secondary domain (and be schedules by the Linux kernel instead of Xenomai's RT kernel) upon invoking a non-rt system call. Non-RT system calls are all system calls that are not implemented by Xenomai. This includes memory allocation (malloc), file access, network access (sockets), process and thread management etc…

You need to make sure that the time critical part of your application runs in the primary domain. One way to do this is to partition an application in two or more tasks, one high priority tasks runs the time critical code and communicate with other lower-priority tasks using Xenomai's IPC objects such as message queues and FIFOs.

The library **libpowerdna_rt**.so implements a version of the PowerDNA API that is safe to call from time critical code running in primary domain.

All real-time examples have the suffix **_rt**. For example **Sample207** is a standard Linux sample program while **Sample207_rt** is a real-time sample program.

8.6. Running a program automatically after boot

Edit the file /etc/rc.local and add an entry for any number of programs that you want to run after the UEIPAC complete its power-up sequence.
In the example below, the /etc/rc.local file is modified to run the Sample201 example at boot time.

```
#!/bin/sh
#
# rc.local
#
# This script is executed at the end of the boot sequence.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#

listlayers > /etc/layers.xml
sync
devtbl

# start Sample201
/usr/local/examples/Sample201 &

exit 0
```

Note that Sample201 is executed in the background ('&' prefix). To stop sample201 you must send the SIGINT signal with the following command (It is equivalent to typing CTRL+C on the console if Sample201 was running in the foreground):

```
killall -SIGINT Sample201
```

United Electronic Industries
The High-Performance Alternative

8.7. Running a program periodically

The UEIPAC comes with **crond** installed to periodically run scripts and programs.

Enable the init script to start **crond** at boot time:

```
mv /etc/rc.d/K30crond /etc/rc.d/S30crond
```

Add a new schedule entry to the cron configuration file:

```
crontab -e
```

Press **i** to switch to insert mode and type the new schedule entry using the following format: <minute> <hour> <day> <month> <dayofweek> <command>

<Minute> - Minutes after the hour (0-59).
<Hour> - 24-hour format (0-23).
<Day> - Day of the month (1-31).
<Month>- Month of the year (1-12).
<Dayofweek>. Day of the week (0-6, where 0 indicates Sunday).

An asterisk in a schedule entry indicates "every". It means that the task will occur on "every" instance of the given field. So a "*" on the Month field indicates the the task will run "every" month of the year. A * in the Minutes field would indicate that the task would run "every" minute.

A comma is used to input multiple values for a field. For example, if you wanted a task to run at hours 12, 15 and 18, you would enter that as "12,15,18".

For example the following entry will append the string “Hello UEIPAC” to the file /tmp/crontest every day at 2:30 and 15:30.

```
30 2,15 * * * echo "Hello UEIPAC" >> /tmp/crontest
```

The High-Performance Alternative

9. Firmware installation and upgrade

9.1. Installing or upgrading the Linux kernel

Your UEIPAC comes with the Linux kernel already installed into flash memory.
It is possible to update that Linux kernel if needed.

You first need to install a TFTP server on your host PC and copy the new kernel image you got from UEI technical support in the TFTP server's directory. Kernel image files are are named:

- cuImage.ueipac5200 for the UEIPAC-300 and UEIPAC-600.
- cuImage.ueipac834x for the UEIPAC-300-1G, UEIPAC-600-1G, UEIPAC-600R and UEIPAC-1200R.

You can find the image of the Kernel that shipped with your UEIPAC in the folder "<UEIPAC SDK directory>/kernel"

That same folders also contains scripts to download the kernel sources and build the kernel yourself, see Appendix E.

Connect to the UEIPAC through the serial port and power-up the cube. Press a key before the 2 seconds countdown ends to enter U-Boot's command line interface.

UEIPAC with Freescale 5200 CPU (100MBit Ethernet)

1. Erase unprotected part of flash memory:
   `erase ffd50000 ffefffff`
2. Configure the UEIPAC's IP address
   `setenv ipaddr <IP address of the UEIPAC>`
3. Configure U-Boot to use your host PC as TFTP server:
   `setenv serverip <IP address of your host PC>`
4. Download the new kernel from the TFTP server
   `tftp 200000 cuImage.ueipacXXXX`
5. Write kernel into flash (make sure you literally type "${filesize}")
   `cp.b 200000 ffd50000 ${filesize}`
6. Set U-Boot's boot command to automatically boot Linux
   `setenv bootm ffd50000`

United Electronic Industries

The High-Performance Alternative

7. Save environment variables to flash
```
saveenv
```
8. Reset and boot the new kernel:
```
reset
```

UEIPAC with Freescale 8347 CPU (1GBit Ethernet)

1. Erase unprotected part of flash memory:
```
erase fe000000 fe1fffff
```
2. Configure the UEIPAC's IP address
```
setenv ipaddr <IP address of the UEIPAC>
```
3. Configure U-Boot to use your host PC as TFTP server:
```
setenv serverip <IP address of your host PC>
```
4. Download the new kernel from the TFTP server
```
tftp 200000 cuImage.ueipacXXXX
```
5. Write kernel into flash (make sure you literally type "${filesize}")
```
cp.b 200000 fe000000 ${filesize}
```
6. Set U-Boot's boot command to automatically boot Linux
```
setenv bootm fe000000
```
7. Save environment variables to flash
```
saveenv
```
8. Reset and boot the new kernel:
```
reset
```

9.2. Initializing an SD card

Your UEIPAC came pre-installed with an SD card containing the root file system necessary to run Linux.
You might want to initialize a new SD card if the factory-installed card becomes unusable or if you decide to upgrade to a faster or bigger one.

Note: You need to run Linux on your host PC to initialize an SD card. This is required because the SD card must be formatted with the ext2 file system.

Make sure automatic mounting is disabled for removable medias.

The High-Performance Alternative

1. Insert the SD card in a USB adapter connected to your host PC.

2. Find out the name of the device node associated with the card. Type the command “dmesg” and look for a message at the end of the log similar to:
`SCSI: device sda: 1984000 512-byte hdwr sectors (1016 MB)`
This message tells us that the device node we are looking for is “/dev/sda”.

3. Un-mount the SD card if necessary
```
sudo umount /dev/sda1
```

4. Erase all partitions from the SD card and create one primary partition using all the space available on the card (the example below uses a 1GB card with 1016 cylinders, use whatever default value is suggested for the last cylinder):
```
fdisk /dev/sda
Command (m for help): d
Selected partition 1
Command (m for help): n
Command action
   e extended
   p primary partition (1-4)
p
Partition number (1-4):1
First Cylinder (1-1016, default 1):1
Last Cylinder … (1-1016, default 1016):1016

Command (m for help): w
```

5. Un-mount the SD card if necessary
```
sudo umount /dev/sda1
```

6. The device node associated with the partition we just created is “/dev/sda1”. Let’s format this new partition:
```
sudo mke2fs /dev/sda1
```

7. CD to a temporary directory and untar the root file system:
```
cd /tmp
sudo tar xvfz <UEIPAC SDK directory>/rfs.tgz
```

8. Mount the new partition (on some Linux distributions it might already be mounted, check with the command ‘df’) then copy the root file system to the SD card:
```
sudo mount /dev/sda1 /mnt
sudo cp –rd /tmp/rfs/* /mnt
```

9. Unmount the SD card and insert it in the UEIPAC. It is now ready to boot.
```
sudo umount /dev/sda1
```

9.3. Running the standard DAQBios firmware

Starting with the 2.0 release, UEIPACs come with both a Linux kernel and DAQBios firmware loaded in flash. You can select which one you want to run by setting a configuration variable in the u-boot boot loader..

Connect to the UEIPAC through the serial port and power-up the cube. Press a key before the 2 seconds countdown ends to enter U-Boot's command line interface.

Configure UEIPAC with Freescale 5200 CPU to run DAQBios firmware

1. Set U-Boot's boot command to start the DAQBios firmware automatically:
```
setenv bootcmd fwjmp
saveenv
```
2. Reset and boot the DAQBios firmware:
```
reset
```

Configure UEIPAC with Freescale 5200 CPU to run Linux

3. Set U-Boot's boot command to start Linux automatically:
```
setenv bootcmd bootm ffd50000
saveenv
```
4. Reset and boot the Linux kernel:
```
reset
```

Configure UEIPAC with Freescale 8347 CPU to run DAQBios firmware

1. Set U-Boot's boot command to start the DAQBios firmware automatically:
```
setenv bootcmd go ff800100
saveenv
```
2. Reset and boot the DAQBios firmware:
```
reset
```

Configure UEIPAC with Freescale 8347 CPU to run Linux

3. Set U-Boot's boot command to start Linux automatically:
```
setenv bootcmd bootm fe000000
saveenv
```

The High-Performance Alternative

4. Reset and boot the Linux kernel:
   reset

10. Install open source software

You can install pretty much any open source software package designed for Linux on your UEIPAC provided that those software packages can be cross-compiled.
The following sections describe a few standard way of cross-compiling software packages.

10.1. Software coming with an autoconf configure script

Most software packages that use autoconf can be configured with the following command:

```
./configure --host=powerpc-604-linux-gnu --build=i686-pc-linux-gnu –
prefix=<root file system>
```

The configure script will then verify that the UEIPAC cross-compiler is operational and create the Makefiles required to build the software package.

To build type:

```
make
```

To install the built binaries, type:

```
make install
```

10.2. Other software

Read the README and INSTALL files that often come with open source packages for instructions about cross-compiling.

If there are no configure script and no instructions you might still be able to build a software package to run on the UEIPAC with the command:

```
CC=powerpc-604-linux-gnu-gcc LD=powerpc-604-linux-gnu-ld
RANLIB=powerpc-604-linux-gnu-ranlib make
```

The High-Performance Alternative

Appendix A RTMAP API

DqRtDmapInit

Syntax:
```
int DqRtDmapInit(int handle ,int* dmapid ,double refreshRate);
```
Input:

int handle	Handle to the IOM
int* dmapid	The identifier of the newly created DMAP.
double refreshRate	Rate at which the IOM will refresh its version of the DMAP.

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_NO_MEMORY	memory allocation error or exceeded maximum table size
DQ_SUCCESS	command processed successfully

Description:

Initialize the specified IOM to operate in DMAP mode at the specified refresh rate.

