Custom Linux: A Porting Guide

Porting LinuxPPC to a Custom SBC

Shie Erlich

<shie@myrealbox.com>

Rafi Yanai – my partner in the porting process

Avi Rubenbach – without whom this wouldn't be possible

Revision History

Revision 2.1	2003–03–08	Revised by: gjf
Modified code example per author		
Revision 2.0	2002–06–13	Revised by: tab
Added GFDL per author		
Revision 1.0	2002–05–13	Revised by: SE
Initial release		

Table of Contents

Table of Contents

Chapter 1. Introduction

1.1. Who needs to read this ?

This guide describes a work in progress, to port Linux to a custom PowerPC−based board. This means making the operating system work on unfamiliar hardware. Anyone who is on the same track might benefit from reading this paper, as it highlights the pitfalls and problematic points along the way.

1.2. What do I need to know (why so much) ?

Before attempting to port Linux, know at least the following: (whenever possible, a link to a proper information source is attached)

- Hardware: know what hardware you've got, how it works (if it works), and how is it initialized. Get all the hardware manuals you can − you'll probably need them. Also, never assume the hardware works the way it supposed to ! Hardware people do the darnest things :−(
- Basic understanding of drivers and how they work in Linux. Programming knowledge of simple drivers is an advantage − but not a must. http://www.tldp.org/HOWTO/Module−HOWTO/index.html
- How to work with Vision−ICE, how configure it and use it to load a binary kernel into the target RAM. Also, at the beginning, you'll need to know how to use ICE to debug in assembly.
- How to compile and configure a Linux kernel. http://www.tldp.org/HOWTO/Kernel−HOWTO.html
- The Linux boot process. http://www.tldp.org/HOWTO/BootPrompt−HOWTO.html
- Working knowledge of C programming is a must. Some assembly is sure to help. Also, it is best to get to know Makefiles. They tend to raise their ugly head once in a while.
- The Internet is your friend. All the information you need is probably on the net. You just have to know how to find it. Google is a good way to start; mailing lists and news groups usually keep the real gold.
- How to install Linux, configure it, administrate it and basically take care of everything it needs. This guide does not cover anything regarding system administration, setting up a server etc.

1.3. The tools

This section describes the tools we used during the process. Most are trivial to install and use. When neccesary, consult the appropriate url or manual.

- *HardHat Linux*: First and foremost, HHLinux, now known as MontaVista Linux, is the distribution we started with. The distribution contains LSPs (same as BSPs) for PowerPC in a number of board configurations. For porting to our board, we took the LSP which is closest in hardware to our Artysyn PMPPC board, and started from there.
- *LXR*: This is THE killer tool, which allowed us to port Linux in a very short time. LXR is a cross referencer, which means it reads a piece of code (the Linux kernel, for example), and then allows browsing the code, searching through it and much more. I cannot emphasize enough how important this tool is. To see what the end result looks like, look at http://lxr.linux.no/source. LXR itself can be downloaded at http://lxr.sf.net
- *VisionICE*: A hardware debugger, which has the ability to stop, run and add breakpoint straight in the CPU. VisionIce is very usuful when no operating system is running, and allows to step in the kernel during boot process. The application can also be used to take a binary image of a kernel, load it into the target's RAM memory and run it − useful when you've got no boot loader.

- *CVS*: A version control system, allows you to keep multiple versions of the code. Other than backing up the code, it allows diffing between different version, and reverting to older version, when needed.
- A terminal program, like HyperTerminal or ProCOMM for Windows", or minicom for Linux.

1.4. The hardware

The board is based on PPC750 (PowerPC) processor. It is 6U VME64 standard. The board is designed to host two PCI Mezzanine cards (CCPMC) − Mezzanine cards that comply with Std CCPMC1386 can be attached.

- COP connector.
- 1 MB of L2 cache.
- CPC700 system controller.
- 128 MB SDRAM with ECC.
- Flash memory, divided to boot flash and user flash.
- NVRAM memory.
- I/O discretes.
- RS232 channels.
- General purpose registers.
- PCI 2.1 local bus.
- 10/100 BaseT ethernet channel.
- VME64 system bus.

1.5. Copyright & License

Chapter 2. Bootcamp: How To Begin ?

