United Electronic Industries

The High-Performance Alternative

# UEIPAC
# Linux Software Development Kit
# User Manual 5.0

December 2020 Edition

# Table of contents

The High-Performance Alternative

# 1 Introduction

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

On the UEIPAC, a Linux or Real-Time Linux[1] kernel replaces the standard "DAQBIOS" firmware in flash memory and uses an SD card, SSD drive, or flash as its local file system. This file system contains the other components of the operating system such as libraries, utilities, initialization scripts, and daemons.

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

The UEIPAC can also be configured to execute user applications after booting-up and can be configured for synchronization of I/O board clocks on multiple chassis using the IEEE-1588 standard or using an external 1PPS reference.

User applications run 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). See diagram 2.

The UEIPAC SDK includes a library dedicated for communicating with UEIPAC I/O boards and a subset of the hosted PowerDNA API, which allows reuse of existing example programs originally designed for hosted systems that communicate with PowerDNA hardware over a network. The PowerDNA server runs automatically, providing access to UEI software tools.

Examples provided with the SDK can be updated to run directly on the UEIPAC with few modifications. See section 8.4 for more information.

[1] Note that this manual only applies to the Linux-based UEIPAC. UEI additionally offers a UEIPAC-VxWorks distribution which contains a BSP to run a VxWorks kernel. Please refer to the UEIPAC-VxWorks manual to learn how to operate a VxWorks-based UEIPAC.

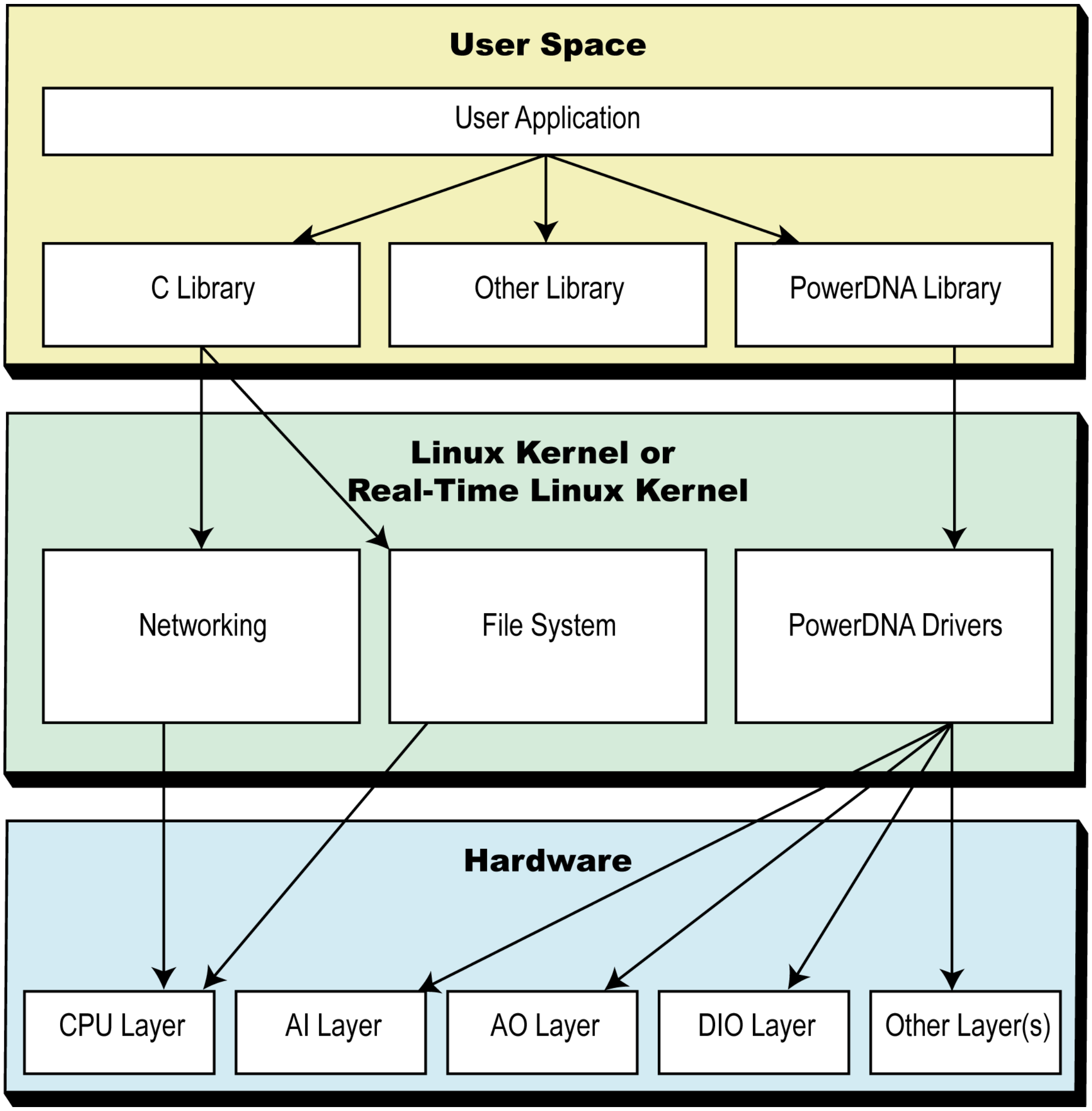


**Figure 1 UEIPAC Layered Architecture**

## 1.1 Kernel Options

Your Linux-based UEIPAC ships with a mainline Linux[2] kernel pre-installed into flash memory. Real-Time options include using the Xenomai Real-Time framework or using a Real-Time Linux kernel, which is provided with your SDK.

Real-Time Linux offers a deterministic form of mainline Linux. It gives faster response times and more importantly, it removes all unbounded latencies.

Real-Time Linux is enabled with the real-time Linux kernel patch, PREEMPT_RT, which improves latencies by maximizing preemptible sections inside the Linux kernel, allowing real-time tasks to be prioritized over the non-real-time Linux kernel threads and interrupt service routines.

Since Real-Time Linux is primarily mainline Linux, it doesn't require any special toolsets; users can use a standard C library, a Linux driver, and POSIX application.

More information about real-time programming is provided in section 8.6. Instructions for updating your kernel are provided in Chapter 9.

**IMPORTANT**: If you are using a **UEIPAC option -11 or -12 (UEIPAC G4 SoloX),** please refer to the *UEIPAC SoloX Software Manual*. The UEIPAC option -11 / -12 is UEI's fourth generation of the UEIPAC, which is based on the dual-core SoloX / i.MX6 series ARM processor. Hardware, kernel options, SDK, installation procedures and more are different for the UEIPAC G4, and much of the information in this manual is not applicable for that product.

[2] UEI additionally offers a UEIPAC-VxWorks distribution which contains a BSP to run a VxWorks kernel. Please refer to the UEIPAC-VxWorks manual to learn how to operate a VxWorks-based UEIPAC.

# 2 Setting up a development system

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

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

Provided development tools include the following:

- GCC cross-compiler targeting the UEIPAC PPC processor
- GNU toolchain tools, such as `make`
- Standard Linux libraries, such as glibc
- PowerDNA library for accessing the various PowerDNA data acquisition devices

## *2.1 Windows host*

The UEIPAC cross-compiler depends on libraries provided by the Cygwin project.

Cygwin is a collection of tools that provide a Linux-like interface and environment for Windows OS and a DLL, which acts as a Linux API layer and provides substantial Linux API functionality.

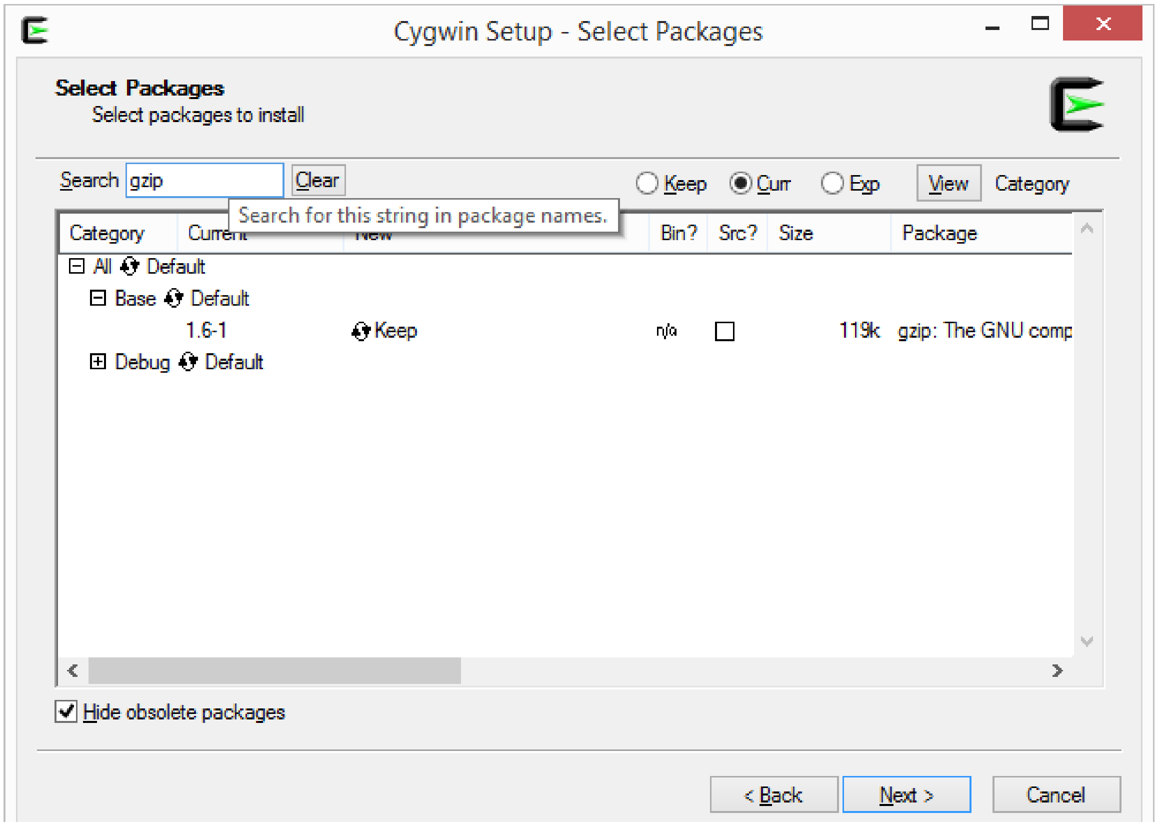
Cygwin is available for free as an open source project. If you don't have Cygwin already installed, download and run the installer **setup_x86.exe** from http://www.cygwin.com.

**NOTE:** UEIPAC software is only compatible with the 32-bit release of Cygwin. When installing from www.cygwin.com, make sure you select **setup_x86.exe** (do not use **setup_x64.exe**).

Running **setup_x86.exe** will install or update Cygwin. Note that the UEIPAC SDK requires three Cygwin packages from the following categories (network utility packages are listed as optional but are referred to in this manual):

- Base: **tar** and **gzip** packages are required.
- Devel: the **make** package is required.
- Net: network utility packages such as **ftp**, **tftp**, **openssh** and **telnet** are optional.

The Cygwin setup window provides a **Search** box, which can be used to find the listed packages and verify they are enabled (see Figure 2).



Cygwin Setup - Select Packages

**Select Packages**

Select packages to install

Search: gzip

Search for this string in package names.

○ Keep ● Curr ○ Exp

| Category | Current | New | Bin? | Src? | Size | Package |
|---|---|---|---|---|---|---|
| ⊟ All ♻ Default | | | | | | |
| ⊟ Base ♻ Default | | | | | | |
| | 1.6-1 | ♻ Keep | n/a | □ | 119k | gzip: The GNU comp |
| ⊞ Debug ♻ Default | | | | | | |

☑ Hide obsolete packages

**Figure 2 Cygwin Setup Window**

To install the UEIPAC SDK on a Windows host:

1. Insert the “UEIPAC SDK” CDROM in your CD drive, and then open a Cygwin command line shell.

2. Change directory to the CD’s root directory (the example below assumes that the CDROM is the D: drive):

```
cd /cygdrive/d
./install.sh
```

The UEIPAC installer modifies the **.bash_profile** file by adding the path of the UEIPAC cross-compiler to your `PATH` variable and creating a new environment variable, **`UEIPACROOT`,** which contains the UEIPAC software installation directory.

To activate the changes to the .bash_profile immediately, you can either close the terminal window and open a new one or type the command below:

```
source ~/.bash_profile
```

## 2.2 Linux host

### 2.2.1 Preparing your 64-bit Linux host

The UEIPAC cross-compiler is a 32-bit program. Note that the cross-compiler requires 32-bit run-time libraries to be installed to run on a 64-bit Linux host.

- Under Ubuntu, use the following command to install libraries:
  ```
  sudo apt-get update
  sudo apt-get install lib32z1
  ```

- Under RedHat, CentOS or Fedora, use the following command to install libraries:
  ```
  sudo yum update
  sudo yum install glibc.i686 zlib.i686
  ```

### 2.2.2 Installing UEIPAC software on your Linux host

To install the UEIPAC SDK on a 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 the CDROM, type:

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

The UEIPAC installer modifies the **.bash_profile** file by adding the path of the UEIPAC cross-compiler to your `PATH` variable and creating a new environment variable, **`UEIPACROOT`,** which contains the UEIPAC software installation directory:

```
#PowerDNA setup: This line was added by the UEIPAC install script
PATH=$PATH:"/home/frederic/uei/ueipac-2.6.0/powerpc-604-linux-gnu/
bin"
export PATH
UEIPACROOT="/home/frederic/uei/ueipac-2.6.0"
export UEIPACROOT
#PowerDNA setup end
```

The **.bash_profile** file is automatically sourced at login.

To activate the changes to the .bash_profile immediately, you can either logout and log back in or type the command below:

```
source ~/.bash_profile
```

You will need to manually update `PATH` and create **`UEIPACROOT`** if your Linux PC is using a different shell interpreter than **bash.**

For example:

- If you are using **csh**, insert **`PATH`** and **`UEIPACROOT`** in **~/.login**
- If you are using **dash,** insert **`PATH`** and **`UEIPACROOT`** in **~/.profile**

## 2.3 SDK directory layout

The following directories and files are included in the SDK file structure:

- *bin*: command line utilities not installed by default on the UEIPAC SD card (mostly Xenomai test programs)
- *doc*: manuals in PDF and HTML format
- *include*: UEIPAC SDK header files
- *kernel*: kernel source code, build scripts, and binary images
- *lib*: UEIPAC SDK shared and static libraries
- *powerpc-604-linux-gnu*: GCC cross compiler
- *rfs*: archive containing the root file system currently installed
- *sdk*: UEIPAC software development kit

# 3 Configuring the UEIPAC

Your PowerDNA/PowerDNR hardware must be pre-configured to run Linux. Hardware configuration includes:

- A Linux kernel loaded in flash memory
  - A root file system contained on any of the following:
    - An SD card inserted in the SD card slot
    - An SSD drive (only available on the UEIPAC-XXX-02 and UEIPAC-XXX-03 versions)
    - Directly in flash (only available on the UEIPAC-XXX-03 versions)

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. If needed, you can reconfigure the serial port for use as a general purpose serial port (see section 6.2).

To connect through the serial port:

1. 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.

2. Run your serial terminal program and configure the serial communication settings accordingly.

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

Once power is connected, you should see a message on your serial console similar to the following:

```
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
```

The console messages are generated from the system U-Boot boot loader. The boot loader pauses for 2 seconds to give the user a chance to alter the configuration, if necessary.

After the countdown ends, U-Boot loads the Linux kernel from flash, uncompresses 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
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
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.

~ #
```

After the boot loader completes, the Linux root prompt (#) will be available in the command line of your serial terminal, allowing you to navigate the file system and enter standard Linux commands, such as `ls`, `ps`, and `cd`.

## 3.2 Root filesystem

A filesystem is the hierarchy of directories where files on a computer system are organized. On Linux systems, the hierarchy starts in a root directory (designated by a forward slash).

The root filesystem is stored on the same partition as the root directory; all other filesystems are mounted in the root filesystem.

The content of the root filesystem includes a minimal set of directories and files required by the system to function (/bin, /sbin, /lib, /dev, /etc).

On the UEIPAC, the root filesystem can be located on different storage hardware depending on the product version (product versions align with different CPU types):

| **UEIPAC Product Version** | **Available Storage Hardware for Root Filesystem** |
|---|---|
| UEIPAC-300,<br>UEIPAC-600, &<br>UEIPAC-700 | Root filesystem must be located on an SD card |
| UEIPAC-300-1G,<br>UEIPAC-600-1G,<br>UEIPAC-700-1G,<br>UEIPAC-600R,<br>UEIPAC-1200R,<br>UEIPAC-400-MIL,<br>UEIPAC-1200-MIL,<br>UEINET-PAC, &<br>UEIPAC-400F | Root filesystem can be located on any of the following:<br>• an SD card<br>• a RAM disk image (stored in flash)<br>• a 24MB partition located in flash |
| UEIPAC-300-1G-02<br>(& all other<br>UEIPAC-xxx-02) | Root filesystem can be located on any of the following:<br>• eUSB SSD drive<br>• an SD card<br>• a RAM disk image (stored in flash)<br>• a 24MB partition located in flash |
| UEIPAC-300-1G-03<br>(& all other<br>UEIPAC-xxx-03) | Root filesystem can be located on any of the following:<br>• eUSB SSD drive<br>• an SD card<br>• a RAM disk image (stored in flash)<br>• a 120MB partition located in flash |

The Linux kernel uses kernel parameters to specify the location of the root filesystem. The boot loader (U-boot) passes the kernel parameters to the kernel as boot arguments.

You can set the boot arguments using the **bootargs** variable.

### 3.2.1 Booting from an SD card

UEIPACs with an SD card installed can boot from the SD card. The default bootup for UEIPAC-XXX-02 versions that have an SSD installed is to boot from the SSD, and the default for UEIPAC-XXX-03 versions is to boot from flash. All other versions boot from the SD card by default.

Note that the root file system located on the SD card uses the EXT3 format.

To use a root filesystem stored on an SD card, the **bootargs** variable must include **root=62:1**.

To power up and power down the UEIPAC cleanly, UEI recommends typing the "`halt`" command before powering down the UEIPAC and the "`reboot`" command to restart the UEIPAC.

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

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

If you get the EXT2-fs warning, 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
```

#### *3.2.1.1 File-system corruption*

Powering down the UEIPAC while it is writing data to a file can cause file system corruption even in a non-related part of the file system.

File corruption can affect files that never get written or even affect files marked as read-only.

The file-system will issue writes in a minimum size, typically 4KB, and a single 4KB block may have data in it that is part of two different files. Those two files might even be in different directories or have different access permissions.

Thus, a simple write to a log file can result in a read and rewrite of part of any file on the partition. When power goes down in the middle of that rewrite, the result is silent data corruption.
File-systems also have to modify a lot of metadata in various places in order to just create a one byte file. A power failure during that operation could, for example, destroy the names of several other files.

You can set-up the UEIPAC to ensure that it survives an uncontrolled power failure by setting up the root file system on a read-only partition and storing temporary files in a RAM disk.

This method ensures that the UEIPAC will always boot unless the SD card itself becomes inoperable (because of wear out or random failure). Note that a small amount of additional memory will be used for temporary file storage, (e.g., log files, lock files).

Keeping system files in a read-only partition has proven reliable in multiple customer applications that incur frequent unscheduled power cycles.

#### *3.2.1.2 Setting-up a root file system as read-only*

See Appendix E for instructions on how to convert a read-write UEIPAC root file system to a read-only one.

#### *3.2.1.3 Restoring an SD card*

It is sometimes necessary to restore an existing SD card or prepare a new one. The following instructions explain how to do that on a UEIPAC:

The first thing is to boot the UEIPAC using an alternate root file system. RAM drive is the only option on most UEIPACs.

1. Boot UEIPAC using a RAM drive (see instructions in section 3.2.4)
2. Run **fdisk**
   Erase all partitions from the SD card and create one primary partition using all the space available on the card:

```
fdisk /dev/sdcard
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
```

3. The device node associated with the partition we just created is "/dev/sdcard1". Format this new partition with **mke2fs** (-j option sets file system type to ext3):

```
mke2fs -j /dev/sdcard1
```

4. Mount the SD card at **/mnt**

```
mount /dev/sdcard1 /mnt
```

5. Copy **<UEIPAC SDK dir>/rfs/rfs.tgz** from your host PC to **/mnt** on the UEIPAC (you can use SCP or FTP)

6. untar the root file system:

```
cd /mnt
gunzip rfs.tgz
tar xvf rfs.tar
```

7. Copy **rfs** folder content to root of SD card

```
mv rfs/* .
rmdir rfs
```

8. Reboot the UEIPAC to start using the newly restored SD card

```
reboot
```

### 3.2.2 Booting from an SSD drive

The default bootup for UEIPAC-XXX-02 versions is to boot from the SSD drive. If they do not have an SSD drive installed, the default boot up is from the SD card. The default for UEIPAC-XXX-03 versions is to boot from flash.

The root file system located on the SSD uses the EXT3 format.

To use a root filesystem stored on an SSD drive, the **bootargs** variable must include **root=/dev/sda1 rw rootwait**.

To power up and power down the UEIPAC cleanly, UEI recommends typing the "`halt`" command before powering down and the "`reboot`" command to restart the UEIPAC.

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

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

If you get the EXT2-fs warning, you must check the file system for errors with the following commands:

```
# mount -o remount,ro /
# e2fsck /dev/sda1
e2fsck 1.38 (30-Jun-2005)
/dev/sda: clean, 702/124160 files, 6632/247872 blocks
# reboot
```

### *3.2.2.1 Restoring an SSD drive*

It is sometimes necessary to restore the SSD drive or prepare a new one. The following instructions explain how to do that on a UEIPAC:

The first thing is to boot the UEIPAC using an alternate root file system. RAM drive is the only option on most UEIPACs.

1. Boot UEIPAC using a RAM drive (see instructions in section 3.2.4)
2. Run **fdisk**
   Erase all partitions from the SSD and create one primary partition using all the space available on the card:

```
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
```

3. The device node associated with the partition we just created is “/dev/sda1”. Format this new partition with **mke2fs** (-j option sets file system type to ext3):

```
mke2fs -j /dev/sda1
```

4. Mount the SD card at **/mnt**

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

5. Copy **<UEIPAC SDK dir>/rfs/rfs.tgz** from your host PC to **/mnt** on the UEIPAC (you can use SCP or FTP)

6. untar the root file system:

```
cd /mnt
gunzip rfs.tgz
tar xvf rfs.tar
```

7. Copy **rfs** folder content to root of the SSD drive

```
mv rfs/* .
rmdir rfs
```

8. Reboot the UEIPAC to start using the newly restored SSD drive

```
reboot
```

9. Press any key to enter U-boot and set **bootargs**

```
=> setenv bootargs console=ttyS0,57600 root=/dev/sda1 rw rootwait
=> saveenv
=> reset
```

### 3.2.3 Booting from an MTD partition (Flash)

A Memory Technology Device (MTD) is a type of device file in Linux for interacting with flash memory.