DqRtDmapAddChannel

Syntax:
```
int DqRtDmapAddChannel(int handle, int dmapid, int dev, int subsystem, uint32* cl, int clSize);
```
Input:

int handle	Handle to the IOM
int dmapid	Identifier of the DMAP
int dev	ID of the device where the channels are located
int subsystem	The subsystem to use on the device (ex: DQ_SS0IN)
uint32* cl	Array containing the channels to add to the DMAP
int clSize	Size of the channel array

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_BAD_DEVN	there is no device with the specified number
DQ_BAD_PARAMETER	the subsystem is invalid for this device
DQ_SUCCESS	command processed successfully

Description:

Add one or more channels to the DMAP.

United Electronic Industries

The High-Performance Alternative

DqRtDmapGetInputMap

Syntax:

```
int DqRtDmapGetInputMap(int handle, int dmapid, int
dev, unsigned char** mappedData);
```

Input:

int handle	Handle to the IOM
int dmapid	Identifier of the DMAP
int dev	ID of the device where the channels are located

Output:

mappedData	pointer to the beginning of the device's input DMAP buffer

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_BAD_DEVN	there is no device with the specified number
DQ_SUCCESS	command processed successfully

Description:

Get pointer to the beginning of the input data map allocated for the specified device

DqRtDmapGetInputMapSize

Syntax:

```
int DqRtDmapGetInputMapSize(int handle, int dmapid,
int dev, int* mapSize);
```

Input:

int handle	Handle to the IOM
int dmapid	Identifier of the DMAP
int dev	ID of the device where the channels are located

Output:

mappedSize	size in bytes of the device's input data map.

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_BAD_DEVN	there is no device with the specified number
DQ_SUCCESS	command processed successfully

Description:

Get the size in bytes of the input map allocated for the specified device

DqRtDmapGetOutputMap

Syntax:

```
int DqRtDmapGetOutputMap(int handle, int dmapid, int dev, unsigned char** mappedData);
```

Input:

int handle	Handle to the IOM
int dmapid	Identifier of the DMAP
int dev	ID of the device where the channels are located

Output:

mappedData	pointer to the beginning of the device's output DMAP buffer

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_BAD_DEVN	there is no device with the specified number
DQ_SUCCESS	command processed successfully

Description:

Get pointer to the beginning of the output data map allocated for the specified device

DqRtDmapGetOutputMapSize

Syntax:

```
int DqRtDmapGetOutputMapSize(int handle, int dmapid, int dev, int* mapSize);
```

Input:

int handle	Handle to the IOM
int dmapid	Identifier of the DMAP
int dev	ID of the device where the channels are located

Output:

mappedSize	size in bytes of the device's output data map.

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_BAD_DEVN	there is no device with the specified number
DQ_SUCCESS	command processed successfully

Description:

Get the size in bytes of the output map allocated for the specified device

DqRtDmapReadScaledData

The High-Performance Alternative

Syntax:

```
int DqRtDmapReadScaledData(int handle, int dmapid, int dev, double* scaledBuffer, int bufferSize);
```

Input:

int handle	Handle to the IOM
int dmapid	Identifier of the DMAP
int dev	ID of the device where the channels are located
int bufferSize	Number of elements in scaledBuffer

Output:

double*scaledBuffer	The buffer containing the scaled data.

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_BAD_DEVN	there is no device with the specified number
DQ_SUCCESS	command processed successfully

Description:

Read and scale data stored in the input map for the specified device.

Note:

The data read is the data transferred by the last call to DqRtDmapRefresh().

This function should only be used with devices that acquire analog data such as the AI-2xx serie.

DqRtDmapReadRawData16

Syntax:

```
int DqRtDmapReadRawData16(int handle, int dmapid, int dev, unsigned short* rawBuffer, int bufferSize);
```

Input:

int handle	Handle to the IOM
int dmapid	Identifier of the DMAP
int dev	ID of the device where the channels are located
int bufferSize	Number of elements in rawBuffer

Output:

unsigned short*rawBuffer	The buffer containing the raw data.

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_BAD_DEVN	there is no device with the specified number
DQ_SUCCESS	command processed successfully

Description:

Read raw data from the specified device as 16 bits integers.

Note:

The data read is the data transferred by the last call to DqRtDmapRefresh().

The High-Performance Alternative

This function should only be used with devices that acquire 16bits wide digital data such as the AI-201.

DqRtDmapReadRawData32

Syntax:

```
int DqRtDmapReadRawData32(int handle, int dmapid, int dev, unsigned int* rawBuffer, int bufferSize);
```

Input:

int handle	Handle to the IOM
int dmapid	Identifier of the DMAP
int dev	ID of the device where the channels are located
int bufferSize	Number of elements in rawBuffer

Output:

unsigned int* rawBuffer	The buffer containing the raw data.

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_BAD_DEVN	there is no device with the specified number
DQ_SUCCESS	command processed successfully

Description:

Read raw data from the specified device as 32 bits integers.

Note:

The data read is the data transferred by the last call to DqRtDmapRefresh().

This function should only be used with devices that acquire 32 bits wide digital data such as the DIO-4xx serie.

DqRtDmapWriteScaledData

Syntax:

```
int DqRtDmapWriteScaledData(int handle, int dmapid, int dev, double* scaledBuffer, int bufferSize);
```

Input:

int handle	Handle to the IOM
int dmapid	Identifier of the DMAP
int dev	ID of the device where the channels are located
int bufferSize	Number of elements in scaledBuffer
double*scaledBuffer	The buffer containing the scaled data to send to the device.

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_BAD_DEVN	there is no device with the specified number

The High-Performance Alternative

DQ_SUCCESS command processed successfully

Description:

Write scaled data to the output map of the specified device.

Note:

The data written will be actually transferred to the device on the next call to DqRtDmapRfresh().

This function should only be used with devices that generate analog data such as the AO-3xx series.

DqRtDmapWriteRawData16

Syntax:

```
int DqRtDmapWriteRawData16(int handle, int dmapid, int
dev, unsigned short* rawBuffer, int bufferSize);
```

Input:

int handle	Handle to the IOM
int dmapid	Identifier of the DMAP
int dev	ID of the device where the channels are located
int bufferSize	Number of elements in rawBuffer
unsigned short*rawBuffer	The buffer containing the raw data to write to the device.

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_BAD_DEVN	there is no device with the specified number
DQ_SUCCESS	command processed successfully

Description:

Write 16 bits wide raw data to the specified device.

Note:

The data written will be actually transferred to the device on the next call to DqRtDmapRfresh().

This function should only be used with devices that generate 16bits wide digital data such as the AO-3xx series.

DqRtDmapWriteRawData32

Syntax:

```
int DqRtDmapWriteRawData32(int handle, int dmapid, int
dev, unsigned int* rawBuffer, int bufferSize);
```

Input:

int handle	Handle to the IOM
int dmapid	Identifier of the DMAP

The High-Performance Alternative

int dev	ID of the device where the channels are located
int bufferSize	Number of elements in rawBuffer
unsigned int* rawBuffer	The buffer containing the raw data to write to the device.

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_BAD_DEVN	there is no device with the specified number
DQ_SUCCESS	command processed successfully

Description:

Write raw data to the specified device as 32 bits integers.

Note:

The data written will be actually transferred to the device on the next call to DqRtDmapRfresh().

This function should only be used with devices that acquire 32 bits wide digital data such as the DIO-4xx series.

DqRtDmapStart

Syntax:

```
int DqRtDmapStart(int handle, int dmapid);
```

Input:

int handle	Handle to the IOM
int dmapid	Identifier of the DMAP

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_SUCCESS	command processed successfully

Description:

Start operations, the cube will update its internal representation of the map at the rate specified in DqRtDmapInit.

DqRtDmapStop

Syntax:

```
int DqRtDmapStop(int handle, int dmapid);
```

Input:

int handle	Handle to the IOM
int dmapid	Identifier of the DMAP

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_SUCCESS	command processed successfully

Description:

The High-Performance Alternative

Stop operations, the cube will stop updating its internal representation of the data map

DqRtDmapRefresh

Syntax:

```
int DqRtDmapRefresh(int handle, int dmapid);
```

Input:

int handle	Handle to the IOM
int dmapid	Identifier of the DMAP

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_SUCCESS	command processed successfully

Description:

Refresh the host's version of the map by downloading the IOM's map.

Note:

The IOM automatically refresh its version of the data map at the rate specified in DqRtDmapInit(). This function needs to be called periodically (a real-time OS might be necessary) to synchronize the host and IOM data maps.

DqRtDmapClose

Syntax:

```
int DqRtDmapClose(int handle, int dmapid);
```

Input:

int handle	Handle to the IOM
int dmapid	Identifier of the DMAP

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_SUCCESS	command processed successfully

Description:

Free all resources allocated by the DMAP operation on the specified IOM.

United Electronic Industries

The High-Performance Alternative

Event API

DqEmbConfigureEvent

Syntax:

```
int DqEmbConfigureEvent(int handle, DQ_EMBEDDED_EVENT event, unsigned int param);
```

Input:

int handle	Handle to the IOM
DQ_EMBEDDED_EVENT event	Event to configure.
unsigned int param	Event specific parameter

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_SUCCESS	command processed successfully

Description:

Configure hardware to notify the specified event.

Possible events are:

DqEmbEventSyncIn:	Digital edge at the syncin connector, set param to 0 for rising edge or 1 for falling edge.
DqEmbEventTimer:	Timer event, set param to desired frequency.

DqEmbWaitForEvent

Syntax:

```
int DqEmbWaitForEvent(int handle, int timeout, DQ_EMBEDDED_EVENT *event);
```

Input:

int handle	Handle to the IOM
int timeout	Timeout in milliseconds
DQ_EMBEDDED_EVENT event	Received event.

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_SUCCESS	command processed successfully

Description:

Wait for any configured event to occur. If no event happens before the timeout expiration the function returns the event "DqEmbEventTimeout".

DqEmbCancelEvent

The High-Performance Alternative

Syntax:

```
     int DqEmbCancelEvent(int handle, DQ_EMBEDDED_EVENT
event);
```

Input:

int handle	Handle to the IOM
DQ_EMBEDDED_EVENT event	Event to cancel

Return:

DQ_ILLEGAL_HANDLE	invalid IOM handle
DQ_SUCCESS	command processed successfully

Description:

Cancel specified event.

The High-Performance Alternative

Using Eclipse IDE to program the UEIPAC

Download and Install Eclipse

There are several ways to install Eclipse with support for C/C++ programming.