2.1. Creating a development environment

The minimum requirments are obviously a development station and a target. However, the recommended way of working is having a third host which acts as a server. The server runs several services such as ftp, telnet, NFS, tftp (if needed) and CVS. The main role of the server is to run CVS and track version control, however once you can boot the target from network, the server will also hold the target images, and filesystem, which makes development much easier.

Regardless, the first step is to install a tool−chain (compiler, linker etc.) for your target. The HardHat Linux cdrom includes all the needed files, and the installation sequence is documented in the HardHat Linux documentation. During the installation, you must select your LSP (basic software for the selected board), and HardHat will install a set of tools and a kernel source tree matching your LSP.

We had a board that had vxWorks running on it, so we setup the target to boot using the standard vxWorks loader. Once the loader initiated, we used visionICE to take−over the target (so that vxWorks won't load an image file) and load a Linux image into the target. What you need to do at this point is get an ICE, connect it to the network and to the target − through a JTAG connection − and install the ICE software on your host.

What should have been done so far:

- A Linux host installed, and the HardHat tool−chain.
- A working target (hardware should be functional)
- ICE is connected to the target and the network and its software usable.
- Optionally, a server running CVS, telnetd, NFS and FTP.

2.2. Compiling the first kernel

If you've installed the Linux kernel that comes with HardHat, then cross−compiling should already be enabled in the kernel `Makefile`. If your kernel is not from the HardHat CD, you should enable cross−compiling in the `Makefile` by defining a CROSS_COMPILE entry in the following manner: (a code segment from the main Makefile)

```
CROSS_COMPILE   = /opt/hardhat/devkit/ppc/7xx/bin/ppc_7xx-
AS              = $(CROSS_COMPILE)as
LD              = $(CROSS_COMPILE)ld
CC              = $(CROSS_COMPILE)gcc
```

The Linux kernel is modular, and allows you to configure it and choose which "blocks" should be compiled with the kernel. In order to do this, first **cd /usr/src/linux** (assuming your kernel source code is installed at /usr/src/linux). Once there, type **make xconfig**.After saving your options, you should **make vmlinux** to create a kernel image suitably for using with VisionICE.

We will not go into more details here, as it's outside the scope of this document. For more information, try http://www.tldp.org/HOWTO/Kernel−HOWTO.html

2.3. Booting the machine

First, configure the terminal program, in our case minicom, the following way: 9600 bps, 8 bits, no parity, 1 stop bit and no flow control of any kind. The serial port in Linux should be `/dev/ttyS0` for `COM1`, `/dev/ttyS1` for `COM2` etc.

Start the target. You should see the vxWorks bootloader on your terminal screen, and should be able to stop the boot sequence by pressing the space bar.

We *cannot use the vxWorks bootloader to load a Linux kernel* since it looks in the ELF header and loads the image to the address written there. However, the Linux kernel, which uses virtual memory, is linked to a high−memory address, and vxWorks can't handle that.

Once the target is stopped, run the VisionICE software and perform the following steps:

- Initialize the target by pressing **Target|Initialize**
- Press **File|Load Executable**. A dialog box will open, asking you to choose a file. Please choose your kernel image (vmlinux). Before pressing **Load**, don't forget to enter a value in the *`+/- Bias`* field.

  The bias field makes it possible to tell ICE to load a certain image in a different address than what's stated in the ELF binary. We wanted to load the kernel into address `0x300000`, and since the binary was linked to `0xC0000000`, we entered `−0xBFD00000`.
- Once the image is loaded succesfully, you can press **Run** or **Step** to start executing your kernel.

After pressing the **Run** button, nothing happened. At that moment, and for some time after, it seemed that nothing was happening and the kernel was stuck. We used ICE to step through the initialization code of the kernel and rule out some potential problems, like virtual memory errors, only to finally discover that the problem was simple: the kernel was indeed booting but since the console (tty) driver had problems, we couldn't see anything!

VisionICE is not the correct tool to use when debugging Linux. ICE doesn't know about virtual memory and protected mode (at least the version we had), and since the Linux kernel turns on virtual memory very early, ICE is only useful for debugging the first assembler statements. After VM is turned on, ICE starts crashing and giving wierd results.