The -03 versions of UEIPAC (with -03 CPU layers) are equipped with 128MB flash where 120MB are available to store an MTD partition.

**Note**: UEIPAC-1G, UEIPAC-R, UEIPAC-F, UEIPAC-MIL and UEIPAC-XXX-02 models are equipped with 32MB of flash where 24MB is available to store an MTD partition; however, please note to do this, users will need to customize the RAM disk image to a bare minimum, (e.g., remove everything under /usr and most kernel modules).

The **bootargs** variable must include **root=/dev/mtdblock0 rw rootfstype=jffs2** to use a root filesystem stored on an MTD partition.
The root file system located on the MTD uses the JFFS2 format.

The JFFS2 file system format is very robust against power loss. (In filesystem terms, "robust" means that data that is just being written when the system goes down may be lost, but the file system itself does not get corrupt and the system can be rebooted without need for any kind of file system check).

Run the commands below to format the MTD partition:

1. Boot from RAM disk. Follow instructions in section 3.2.4.2 *Loading the RAM disk image to flash*.

2. Erase the partition
   ```
   flash_eraseall -j /dev/mtd0
   ```

3. Mount the partition
   ```
   mount -t jffs2 /dev/mtdblock0 /mnt
   ```

4. Copy **<UEIPAC SDK dir>/rfs/rfs.tgz** from your host PC to /mnt on the UEIPAC (you can use SCP or FTP)

5. untar the root file system:
   ```
   cd /mnt
   gunzip rfs.tgz
   tar xvf rfs.tar
   ```

6. Copy **rfs** folder content to root of the MTD partition
   ```
   mv rfs/* .
   rmdir rfs
   ```

7. Reboot the UEIPAC
   ```
   reboot
   ```

8. Press any key to enter U-boot and set **bootargs**
   ```
   => setenv bootargs console=ttyS0,57600 root=/dev/mtdblock0 rw
   rootfstype=jffs2
   => setenv bootcmd bootm F8000000
   => saveenv
   => reset
   ```

### 3.2.4 Booting from a RAM disk

Booting from a RAM disk is faster than any of the other methods of booting; however, the RAM disk size is limited to 16Mbytes and any data written to the RAM disk is lost when the system shuts down or reboots.

The RAM disk is very useful if, for example, you want to reinitialize the SD card or want to use an NFS share for persistent storage.

The RAM disk can only fit in the flash memory of UEIPAC models based on the 8347 CPU (UEIPAC-1G, UEIPAC-R, UEIPAC-F, or UEIPAC-MIL). The UEIPAC models based on the 5200 CPU (only UEIPAC-300/-600/-700) need to upload the RAM disk image via TFTP each time they boot.

#### *3.2.4.1 Customizing the RAM disk image*

Customizing the RAM drive image is necessary to:

- add your program files to the disk image
- change the default IP address
- tweak the startup script if you wish to start a program automatically

Customization can only be done on a Linux PC. Some versions of Linux might require you to install the uboot mkimage utility to proceed with the customization procedure.

- To install uboot mkimage under Ubuntu or Debian, type:
  ```
  $sudo apt-get install uboot-mkimage
  ```

To customize the RAM disk image, do the following:

1. Extract compressed RAM disk image from the uImage file. The following command converts the **uRamdisk-x.y.z** file to **ramdisk.gz**
   ```
   $ dd if=uRamdisk-x.y.z bs=64 skip=1 of=ramdisk.gz
   21876+1 records in
   21876+1 records out
   ```
2. Uncompress RAM disk image
   ```
   $ gunzip -v ramdisk.gz
   ramdisk.gz:      66.6% -- replaced with ramdisk
   ```
3. Mount RAM disk image
   ```
   $ mount -o loop –t ext2 ramdisk /mnt
   ```

Now files in the `/mnt` directory can be added, removed, or modified.

Once you are done, you can re-pack the RAM disk into a U-Boot image:

1. Unmount RAM disk image:
   ```
   $ umount /mnt
   ```
2. Compress RAM disk image:
   ```
   $ gzip -v9 ramdisk
   ramdisk:         66.6% -- replaced with ramdisk.gz
   ```
3. Create new U-Boot image:
   ```
   $ mkimage -T ramdisk -C gzip -n 'My UEISIM RAM disk' -d ramdisk.gz
   new-uRamdisk-x.y.z
   Image Name:   UEIPAC RAM disk
   Created:      Wed Apr 11 17:32:41 2012
   Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
   Data Size:    14991743 Bytes = 14640.37 kB = 14.30 MB
   Load Address: 0x00000000
   Entry Point:  0x00000000
   ```

#### *3.2.4.2 Loading the RAM disk image to flash*

Follow the steps below to upload the RAM disk to memory and boot from it.

1. Copy the < UEIPAC SDK directory>/rfs/uRamdisk-x.y.z file to the root directory of your TFTP server.
2. Power-up your UEIPAC and press any key to enter U-Boot.
3. Configure the UEIPAC’s IP address:
   `setenv ipaddr <IP address of the UEIPAC>`
4. Configure U-Boot to use your host PC as TFTP server:
   `setenv serverip <IP address of your host PC>`

**NOTE:** U-Boot only supports 1Gbps link speed on 8347 based UEIPACs. Make sure your host PC supports 1Gbps or connect a 1Gbps network switch between the UEIPAC and your host PC.

**NOTE:** After uploading the RAM disk using `tftp`, the number of bytes transferred will print in the stdio messages. Note the number of bytes transferred.

5. Upload RAM disk:
   `tftp 4000000 uRamdisk-x.y.z`
   Console messages similar to the following will display in the serial window:

```
=> tftp 4000000 UserRamdisk-4.0.0
from server 192.168.100.59; our IP address is 192.168.100.2 Filename
'UserRamdisk-4.0.0'.
Load address: 0x4000000
Loading:
*.###################################################################
###################################################################
<...>
############
done
Bytes transferred = 14991807 (e4c1bf hex)
```

**NOTE:** The UEIPAC-XXX-03 models have a different start address than the other 8347-based models.

- **Use step 6** for UEIPAC-300-1G, UEIPAC-600-1G, UEIPAC-700-1G, UEIPAC-600R, UEIPAC-1200R, UEIPAC-400-MIL, UEIPAC-1200-MIL, UEINET-PAC, UEIPAC-400F & UEIPAC-xxx-02 models.
- **Use step 7** for UEIPAC-xxx-03 models.
- **Skip step 6&7** for models based on 5200 CPU (UEIPAC-300, UEIPAC-600, & UEIPAC-700)

6. On standard -1G Cubes, -R/F RACKs, -MIL & -02 models
   but NOT -03 models or UEIPAC-300/600/700 models,
   erase flash sectors and copy the RAM disk to flash (at start address FE300000):

   **NOTE:** The parameters for the flash erase command include the start address of the first sector and the end address of the last sector to erase. Flash sectors are erased in 128kB chunks. The start address is a constant, and the end address of the last sector is calculated based on the image size:

   **Start address**: FE300000

   **Image size**: number of bytes uploaded to the RAM disk in step 5. The image size is printed to the screen when the image is finished uploading. In the example in step 5, the image size is ***e4c1bf*** hex:

   a. Add the image size to the base address to calculate the end address of the image.
      For example:
      FE300000(base address) + E4C1BF(image size) = FF14C1BF(end address)

   b. Round the end address to the nearest end sector boundary. Since sectors are in 128kB chunks (from 00000 to 1FFFF hex), the end sector boundary can be calculated by performing a logical OR of 1FFFF on the address calculated in step a.
      For example:
      FF14C1BF(end address) OR 1FFFF = FF15FFFF(end sector boundary for image)

   c. Erase the sectors from the base address to the end sector address:
      ```
      erase fe300000 ff15ffff
      ```

   d. Copy image from RAM to flash:
      ```
      cp.b 4000000 fe300000 ${filesize}
      ```

7. On UEIPAC XXX-03 models only, erase flash sectors and copy the RAM disk to flash (at start address F8300000):

   **NOTE:** The parameters for the flash erase command include the start address of the first sector and the end address of the last sector to erase. Flash sectors are erased in 128kB chunks. The start address is a constant, and the end address of the last sector is calculated based on the image size:

   **Start address**: F8300000

   **Image size**: number of bytes uploaded to the RAM disk in step 5. The image size is printed to the screen when the image is finished uploading. In the example in step 5, the image size is ***e4c1bf*** hex:

   a. Add the image size to the base address to calculate the end address of the image.
   For example:
   F8300000(base address) + E4C1BF(image size) = F914C1BF(end address)

   b. Round the end address to the nearest end sector boundary. Since sectors are in 128kB chunks (from 00000 to 1FFFF hex), the end sector boundary can be calculated by performing a logical OR of 1FFFF on the address calculated in step a.
   For example:
   F914C1BF(end address) OR 1FFFF = F915FFFF(end sector boundary for image)

   c. Erase the sectors from the base address to the end sector address:

   ```
   erase f8300000 f915ffff
   ```

   d. Copy image from RAM to flash (enter ${filesize} exactly as written):

   ```
   cp.b 4000000 f8300000 ${filesize}
   ```

8. Update the `bootargs` variable to tell the kernel that its root file system is a RAM disk (`ramdisk_size` is required if the RAM disk size grows past 64MB):

   For UEIPAC 300, UEIPAC 600, UEIPAC-700 models (based on 5200 CPU):

   ```
   setenv bootargs console=ttyPSC0,57600
   ramdisk_size=<your RAM disk size> root=/dev/ram0 rw
   ```

   For all other UEIPACs (1G Cubes and R/F RACK models based on 8347 CPU):

   ```
   setenv bootargs console=ttyS0,57600
   ramdisk_size=<your RAM disk size> root=/dev/ram0 rw
   ```

9. Change boot command to unpack the RAM disk in memory before starting the kernel:

   For UEIPAC-300, UEIPAC-600, UEIPAC-700 models (based on 5200 CPU),
   RAM disk must be loaded from RAM

   ```
   setenv bootcmd bootm ffc10000 4000000
   ```

   For UEIPAC-600R/1200R, UEIPAC-300/600/700-1G, UEIPAC-400/1200-MIL, UEINET-PAC, UEIPAC-400F, and UEIPAC-XXX-02 versions
   (1G Cubes, R/F RACKs, and -02 models all based on 8347 CPU),
   RAM disk can be loaded from flash

   ```
   setenv bootcmd bootm fe000000 fe300000
   ```

   For UEIPAC-XXX-03 versions
   (all -03 versions of the 1G Cubes, R/F RACKs models based on 8347 CPU),
   RAM disk can be loaded from flash

   ```
   setenv bootcmd bootm f8000000 f8300000
   ```

10. Save environment to make changes permanent and reset:

    ```
    saveenv
    reset
    ```

### 3.2.5 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.

### 3.2.6 Revert to booting from an SD card

Follow the steps below to revert back to the default of booting from the SD card.

1. Power-up your UEIPAC and press any key to enter U-Boot.

2. Update the `bootargs` variable to tell the kernel that its root file system is an SD card:
   For UEIPAC-300, UEIPAC-600, & UEIPAC-700 models (based on 5200 CPU):
   Type:
   ```
   setenv bootargs console=ttyPSC0,57600 root=62:1 rw
   ```
   For all other UEIPACs (1G Cubes and R/F RACK models, and all -02 and -03 versions based on 8347 CPU):
   Type:
   ```
   setenv bootargs console=ttyS0,57600 root=62:1 rw
   ```

3. Change the boot command to start kernel from the SD card:
   For UEIPAC-300, UEIPAC-600, & UEIPAC-700 models (based on 5200 CPU):
   Type:
   ```
   setenv bootcmd bootm ffc10000
   ```
   For UEIPAC-600R/1200R, UEIPAC-300/600/700-1G, UEIPAC-400/1200-MIL, UEINET-PAC, UEIPAC-400F, and UEIPAC-XXX-02 versions
   (1G Cubes, R/F RACKs, and -02 models all based on 8347 CPU),
   Type:
   ```
   setenv bootcmd bootm fe000000
   ```
   For UEIPAC-XXX-03 versions
   (all -03 versions of the 1G Cubes, R/F RACKs models based on 8347 CPU):
   Type:
   ```
   setenv bootcmd bootm f8000000
   ```

4. Save environment to make changes permanent and reset:
   ```
   saveenv
   reset
   ```

## *3.3 Configuring the Network*

### 3.3.1 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 <IP address>
```

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

#### *3.3.1.1 Configuring the primary Ethernet port*

Some UEIPACs are equipped with dual Ethernet controllers (UEIPAC-600R/1200R, UEIPAC-300/600/700-1G, UEIPAC-400/1200-MIL, UEINET-PAC, UEIPAC-400F, UEIPAC-XXX-02 & UEIPAC-XXX-03). Those two Ethernet ports are identical from the hardware point of view.

**eth0** (NIC1) is the primary port and **eth1** (NIC2) is the auxiliary port by default. You can change the role of each port with the procedure below:

1. Stop network service:
   ```
   ~# /etc/init.d/network stop
   ```

2. Edit **/etc/network.conf** and replace the line *DEVICE=eth0* with *DEVICE=eth1*

3. Start network service:
   ```
   ~# /etc/init.d/network start
   ```

#### *3.3.1.2 Configuring the auxiliary Ethernet port*

Note that **`setip`** only configures the primary port (eth0)

To configure the auxiliary port (eth1), use **ifconfig**:

```
ifconfig eth1 <IP address>
```

Insert the **ifconfig** command in **/etc/rc.local** to make the change permanent upon reboot.

Note that you shouldn’t configure both Ethernet ports to be on the same subnet (for example eth0:192.168.100.2 and eth1:192.168.100.3). This will confuse the kernel packet routing.

### 3.3.2 Changing the default packet size (MTU)

You can change the MTU parameter for an ethernet port (default MTU is 1500 bytes) with the **ifconfig** command.

For example, to change MTU for eth0 to 9000 bytes:

```
  ifconfig eth0 mtu 9000
```

The command will generate an **Invalid Argument** message if you set the MTU value too high. The highest value tolerated on current hardware is 9500 bytes.

Insert the command in **/etc/rc.local** to make the change permanent upon reboot.

### 3.3.3 Configuring dynamic IP address (using a DHCP server)

If you have a DHCP server available, you can configure the UEIPAC to automatically fetch an IP address when it boots up:

- Edit the file /etc/network.conf file and change the line:

  ```
  DHCP=no
  ```

  to:

  ```
  DHCP=yes
  ```

Setting DHCP=yes to enable DHCP configures eth0.

To configure DHCP on eth1 for UEIPACs equipped with dual Ethernet controller (UEIPAC-600R/1200R, UEIPAC-300/600/700-1G, UEIPAC-400/1200-MIL, UEINET-PAC, UEIPAC-400F, UEIPAC-XXX-02 or UEIPAC-XXX-03), use `udhcpc`:

```
udhcpc –i eth1 –s /etc/udhcp/default.script
```

After configuring DHCP on eth0 or eth1, you must restart the network to activate changes:

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

Note that automated discovery mechanisms are not supported. After DHCP is configured and the UEIPAC is restarted, you can learn the dynamically assigned IP address by connecting to the UEIPAC over the serial connection and typing `ifconfig` at the serial prompt.

Alternatively, you could configure the DHCP server to assign a fixed address to the UEIPAC or you could assign a hostname using the `udhcpc -x` option:

```
udhcpc -i eth1 -x hostname:<newHostName> -s /etc/udhcp/default.script
```

Once the hostname is configured, you can use that name to log into the UEIPAC:

```
$telnet <newHostName>.<yourDomain>
```

For example, to assign a `myPAC` hostname to eth1 that will be permanent upon reboot, edit the /etc/rc.local file and add the `udhcpc` command with the `-x` option:

```
udhcpc -i eth1 -x hostname:myPAC -s /etc/udhcp/default.script
```

Upon reboot, you connect to the UEIPAC using the `myPAC` hostname and your domain:

```
$telnet myPAC.ueidaq.com
Trying 192.168.0.227…
Connect to myPAC.ueidaq.com.

ueipac login:
```

### 3.3.4 Name resolution

If your UEIPAC uses a static address, you need to edit the /etc/resolv.conf file 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 immediately.

### 3.3.5 Connecting through Telnet

Once the IP address is configured, you can connect to the UEIPAC over the network instead of the serial port if you choose. Telnet is a network protocol that allows access between your host PC and the UEIPAC using the exact same command line interface as the serial connection.

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

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

Type the “exit” command to logout.

### 3.3.6 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.

You can avoid typing the password each time you login using SSH keys:

1. Create private and public SSH keys on your host PC
   ```
   ssh-keygen –t rsa
   ```

2. Copy the public key to /.ssh on the UEIPAC
   ```
   scp ~/.ssh/id_rsa.pub root@<IP address>:/.ssh/authorized_keys
   ```

   **NOTE:** If your UEIPAC does not have the ~/.ssh directory, the `scp` will display an error message stating the directory does not exist. If that happens, telnet or SSH onto the UEIPAC, and type: `mkdir ~/.ssh`
   and then type the `scp` command again on your host PC.

3. You can now log on the UEIPAC using SSH without password

### 3.3.7 Configuring DHCP server

The UEIPAC comes with the minimal DHCP server, **udhcpd**. You can use it when the UEIPAC is a server to assign IP addresses to clients. This is useful when configuring UEIPAC as a wifi access point so that it can assign IP addresses to the wifi devices connecting to the access point.

Create an /etc/udhcpcd.conf file to specify the network interface that will lease the IP addresses and the block of IP addresses that will be leased.

```
# The start and end of the IP lease block
start          192.168.2.20
end            192.168.2.254

# The interface that udhcpd will use
interface eth0
```

## 3.4 Configuring date and time

### 3.4.1 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 date 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 one of the following commands:

```
date –s MMDDhhmm
date –s YYYYMMDDhhmm.ss
```

As an example, “date –s 06021405” sets 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 –u
```

### 3.4.2 Changing the time zone

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

As an example, type the following to set `TZ`:

```
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 /etc/profile file.

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

### 3.4.3 Connecting to an NTP server

The "`rdate`" utility can be used to retrieve the time from an NTP server.

The following command prints the time returned by the NTP server:

```
rdate –p <NTP server IP address>
```

The following command changes the UEIPAC 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 -u
```

## 3.5 Changing the password

Type the following command and enter your new password two times when prompted:

```
passwd
```

You can now logout and login with your new password.

## 3.6 Configuring the web server

The UEIPAC comes enabled with a simple web server. HTML pages can be copied to the folder **/www** to make them accessible from a remote web browser.

## 3.7 System logger

UEIPAC is configured with the system logger disabled by default to avoid unnecessary access to the file system.

You can enable the system logger after adding the **`syslogd`** command to **/etc/rc.local**: Log messages will be written to the **/var/log/messages** file.

You can also enable the kernel logger to log all kernel messages (which are by default printed on the serial console) after adding the **`klogd`** command to **/etc/rc.local**

Lastly, to write your own messages to the system logger, include **<syslog.h>** in your program and call the POSIX APIs **openlog(), syslog(),** and **closelog()**.

Note that **`syslogd`** won’t work on a read-only file system because it needs to create a socket in /dev (/dev/log). A solution to this issue is to create a symbolic link named /dev/log that references a file in the /tmp folder.

```
  ln –s /dev/log /tmp/.devlog
```

# 4 Transferring files

You can use any of the NFS, FTP, SSH or TFTP protocols 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". Logging in as root provides read and write access to the entire file system.

## 4.4 SSH

The UEIPAC also comes with the **dropbear** SSH server 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>
```

## 4.6 Windows shared directory

To mount a directory shared by a Windows computer or a Network Attached Storage (NAS), do the following:

1. Load the **cifs** kernel module:

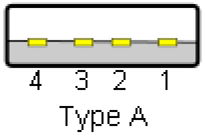
```
modprobe cifs
```

2. Mount the network share:

```
mount –t cifs //hostip/share /mnt -o username=<user>,password=<pass>
```

# 5 Connecting USB devices

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



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. They range from the smallest USB flash drive to enclosures used to connect ATA or SATA hard-drives.

The UEIPAC supports all USB mass storage devices that 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
```

Load the USB storage kernel driver with the command below:

```
~# modprobe usb-storage
```

Note that you must append the string **usb-storage** at the end of the **/etc/modules** file to automatically load this kernel module at boot time.

This should display kernel messages similar to the messages below:

```
[  288.462755] Initializing USB Mass Storage driver...
[  288.473169] scsi0 : usb-storage 1-1:1.0
[  288.482325] usbcore: registered new interface driver usb-storage
[  288.494529] USB Mass Storage support registered.
[  289.483586] scsi 0:0:0:0: Direct-Access     SanDisk  Cruzer
8.02 PQ: 0 ANSI: 0 CCS
[  289.503867] sd 0:0:0:0: [sda] 62562239 512-byte logical blocks:
(32.0 GB/29.8 GiB)
[  289.522154] sd 0:0:0:0: [sda] Write Protect is off
[  289.532494] sd 0:0:0:0: [sda] No Caching mode page present
[  289.543548] sd 0:0:0:0: [sda] Assuming drive cache: write through
[  289.559485] sd 0:0:0:0: [sda] No Caching mode page present
[  289.570534] sd 0:0:0:0: [sda] Assuming drive cache: write through
[  289.585852]  sda: sda1
[  289.594927] sd 0:0:0:0: [sda] No Caching mode page present
[  289.605996] sd 0:0:0:0: [sda] Assuming drive cache: write through
[  289.618236] sd 0:0:0:0: [sda] Attached SCSI removable disk
```

Note the device node name assigned to this USB device is in the format “sdxn”:

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

The kernel message above shows 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 unmount the file system before unplugging 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

### 5.2.1 Load kernel modules

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

```
modprobe rtl8187
modprobe rt2x00usb
modprobe rt2500usb
modprobe rt73usb
```

Wifi network interface names follow the format of **wlan0**, **wlan1,** etc.