If you are already using Eclipse (for java programming for example) you can keep your existing Eclipse and just install the additional plug-ins CDT (C/C++ developer tools) and TM (Target management).
Otherwise, download the :**Eclipse IDE for C/C++ developers** package available at http://www.eclipse.org/downloads.

Unzip the package in a folder of your choice (for example "c:\eclipse\" under Windows or "/opt/eclipse" under Linux) and run the program **eclipse.exe** to start Eclipse

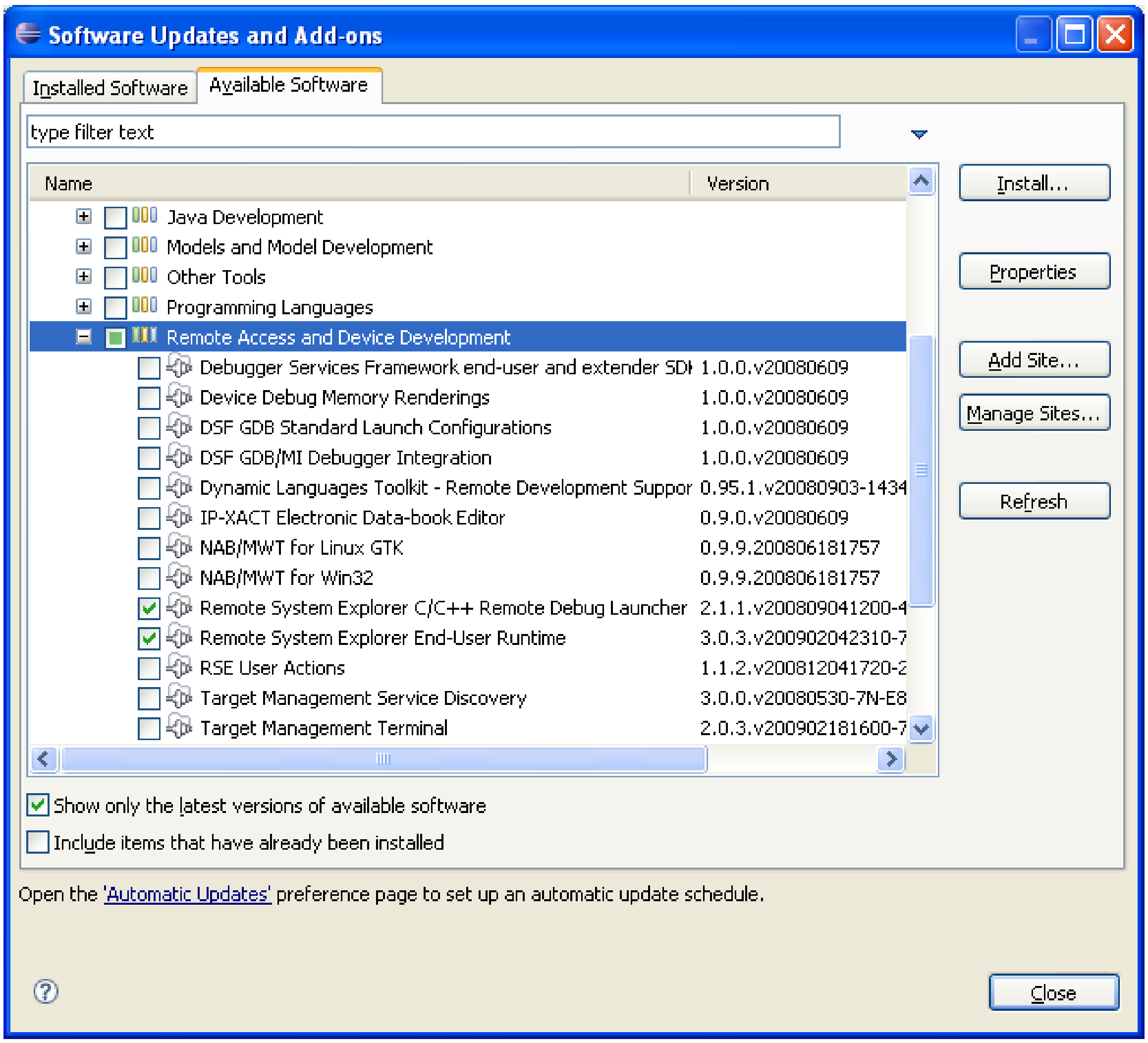
Install additional plug-ins

Eclipse CDT is pre-configured to build, run and debug programs on the host PC. However we want to run and debug programs on the UEIPAC and we need a few additional plug-ins to do that.

The TM (target management) plug-in is very useful to transfer files between the PC and the UEIPAC from the IDE.

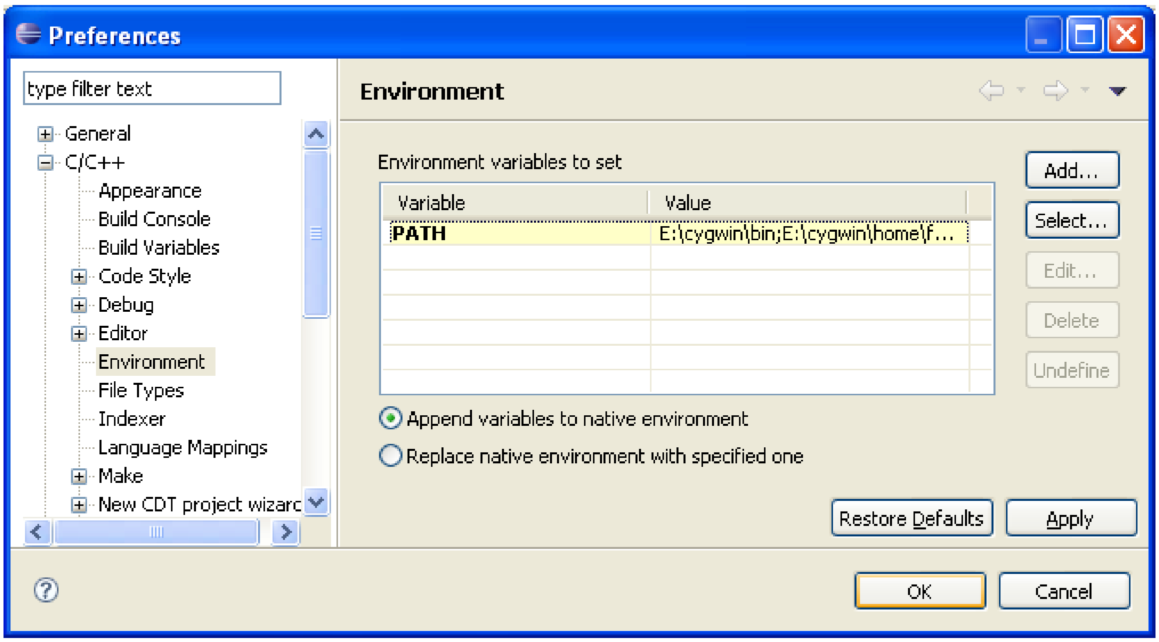
In Eclipse select the menu **Help/Software updates...** In the **Available Software** tab select **Ganymede Update Site/Remote Access and Device Development**
Check the packages **Remote System Explorer End User Runtime** and **Remote System Explorer C/C++ Remote Debug Launcher** then click on **Install…**

United Electronic Industries
The High-Performance Alternative



Set-up preferences

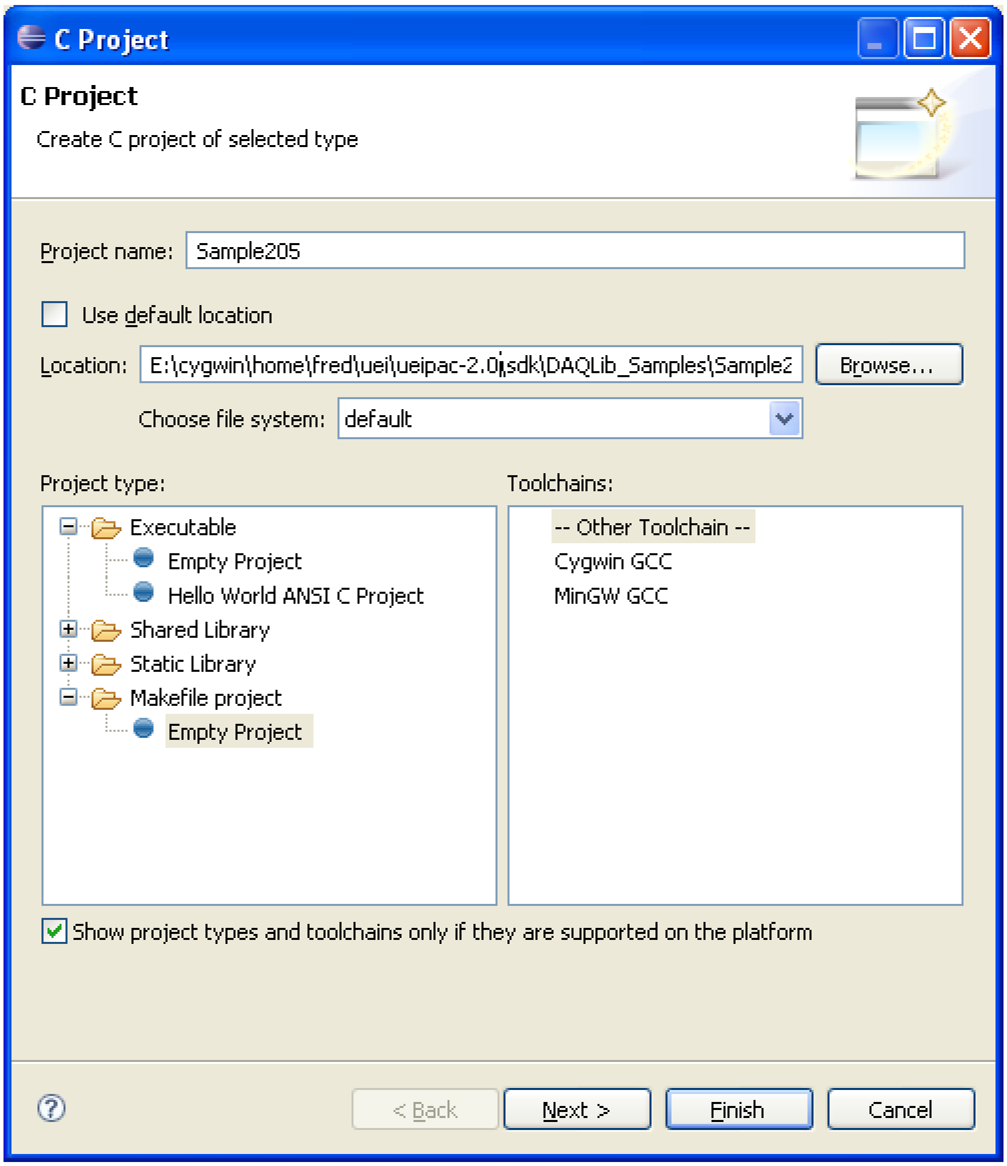
Edit Eclipse preferences to add the path to the cygwin tools (such as make) and the UEIPAC cross-compiler.
Select the menu option **Window/Preferences** then click on **C/C++/Environment**.
Add a variable named **PATH** with value set to the cygwin bin directory and the powerpc-604-linux-gnu/bin directory.
For example: **c:\cygwin\bin;c:\cygwin\home\fred\uei\ueipac-2.0.0\powerpc-604-linux-gnu/bin**



Open and Build examples

Select the menu option **File/New C Project…**
Select the project type **Makefile Project/Empty Project**

Type a project name
Un-check **Use Default Location**
Browse to the location of the example you wish to build (examples are located <Cygwind directory>\home\<your user name>\uei\ueipac-2.0\sdk\DAQLib_Samples).