Chapter 3. Booting In The Dark

3.1. Debugging with print_str()

As stated in the previous chapter, the machine starts to boot, but nothing happens. At least, nothing that we can see. The screen is blank and no kernel messages appear. At this point, you have to ask yourself, is it really booting?

Since the console wouldn't start, and ICE died real fast, we had no choice. We had to debug somehow, and the oldest way is good here – printing to the screen. Obviously, we couldn't use `printk()`, so we wrote a short function which pushes characters straight into the serial port. We used the boot process "map" shown in the previous section, and inserted some prints along the way. This helped us to know at what stage we are completing and where we're dying. The following piece of code prints a single character to the serial port, by polling it and waiting for it to be free.

```
                                 /* tx holding reg empty or tx    */
#define LSR_THREMPTY 0x20        /* fifo is empty (in fifo mode ) */
#define THR_REG      0x00        /* Transmit holding reg */
#define LSR_REG      0x05        /* Line status reg */
#define COM1_ADDRESS 0xFF600300  /* == replace with your UART address */

void print_char (char ch) {
        volatile  unsigned char status = 0;
        /* wait until txempty */
        while ((status & LSR_THREMPTY ) == 0)
                status = *((volatile unsigned char *)(COM1_ADDRESS + LSR_REG));

        *((volatile unsigned char *)(COM1_ADDRESS + THR_REG)) = ch;
}
```

There's a better code for printing directly to the serial port, however, it's a bit more complicated. You can find it in arch/ppc/boot/common/misc–common.c, using `puts()` or `putc()`.

3.2. Modifying code using compiler flags

Although it is not a porting issue, the way you modify your code matters. It's easier if you do it right the first time. The Linux kernel uses standard configuration flags CONFIG_XXXX (like CONFIG_PPC, CONFIG_ISA etc), which are used to mark a certain machine, architecture or device. We defined ourselves a new flag (let's call it CONFIG_TESTMACH), and surrounded our new/modified code with these flags:

```
....original code....
#ifdef CONFIG_TESTMACH
....modified code....
#else
....original code....
#endif /* CONFIG_TESTMACH */
```

To "activate" our code, we added the new flag to the kernel configuration file – `.config` – by adding **CONFIG_TESTMACH=y** to it. In the first stage, this solution allows you a quick way to find the code you changed, but later the flag you chose will allow you to add your code into the kernel tree and into the configuration program (**make xconfig**).

3.3. Getting the console to work

3.3.1. Forcing the kernel to boot our−way

Once we discovered the kernel was indeed booting, but the console wasn't printing, it was time to begin. First, we forced the kernel to boot using a specified configuration for the serial port, in our case `9600n1`, and did not allow any command line options or boot time considerations etc.

The first place to go is `drivers/char/tty_io.c`, to `console_init()`. This function determines the console configuration at startup. Here's a small part of it:

```
memset(, 0, sizeof(struct termios));
memcpy(tty_std_termios.c_cc, INIT_C_CC, NCCS);
tty_std_termios.c_iflag = ICRNL | IGNPAR;
tty_std_termios.c_oflag = OPOST | ONLCR;
tty_std_termios.c_cflag = CLOCAL | B9600 | CS8 | CREAD;
tty_std_termios.c_cflag &= ~(CRTSCTS);
tty_std_termios.c_lflag = ISIG | ICANON | ECHO | ECHOE | ECHOK | ECHOCTL | ECHOKE | IEXTEN;
tty_std_termios.c_iflag = ICRNL | IXON;
tty_std_termios.c_oflag = OPOST | ONLCR;
tty_std_termios.c_cflag = B38400 | CS8 | CREAD | HUPCL;
tty_std_termios.c_lflag = ISIG | ICANON | ECHO | ECHOE | ECHOK | ECHOCTL | ECHOKE | IEXTEN;
```

The first (naive) thing we tried, was to configure the console the way we wanted. *Of course, this didn't help us much ;−)*

Disappointed but not discouraged, we remembered that we didn't have a bootloader yet, and that we didn't really know if any option was being passed on to the kernel. "Maybe the kernel gets some garbage for command line?" we (again, naively) thought. So we tried to stop the kernel from parsing command−line options, and manually inserted our command line. *This didn't help us much ;−)*