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

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

```
lo        no wireless extensions.

eth0      no wireless extensions.

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
```

### 5.2.2 Connection to an open access point

To connect to an open access point, use the following procedure:

1. Specify that you want to connect as a client to a network with an access point:
   ```
   iwconfig wlan0 mode managed
   ```

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

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

4. You can now scan the access points accessible by your wifi interface:
   ```
   iwlist wlan0 scan
   ```

5. If there is a DHCP server on your network, get an IP address for your wifi interface:
   ```
   udhcpc –i wlan0 –s /etc/udhcp/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
   ```

### 5.2.3 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 uses 26 hex characters, and 64-bit WEP uses 10.

### 5.2.4 Connection to an access point with WPA/WPA2 security

1. Generate the pre-shared key using the password of the access point.
   ```
   wpa_passphrase <name of your access point> <access point password>
   ```

2. Edit the **/etc/wpa_supplicant.conf** file and update the following fields:
   - **ssid** : the ID of your access point
   - **psk** : the pre-shared key generated with **wpa_passphrase**
   - **proto** : **WPA** for WPA security and **RSN** for WPA2 security
   - **key_mgmt** : **WPA-PSK**
   - **pairwise** : **TKIP** for WPA and **TKIP CCMP** for WPA2
   - **group** : **TKIP** for WPA and **TKIP CCMP** for WPA2

The following is an example **/etc/wpa_supplicant.conf** file:

```
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 pre-shared key generated with wpa_passphrase here>
  priority=2
 }
```

3. 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
```

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

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

5. 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
```

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

```
udhcpc –i wlan0 –s /etc/udhcp/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
```

### 5.2.5 Direct connection to another computer in ad-hoc mode

1. Specify that you want to connect in ad-hoc mode:
   ```
   iwconfig wlan0 mode ad-hoc
   ```
2. Set the ESSID of the access point:
   ```
   iwconfig wlan0 essid <name of your access point>
   ```

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

4. If there is a DHCP server on your network, get an IP address for your wifi interface:
   ```
   udhcpc –i wlan0 –s /etc/udhcp/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 and supports USB modems connected to the UEIPAC USB port and embedded mini PCI Express modems connected to a CAR-550 carrier card.

Information in this section is based on using a Sierra wireless MC8790 card, which offers UMTS/HSPA and quad-band GSM/GPRS/EDGE network access for roaming on high-speed networks worldwide.

### 5.3.1 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).

### 5.3.2 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.

#### 5.3.2.1 *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
```

#### 5.3.2.2 *Configure provider*

The system is pre-configured to connect to AT&T network. If you are using a different provider, edit the /etc/ppp/peers/gsm_chat file:

1. Look for the following line:
   ```
   OK      'AT+CGDCONT=1,"IP","ISP.CINGULAR"'
   ```

2. Replace it with the Access Point Name (APN) 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"'
   ```

See Table 1 for example APNs for several European countries.

**Table 1 Examples of Providers & Access Point Names**

| Country | Provider | APN | Authentication | Phone Number | User | Password |
|---|---|---|---|---|---|---|
| Austria | A1 | at+cgdcont=1,"IP","a1.net" | PAP/CHAP | *99***1# | ppp@A1net.at | ppp |
| Belgium | Mobistar | at+cgdcont=1,"IP","web.pro.be" | Terminal based | *99# | mobistar | mobistar |
| France | Orange | at+cgdcont=1,"IP","orange.fr" | Terminal based | *99***1# | orange | orange |
| Germany | D2 Vodafone | at+cgdcont=1,"IP","web.vodafone.de" | PAP/CHAP | *99***1# | none | none |
| Netherlands | KPN | at+cgdcont=1,"IP","internet" | Terminal based | *99***1# | Internet | none |
| Netherlands | Orange | at+cgdcont=1,"IP","internet","",0,0 | Terminal based | *99***1# | none | none |
| Netherlands | Vodafone | at+cgdcont=1,"IP","web.vodafone.nl" | Terminal based | *99# | vodafone | vodafone |

### *5.3.2.3 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 `dmesg` command 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)
```

Verify that ppp0 was assigned an IP address.

You can now connect to the internet from your UEIPAC.

### 5.3.3 Automatic startup

To automatically load the kernel modules, edit the /etc/modules file 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 /etc/rc.d directory using the following command:

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

## 5.4 Serial Port

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

### 5.4.1 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. Refer to the **SampleLinuxSerialPort** sample for example C code showing how to program a standard Linux serial port. Sample code can be found in the following directory:

- `< UEIPAC SDK directory>/sdk/examples`

### 5.4.2 Automatic startup

To automatically load the kernel modules, edit the /etc/modules file 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

### 5.5.1 Prerequisite

LibUSB uses usbfs which is 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.

#### *5.5.1.1 Mount USBFS manually*

Type the following command to mount USBFS:

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

#### *5.5.1.2 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
```

### 5.5.2 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.

# 6 Serial Port

## 6.1 UEI Serial Server

UEI Serial Server makes PowerDNx serial devices (such as SL-501 and SL-508) accessible as standard Linux serial ports that can be programmed using the POSIX termios API.

The mapping configuration file is a text file with a [settings] section for global parameters and a [ttyUEI??] section for each mapped serial port.

For example:

```
[settings]
timeoutms=1000
retrycount=4
pollperiodms=10

[ttyUEI0]
ipAddress=127.0.0.1
device=2
port=0
#mode: 0=RS-232, 1=RS-485HD, 2=RS-485-FD
mode=0
baudRate=9600
#parity: 0=none, 1=odd, 2=even
parity=0
#stop Bits: 0=no stop bit, 1=1 stop bit, 2=1.5 stop bit
stopBits=0
#data bits: 5,6,7 or 8 data bits
dataBits=8

[ttyUEI1]
ipAddress=127.0.0.1
device=2
port=1
mode=0
baudRate=57600
parity=1
stopBits=1
```

```
    dataBits=7
```

This example configuration file configures the serial server to return an error if it cannot communicate with the IOM after **timeoutms** milliseconds.
The server can retry communication for **retrycount** times before giving up.
The server will periodically poll serial ports for new incoming data using the **pollperiodms** value to specify the period in milliseconds.

This file creates two virtual serial ports /dev/ttyUEI0 and /dev/ttyUEI1 to control physical ports 0 and 1 on device 2 located on the UEIPAC.

- **/dev/ttyUEI0** is configured to run at 9600 bits per sec, no parity, no stop bits and 8 data bits
- **/dev/ttyUEI1** is configured to run at 57600 bits per sec, parity odd, 1 stop bits and 7 data bits

Note that the communication settings are only default values. The serial port will be re-configured to use whatever communication settings you specify when opening the port from your application.

Run the serial server with the following command:

```
   ueiserialserver <config file name>
```

Once the server is started, you can use the **/dev/ttyUEI??** nodes like any other serial port with the termios API or any other program designed to access serial ports.

The UEIPAC comes with **microcom** installed on the SD card. You can run microcom to test the serial ports.

Start the serial server with at least two configured ports: /dev/ttyUEI0 and /dev/ttyUEI1

For the following example, we will assume that the two serial ports are connected with a NULL modem cable.

Open two separate command line shells and start the **microcom** program for each of the Serial ports you wish to test:

```
  microcom -s 19200 /dev/ttyUEI0

  microcom -s 19200 /dev/ttyUEI1
```

If both serial ports are tied with a NULL modem cable, anything you type in one of the session will appear on the other session.

## 6.2 Using the CPU layer’s serial port for general purpose

To use the CPU layer’s serial port for general purpose, edit the /etc/inittab file 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.

# 7 Testing the I/O layers

## 7.1 devtbl

The `devtbl` command will print a list of the I/O layers that are detected in the 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 examples are 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 at least one example for each supported I/O layer. Sample code examples are named "SampleXXX" (where XXX is the model ID of each layer).

To access the samples, change directory to "<UEIPAC SDK directory>/sdk/examples" and copy the chosen example to your UEIPAC using one of the methods described in section 4, (e.g., telnet, ssh, etc.).

For example using FTP:

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

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

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

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

```
/tmp # ./Sample207 -d 2 -c "3,5,7"
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 or 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 PowerDNA C API.
It only works in immediate, RTDMAP and RTVMAP modes. ACB, Messaging and Asynchronous modes are not supported.

PowerDNA server is automatically started by inetd.
Inetd monitors incoming packets and automatically starts the appropriate server (FTP, WWW, SSH and PDNA).

If you need to stop the PowerDNA server from automatically starting, edit `/etc/inetd.conf` and comment out the pdnaserver line. You can also disable other network services this way (FTP, SSH, etc.).

# 8 Application development

## 8.1 Prerequisites

Verify the “<UEIPAC SDK directory>/powerpc-604-linux-gnu/bin” directory is added to your PATH environment variable. This will allow you to invoke the GCC cross compiler without having to specify its full path.

The cross compiler 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).

Note that application development on the UEIPAC uses the PowerDNA API library. In conjunction with this chapter, please refer to the “PowerDNA API Reference Manual” for API descriptions.

## 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-gcc”.

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 or RACK:

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

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 in the console window.

## 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”. The debugger 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-absolute-prefix <UEIPAC Install Dir>
set solib-search-path <UEIPAC Install Dir>/powerpc-604-linux-gnu/
powerpc-604-linux-gnu/lib
```

   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.

The program is now in “running” state and GDB paused its execution. The following debug actions are available:

6. Insert a breakpoint at the beginning of the “main” function:
   `break main`
7. Resume execution with the `cont` command (GDB will pause the execution again when entering the “main” function).
8. 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 connects to the target and pauses the execution in the main function each time you start gdb:

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

To learn how to fully use the GDB debugger, the GDB documentation is available at [http://sourceware.org/gdb/documentation/](http://sourceware.org/gdb/documentation/).

## 8.4 PowerDNA Library

The PowerDNA library implements the API used to program the PowerDNA I/O layers:

- Source code is installed in "<UEIPAC SDK directory>/sdk/DAQLib".
- Examples are located in "<UEIPAC SDK directory>/sdk/examples".
- Documentation is located in "<UEIPAC SDK directory>/doc".

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:

| Mode | Description |
|---|---|
| **Immediate** | This 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 cannot achieve maximum performance with this mode<br>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.<br>DMAP mode examples are named "SampleDMapXXX" |
| **Buffered (ACB)** | Allows access to AI, AO, DIO and CT layers at full speed.<br>It is designed to correct communication errors that might happen on the network link. The error correction mechanism will cause issues with real-time deadlines<br>ACB mode examples are named "SampleACBXXX" |

| Mode | Description |
|---|---|
| **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 mechanism will cause issues with real-time deadlines<br>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.<br>VMAP mode examples are named “SampleVMapXXX” |
| **Autonomous Variable Size Data Mapping (AVMAP):** | Allows access to all layers at full speed, transferring incoming and outgoing data in buffers in one call.<br>AVMAP mode examples are named “SampleAVMapXXX” |
| **Asynchronous** | Allows I/O layers to asynchronously notify the user application upon hardware or board-specific events<br>Asynchronous mode examples are named “SampleAsyncXXX” |

The UEIPAC SDK supports the immediate (also known as “point by point”), DMAP, VMAP, and AVMAP modes to control the “local” layers and Async mode on select layers, as well.

The other modes (ACB and MSG) 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 run 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 |

| | Firmware Running on the IO Module | | |
|---|---|---|---|
| MSG | Yes | No | Yes |
| VMAP | Yes | Yes | Yes |
| AVMAP | Yes | Yes | Yes |
| Asynchronous | *Select I/O layers (refer to sample code) | *Select I/O layers (refer to sample code) | *Select I/O layers (refer to sample code) |

### 8.4.1 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.

#### *8.4.1.1 Initialization, miscellaneous API*

The following 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

#### *8.4.1.2 Offline Data Conversion API*

UEI recommends logging acquired data in its raw format to save UEIPAC CPU cycles, and then later scaling the data on a host PC.

Offline data conversion APIs include all DqConv*** API. These offline data conversion API are available with UEIPAC versions 3.4.2 and later.

#### *8.4.1.3 Immediate mode API*

Immediate Mode 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.

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

#### *8.4.1.4 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 that use RTDMAP mode are called DqRtDmap***.

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

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

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

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

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

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

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

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

8. Stop the devices and free all resources:

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

#### 8.4.1.5 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 layers' FIFOs is accessed in one operation.

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

The following is a quick tutorial on using the RTVMAP API (handling of error codes is omitted):

1. Initialize the VMAP to acquire/generate data at 1kHz:
   ```
   DqRtVmapInit(handle, vmapid, 1000.0);
   ```

2. Add channels from the first set of input ports of a messaging layer (serial, 1553, ARINC, etc.) at device 0 as follows:
   ```
   //configure input ports 0, 1, 2, and 3
   int channels[] = {0, 1, 2, 3 };
   int flags[] = {0, 0, 0, 0 };

   DqRtVmapAddChannel(handle, vmapid,0,DQ_SS0IN,channels,flags,4);
   ```

3. Add channels from the first analog input, analog output, and/or digital I/O layer (analog input (input subsystem) in this example), for device 1, as follows:
   ```
   //initialize a VMAP channel, which for AI/AO/DIO layers is a
   // virtual channel streaming all physical channels (interleaved)

   int vmapChannel = 0; // the actual value doesn't matter
   int flag = 0;

   DqRtVmapAddChannel(handle, vmapid, 1, DQ_SS0IN, &vmapChannel,
   &flag, 1);

   //add the input channels 0, 1, 2, 3, and 4
   int channels[] = {0, 1, 2, 3, 4 };
   int flags[] = {0, 0, 0, 0, 0 };

   DqRtVmapSetChannelList(handle, vmapid, 1, DQ_SS0IN, channels,
   flags, 5);
   ```

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

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

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

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

   **NOTE**: On UEIPAC versions 3.4.2 and later, users can log data in a raw format to save CPU cycles and later scale the data offline on the host PC.
   Refer to `DqConv***` data conversion API for descriptions.

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

#### *8.4.1.6 AVMAP API*

In AVMAP mode, the UEIPAC continuously updates data in buffers.
Each input layer is programmed to acquire data from its internal FIFO at a rate paced by its hardware clock.
The user application is notified when a user-programmable amount of data is in the FIFO and ready for retrieval.

The content of all the layers' FIFOs is accessed in one operation.

Supported APIs to use AVMAP mode are DqRtVmap***.

The following is a quick tutorial on using the AVMAP API (handling of error codes is omitted):

1. Initialize the VMAP to acquire/generate data at 1kHz:
```
DqRtVmapInit(handle, vmapid, 1000.0);
```

2. Add channels from the first input subsystem of device 0 (for analog I/O boards, the last parameter is “1” which represents 1 FIFO holding data for all channels):
```
int channels[] = {0, 1, 2, 3 };
DqRtVmapAddChannel(handle, vmapid, 0, DQ_SS0IN, channels, flags,
1);
```

3. For analog input boards, add the number of actual channels per I/O board from the first input subsystem of device 0:
```
// in header
#define CHANNELS 4
//in main program
DqRtVmapSetChannelList(handle, vmapid, 0, DQ_SS0IN, channels,
CHANNELS);
```

4. Set up scan rate for individual device/subsystem:
```
DqRtVmapSetScanRate(handle, vmapid, 0, DQ_SS0IN, 1000.0);
```

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

6. Set FIFO watermark level to trigger data transfer:
```
wtrmrk = 10;
DqRtAsyncVMapSetWatermark (handle, 0, vmapid, wtrmrk);
```

7. Enable the AVMAP:
```
DqRtAsyncXMapEnable(handle, vmapid);
```

8. Issue software trigger to trigger board:
```
DqCmdSwTrigger(handle, vmapid);
```

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

10. Retrieve data acquired by device, this call blocks until there is enough data in the FIFO:
```
DqRtVmapRefresh(handle, vmapid, 0);
```

11. Get retrieved data acquired by device 0:
```
DqRtVmapGetInputData(handle, vmapid, 0, numScans*sizeof(uint16),
&data_size, &avl_size, (uint8*)bdata);
```

**NOTE**: On UEIPAC versions 3.4.2 and later, users can log data in a raw format to save CPU cycles and later scale the data offline on the host PC.
Refer to `DqConv***` data conversion API for descriptions.

12. Stop the devices and free all resources:

```
DqRtAXMapEnable(handle, FALSE);
DqRtVmapStop(handle, vmapid);
DqRtVmapClose(handle, vmapid);
```

### 8.4.1.7 Asynchronous Event API

The Asynchronous API allows you to get notified in your application when a user-programmable asynchronous event occurs.

The types of events supported varies depending on which type of I/O board you are configuring.

The following are examples of events supported for different I/O board types:

| **Type of I/O Board** | **Type of Asynchronous Events**[3] | **Examples of Supported Boards** |
|---|---|---|
| Digital I/O Boards | Edge detection / Change of state and Periodic Events | DNx-DIO-401/3/4/5/6, DNx-DIO-449 |
| Serial I/O Boards | Timeout conditions, RX or TX done, and pattern detection Events | DNx-SL-501, DNx-SL-508 |
| CAN-Bus Boards | FIFO watermark and bus error Events | DNx-CAN-503 |
| Counter/Timer Boards | Count complete Events | DNx-CT-601, DNx-CT-602 |
| Avionics Protocol Boards | Many protocol-specific Events | DNx-1553-553,<br>DNx-429-516 |

The following is a quick tutorial on using the event API for a digital I/O board, located as device 0 in the IOM and programmed to read the states of all digital input pins on a rising or falling change of state (COS) on port 0 pin 0 (handling of error codes is omitted):

1. Open asynchronous communication with device drivers:
   `DqAddIOMPort(hd, &a_handle, DQ_UDP_DAQ_PORT_ASYNC, 1000);`

2. Set up channels as inputs or outputs
   (this example uses all channels as inputs, which is the default; otherwise, configure I/O directionality: for example, for a DIO-403 use `DqAdv403SetIo()`).

3. Initialize events of device 0 by clearing buffer:
   `DqAdv403ConfigEvents(a_handle, 0, EV403_CLEAR, 0, 0);`

[3] Refer to SampleAsyncXXX sample code for descriptions of types of events supported and specific usage.

4. Configure a COS event on a rising or falling edge on port 0 pin 0 for device 0:

```
uint8 pos_edge_masks[DQ_DIO403_PORTS] = { 0, 0, 0, 0, 0, 0 };
uint8 neg_edge_masks[DQ_DIO403_PORTS] = { 0, 0, 0, 0, 0, 0 };
pos_edge_masks[0] = 1 << 0;
neg_edge_masks[0] = 1 << 0;
DqAdv403ConfigEvents(a_handle, 0, EV403_DI_CHANGE, pos_edge_masks,
neg_edge_masks);
```

   NOTE: To configure periodic events, use the `DqAdv403ConfigEvents32()` API. Also note that each I/O board type has its own `DqAdvXXXConfigEvents()` API that is used for event configuration.

5. Enable events on device 0:

```
DqRtAsyncEnableEvents(a_handle, 0, 1);
```

6. Wait for the next event on device 0. If no event occurs after 1 second, the function returns the error code, "`DQ_TIMEOUT_ERROR`":

```
ret = DqCmdReceiveEvent(a_handle, 0, 1000000, &pEvent, &size);
```

7. Process the event:

   The `DqCmdReceiveEvent()` API returns a `pEvent` structure of size `size`. This structure will vary with each I/O board type. Refer to `DqAdvXXXConfigEvents()` API for descriptions of the members of correlating `pEvent` structures.

   Note that the returned `pEvent` will always contain the event that was received `(pEvent->event)` and any data associated with the event `(pEvent->data[])`.

   The data returned `(pEvent->data[])` may need `ntohl` conversion.

   UEI provides byte order conversion API that detect and convert for the correct endianness. For example, use UEI `DqNtohl()` for the conversion of the timestamp:

```
pEv403 = (pEV403_ID)pEvent->data;
tstamp = DqNtohl(hd, pEv403->tstamp);
```

8. To finish, disable asynchronous events on device 0:

```
DqRtAsyncEnableEvents(hd, 0, 0);
```

9. Close asynchronous communication:

```
DqCloseIOM(a_handle);
```

NOTE: The Asynchronous Event API described in this section was introduced in version 4.0.8. For UEIPAC releases prior to 4.0.8, the UEIPAC only supported the event API described in Appendix A, which only applied to programs running on a UEIPAC, accessing local UEIPAC hardware.

#### *8.4.1.8 Unsupported APIs*

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

### 8.4.2 Building and running the examples

To build and run examples, change your current directory to "<UEIPAC SDK directory>/sdk/examples" and type *make* to make sure that your setup can build the samples correctly.

If you get any errors 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 them.

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

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

```
Sample201 –d 1 –c "0,2,4"
```

### 8.4.3 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 to 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 –l powerdna
myprogram.o –o myprogram
```

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 Synchronization

UEIPACs include hardware and software resources to synchronize one or more chassis to an external source.

UEIPACs can be synchronized using a common pulse-per-second (PPS) reference signal routed via 10-pin sync port at the front of the chassis.

UEIPACs using the Real-time Linux kernel can alternatively be synchronized using the IEEE-1588 Precision Time Protocol (PTP) standard over Ethernet.

You configure PTP or PPS synchronization in your application using APIs that begin with DqSync***. Refer to the PowerDNA API Reference Manual for API descriptions.

### 8.5.1 PTP Synchronization

PTP synchronization is supported UEIPACs equipped with -02 and -03 CPU layers with Logic 02.12.46 (2018) or later and running the Real-time Linux kernel.

The IEEE-1588 PTP standard defines the protocol for establishing a master/slave relationship between a reference clock (the PTP grandmaster) and all other devices in the system that will synchronize their clocks to the master clock (slaves). The master/slave hierarchy is established through an exchange of PTP packets containing clock attributes (clock accuracy, etc.). Once the best master (grandmaster) clock is established, the grandmaster periodically sends synchronization packets containing timestamps to each slave UEIPAC for local clock alignment.

UEIPACs support the following PTP configuration:

- PTP v2 (IEEE 1588-2008)
- PTP over IPv4/UDP (Annex D)
- Multicast transmission mode (unicast point-to-point is not currently supported)
- Hardware timestamping
- Capable of being a PTP slave device (UEIPACs as the PTP grandmaster is not currently supported)
- Capable of connecting via an end-to-end boundary clock (transparent clocks are not currently supported)
- Supported for I/O board synchronization only

Note that the UEIPAC’s real-time clock cannot be synchronized to PTP time. For example, typing `date` at the UEIPAC’s Linux prompt will not return PTP time.

### 8.5.2 External PPS Synchronization

Synchronization to an external PPS reference is supported on UEIPACs having CPU Logic 02.12.2D (2017) or later.

UEIPACs can synchronize to an external pulse per second reference provided by a master PPS source. Typically this is a one pulse per second (1PPS) signal. The 1PPS is routed to each chassis and input over the 10-pin sync port at the front of the UEIPAC.

Once locked to a common 1PPS pulse, UEIPACs can be programmed to generate internal I/O board clocks and triggers for all configured boards in a system, synchronized to the common reference. This allows time alignment among each of the I/O boards on all the synchronized chassis, as well as allowing the synchronization of timestamps.

UEIPACs are capable of internally generating a 1PPS pulse and acting as the 1PPS master or receiving a 1PPS pulse and acting as a 1PPS slave.

## *8.6 Real-Time Programming*

The UEIPAC supports Real-Time Linux and the Xenomai Real-Time framework.

### 8.6.1 Programming with Real-Time Linux

The UEIPAC can be updated to a Real-Time Linux kernel. Real-Time Linux is a single-kernel real-time solution to mainline Linux that removes all unbounded latencies and provides faster response times.

Real-Time Linux is primarily mainline Linux but additionally includes the PREEMPT_RT patch. PREEMPT_RT maximizes preemptible sections of the Linux kernel by reworking the kernel spinlock primitives. Additionally, PREEMPT_RT runs interrupt handlers in kernel threads, allowing threads to be interruptible, prioritized, and preempted.