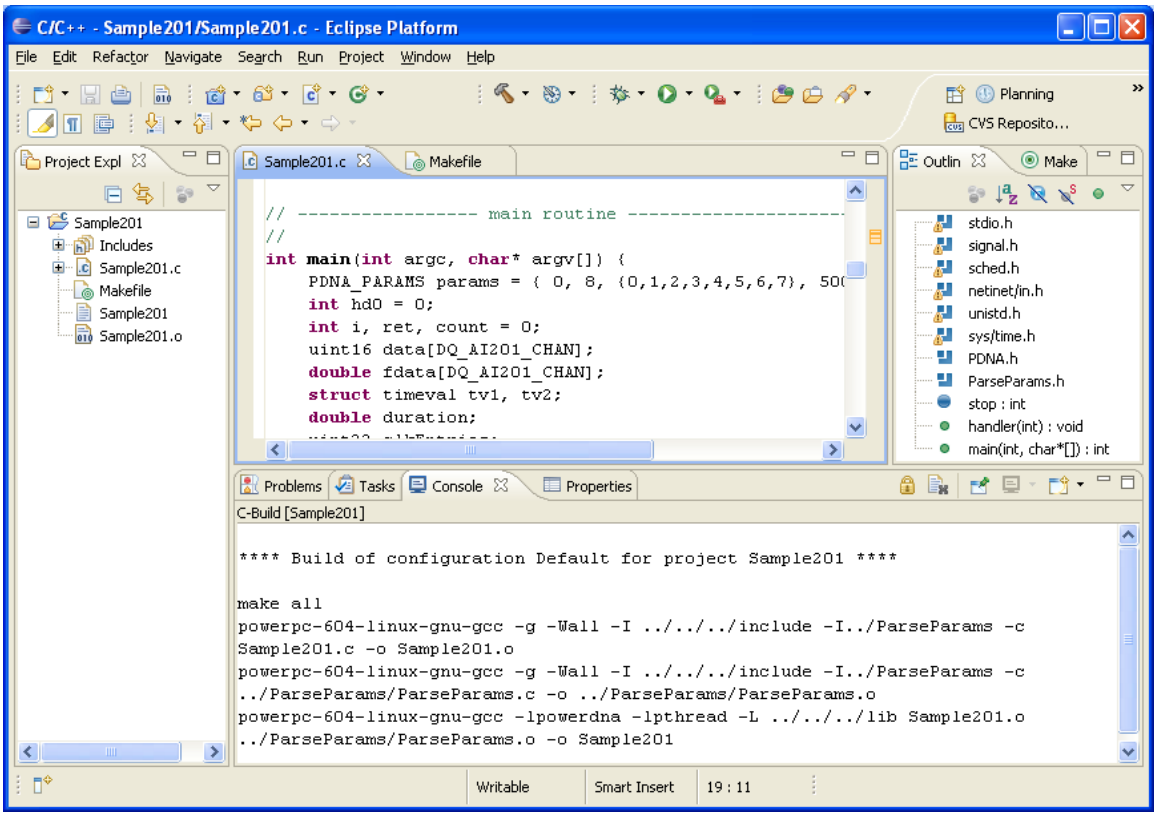
Click **Finish** to create the project

Edit the Makefile of the project and add the following line at the beginning of the file

```
UEIPACROOT = /home/<your user name>/uei/ueipac-2.0
```

Select the menu **Project/Build Project** to build the example.

The High-Performance Alternative

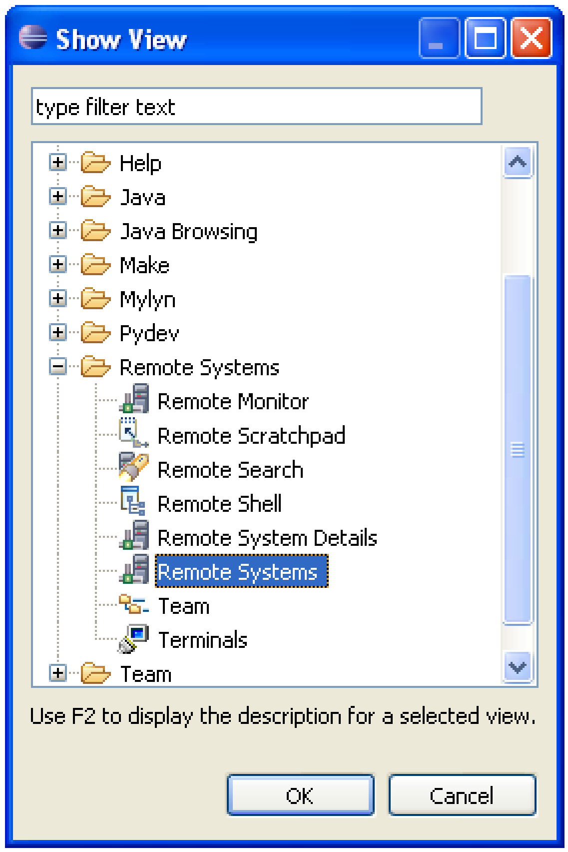


Download program to target

Add a **Remote Systems** view to your current perspective:

Select the menu **Window/Show View/Other…**

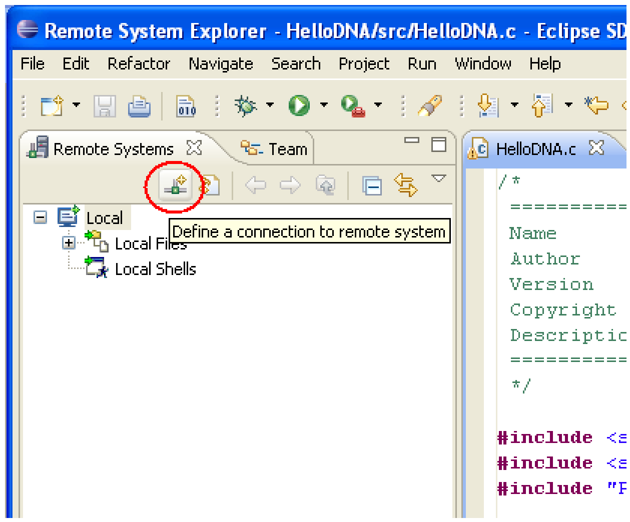
The High-Performance Alternative

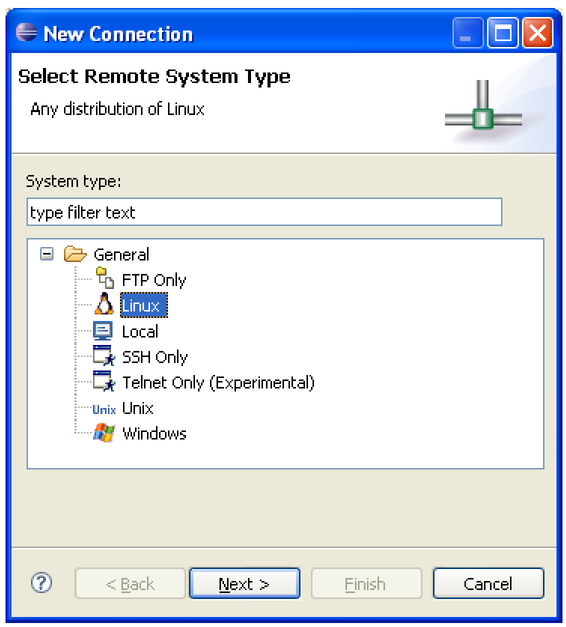


Select **Remote Systems** and Click on **OK**.

In the **Remote Systems** view, click the **Define a connection to a remote system** button

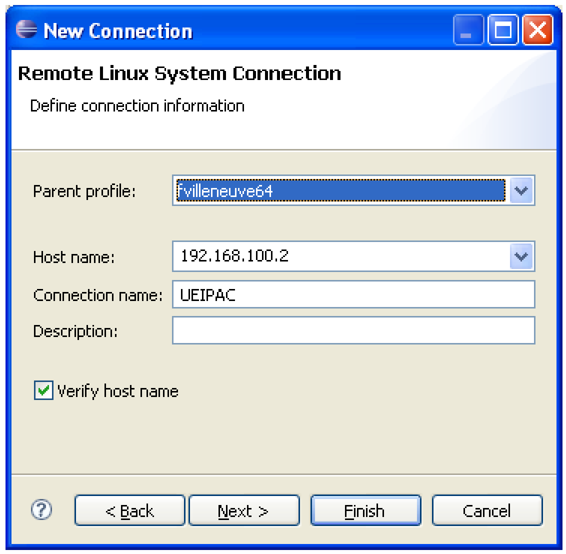
The High-Performance Alternative





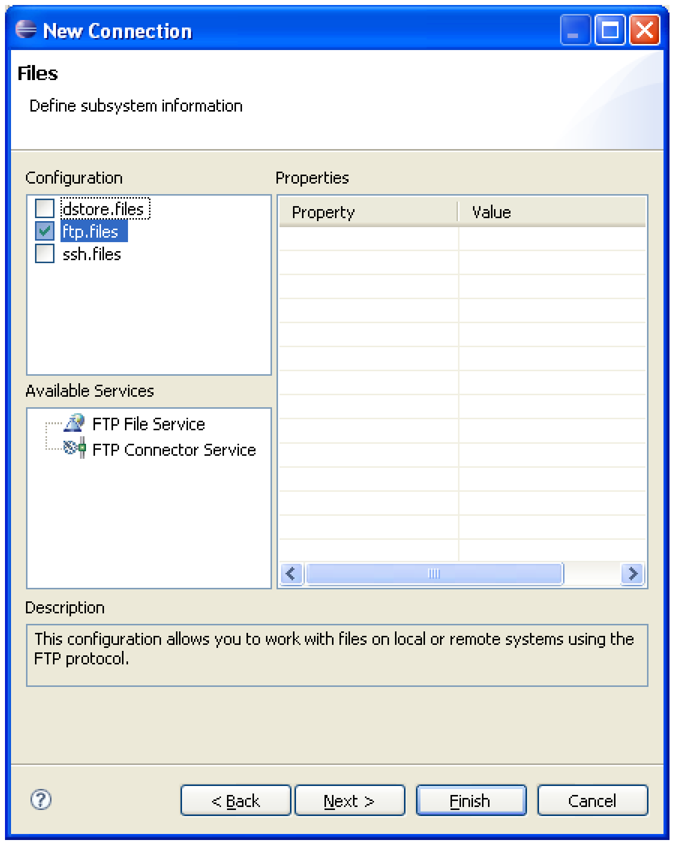
Select **Linux** then click on **Next**.