Mainline Linux is continually incorporating PREEMPT_RT patch components and will eventually include all PREEMPT_RT functionality, making the patch unnecessary in time.

Features that PREEMPT_RT implement to accommodate real-time performance include the following:

- Using high resolution timers (which were included in mainline a few years ago)
- Modifying kernel locking primitives to be preemptible (spinlocks, mutex)
- Converting interrupt handlers into preemptible kernel threads

Real-Time Linux doesn’t require any special toolsets, and users can use a standard C library, a Linux driver, and POSIX application.

Users should be familiar with real-time POSIX for best results.

All mainline Linux examples provided by UEI (i.e., SampleDMAP* and SampleVMAP*) can be used as a base for your real-time application.

To optimize your application for real-time execution, consider the following:

1. Perform non-real-time tasks first, such as opening files, allocating memory, and opening the IOM (calling `DqOpenIOM`).

2. Use a real-time scheduling policy (e.g., `SCHED_FIFO` or `SCHED_RR`). Priority is set as a value from 1 to 99, where higher value priorities are scheduled before lower priority threads.

   ```
   struct sched_param schedp;

   // Configure this process to run with the real-time scheduler
       memset(&schedp, 0, sizeof(schedp));
       schedp.sched_priority = 99;     // highest priority
       sched_setscheduler(0, SCHED_FIFO, &schedp);

   //if you want to use pthreads, consider pthread_setschedparam()
   ```

3. Lock the address space, (e.g., call `mlock` or `mlockall` function):

   ```
   // no memory-swapping for this program
       mlockall(MCL_CURRENT | MCL_FUTURE);
   ```

4. Set up periodic real-time tasks, (e.g., initialization for data collection).

5. Use `CLOCK_MONOTONIC` for timing your real-time tasks.
   - `CLOCK_MONOTONIC` starts ticking when the kernel is started and can’t be adjusted.
   - `CLOCK_REALTIME` is relative to 01/01/1970 and can be adjusted (manually, via NTP or PTP), which will disrupt a program’s timing.

   ```
   struct timespec next;

   // get absolute time at the beginning of the loop
   //     the thread will wake-up periodically at
   //     T1=next+period, T2=next+2*period
   clock_gettime(CLOCK_MONOTONIC, &next);
   ```

```
while (1)
{

   // Add period interval to previous wake-up time; this will
   //  give the ‘next’ wake-up time
   timespec_add_ns(&next, periodns);

   // do your work here

   // set the absolute time at which the kernel will
   //   wake up the thread
   clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &next, NULL);

}
```

6. Enable priority inheritance if needed when using POSIX mutex.

7. Release resources and exit.

### 8.6.2 Programming with Xenomai Real-Time framework

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 also delivered later to the Linux kernel when the real-time kernel is done.
- Upon receiving an interrupt, the real-time kernel can schedule 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 real-time tasks to run 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 by 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 starts in primary domain; however, it will jump to secondary domain (and be scheduled 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.

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: the high priority tasks run 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 Xenomai examples have the suffix **_rt**. For example **Sample207** is a standard Linux sample program while **Sample207_rt** is a Xenomai real-time sample program.

## 8.7 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 completes the 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. This is equivalent to typing CTRL+C on the console if Sample201 was running in the foreground):

```
killall –SIGINT Sample201
```

## 8.8 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 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".

As an 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
```

# 9 Firmware installation and upgrade

## 9.1 Installing or upgrading the Linux kernel

Your UEIPAC comes with a Linux kernel already installed into flash memory.

It is possible to update that Linux kernel if needed.

Note that when installing or upgrading the kernel or firmware through U-Boot and Ethernet, you must connect to the UEIPAC via a Gigabit network switch.

You first need to install a TFTP server on your host PC and copy the new kernel image you received from UEI technical support to the TFTP server's directory.

Kernel image files are named:

| **UEIPAC Product Version** | **Linux Kernel image**<br>**(with Xenomai support)** | **Real-Time Linux Kernel image**<br>**(with PREEMPT_RT patch)** |
|---|---|---|
| UEIPAC-300,<br>UEIPAC-600, &<br>UEIPAC-700 | cuImage.ueipac5200 | N/A |
| UEIPAC-300-1G,<br>UEIPAC-600-1G,<br>UEIPAC-700-1G,<br>UEIPAC-600R,<br>UEIPAC-1200R,<br>UEIPAC-400-MIL,<br>UEIPAC-1200-MIL,<br>UEINET-PAC,<br>UEIPAC-400F,<br>UEIPAC-300-1G-02,<br>(& all other<br>UEIPAC-xxx-02) | cuImage.ueipac834x | cuImage.ueipac834x-rt |
| UEIPAC-300-1G-03<br>(& all other<br>UEIPAC-xxx-03) | cuImage.ueipac834x-03 | cuImage.ueipac834x_rt-03 |

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

That same folder also contains scripts to download the kernel sources and build the kernel yourself, see Appendix G..
To start the installation or upgrade, connect to the UEIPAC through the serial port and power-up the Cube or RACK. Press any key before the 2 seconds countdown ends to enter U-Boot's command line interface.

To continue with your installation or upgrade, follow one of the procedures provided below (choose which procedure based on which UEIPAC you have).

> **NOTE:** U-Boot only supports 1Gbps link speed on 8347 based UEIPACs. Make sure your host PC supports 1Gbps or connect a 1Gbps network switch between the UEIPAC and your host PC.

### 9.1.1 UEIPAC with Freescale 5200 CPU (100 MBit Ethernet)

The following procedure is used to update the kernel for the UEIPAC-300, UEIPAC-600, and UEIPAC-700 systems.

1. Erase the unprotected part of flash memory:
   ```
   erase ffc10000 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.ueipac5200
```

5. Write kernel into flash (make sure you literally type “${filesize}”):

```
cp.b 200000 ffc10000 ${filesize}
```

6. Set U-Boot's boot command to automatically boot Linux:

```
setenv bootcmd bootm ffc10000
```

7. Save environment variables to flash:

```
saveenv
```

8. Reset and boot the new kernel:

```
reset
```

### 9.1.2 UEIPAC and UEIPAC-XXX-02 versions with Freescale 8347 CPU (1GBit Ethernet)

The following procedure is used to update the kernel for the UEIPAC-600R/1200R, UEIPAC-300/600/700-1G, UEIPAC-400/1200-MIL, UEINET-PAC, UEIPAC-400F, and UEIPAC-XXX-02 systems.

1. Erase the unprotected part of flash memory:
   ```
   erase fe000000 fe2fffff
   ```
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
   - For mainline Linux: `tftp 200000 cuImage.ueipac834x`
   - For Real-Time Linux: `tftp 200000 cuImage.ueipac834x-rt`
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 bootcmd bootm fe000000
   ```
7. Save environment variables to flash
   ```
   saveenv
   ```
8. Reset and boot the new kernel:
   ```
   reset
   ```

### 9.1.3 UEIPAC-XXX-03 versions (with Freescale 8347 CPU, 1GBit Ethernet)

The following procedure is used to update the kernel for the UEIPAC-XXX-03 systems.

1. Erase the unprotected part of flash memory:
   ```
   erase f8000000 f82fffff
   ```
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
   - For mainline Linux: `tftp 200000 cuImage.ueipac834x-03`
   - For Real-Time Linux: `tftp 200000 cuImage.ueipac834x_rt-03`
5. Write kernel into flash (make sure you literally type "${filesize}")
   ```
   cp.b 200000 f8000000 ${filesize}
   ```
6. Set U-Boot's boot command to automatically boot Linux
   ```
   setenv bootcmd bootm f8000000
   ```

7. Save environment variables to flash
   `saveenv`

8. Reset and boot the new kernel:
   `reset`

## 9.2 Initializing an SD card

Your UEIPAC is delivered 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.

### 9.2.1 On a Linux PC

Note that 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.

You can either type the commands below to manually format and initialize an SD card or you can run scripts included in the UEIPAC SDK to automate the procedure.

#### *9.2.1.1 Automated Procedure*

The UEIPAC SDK includes scripts to automatically partition and initialize a one or two partition SD card:

- <ueipac sdk dir>/rfs/createsdcard.sh creates one ext2 partitions and copies all system files.
- <ueipac sdk dir>/rfs/createsdcard_2parts.sh creates two ext2 partitions and copies all system files to the first one. The second partition is entirely available to store user data.
- <ueipac sdk dir>/rfs/createsdcard_vfat.sh creates one VFAT and one ext2 partition and copies all system files to the second one (otherwise it confuses windows and you can't read the vfat partition on a windows PC). The first partition is entirely available to store user data.

#### *9.2.1.2 Manual Procedure*

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

2. Find 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 sdb: 1984000 512-byte hdwr sectors (1016 MB)
```

This message tells us that the device node we are looking for is “/dev/sdb”.

3. Unmount the SD card if necessary

```
sudo umount /dev/sdb1
```

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/sdb
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. Unmount the SD card if necessary

```
sudo umount /dev/sdb1
```

6. The device node associated with the partition we just created is “/dev/sdb1”. Format this new partition with **mke2fs** (-j option sets file system type to ext3):

```
sudo mke2fs -j /dev/sdb1
```

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. Verify this with the `df` command) then copy the root file system to the SD card:

```
sudo mount /dev/sdb1 /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/sdb1
```

### 9.2.2 On the UEIPAC itself

Boot the UEIPAC from the RAM disk instead of the SD card (follow instructions detailed in chapter 3.2).

1. Set the IP address:
```
setip <IP address of the UEIPAC>
```
2. Format the SD card:
```
mke2fs -j /dev/sdcard1
```
3. Mount the SD card:
```
mount /dev/sdcard1 /mnt
```
4. Transfer the root file system image to the UEIPAC from a Linux or Windows PC:
```
scp rfs-x.y.z.tgz root@<IP address of UEIPAC>:/mnt
```
5. Uncompress the image:
```
gunzip /mnt/rfs-x.y.z.tgz
tar xvf /mnt/rfs-x.y.z.tar
mv /mnt/rfs/* /mnt
sync
```

## 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 or RACK. Press any key before the 2 seconds countdown ends to enter U-Boot's command line interface.

### 9.3.1 Configure UEIPAC with Freescale 5200 CPU to run DAQBios firmware

Configure UEIPAC-300, UEIPAC-600, & UEIPAC-700 systems with the following:

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
```

### 9.3.2 Configure UEIPAC with Freescale 5200 CPU to run Linux

Configure UEIPAC-300, UEIPAC-600, & UEIPAC-700 systems with the following:

1. Set U-Boot's boot command to start Linux automatically:
```
setenv bootcmd bootm ffc10000
saveenv
```
2. Reset and boot the Linux kernel:
```
reset
```

### 9.3.3 Configure UEIPAC and UEIPAC-XXX-02 with Freescale 8347 CPU to run DAQBios firmware

Configure UEIPAC-600R/1200R, UEIPAC-300/600/700-1G, UEIPAC-400/1200-MIL, UEINET-PAC, UEIPAC-400F, and UEIPAC-XXX-02 systems with the following:

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
   ```

### 9.3.4 Configure UEIPAC and UEIPAC-XXX-02 with Freescale 8347 CPU to run Linux

Configure UEIPAC-600R/1200R, UEIPAC-300/600/700-1G, UEIPAC-400/1200-MIL, UEINET-PAC, UEIPAC-400F, and UEIPAC-XXX-02 systems with the following:

1. Set U-Boot’s boot command to start Linux automatically:
   ```
   setenv bootcmd bootm fe000000
   saveenv
   ```
2. Reset and boot the Linux kernel:
   ```
   reset
   ```

### 9.3.5 Configure UEIPAC-XXX-03 with Freescale 8347E CPU to run DAQBios firmware

Configure UEIPAC-XXX-03 systems with the following:

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
   ```

### 9.3.6 Configure UEIPAC-XXX-03 with Freescale 8347E CPU to run Linux

Configure UEIPAC-XXX-03 systems with the following:

1. Set U-Boot’s boot command to start Linux automatically:
   ```
   setenv bootcmd bootm f8000000
   saveenv
   ```
2. Reset and boot the Linux kernel:
   ```
   reset
   ```

# 10 Third-party software

## 10.1 Third-party libraries installed by default on UEIPAC

The libraries below typically implement C APIs that you can call from your own program.

### 10.1.1 zeromq

ØMQ (also known as ZeroMQ, 0MQ, or zmq) is an embeddable networking library.

### 10.1.2 libmodbus

Libmodbus provides a C API to implement MODBUS/TCP or MODBUS/RTU slaves and masters.

### 10.1.3 expat

Expat is an XML parsing library.

### 10.1.4 sqlite

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.

### 10.1.5 gpsd

gpsd is a utility that can listen to a GPS or AIS receiver and re-publish the positional data in a simpler format.

### 10.1.6 GSL

The GNU Scientific Library (GSL) is a numerical library for C and C++ programmers.

### 10.1.7 libusb

libusb is a C library that gives applications easy access to USB devices.

### 10.1.8 mosquitto

Mosquitto is an open source message broker that implements the MQ Telemetry Transport protocol (MQTT).

### 10.1.9 audiofile

The Audio File Library is a C-based library for reading and writing audio files in many common formats.

## 10.2 Building third-party software with buildroot

1. Download buildroot at www.buildroot.org
2. Run **make menuconfig**
3. Go to Target options menu
   Set Architecture to **PowerPC**
   Set Architecture variant to **603**
4. Go to Toolchain menu
   Set Toolchain type to **external toolchain**
   Set Toolchain path to <Type your UEIPAC SDK install path here>/powerpc-604-linux-gnu
   Set Toolchain prefix to **powerpc-604-linux-gnu**
   Set Toolchain kernel header version to **3.2.x**
   Set External Toolchain version to **6.x**
   Set External Toolchain C Library to **glibc**
   Enable RPC support
   Enable C++ support
5. Go to Target package menu
6. Select the software package(s) you want to build
7. Save and Exit
8. Run **make**

The built files are accessible in the **target** folder.

## 10.3 Building third-party software from source

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 methods of cross-compiling software packages.

### 10.3.1 Software with an autoconf configure script

Most software packages that use autoconf can be configured with the following command on a Linux PC:

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

Use the following command on a Window/Cygwin PC:

```
./configure –-host=powerpc-604-linux-gnu –-build=i686-pc-cygwin
–-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
```

### 10.3.2 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 scripts 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
```

# Appendix A. Event API

## A.1 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. |

## A.2 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”.

## A.3 DqEmbCancelEvent

**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.

# Appendix B. Using Eclipse IDE to program the UEIPAC

## B.1 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. The procedures described in this appendix are based on the Eclipse IDE 2018-09 release; for other releases, the displays and instructions may vary slightly.

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

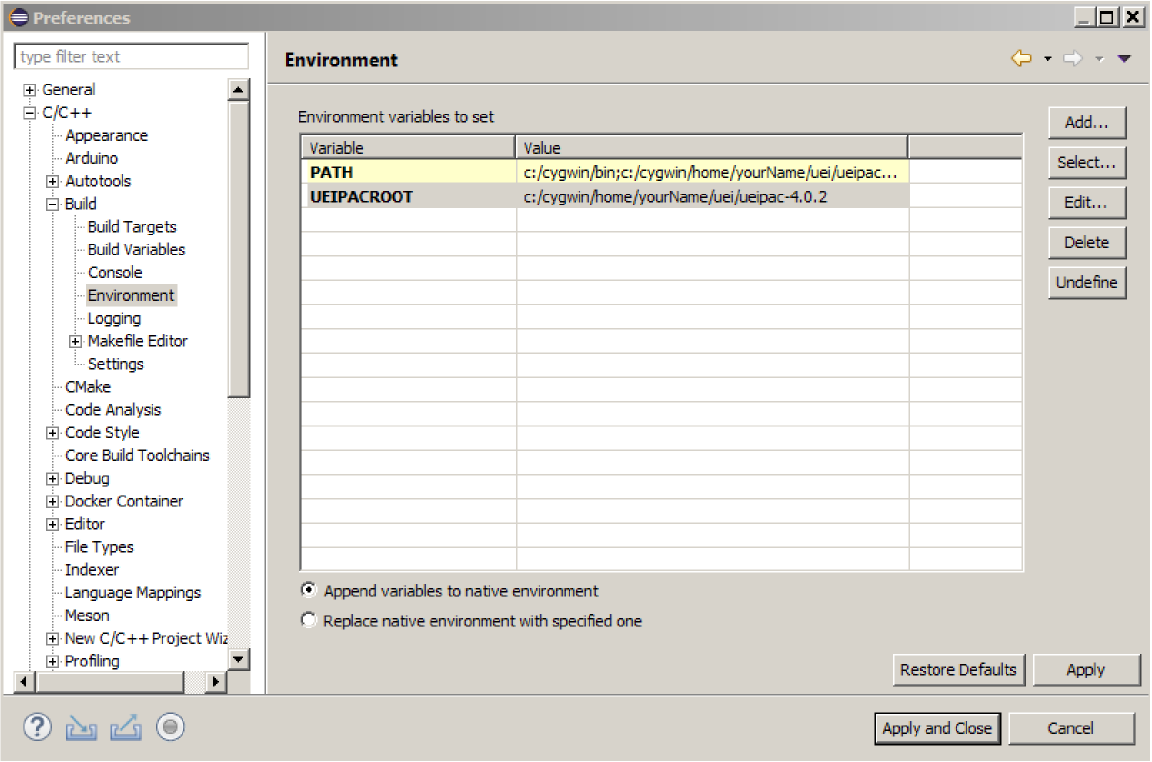
## B.2 Set up preferences

Edit Eclipse preferences to add the path to the Cygwin tools (such as make) and the UEIPAC cross-compiler.

1. Select the menu option **Window** » **Preferences,** and then in the left sidebar click **C/C++** » **Build** » **Environment**.

2. Click **Add** to 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/<username>/uei/ueipac-4.0.2/powerpc-604-linux-gnu/bin**

3. Click **Add** to add a variable named **UEIPACROOT** with value set to the UEIPAC SDK install directory.
   **c:/cygwin/home/<username>/uei/ueipac-4.0.2**



Environment variables to set

| Variable | Value |
|---|---|
| **PATH** | c:/cygwin/bin;c:/cygwin/home/yourName/uei/ueipac... |
| **UEIPACROOT** | c:/cygwin/home/yourName/uei/ueipac-4.0.2 |

● Append variables to native environment

○ Replace native environment with specified one

**Figure 3 Setting up environment**

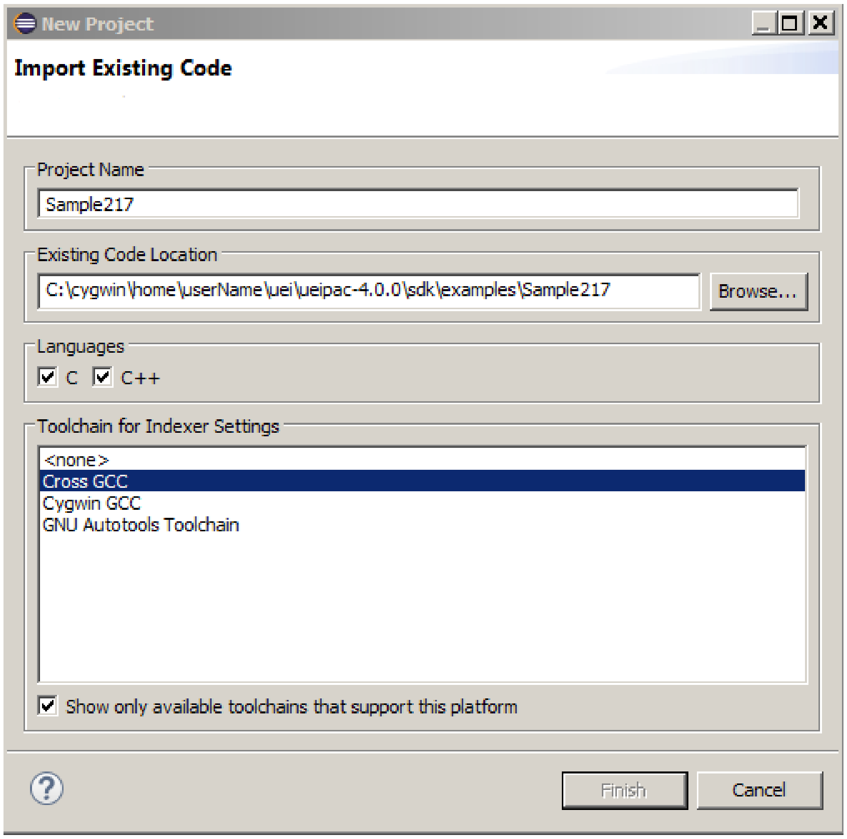
4. Click **Apply** and then **OK** to store Environment variables.

## *B.3 Open and build examples*

1. Select the menu option **File » New » Makefile Project with Existing Code.**

2. Type a project name.

3. Browse to the location of the example you wish to build (examples are located <Cygwin directory>\home\<your user name>\uei\ueipac-x.y.z\sdk\examples).



Project Name: Sample217

Existing Code Location: C:\cygwin\home\userName\uei\ueipac-4.0.0\sdk\examples\Sample217

Languages: ☑ C ☑ C++

Toolchain for Indexer Settings:

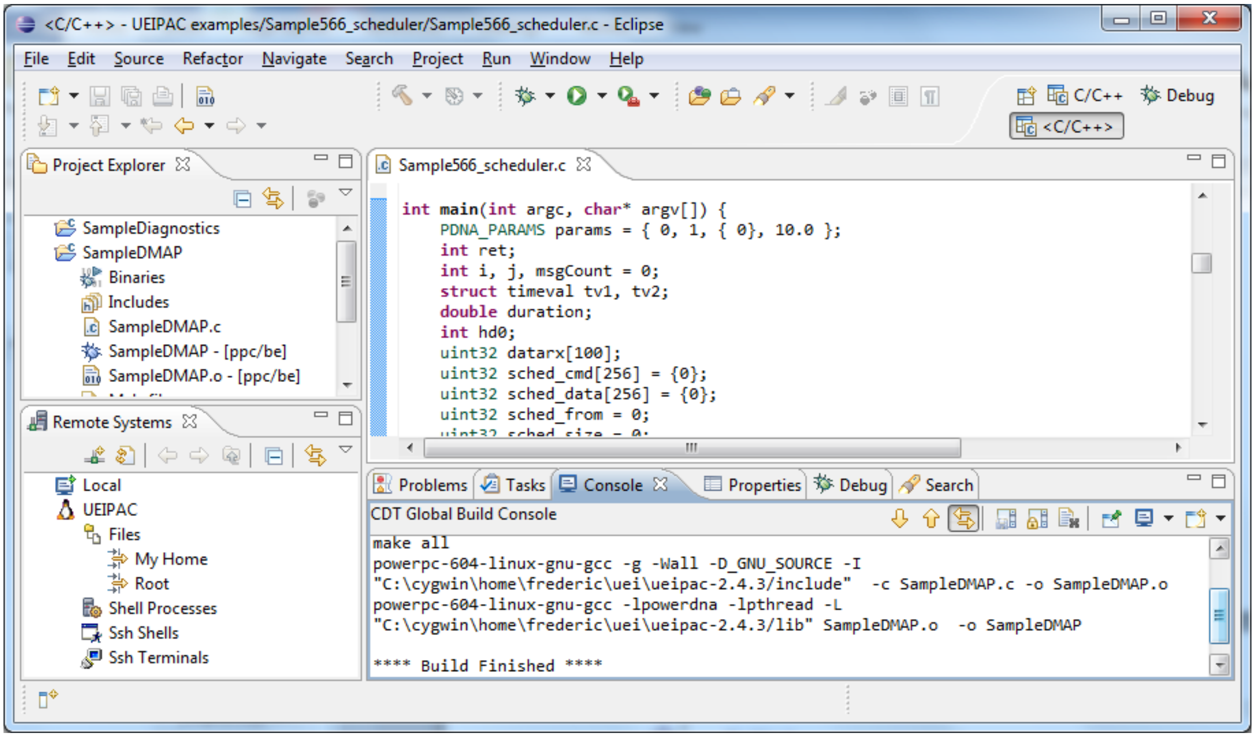
- <none>
- Cross GCC
- Cygwin GCC
- GNU Autotools Toolchain

☑ Show only available toolchains that support this platform

**Figure 4 Eclipse Import Existing Code Window**

4. Click **Finish** to create the project.

5. Select the **Project » Build Project** menu to build the example. Note that this build may produce errors.