The High-Performance Alternative



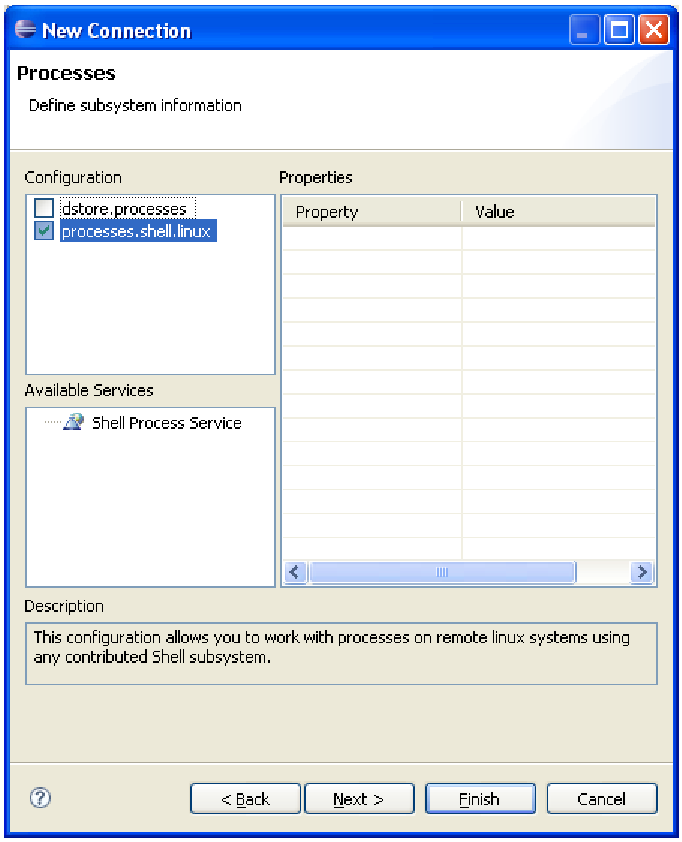
Enter the IP address of your UEIPAC as **Host Name** and pick a **Connection name**.
Click on **Next**

The High-Performance Alternative

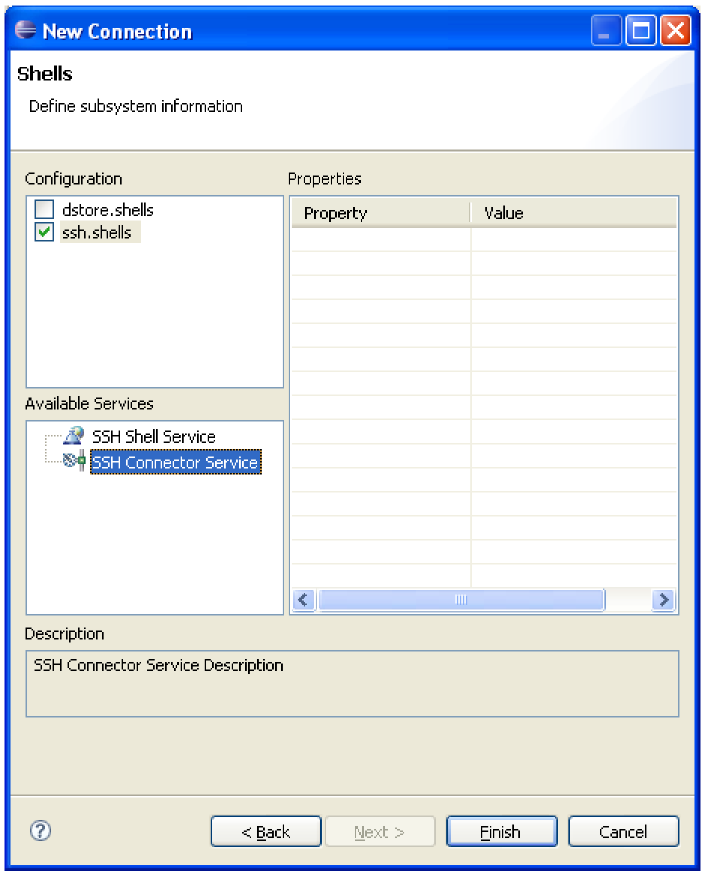


We will use FTP to transfer files to/from the UEIPAC. Select **ftp.files**. Select **FTP Connector Service** and set the FTP **passive** setting to **true.** Click on **Next**.

The High-Performance Alternative



We will use a remote shell to control the processes running on the UEIPAC. Select **processes.shell.linux** and click on **Next**.

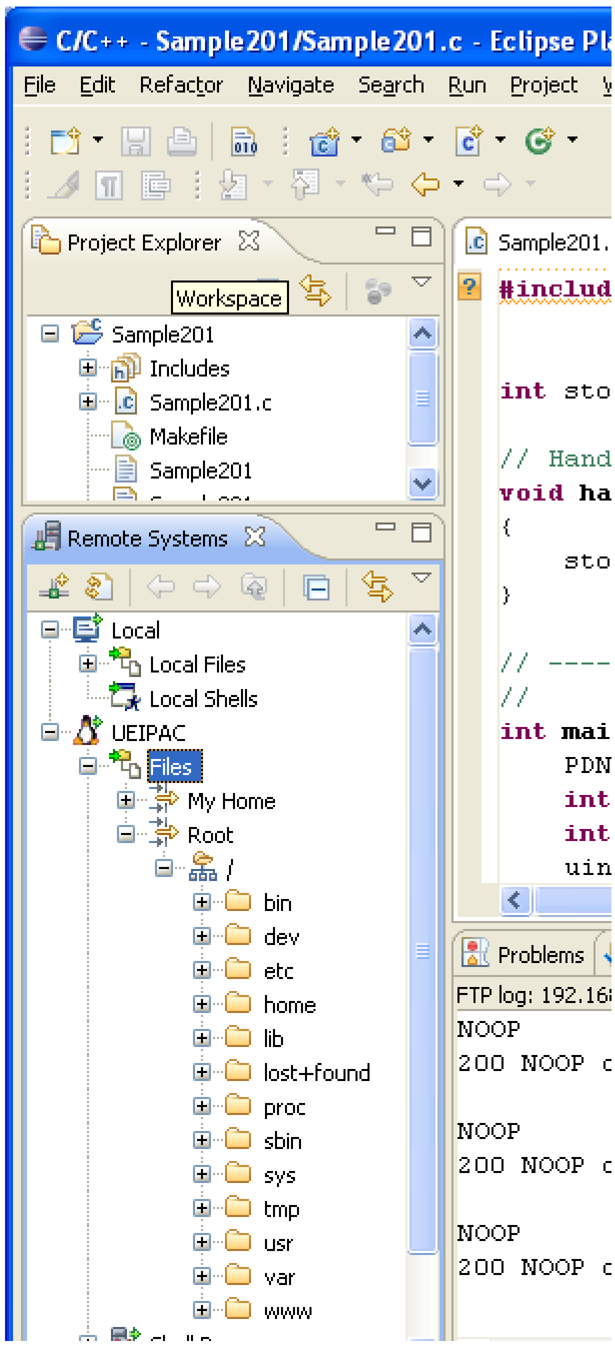


We will use SSH as remote shell to control the UEIPAC.
Select **ssh.shells** and click on “Finish”.

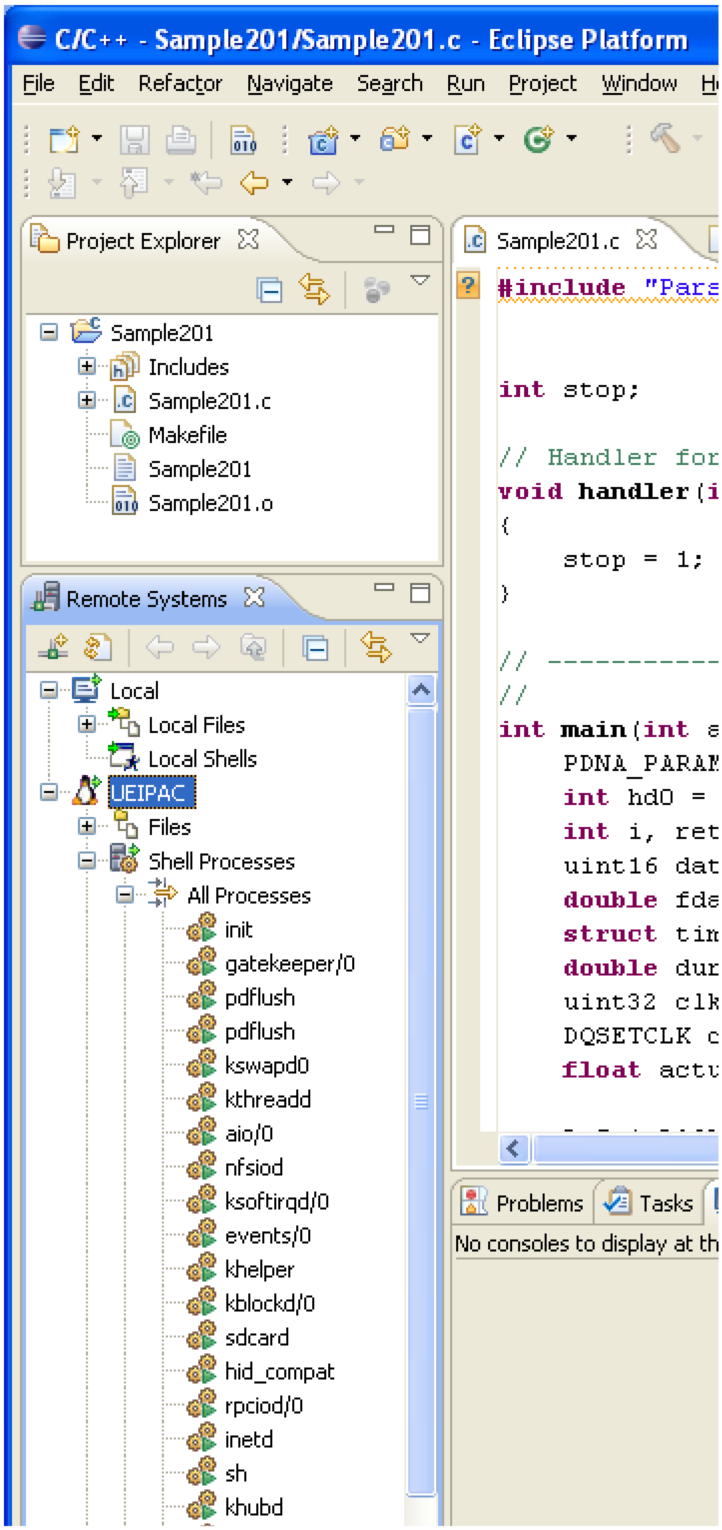
The UEIPAC will now appear in the **Remote Systems** pane on the left.
Let’s test the connection by navigating files on the UEIPAC file system. Click on **UEIPAC/Files/Root**:

The High-Performance Alternative

Enter "root" as **User ID** and **Password**



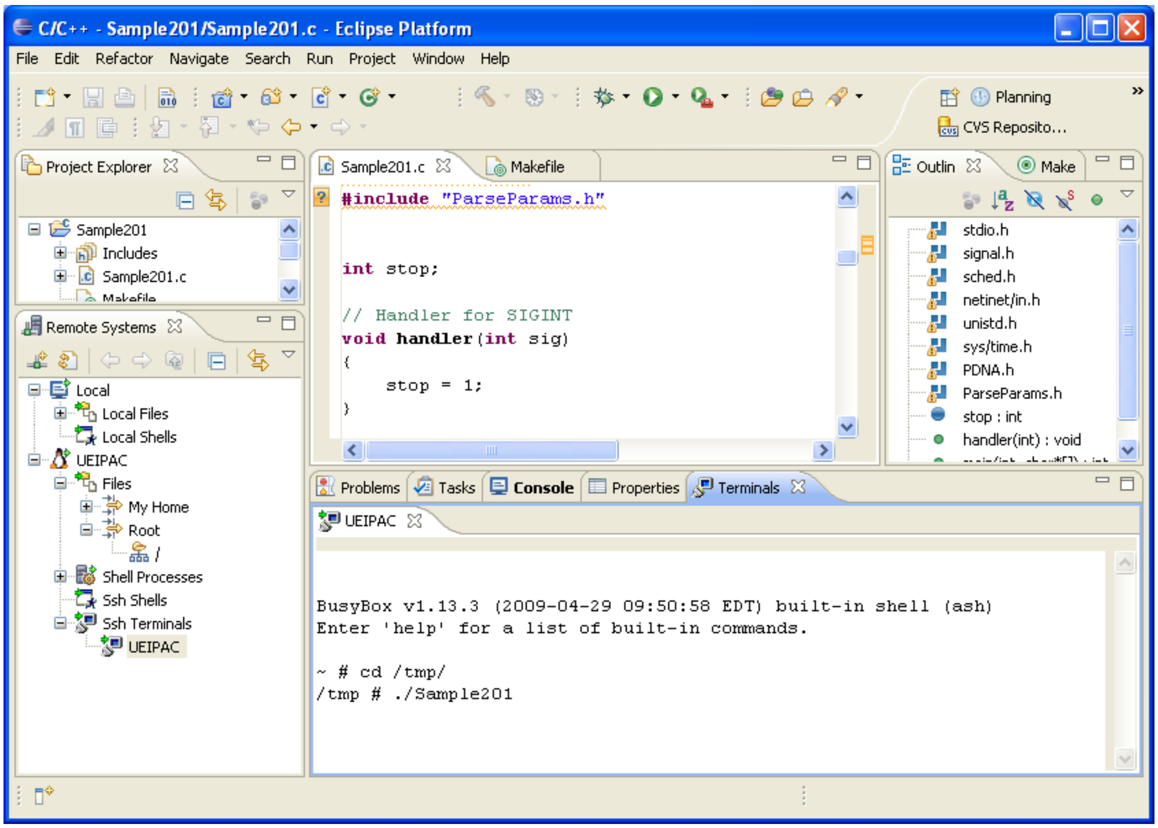
Select **UEIPAC/Shell Process/All Processes** to view the processes running on the UEIPAC


C/C++ - Sample201/Sample201.c - Eclipse Platform
File Edit Refactor Navigate Search Run Project Window
Project Explorer
Sample201
Includes
Sample201.c
Makefile
Sample201
Sample201.o
Remote Systems
Local
Local Files
Local Shells
UEIPAC
Files
Shell Processes
All Processes
init
gatekeeper/0
pdflush
pdflush
kswapd0
kthreadd
aio/0
nfsiod
ksoftirqd/0
events/0
khelper
kblockd/0
sdcard
hid_compat
rpciod/0
inetd
sh
khubd
Sample201.c
int stop;
// Handler for
void handler(i
{
stop = 1;
}
int main(int
int hd0 =
int i, ret
Problems
Tasks
No consoles to display at th

United Electronic Industries

The High-Performance Alternative

Righ-click on **UEIPAC/Ssh Terminals** and select **Launch Terminal** to open a remote shell session on the UEIPAC

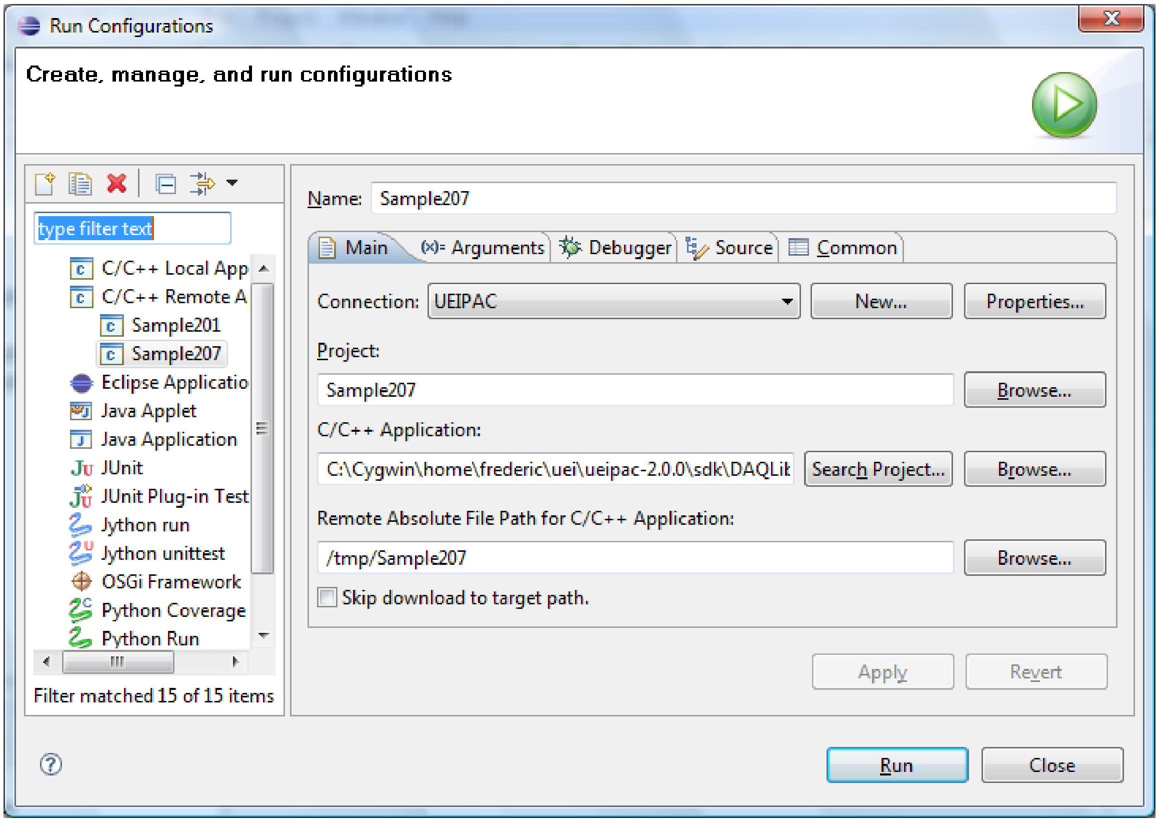


Execute program

Select the **Run>Show Run Dialog…** menu option to open the Run dialog box.