```
int main(int argc, char* argv[]) {
    PDNA_PARAMS params = { 0, 1, { 0}, 10.0 };
    int ret;
    int i, j, msgCount = 0;
    struct timeval tv1, tv2;
    double duration;
    int hd0;
    uint32 datarx[100];
    uint32 sched_cmd[256] = {0};
    uint32 sched_data[256] = {0};
    uint32 sched_from = 0;
    uint32 sched_size = 0;
```

```
make all
powerpc-604-linux-gnu-gcc -g -Wall -D_GNU_SOURCE -I
"C:\cygwin\home\frederic\uei\ueipac-2.4.3/include"  -c SampleDMAP.c -o SampleDMAP.o
powerpc-604-linux-gnu-gcc -lpowerdna -lpthread -L
"C:\cygwin\home\frederic\uei\ueipac-2.4.3/lib" SampleDMAP.o  -o SampleDMAP

**** Build Finished ****
```

**Figure 5 Eclipse Build Project Window**

The indexer will report errors about header files it can’t find.

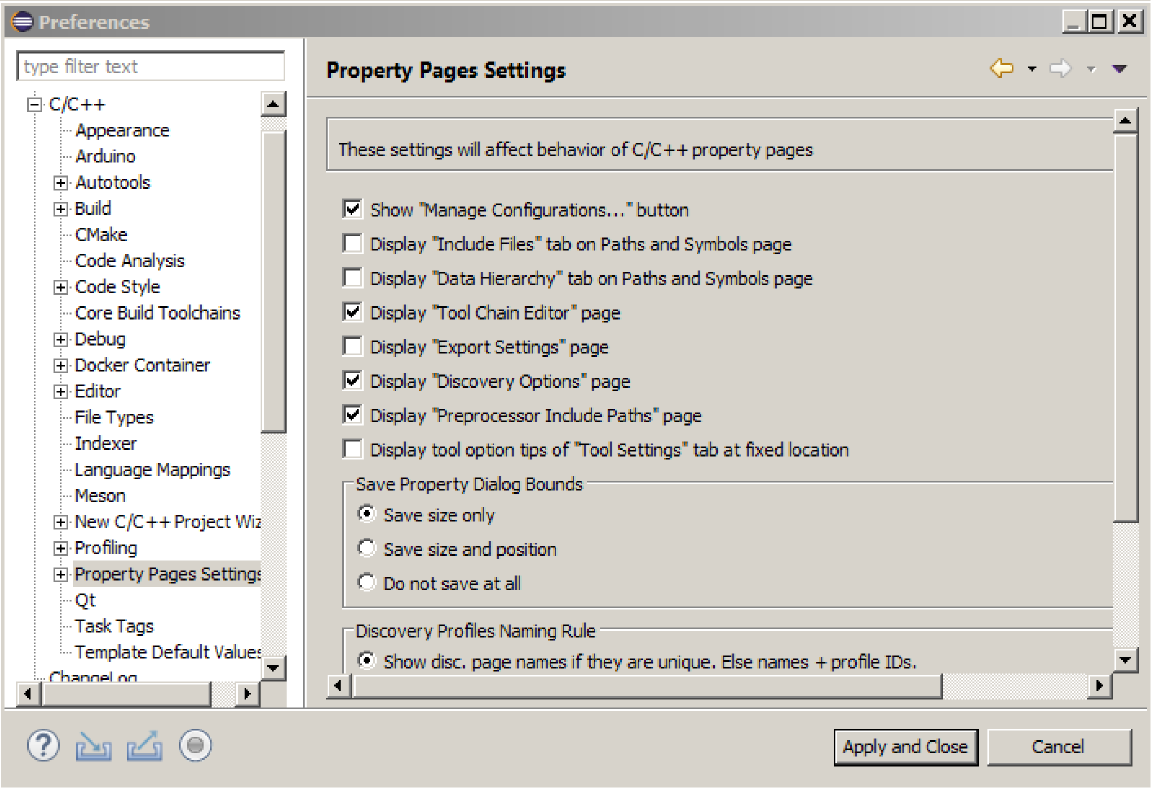
The following procedure configures discovery options to allow the indexer to find required programs:

1. Select the menu option **Window » Preferences.**

2. In the left sidebar, select **C/C++**, and then click **Property Pages Settings.**

3. Click the **Display “Discovery Options” page** check box to enable. (Refer to Figure 6 below)

4. Click **Apply and Close**.



Property Pages Settings

These settings will affect behavior of C/C++ property pages

☑ Show "Manage Configurations..." button
☐ Display "Include Files" tab on Paths and Symbols page
☐ Display "Data Hierarchy" tab on Paths and Symbols page
☑ Display "Tool Chain Editor" page
☐ Display "Export Settings" page
☑ Display "Discovery Options" page
☑ Display "Preprocessor Include Paths" page
☐ Display tool option tips of "Tool Settings" tab at fixed location

Save Property Dialog Bounds
● Save size only
○ Save size and position
○ Do not save at all

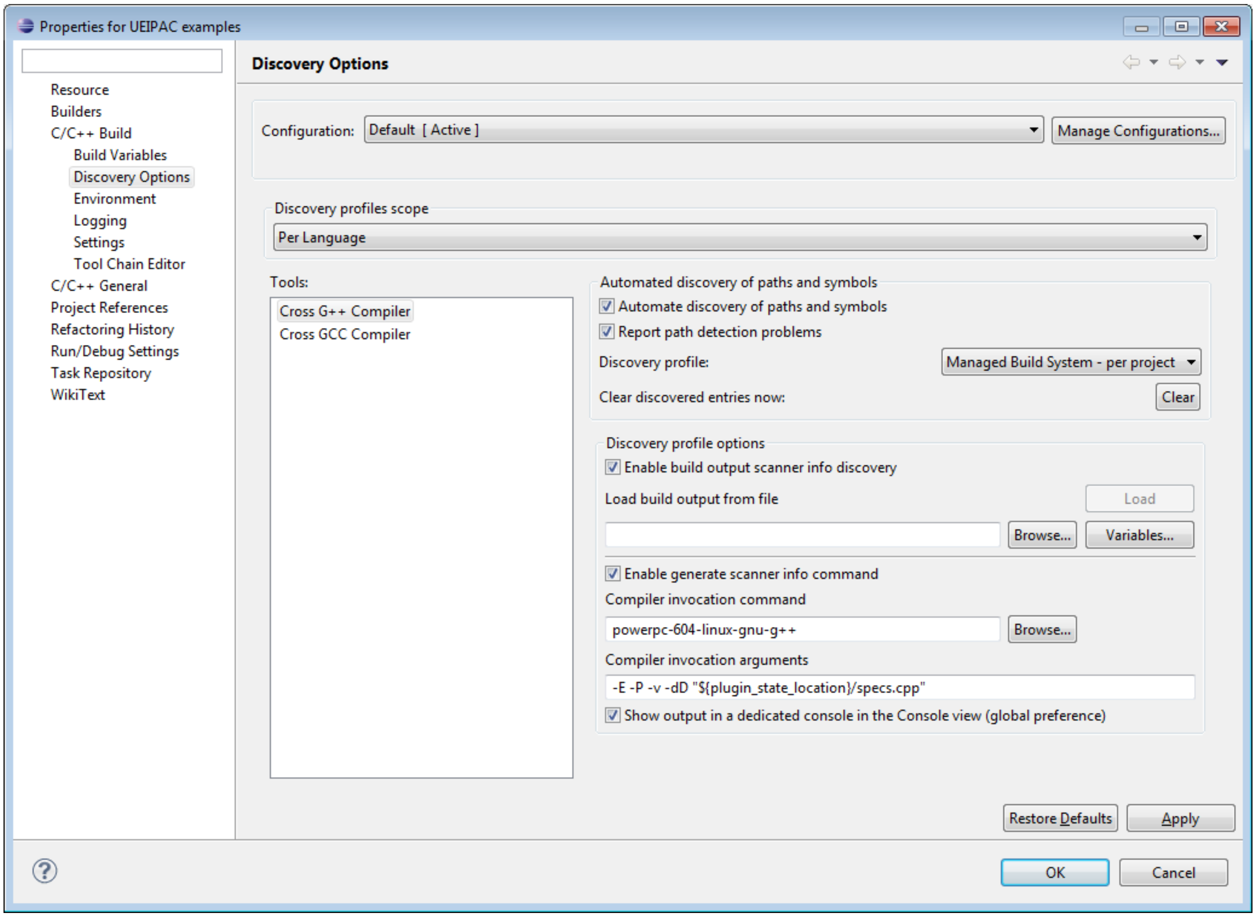
Discovery Profiles Naming Rule
● Show disc. page names if they are unique. Else names + profile IDs.

**Figure 6 Eclipse Property Pages Settings Window**

5. Select the menu option **Project » Properties** and in the left sidebar, click **C/C++ Build » Discovery Options**.

6. Verify **Automate discover of paths and symbols** is checked, and **Discovery profile** is set to **Managed Build System – per project**. (Refer to Figure 7 below).

7. Change the Compiler invocation commands to powerpc-604-linux-gnu-g++ and powerpc-604-linux-gnu-gcc:



Discovery Options

Configuration: Default [ Active ]

Discovery profiles scope: Per Language

Tools:
Cross G++ Compiler
Cross GCC Compiler

Automated discovery of paths and symbols
☑ Automate discovery of paths and symbols
☑ Report path detection problems
Discovery profile: Managed Build System - per project
Clear discovered entries now:

Discovery profile options
☑ Enable build output scanner info discovery
Load build output from file
☑ Enable generate scanner info command
Compiler invocation command
powerpc-604-linux-gnu-g++
Compiler invocation arguments
-E -P -v -dD "${plugin_state_location}/specs.cpp"
☑ Show output in a dedicated console in the Console view (global preference)

**Figure 7 Eclipse Discovery Options Window**

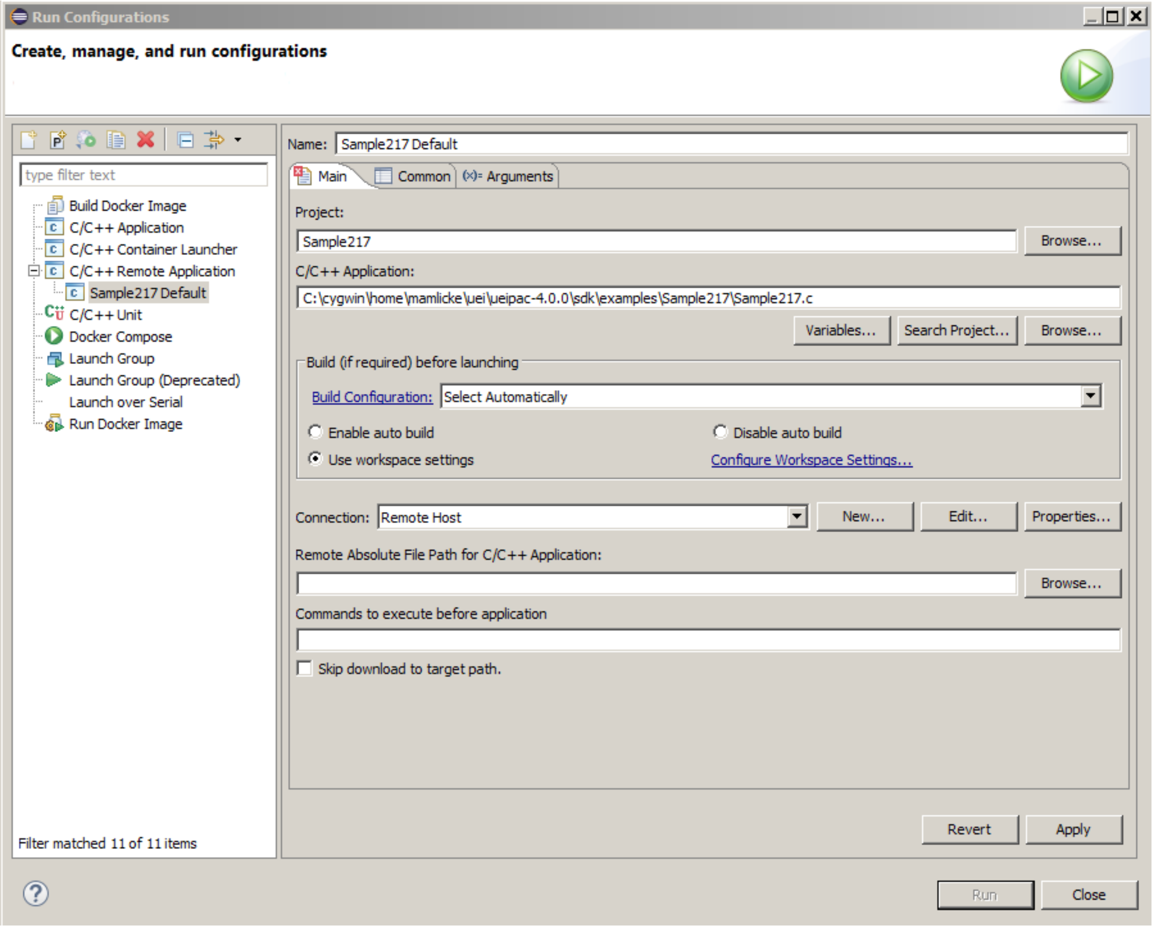
8. Check **Show output in a dedicated console in the Console view** to see output messaging in the Eclipse Console pane.

9. Click **Apply** and **OK**. The indexer will automatically find all header files next time you build the project.

## B.4 Execute a program

1. Select the **Run » Run Configurations…** menu option.

2. Select the **C/C++ Remote Application** option, and right-click to select **New Configuration** to open a new remote launch configuration. The **Create, manage, and run configurations** Sample pane will open:



Run Configurations

Create, manage, and run configurations

Name: Sample217 Default

Project: Sample217

C/C++ Application: C:\cygwin\home\mamlicke\uei\ueipac-4.0.0\sdk\examples\Sample217\Sample217.c

Build (if required) before launching

Build Configuration: Select Automatically

○ Enable auto build ○ Disable auto build

● Use workspace settings

Connection: Remote Host

Remote Absolute File Path for C/C++ Application:

Commands to execute before application

□ Skip download to target path.

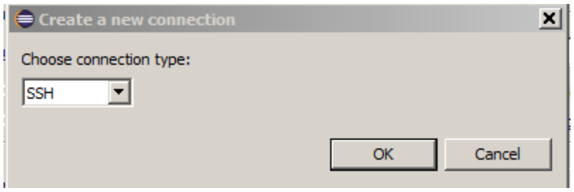
Filter matched 11 of 11 items

3. Enter a name for this new launch configuration. The default is `<sampleName>_Default`.

4. Verify that the **Project** is set correctly, or press **Browse…** to select the right project.

5. Verify that the **C/C++ Application** is set to the binary built from your project.

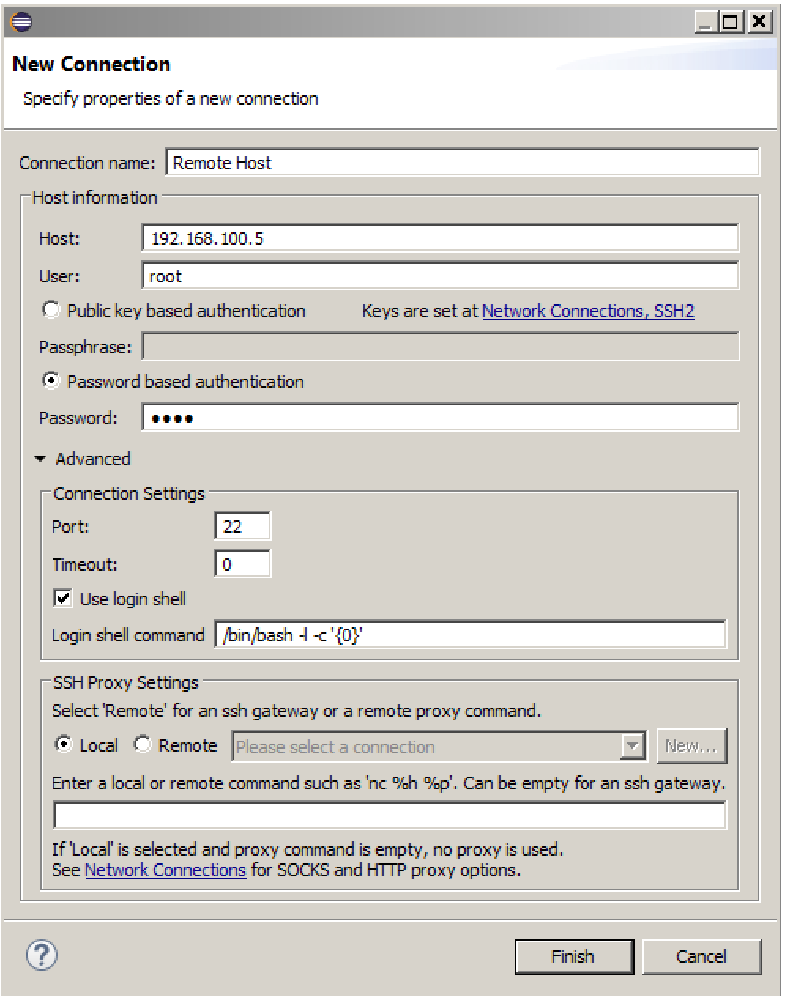
6. Set up your connection to the UEIPAC Remote Host by clicking **New**. A Create a new connection window will open:



Create a new connection

Choose connection type: SSH

7. Select SSH and click OK. A New Connection window will open:



**New Connection**

Specify properties of a new connection

Connection name: Remote Host

Host information

Host: 192.168.100.5

User: root

○ Public key based authentication    Keys are set at Network Connections, SSH2

Passphrase:

● Password based authentication

Password: ••••

Advanced

Connection Settings

Port: 22

Timeout: 0

☑ Use login shell

Login shell command /bin/bash -l -c '{0}'

SSH Proxy Settings

Select 'Remote' for an ssh gateway or a remote proxy command.

● Local ○ Remote  Please select a connection

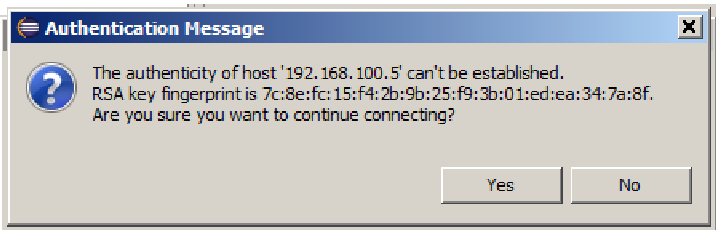
Enter a local or remote command such as 'nc %h %p'. Can be empty for an ssh gateway.

If 'Local' is selected and proxy command is empty, no proxy is used.
See Network Connections for SOCKS and HTTP proxy options.

8. Verify Password based authentication is checked.

9. Enter the IP address of the UEIPAC for the Host field, and enter the User name and Password. The default username and password for the UEIPAC is root and root. Click Finish.

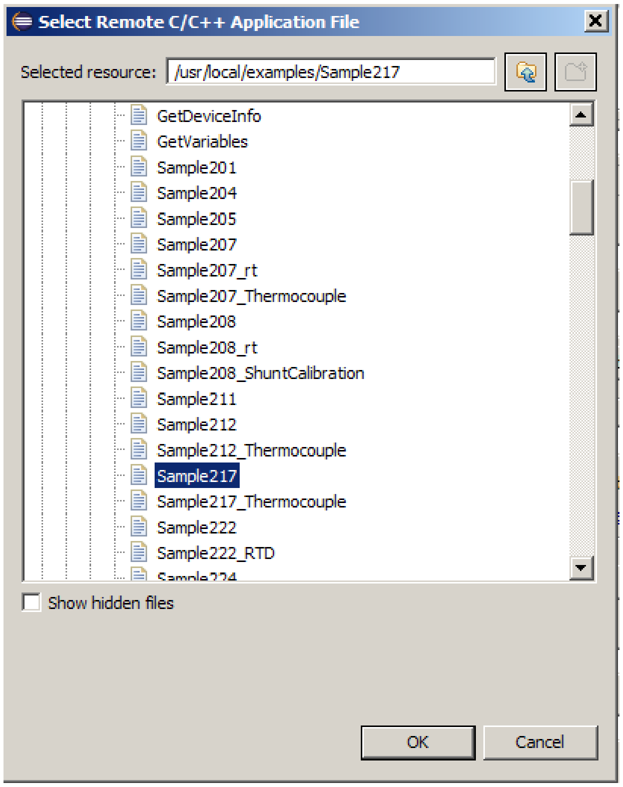
10. If you get Authentication message, click Yes.



Authentication Message

The authenticity of host '192.168.100.5' can't be established.
RSA key fingerprint is 7c:8e:fc:15:f4:2b:9b:25:f9:3b:01:ed:ea:34:7a:8f.
Are you sure you want to continue connecting?

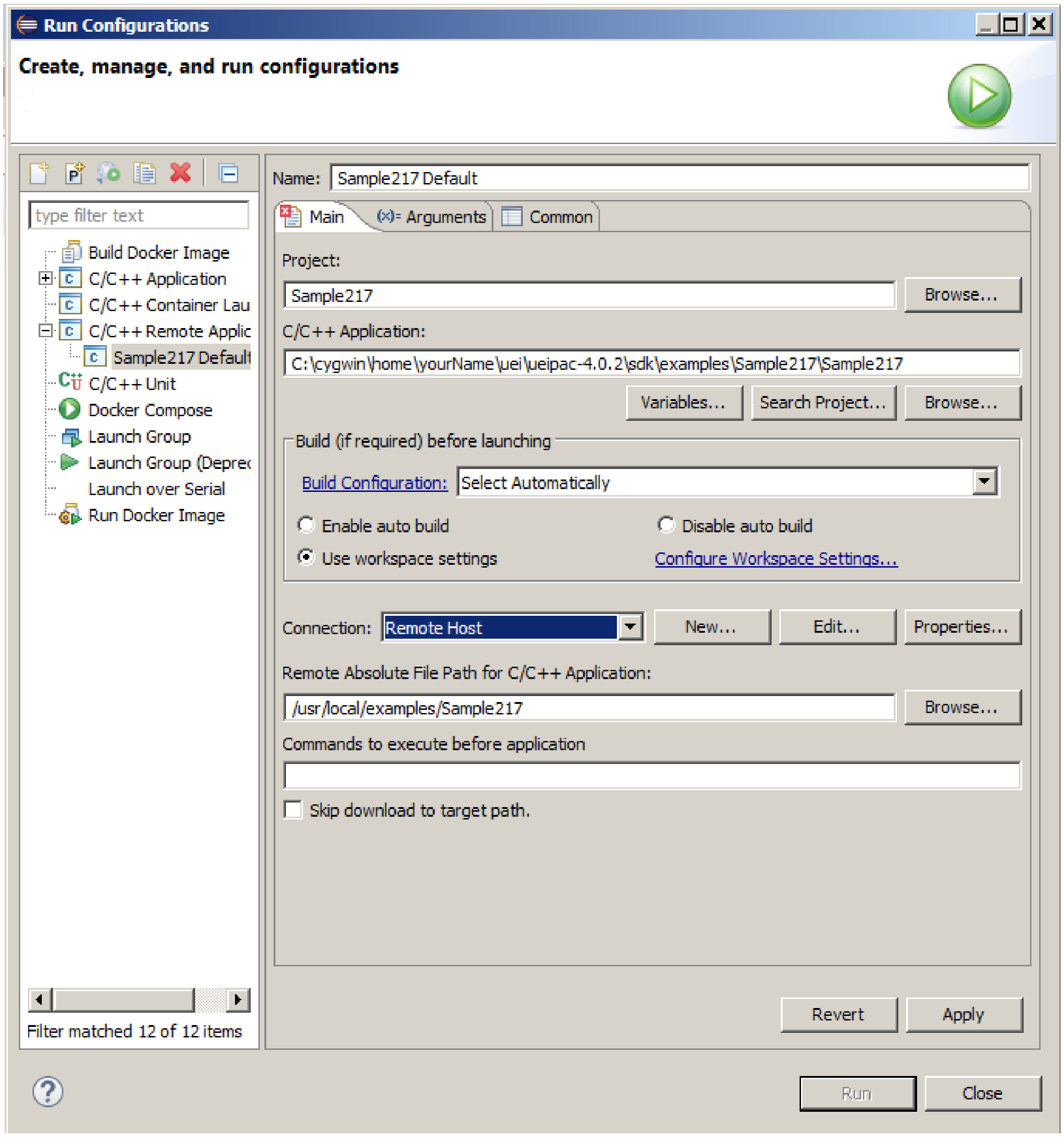
11. Click Browse for Remote Absolute File Path for C/C++ Application and select the directory:



Select Remote C/C++ Application File

Selected resource: /usr/local/examples/Sample217

12. Click **Apply**:



Run Configurations

Create, manage, and run configurations

Name: Sample217 Default

Project: Sample217

C/C++ Application: C:\cygwin\home\yourName\uei\ueipac-4.0.2\sdk\examples\Sample217\Sample217

Build (if required) before launching

Build Configuration: Select Automatically

○ Enable auto build ○ Disable auto build

● Use workspace settings

Connection: Remote Host

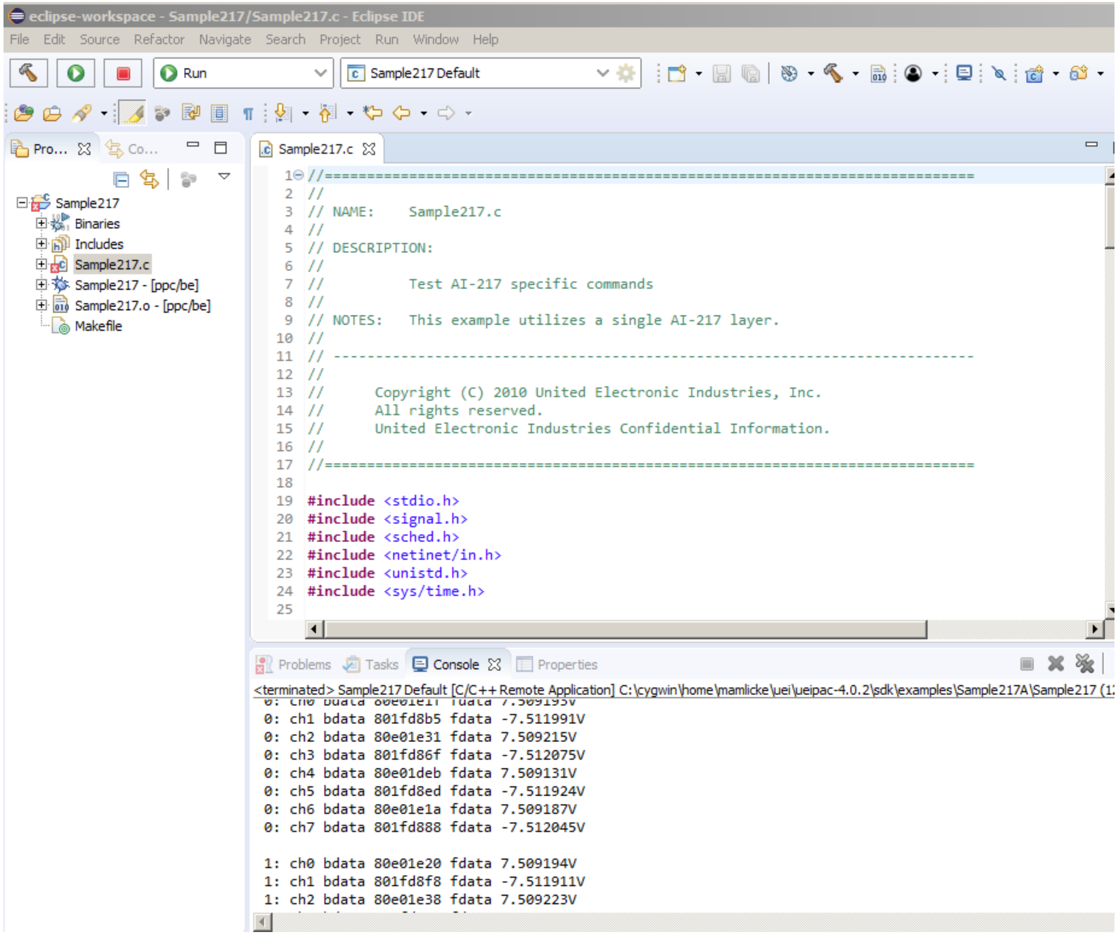
Remote Absolute File Path for C/C++ Application: /usr/local/examples/Sample217

Commands to execute before application:

□ Skip download to target path.

Filter matched 12 of 12 items

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