Select the **C/C++ Remote Application** option and press the **New** button to create a new remote launch configuration:

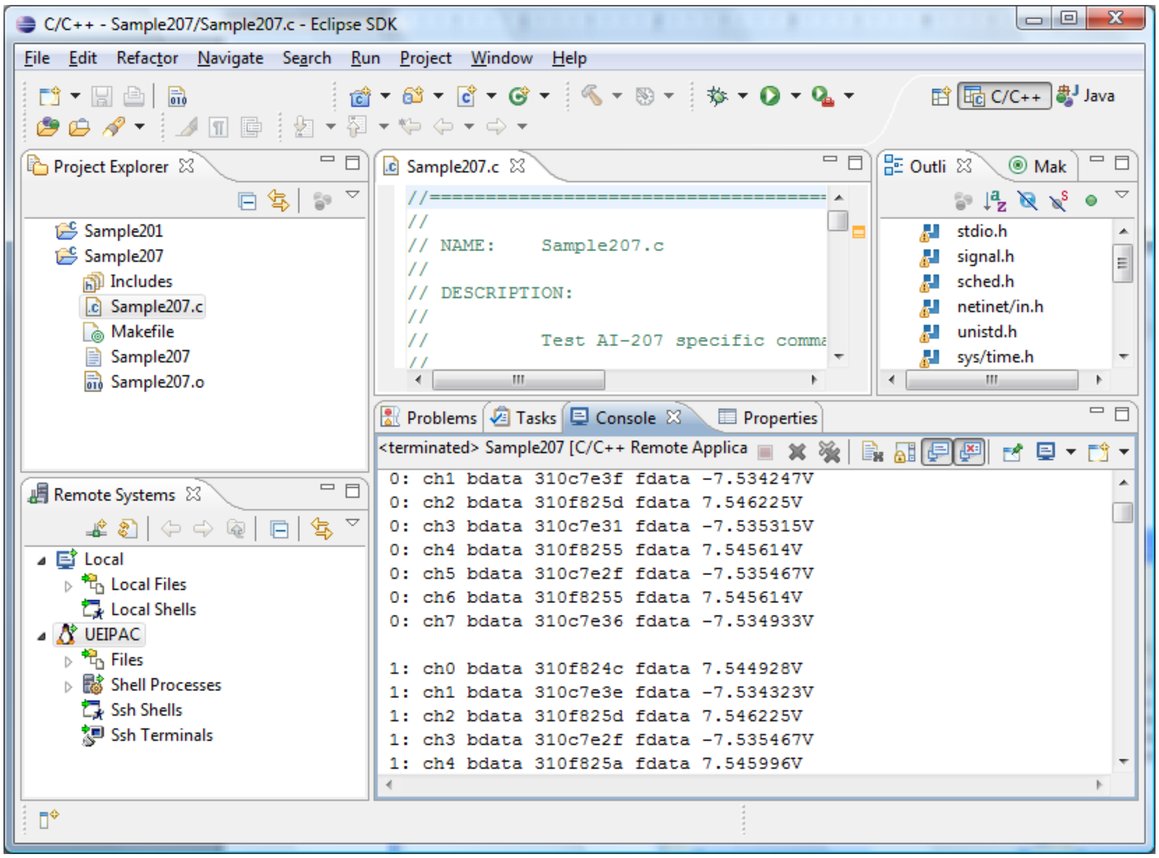
The High-Performance Alternative



Enter a name for this new launch configuration.
Set the Connection to **UEIPAC** previously defined.
Verify that the project is set properly or press **Browse…** to select the right project.
Verify that the **C/C++ Application** is set to the binary built from your project.
Set the **Remote Absolute File Path** to the path of the executable on the remote target.

Press **Run** to download the binary to the UEIPAC and execute it. You will see the result in the Console:

The High-Performance Alternative



Debugging your program on the UEIPAC

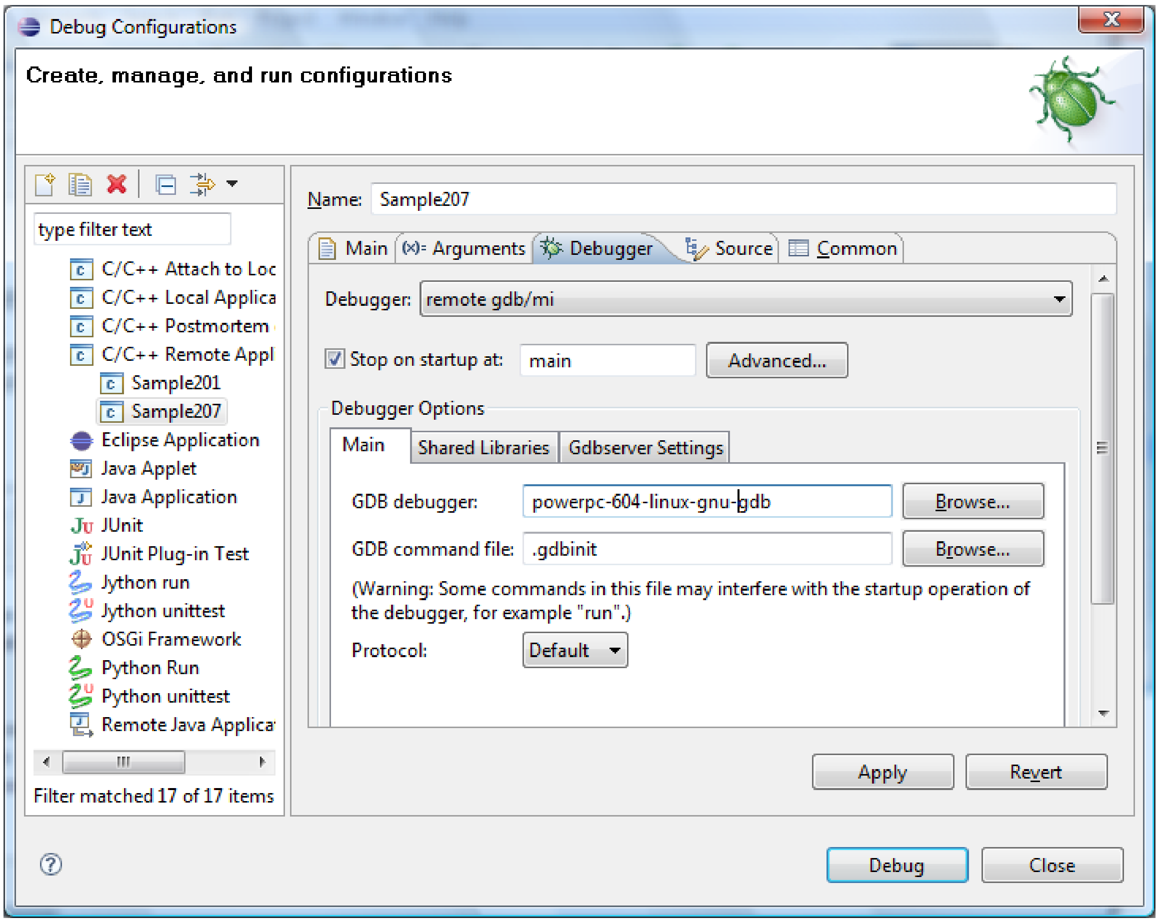
The UEIPAC examples are already compiled with debug information. Make sure that your program does too (add the option **–g** to the compiler flags).

Select the **Run>Show Debug Dialog…** menu option to open the Debug dialog box.

Select the **C/C++ Remote Application/Sample201** previously created.

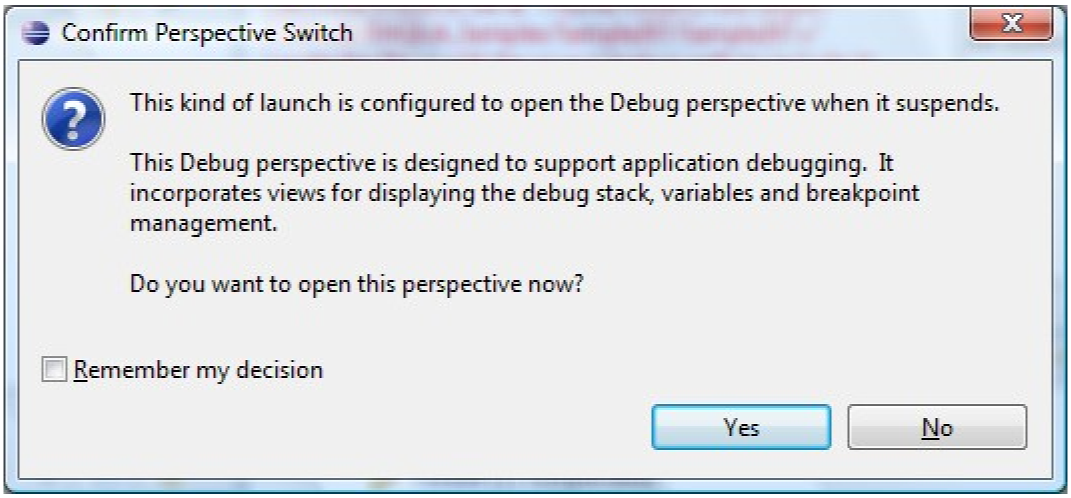
In the **Debugger** tab set **GDB debugger** to **powerpc-604-linux-gnu-gdb**

The High-Performance Alternative



Click on **Debug** to download the program to the UEIPAC and start debugging it.

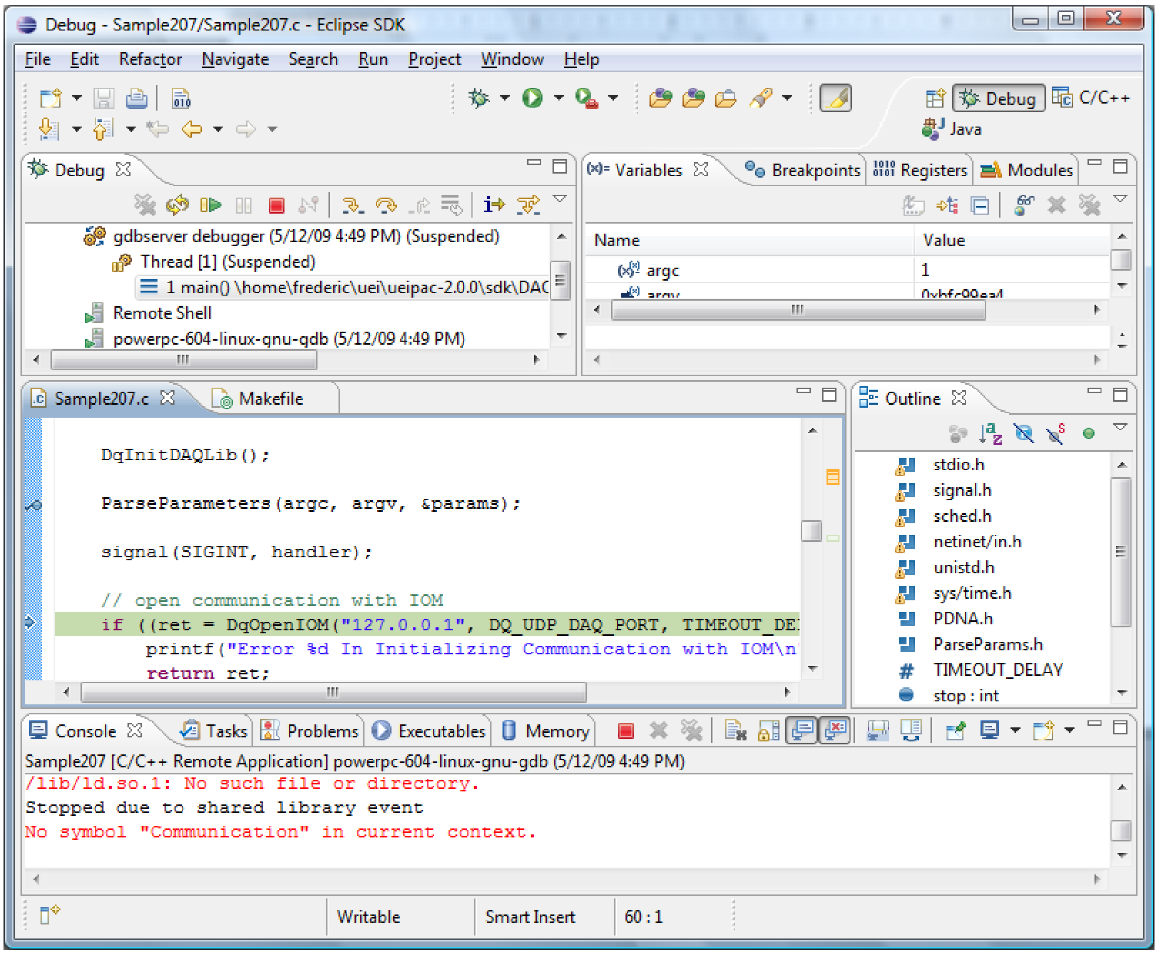
Eclipse will suggest that you switch to the **Debug perspective**, click on **Yes**



The debugger will pause the program execution at the beginning of main().

Set a breakpoint on a line in main() (Right-click on the line and select **Toggle breakpoint**) then press **F8** to resume execution.

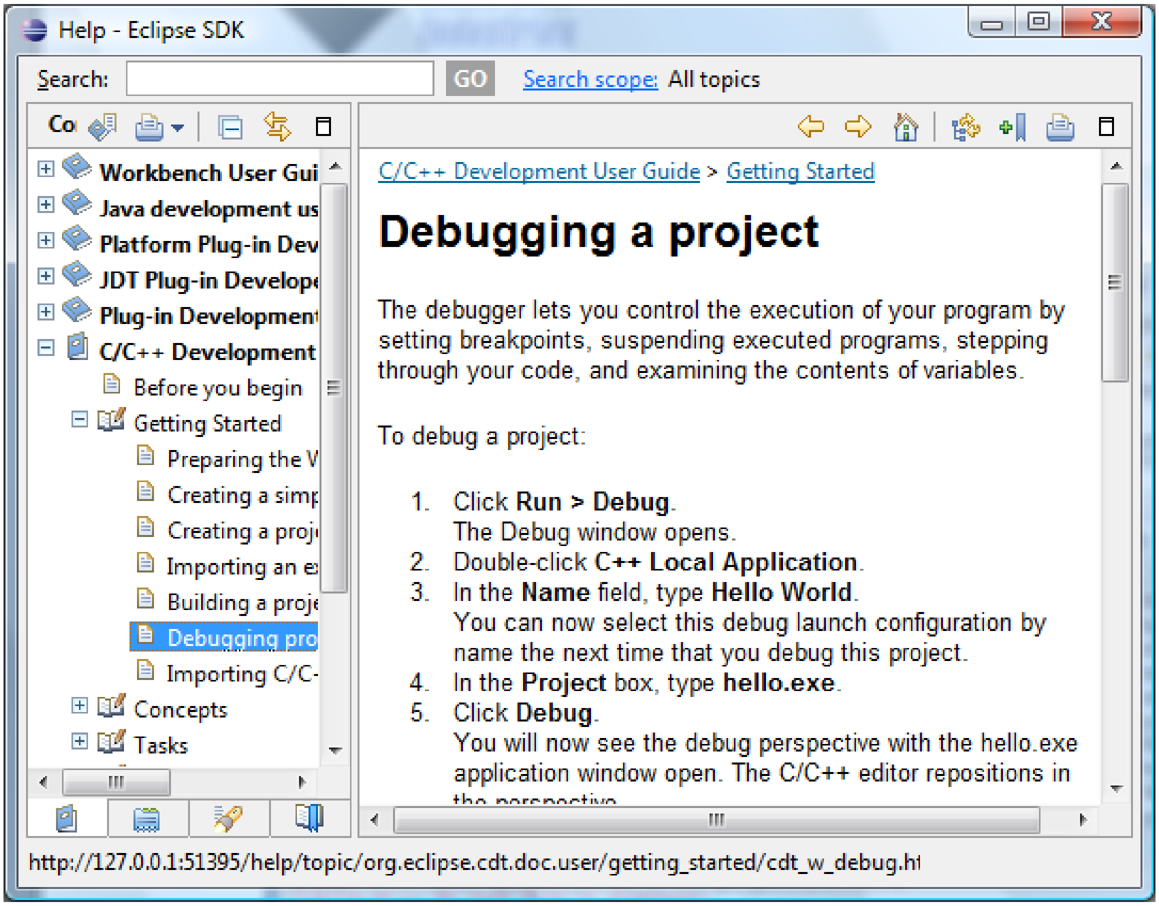
The debugger will pause the program again at the line where the breakpoint was set.



You can now execute the program step by step pressing the keys **F5** and **F6**

More information about debugging programs is available in Eclipse's online help. Select the menu option **Help/Help content**

The High-Performance Alternative



United Electronic Industries
The High-Performance Alternative

Booting from NFS

Configure shared RFS on host PC

- Install an NFS server on your Linux machine
- Un-tar the rfs.tgz file that comes on the UEIPAC CD-ROM
- Share the rfs directory (usually done by adding an entry in the */etc/exports* file) */etc/exports* file should look like this:

```
/home/frederic/UEIPAC/rfs
192.168.100.0/255.255.255.0(rw,sync,no_subtree_check,no_root_squa
sh)
```

- Remove the file *rfs/etc/rc.d/S10network* (kernel does the network configuration while booting and overwriting it will kill the NFS session)
- Create the directory *rfs/etc/mnt* (used to mount the sd card later)
- Edit the file *rfs/etc/fstab* and change the mount point for */dev/sdcard1* to */mnt* *rfs/etc/fstab* should look like this:

```
/dev/sdcard1 /mnt ext2 defaults,noatime 1 1
none /proc proc defaults 0 0
none /sys sysfs defaults 0 0
none /dev/pts devpts defaults 0 0
```

This will make the sd card accessible under */mnt* when the UEIPAC boots over NFS

Configure Uboot

- Power-up the UEIPAC and press a key to enter U-Boot.
- Type the following commands:

```
setenv gateway <your gateway ip address>
setenv netmask <your netmask>
setenv consoledev ttyPSC0
setenv baudrate 57600
setenv netdev eth0
setenv rootpath <The remote path where rfs is located on your
host PC>
run nfsargs
run addip
setenv bootargs ${bootargs} console=${consoledev},${baudrate}
saveenv
printenv
```

- Verify that your *bootargs* variable looks like this:

```
bootargs=root=/dev/nfs rw
```

The High-Performance Alternative

```
nfsroot=192.168.100.1:/home/frederic/UEIPAC/rfs
console=ttyPSC0,57600
ip=192.168.100.2:192.168.100.1::255.255.255.0::eth0:off panic=1
```

- Reset the UEIPAC which will now find its root file system on the NFS share
```
reset
```

The High-Performance Alternative

Building the Linux kernel

Note that you can only build the UEIPAC Linux kernel on a PC running Linux connected to the internet.
Make sure that you have the following tools installed:

- git
- make
- patch
- UBoot mkimage

Use the package manager of your linux distribution to install those tools.

For example, use the following commands on Ubuntu:

```
sudo apt-get install git
sudo apt-get install make
sudo apt-get install patch
sudo apt-get install uboot-mkimage
```

Download and patch kernel source

At a command prompt, change the current directory to **<UEIPAC SDK directory>/kernel**

The UEIPAC kernel includes Xenomai real-time extension. The first step is to download and build Xenomai.

Run the **build_xenomail.sh** script.

```
./build_xenomai.sh
```

Then run the **get_kernel.sh** script with the option **–cpu** set to the CPU of your UEIPAC:

For UEIPAC-300, UEIPAC-600:

```
./get_kernel.sh –cpu 5200
```

For all other UEIPACs:

```
./get_kernel.sh –cpu 834x
```

This script might take a long time to execute depending on the speed of your internet connection.

Once the script is finished, you will find a new directory **linux-DENX-v2.6.28.5** containing the kernel source with Xenomai and UEIPAC patches applied.

Configure and build the kernel for UEIPAC-300 and UEIPAC-600

Change the current directory to the linux source directory

1. Configure kernel with default settings:
```
make ARCH=powerpc CROSS_COMPILE=powerpc-604-linux-gnu-
52xx/ueipac5200_defconfig
```
2. Customize kernel configuration:
```
make ARCH=powerpc CROSS_COMPILE=powerpc-604-linux-gnu-
menuconfig
```
3. Compile the kernel:
```
make ARCH=powerpc CROSS_COMPILE=powerpc-604-linux-gnu-
cuImage.ueipac5200
```

You can find the build kernel in **arch/powerpc/boot/cuImage.ueipac5200**

Configure and build the kernel for UEIPAC-300-1G, UEIPAC-600-1G and RACK versions

Change the current directory to the linux source directory

1. Configure kernel with default settings:
```
make ARCH=powerpc CROSS_COMPILE=powerpc-604-linux-gnu-
83xx/ueipac834x_defconfig
```
2. Customize kernel configuration:
```
make ARCH=powerpc CROSS_COMPILE=powerpc-604-linux-gnu-
menuconfig
```
3. Compile the kernel:
```
make ARCH=powerpc CROSS_COMPILE=powerpc-604-linux-gnu-
cuImage.ueipac834x
```
4. Compile modules:
```
make ARCH=powerpc CROSS_COMPILE=powerpc-604-linux-gnu-
modules
```
5. Install modules:
```
make ARCH=powerpc CROSS_COMPILE=powerpc-604-linux-gnu-
INSTALL_MOD_PATH=<Your module install path>
modules_install
```

You can find the built kernel in **arch/powerpc/boot/cuImage.ueipac834x** and the modules in whatever directory you assigned to the **INSTALL_MOD_PATH** variable.

The High-Performance Alternative