```
//============================================================================
//
// NAME:    Sample217.c
//
// DESCRIPTION:
//
//          Test AI-217 specific commands
//
// NOTES:   This example utilizes a single AI-217 layer.
//
// ---------------------------------------------------------------------------
//
//      Copyright (C) 2010 United Electronic Industries, Inc.
//      All rights reserved.
//      United Electronic Industries Confidential Information.
//
//============================================================================

#include <stdio.h>
#include <signal.h>
#include <sched.h>
#include <netinet/in.h>
#include <unistd.h>
#include <sys/time.h>

```

```
<terminated> Sample217 Default [C/C++ Remote Application] C:\cygwin\home\mamlicke\uei\ueipac-4.0.2\sdk\examples\Sample217A\Sample217 (1
0: ch0 bdata 80e01e1f fdata 7.509193V
0: ch1 bdata 801fd8b5 fdata -7.511991V
0: ch2 bdata 80e01e31 fdata 7.509215V
0: ch3 bdata 801fd86f fdata -7.512075V
0: ch4 bdata 80e01deb fdata 7.509131V
0: ch5 bdata 801fd8ed fdata -7.511924V
0: ch6 bdata 80e01e1a fdata 7.509187V
0: ch7 bdata 801fd888 fdata -7.512045V

1: ch0 bdata 80e01e20 fdata 7.509194V
1: ch1 bdata 801fd8f8 fdata -7.511911V
1: ch2 bdata 80e01e38 fdata 7.509223V
```

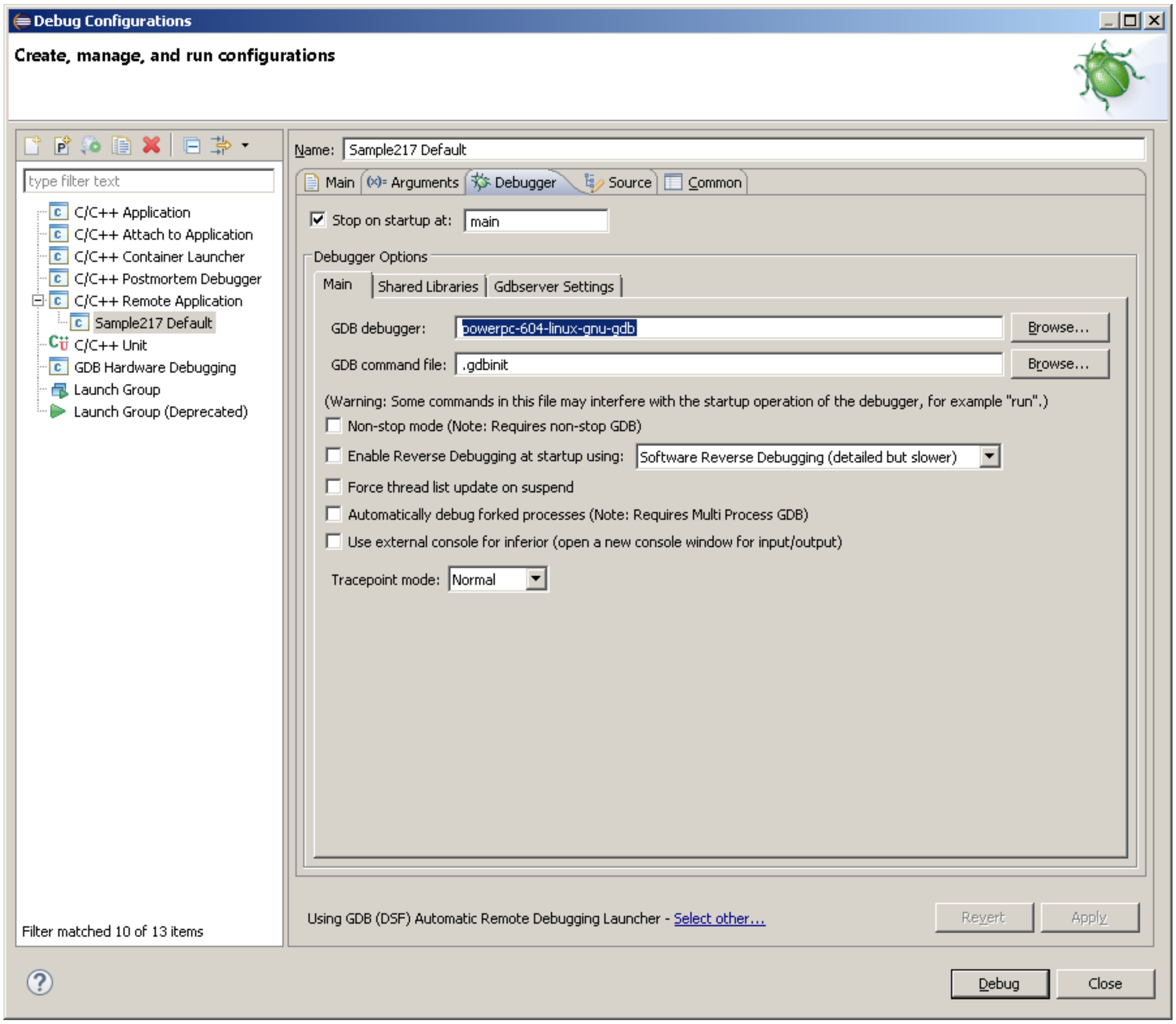
## B.5 Debugging your program on the UEIPAC

The UEIPAC examples are already compiled with debug information, and the Makefile in each sample directory includes the **–g** compiler flag to produce debug information when compiling updated samples.

1. Select the **Run » Debug Configurations…** menu option to open the Debug dialog box.

2. Select the **C/C++ Remote Application**, and right-click to select **New Configuration**. The debug name created is your project name with "_Default" appended. In this example, the debug name is Sample217_Default; however, you can rename the debug configuration to whatever you wish.
(Refer to the figure on the following page).

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



Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Container Launcher
- C/C++ Postmortem Debugger
- C/C++ Remote Application
  - Sample217 Default
- C/C++ Unit
- GDB Hardware Debugging
- Launch Group
- Launch Group (Deprecated)

Filter matched 10 of 13 items

Name: Sample217 Default

☑ Stop on startup at: main

Debugger Options

GDB debugger: powerpc-604-linux-gnu-gdb

GDB command file: .gdbinit

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

☐ Non-stop mode (Note: Requires non-stop GDB)

☐ Enable Reverse Debugging at startup using: Software Reverse Debugging (detailed but slower)

☐ Force thread list update on suspend

☐ Automatically debug forked processes (Note: Requires Multi Process GDB)

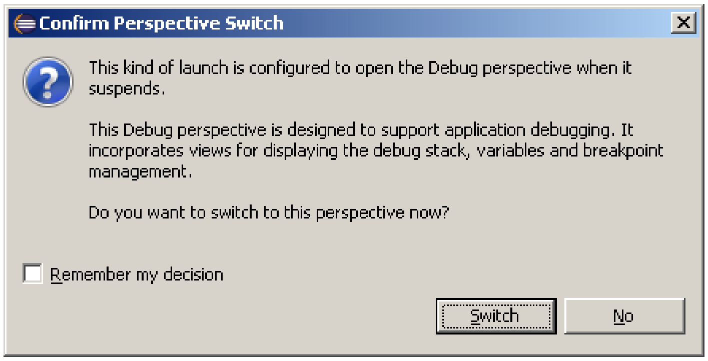
☐ Use external console for inferior (open a new console window for input/output)

Tracepoint mode: Normal

Using GDB (DSF) Automatic Remote Debugging Launcher - Select other...

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

5. Eclipse will suggest that you switch to the **Debug perspective**, click **Switch.**



This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

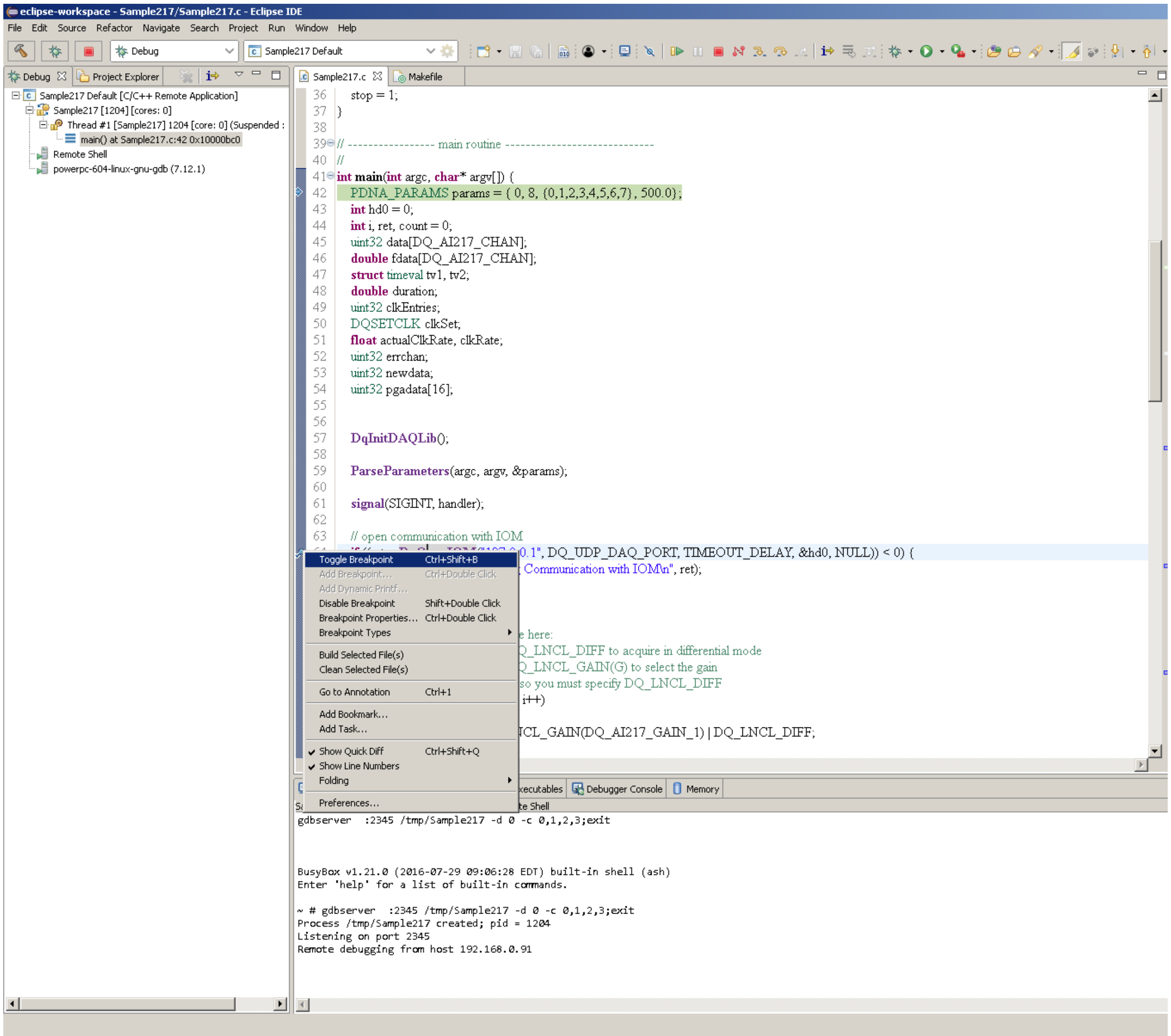
Do you want to switch to this perspective now?

□ Remember my decision

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

Set a breakpoint on a line in main() (Right-click 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.


eclipse-workspace - Sample217/Sample217.c - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Sample217 Default
Debug
Project Explorer
Sample217 Default [C/C++ Remote Application]
Sample217 [1204] [cores: 0]
Thread #1 [Sample217] 1204 [core: 0] (Suspended :
main() at Sample217.c:42 0x10000bc0
Remote Shell
powerpc-604-linux-gnu-gdb (7.12.1)
Sample217.c
Makefile
Toggle Breakpoint Ctrl+Shift+B
Add Breakpoint... Ctrl+Double Click
Add Dynamic Printf...
Disable Breakpoint Shift+Double Click
Breakpoint Properties... Ctrl+Double Click
Breakpoint Types
Build Selected File(s)
Clean Selected File(s)
Go to Annotation Ctrl+1
Add Bookmark...
Add Task...
✓ Show Quick Diff Ctrl+Shift+Q
✓ Show Line Numbers
Folding
Preferences...
xecutables
Debugger Console
Memory
te Shell

```
    stop = 1;
}

// ---------------- main routine ----------------------------
//
int main(int argc, char* argv[]) {
    PDNA_PARAMS params = { 0, 8, {0,1,2,3,4,5,6,7}, 500.0};
    int hd0 = 0;
    int i, ret, count = 0;
    uint32 data[DQ_AI217_CHAN];
    double fdata[DQ_AI217_CHAN];
    struct timeval tv1, tv2;
    double duration;
    uint32 clkEntries;
    DQSETCLK clkSet;
    float actualClkRate, clkRate;
    uint32 errchan;
    uint32 newdata;
    uint32 pgadata[16];


    DqInitDAQLib();

    ParseParameters(argc, argv, &params);

    signal(SIGINT, handler);

    // open communication with IOM
                           0.1", DQ_UDP_DAQ_PORT, TIMEOUT_DELAY, &hd0, NULL)) < 0) {
                           Communication with IOM\n", ret);



                           e here:
                           Q_LNCL_DIFF to acquire in differential mode
                           Q_LNCL_GAIN(G) to select the gain
                           so you must specify DQ_LNCL_DIFF
                           i++)

                           CL_GAIN(DQ_AI217_GAIN_1) | DQ_LNCL_DIFF;
```

```
gdbserver  :2345 /tmp/Sample217 -d 0 -c 0,1,2,3;exit



BusyBox v1.21.0 (2016-07-29 09:06:28 EDT) built-in shell (ash)
Enter 'help' for a list of built-in commands.

~ # gdbserver  :2345 /tmp/Sample217 -d 0 -c 0,1,2,3;exit
Process /tmp/Sample217 created; pid = 1204
Listening on port 2345
Remote debugging from host 192.168.0.91
```

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

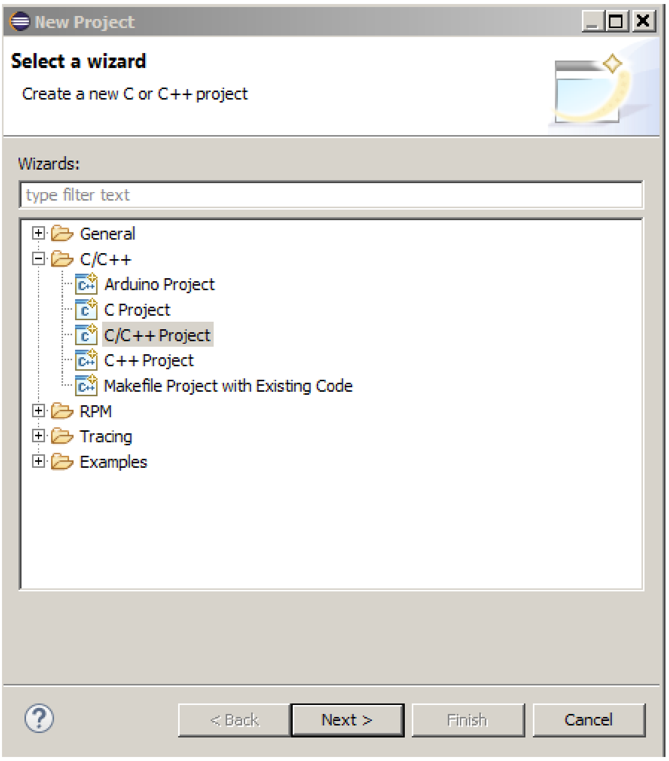
More information about debugging programs is available in Eclipse’s online help. Select the menu option **Help » Help Contents.**
In the Help window, search for “Debugging a project”.

# Appendix C. Creating a new Eclipse project for UEIPAC

The following procedure creates a new UEIPAC project in Eclipse. If you are downloading Eclipse for the first time or setting up Eclipse for UEIPAC projects that already have existing code, please refer to above.

## C.1 *Create a new project*

1. Select the menu option **File » New » Project**.



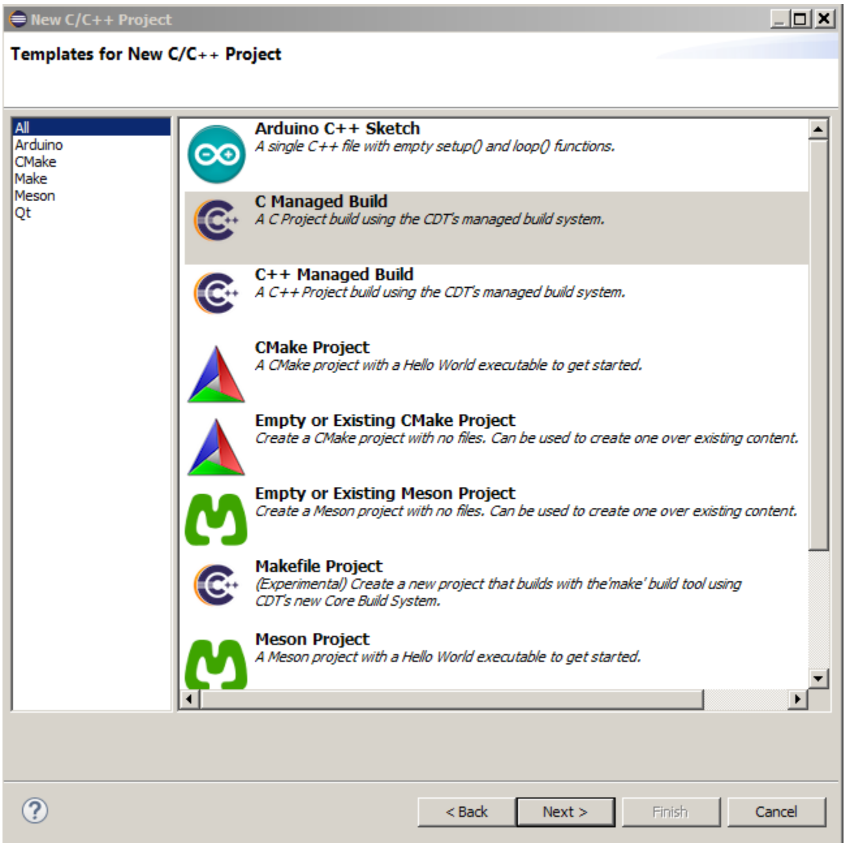
New Project

**Select a wizard**

Create a new C or C++ project

2. Select **C/C++ >> C/C++ Project**, and click **Next** >.

3. Select a template, and click **Next** >:



Templates for New C/C++ Project

**Arduino C++ Sketch**
*A single C++ file with empty setup() and loop() functions.*

**C Managed Build**
*A C Project build using the CDT's managed build system.*

**C++ Managed Build**
*A C++ Project build using the CDT's managed build system.*

**CMake Project**
*A CMake project with a Hello World executable to get started.*

**Empty or Existing CMake Project**
*Create a CMake project with no files. Can be used to create one over existing content.*

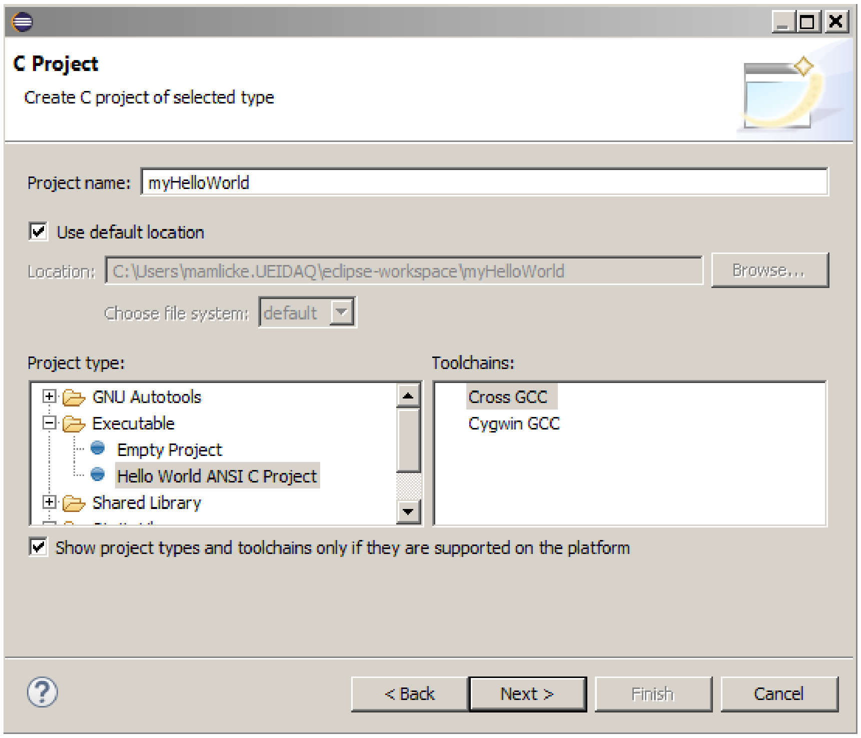
**Empty or Existing Meson Project**
*Create a Meson project with no files. Can be used to create one over existing content.*

**Makefile Project**
*(Experimental) Create a new project that builds with the'make' build tool using CDT's new Core Build System.*

**Meson Project**
*A Meson project with a Hello World executable to get started.*

]

4. Enter a **Project name**, select **Cross GCC** and click **Next.**



**C Project**

Create C project of selected type

Project name: myHelloWorld

☑ Use default location

Location: C:\Users\mamlicke.UEIDAQ\eclipse-workspace\myHelloWorld

Choose file system: default

Project type:

- GNU Autotools
- Executable
  - Empty Project
  - Hello World ANSI C Project
- Shared Library

Toolchains:

- Cross GCC
- Cygwin GCC

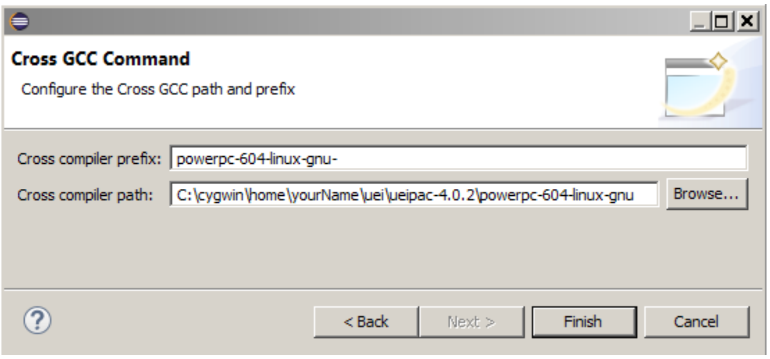
☑ Show project types and toolchains only if they are supported on the platform

5. Enter **Author** and basic properties in the **Basic Settings** window, and click **Next**.

6. Click **Next** in the **Select Configurations** window. Paths will be configured in a later step.

7. In the **Cross GCC Command** window, set **prefix** to **powerpc-604-linux-gnu-** (don’t forget the ‘-’ at the end of the prefix).

8. Set **path** to **<UEIPAC SDK folder>/powerpc-604-linux-gnu/bin**, and then click **Finish.**



**Cross GCC Command**

Configure the Cross GCC path and prefix

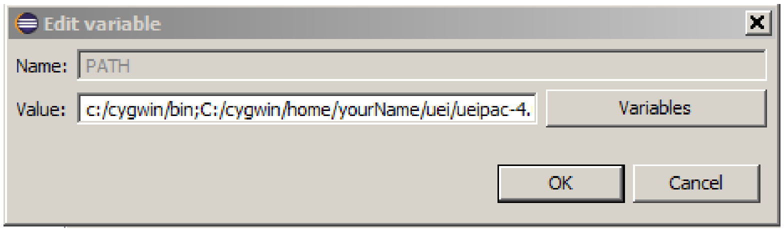
Cross compiler prefix: powerpc-604-linux-gnu-

Cross compiler path: C:\cygwin\home\yourName\uei\ueipac-4.0.2\powerpc-604-linux-gnu

## C.2 Configure the environment

Eclipse needs to know where the **make** utility and the compiler binaries are located (even though they were already set during the project configuration).

1. In the Project Explorer pane, right-click your project and select **Properties.**
2. Select **C/C++ Build,** and then click **Environment.**
3. Click **PATH,** and then click **Edit…** An Edit variable window will open:
4. Append **c:\cygwin\bin; c:\cygwin\home\<username>\uei\ueipac-x.y.z\powerpc-604-linux-gnu\bin;** to the existing value, and click **OK**.



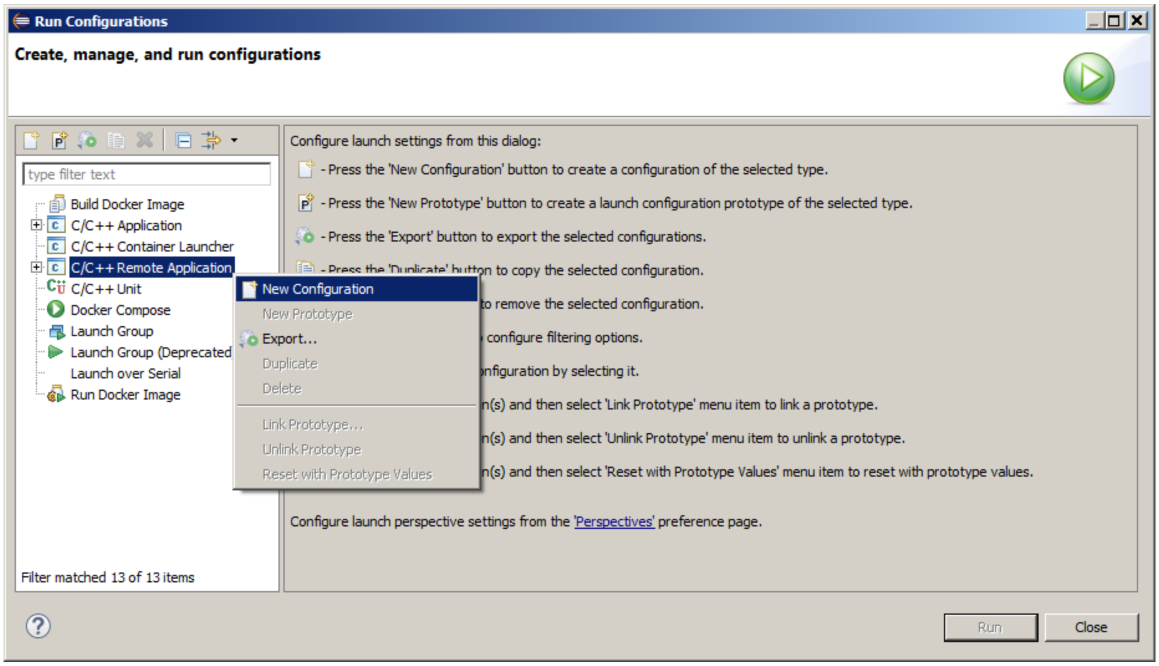
Edit variable

Name: PATH

Value: c:/cygwin/bin;C:/cygwin/home/yourName/uei/ueipac-4.

## C.3 Build and run

1. Select the **Run » Run Configurations…** menu option to open the Run dialog box.
2. Select the **C/C++ Remote Application** option, and right-click **New Configuration** to create a new remote launch configuration:



Configure launch settings from this dialog:

- Press the 'New Configuration' button to create a configuration of the selected type.
- Press the 'New Prototype' button to create a launch configuration prototype of the selected type.
- Press the 'Export' button to export the selected configurations.
- Press the 'Duplicate' button to copy the selected configuration.

to remove the selected configuration.

configure filtering options.

nfiguration by selecting it.

n(s) and then select 'Link Prototype' menu item to link a prototype.

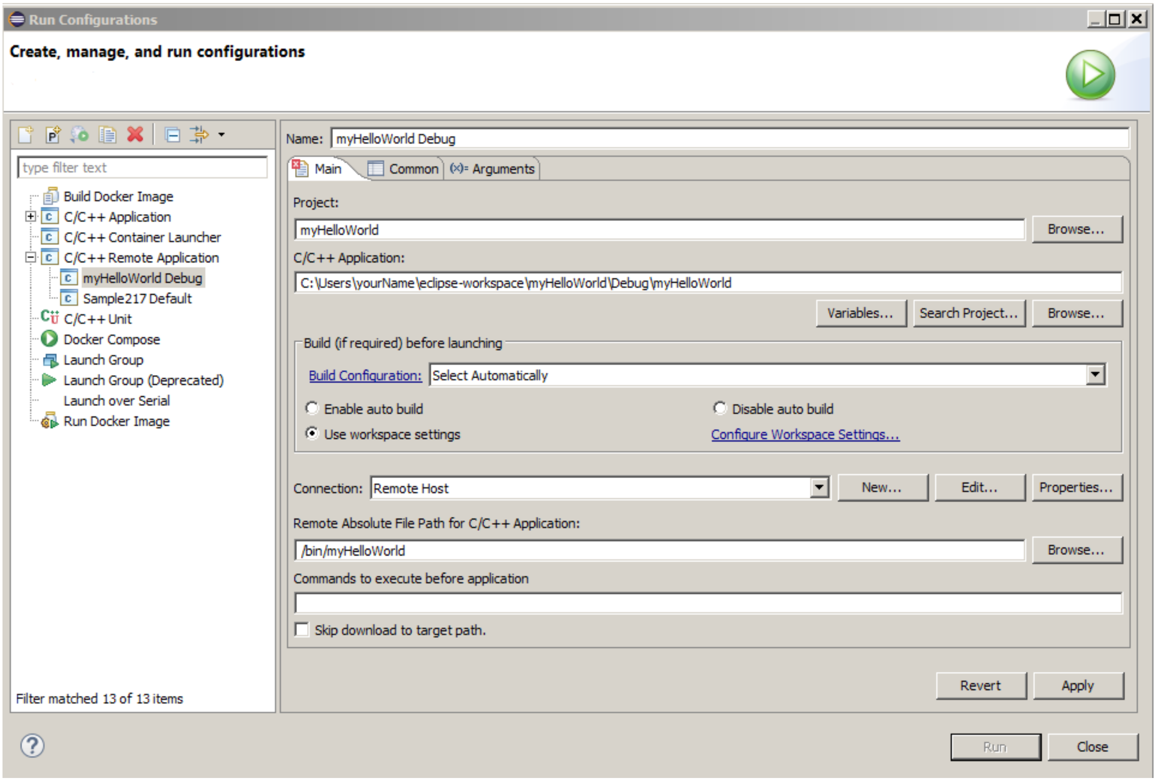
n(s) and then select 'Unlink Prototype' menu item to unlink a prototype.

n(s) and then select 'Reset with Prototype Values' menu item to reset with prototype values.

Configure launch perspective settings from the 'Perspectives' preference page.

3. Verify that the **C/C++ Application** is set to the binary built from your project. (This is set to Debug\HelloUEIPAC in the example below).

4. Enter the path and name in the **Remote Absolute File Path for C/C++ Application** to set the location where the application should be uploaded on the UEIPAC file system. (This is set to /tmp/HelloUEIPAC in the example below).



Run Configurations

Create, manage, and run configurations

Name: myHelloWorld Debug

Project: myHelloWorld

C/C++ Application: C:\Users\yourName\eclipse-workspace\myHelloWorld\Debug\myHelloWorld

Build (if required) before launching

Build Configuration: Select Automatically

○ Enable auto build ○ Disable auto build

● Use workspace settings

Connection: Remote Host

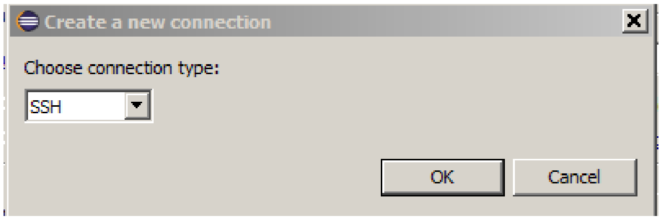
Remote Absolute File Path for C/C++ Application: /bin/myHelloWorld

Commands to execute before application

□ Skip download to target path.

Filter matched 13 of 13 items

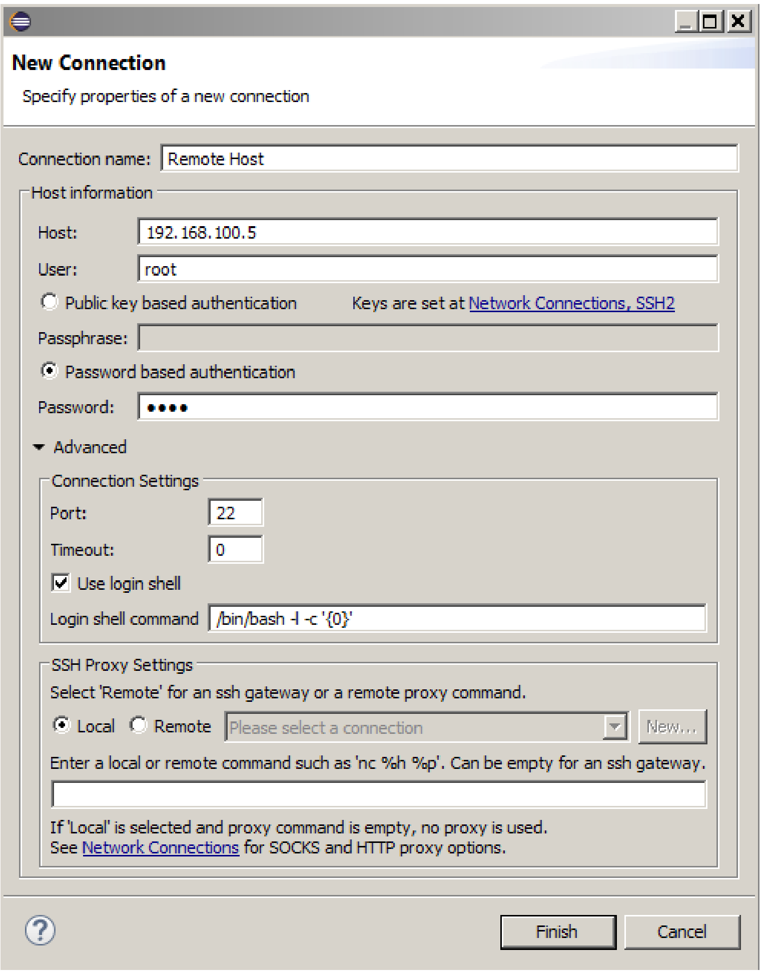
5. Click **New…** A Create a new connection window will open:



Create a new connection

Choose connection type: SSH

6. Select SSH and click OK. A New Connection window will open:



New Connection

Specify properties of a new connection

Connection name: Remote Host

Host information

Host: 192.168.100.5

User: root

○ Public key based authentication — Keys are set at Network Connections, SSH2

Passphrase:

● Password based authentication

Password: ••••

Advanced

Connection Settings

Port: 22

Timeout: 0

☑ Use login shell

Login shell command /bin/bash -l -c '{0}'

SSH Proxy Settings

Select 'Remote' for an ssh gateway or a remote proxy command.

● Local ○ Remote — Please select a connection — New...

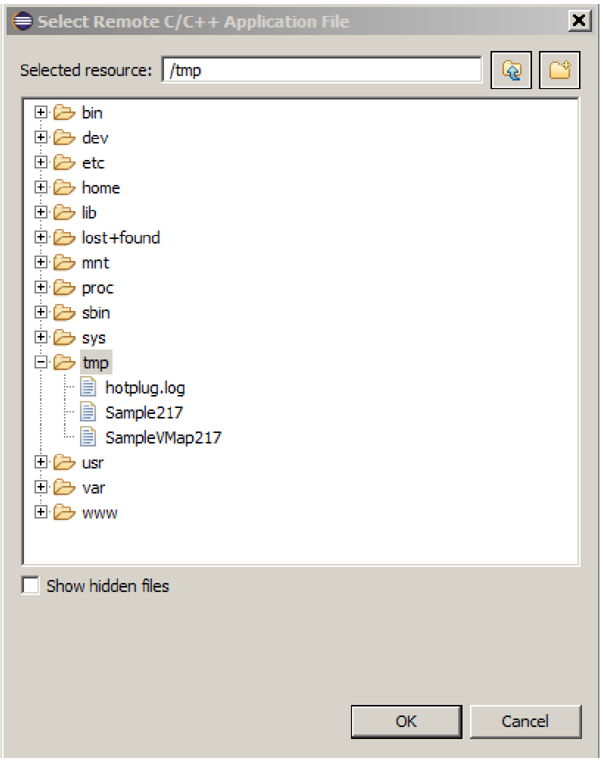
Enter a local or remote command such as 'nc %h %p'. Can be empty for an ssh gateway.

If 'Local' is selected and proxy command is empty, no proxy is used.
See Network Connections for SOCKS and HTTP proxy options.

7. Verify Password based authentication is checked.

8. Enter the IP address of the UEIPAC for the Host field, and enter the User name and Password. The default username and password for the UEIPAC is root and root. Click Finish.

9. Click **Browse** for Remote Absolute File Path for C/C++ Application and select the directory:



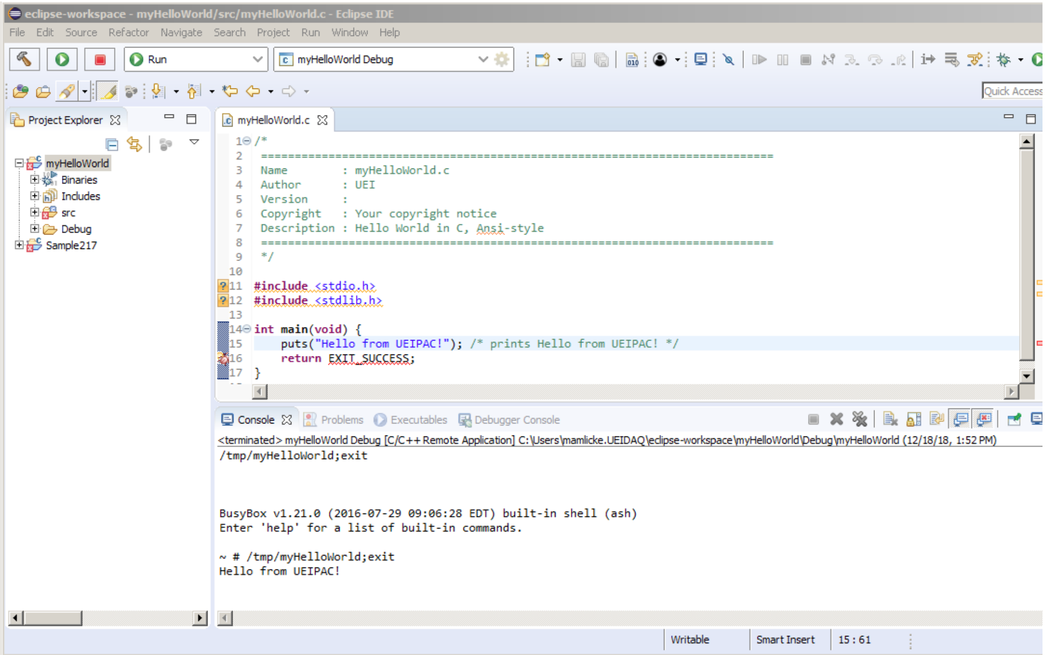
Select Remote C/C++ Application File

Selected resource: /tmp

```
bin
dev
etc
home
lib
lost+found
mnt
proc
sbin
sys
tmp
    hotplug.log
    Sample217
    SampleVMap217
usr
var
www
```

☐ Show hidden files

10. Click **Apply**, and then click **Run**:



eclipse-workspace - myHelloWorld/src/myHelloWorld.c - Eclipse IDE

```
/*
 ============================================================================
 Name        : myHelloWorld.c
 Author      : UEI
 Version     :
 Copyright   : Your copyright notice
 Description : Hello World in C, Ansi-style
 ============================================================================
 */

#include <stdio.h>
#include <stdlib.h>

int main(void) {
	puts("Hello from UEIPAC!"); /* prints Hello from UEIPAC! */
	return EXIT_SUCCESS;
}
```

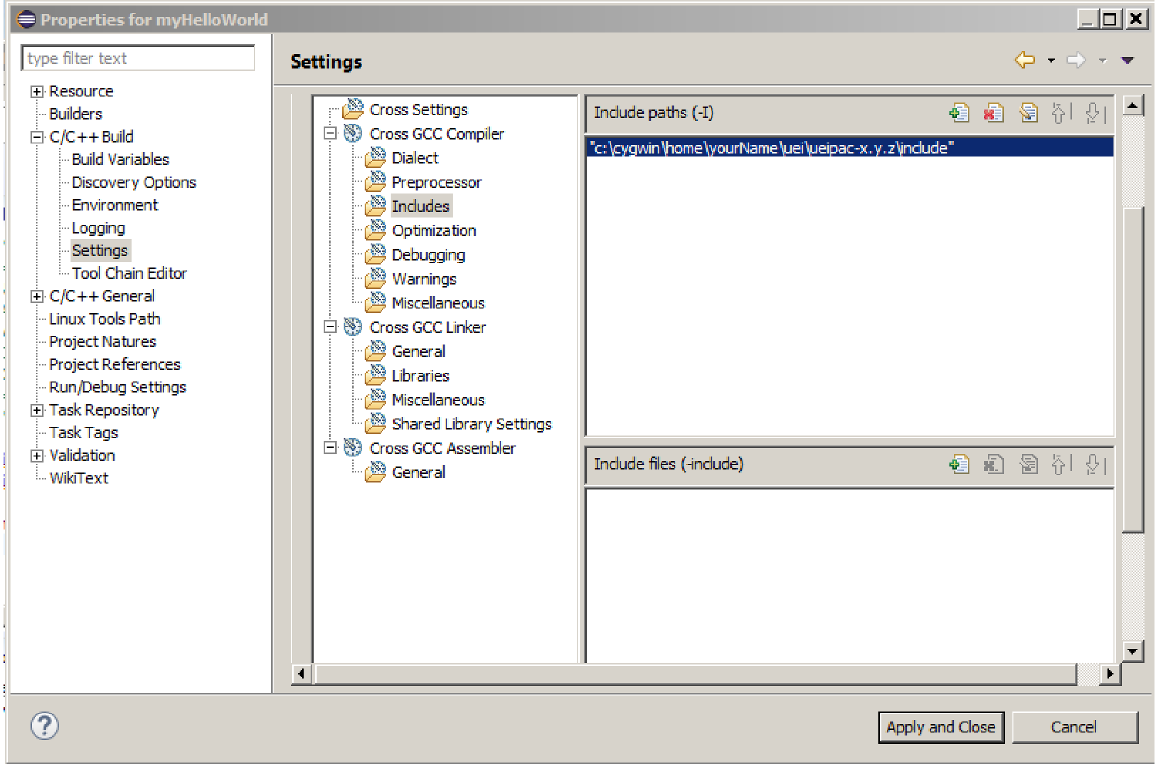
```
<terminated> myHelloWorld Debug [C/C++ Remote Application] C:\Users\mamlicke.UEIDAQ\eclipse-workspace\myHelloWorld\Debug\myHelloWorld (12/18/18, 1:52 PM)
/tmp/myHelloWorld;exit



BusyBox v1.21.0 (2016-07-29 09:06:28 EDT) built-in shell (ash)
Enter 'help' for a list of built-in commands.

~ # /tmp/myHelloWorld;exit
Hello from UEIPAC!
```

## C.4 Adding DNA API calls to your program

1. Right-click your project, and select **Properties.**

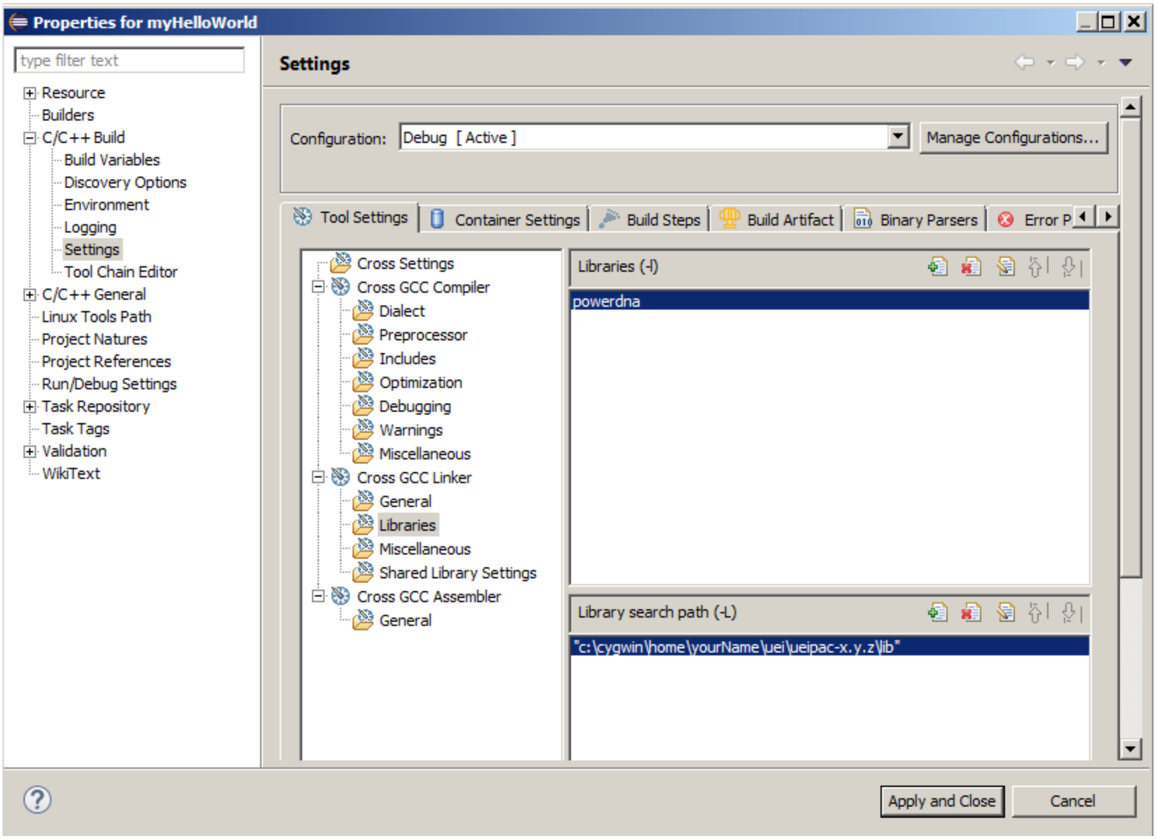
2. Select **C/C++ Build,** and then click **Settings.**

3. Click **Cross GCC Compiler » Includes** and add an include path set to **c:\cygwin\home\<username>\uei\ueipac-x.y.z\include.**



Include paths (-I): "c:\cygwin\home\yourName\uei\ueipac-x.y.z\include"

4. Click **Cross GCC Linker » Libraries** and add a library set to **powerdna** and library search path  set to **c:\cygwin\home\<username>\uei\ueipac-x.y.z\lib.**



Configuration: Debug [ Active ]

Libraries (-l): powerdna

Library search path (-L): "c:\cygwin\home\yourName\uei\ueipac-x.y.z\lib"

Add PowerDNA library calls to your code, build and run.

# Appendix D. Booting from NFS

## D.1 Configure shared RFS on host PC

1. Install an NFS server on your Linux machine.

2. Untar the rfs.tgz file that comes on the UEIPAC CD-ROM.

3. 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_squas
   h)
   ```

4. Remove the file *rfs/etc/rc.d/S10network* (kernel does the network configuration while booting and overwriting it will kill the NFS session).

5. Create the directory *rfs/etc/mnt* (used to mount the SD card later).

6. 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.

## D.2 Configure U-Boot

1. Power-up the UEIPAC and press a key to enter U-Boot.

2. Type the following to set the console type, which depends on what type of CPU is in your UEIPAC:
   For UEIPAC-300, UEIPAC-600, & UEIPAC-700 models (based on 5200 CPU):

   ```
     setenv consoledev ttyPSC0
   ```

   For all other UEIPACs (1G Cubes and R/F RACK models based on 8347 CPU):

   ```
     setenv consoledev ttyS0
   ```

3. Type the following commands :

```
setenv gateway <your gateway ip address>
setenv netmask <your netmask>
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
```

4. Verify that your `bootargs` variable looks like this:

   For UEIPAC-300, UEIPAC-600, & UEIPAC-700 models (based on 5200 CPU):

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

   For all other UEIPACs (1G Cubes and R/F RACK models based on 8347 CPU):

```
  bootargs=root=/dev/nfs rw
  nfsroot=192.168.100.1:/home/frederic/UEIPAC/rfs
  console=ttyS0,57600
  ip=192.168.100.2:192.168.100.1::255.255.255.0::eth0:off panic=1
```

5. Reset the UEIPAC which will now find its root file system on the NFS share

```
reset
```

# Appendix E. Building the Linux kernel

## *E.1 Install tools*

Note that you can only build the UEIPAC Linux kernel on a PC running Linux connected to the internet.

Before you begin, verify 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
```

## *E.2 Build the kernel for UEIPAC-300, UEIPAC-600, UEIPAC-700*

The UEIPAC kernel for the UEIPAC-300, UEIPAC-600, & UEIPAC-700 includes a Xenomai real-time extension.

1. Change to the kernel directory:
   ```
   cd <UEIPAC SDK directory>/kernel
   ```

2. Run the **build_xenomail.sh** script.
   ```
   ./build_xenomai.sh
   ```

3. Run the **get_kernel.sh** script:
   ```
   ./get_kernel.sh -xenomai –cpu 5200
   ```

   NOTE: 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 containing the kernel source with patches applied: linux-v4.9.51 (or whatever the current version is of the Linux kernel with Xenomai patches).

4. Change the current directory to the Linux source directory.

5. Run the **build_kernel.sh** script:
   ```
   ./build_kernel.sh -xenomai –cpu 5200
   ```

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

NOTE: The build scripts build the default UEIPAC kernel. You can customize the kernel configuration by using the menuconfig tool:

```
make ARCH=powerpc CROSS_COMPILE=powerpc-604-linux-gnu- menuconfig
```

You can then follow the build script to compile the kernel and modules and install the modules.

## E.3 Build the kernel for UEIPAC-XXX-1G, RACK and UEIPAC-XXX-02 versions with 8347 CPU

The following procedure is used to download, configure and build the kernel for the UEIPAC-600R/1200R, UEIPAC-300/600/700-1G, UEIPAC-400/1200-MIL, UEINET-PAC, UEIPAC-400F, and UEIPAC-XXX-02 systems.

Note that with these products, you have the option of using the Linux kernel with Xenomai patches or the Real-Time Linux kernel with the PREEMPT_RT patch.

1. Change to the kernel directory:
   ```
   cd <UEIPAC SDK directory>/kernel
   ```

2. If using the Linux kernel with Xenomai patches, run the **build_xenomail.sh** script.
   ```
   ./build_xenomai.sh
   ```

3. Run the **get_kernel.sh** script:
   - For Linux with Xenomai, run:
     ```
     ./get_kernel.sh -xenomai –cpu 834x
     ```
   - For Real-Time Linux, run:
     ```
     ./get_kernel.sh -rt –cpu 834x
     ```

> NOTE: 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 `linux-v<LINUX_VER>` directory containing the kernel source with patches applied:

- For Linux with Xenomai: `linux-v4.9.51` (or whatever the current version is of the Linux kernel with Xenomai patches).
- For Real-Time Linux: `linux-v4.4.115` (or whatever the current version is of the Real-Tim Linux kernel).

4. Change the current directory to the Linux source directory.
5. Run the **build_kernel.sh** script:
   - For Linux with Xenomai, run:
     ```
     ./build_kernel.sh -xenomai –cpu 834x
     ```
   - For Real-Time Linux, run:
     ```
     ./build_kernel.sh -rt –cpu 834x
     ```

You can find the built kernel in **arch/powerpc/boot/cuImage.ueipac834x** and the modules in **module_ueipac834x**.

NOTE: The build scripts build the default UEIPAC kernel. You can customize the kernel configuration by using the menuconfig tool:

```
make ARCH=powerpc CROSS_COMPILE=powerpc-604-linux-gnu- menuconfig
```

You can then follow the build script to compile the kernel and modules and install the modules.

## E.4 Build the kernel for UEIPAC-XXX-03 versions

The following procedure is used to download, configure and build the kernel for the UEIPAC-XXX-03 systems.

Note that with these UEIPAC-XXX-03 systems, you have the option of using the Linux kernel with Xenomai patches or the Real-Time Linux kernel with the PREEMPT_RT patch.

1. Change to the kernel directory:
   ```
   cd <UEIPAC SDK directory>/kernel
   ```

2. If using the Linux kernel with Xenomai patches, run the **build_xenomail.sh** script.

```
./build_xenomai.sh
```

3. Run the **get_kernel.sh** script:
    - For Linux with Xenomai, run:
      ```
      ./get_kernel.sh -xenomai –cpu 834x -o 3
      ```
    - For Real-Time Linux, run:
      ```
      ./get_kernel.sh -rt –cpu 834x –o 3
      ```

   NOTE: 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 `linux-v<LINUX_VER>` directory containing the kernel source with patches applied:

- For Linux with Xenomai: `linux-v4.9.51` (or whatever the current version is of the Linux kernel with Xenomai patches).
- For Real-Time Linux: `linux-v4.4.115` (or whatever the current version is of the Real-Tim Linux kernel).

4. Change the current directory to the Linux source directory.

5. Run the **build_kernel.sh** script:
    - For Linux with Xenomai, run:
      ```
      ./build_kernel.sh -xenomai –cpu 834x -o 3
      ```
    - For Real-Time Linux, run:
      ```
      ./build_kernel.sh -rt –cpu 834x -o 3
      ```

You can find the built kernel in **arch/powerpc/boot/cuImage.ueipac834x** and the modules in **module_ueipac834x**.

NOTE: The build scripts build the default UEIPAC kernel. You can customize the kernel configuration by using the menuconfig tool:

```
make ARCH=powerpc CROSS_COMPILE=powerpc-604-linux-gnu- menuconfig
```

You can then follow the build script to compile the kernel and modules and install the modules.

# Appendix F. Converting root file system to read only

## F.1 Modify RFS on SD card

1. Edit /etc/fstab as follow to mount a ram disk at /var (ram disk maximum size is set to 2MBytes):

```
/dev/sdcard1    /           ext3    defaults,noatime    1   1
none            /proc       proc    defaults            0   0
none            /sys        sysfs   defaults            0   0
none            /dev/pts    devpts  defaults            0   0
tmpfs           /var        tmpfs   defaults,size=2M    0   0
```

2. Create a new script /etc/varsetup.sh with the content below. It sets up the folders needed in /var and maps a few writable folders at /tmp, /mnt and /home

```
mkdir /var/tmp
mkdir /var/log
mkdir /var/lib
mkdir /var/lib/misc
mkdir /var/spool
mkdir /var/spool/cron
mkdir /var/spool/cron/crontabs
mkdir /var/run
mkdir /var/lock
mkdir /var/mnt
mkdir /var/home

mount --bind /var/tmp /tmp
mount --bind /var/mnt /mnt
mount --bind /var/home /home
```

3. Edit /etc/inittab as follow to execute varsetup.sh

```
# Mount all filesystem listed in /etc/fstab
::sysinit:/bin/mount –a

# Create and mount non-persistent folders
::sysinit:/etc/varsetup.sh

# Configure local network interface
::sysinit:/sbin/ifconfig lo 127.0.0.1 up
::sysinit:/sbin/route add -net 127.0.0.0 netmask 255.0.0.0 lo

# run rc scripts
::sysinit:/etc/rcS
```

```
# Start a shell on the console
ttyS0::respawn:-/bin/sh

# unmount root file system when shutting-down
::shutdown:/bin/umount -a -r
```

4. Create symbolic links to files stored in /etc that need to be kept writeable.

```
ln –s /var/resolv.conf /etc/resolv.conf
ln –s /var/layers.xml /etc/layers.xml
```

## F.2 Configure U-Boot

1. Connect the console serial port, power-up the UEIPAC and press a key to enter U-Boot.

2. Type the following commands to load the root file system as read-only:

```
setenv bootargs console=ttyS0,57600 root=62:1 ro
saveenv
```

# Appendix G. Updating RAM disk image

The UEIPAC software CDROM contains a RAM disk image **uRamdisk-x.y.z** that you can upload to flash and boot from on 1G and R UEIPAC models (see section 3.2).

This appendix explains how to modify this image to add your own files (typically your program and associated configuration files).

This operation can only be done on a Linux workstation. You also need to install the uboot mkimage utility. For example, with Ubuntu type:

```
bash$ sudo apt-get install uboot-mkimage
```

1. Extract compressed RAM disk image from uImage file. The following command converts the file **uRamdisk-x.y.z** to **ramdisk.gz:**

   ```
   bash$ dd if=uRamdisk-x.y.z bs=64 skip=1 of=ramdisk.gz
   21876+1 records in
   21876+1 records out
   ```

2. Uncompress RAM disk image:

   ```
   bash$ gunzip -v ramdisk.gz
   ramdisk.gz:      66.6% -- replaced with ramdisk
   ```

3. Mount RAM disk image:

   ```
   bash$ mount -o loop ramdisk /mnt/tmp
   ```

Now you can add, remove, or modify files in the `/mnt/tmp` directory. Once you are done, you can re-pack the RAM disk into a U-Boot image:

1. Unmount RAM disk image:

   ```
   bash$ umount /mnt/tmp
   ```

2. Compress RAM disk image :

   ```
   bash$ gzip -v9 ramdisk
   ramdisk:         66.6% -- replaced with ramdisk.gz
   ```

3. Create new U-Boot image:

   ```
   bash$ mkimage -T ramdisk -C gzip -n 'UEIPAC RAM disk' -d
   ```

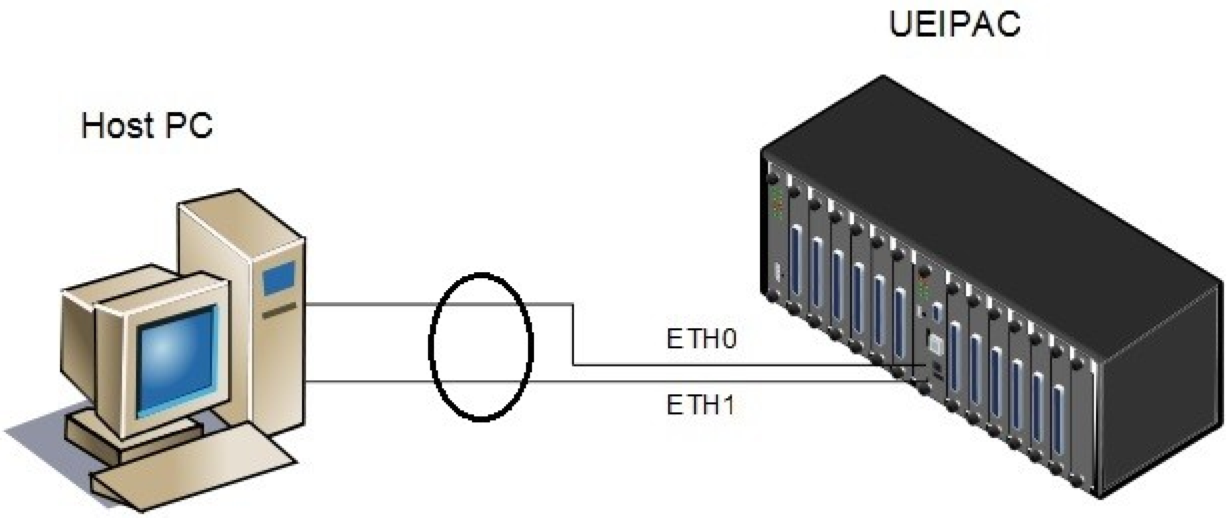
```
ramdisk.gz new-uRamdisk-x.y.z
Image Name:   UEIPAC RAM disk
Created:      Wed Apr 11 17:32:41 2012
Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
Data Size:    2425561 Bytes = 2368.71 kB = 2.31 MB
Load Address: 0x00000000
Entry Point:  0x00000000
```

# Appendix H. Bonding/Teaming Ethernet ports

Teaming/Bonding describes various methods to combine multiple network links in parallel to provide redundancy and/or increase data throughput.

This chapter describes the configuration of a fault tolerant link between a UEIPAC and a host PC. Bonding is only possible on UEIPACs equipped with two independent network ports.

In this mode, only one network adapter (the primary) is active. The secondary adapter takes over as soon as it detects that the primary adapter can't connect to its peer.



1. Power-up your UEIPAC and open a serial terminal program.
2. Tear-down existing network connections.

```
ifconfig eth0 down
ifconfig eth1 down
```

3. Load bonding kernel module with parameters set to the following:

- monitor the link to the host PC every 500ms
- fault tolerance mode
- use **eth0** as primary connection

```
modprobe bonding arp_ip_target=192.168.103.1 arp_interval=500
mode=active-backup primary=eth0
```

4. Bring-up **bond0** connection.

```
ifconfig bond0 up
```

5. Assign **eth0** and **eth1** as slaves to **bond0.**

```
ifenslave bond0 eth0 eth1
```

6. Configure an IP address for **bond0.**

```
ifconfig bond0 192.168.103.2 netmask 255.255.255.0
```