3.3.2. Non−standard hardware − just say no!

At that point, we didn't have a console, but we had time. So we dove a bit deeper into the console issues. Looking at `drivers/char/serial.c`, we came across `serial_console_setup()`. This function, apart from parsing command−line options, also configures the serial port by writing directly to it. Our hardware people decided it was a good time to let us know that our serial port wasn't standard. The lines that are used for flow control were not connected. We decided to remark−out the following line, which sets the RTS and DTR lines high, because we just didn't have them.

```
serial_out(info, UART_MCR, UART_MCR_DTR | UART_MCR_RTS);
```

Ofcourse, this didn't help us much :−(The lesson learned here was *check, check, check your hardware!*. Custom boards might not be standard, and the porting will go a lot quicker if you know about it.

3.3.3. Let there be light: calculating baud rate

Finally, we decided to check the baudrate. Did Linux mean what we thought it meant when it said 9600? Possibly not, since we didn't know how it computed that value. We've noticed that the file(s) `include/asm−ppc/pmppc_serial.h` (replace pmppc with your board name) included a definition of

`BAUDBASE`, which is later used for everything regarding serial ports. It was computed using the board's local bus frequency, bus clock to system clock ratio etc. This seemed wrong, so we checked out what the base baud was in a vxWorks system we had running on the board, and changed it to:

```
/*
 * system clock = 33Mhz, serial clock = system clock / 4
 * the following must hold: (divisor * BaudRate) == (System clock / 64)
 */
#define BASE_BAUD (33000000 / 4 / 16)
```

A quick compilation, and a reboot later we had a booting kernel visible through our serial port. Success!

Chapter 4. Linux Still Isn't Booting

4.1. Memory probing, RTC and decrementors

Now that the console was working, we could see the real problems. The system wasn't booting yet. Since we were working with C code, we traced the code, and found that a function called `sdram_size()` wasn't completing correctly. The function probed a register for the size of the RAM, a register our board doesn't have. We made the function return a given value of 128MB, it's an ugly hack, but our board doesn't have a way of knowing the amount of RAM.

Wwe had the same problems with a bunch of functions called todc_XXXX, mainly `todc_get_rtc_time()`, `todc_set_rtc_time()`, and `time_init()` since we don't have a RTC (real−time clock) chip on our board, and those functions were using it. For the time being, we made the `todc_XXX` function only set and get a constant date and time, since our board doesn't have a bios battery and so cannot keep time when powered off.

Once all this was done, we found `todc_calibrate_descr()`, which again uses the RTC chip. We had to replace that function with our own:

```
void calibrate_decr() {
        int freq, divisor;
        freq = bus_freq();
        divisor = 4;
        tb_ticks_per_jiffy = freq / HZ / divisor;
        tb_to_us = mulhwu_scale_factor(freq / divisor, 1000000);
}
```

4.2. Big−little endian (we should have known)

4.2.1. Probing the CPC700

Finally, we reached the PCI−probing part of the boot process, only to discover that it didn't work. We tried communicating with the CPC700 using `cpc700_read_local_pci_cfgb()`, which was supplied along with the PMPPC's LSP, and tried to read CPC's config register. We should have gotten `0x1014`, which is the vendor ID, but we didn't. We realized that we were talking little−endian and the CPC was listening in big−endian. We made a small patch to the functions, so that we spoke big−endian to the CPC700. We could then read the vendor ID correctly, but the rest of it still didn't work. We didn't want to alter the code so that everything would be done big−endian style.

4.2.2. Making CPC700 speak little−endian

We discovered that the CPC700 can be initialized to do automatic byte−swapping, which does little−to−big endian convertion on the fly. As it seems, our board was initialized to do just that. We added a small code segment in `setup_arch()`, which checks if byte−swapping is enabled, and if so, disables it:

```
while (cnt<2) {
        cpc700_read_local_pci_cfgb(0, );
        cpc700_read_local_pci_cfgb(1, );
        if (l == 0 && h == 0) {
                if (cnt == 0) {
```

```
                        printk("CPC700 byte swaping enabled - trying to disable ... ");
                        cpc700_write_pifcfg_be(0x18, 0); /* disable byte-swapping */
                } else {
                        printk("FAILED !!\n");
                        break;
                }
        } else {cd
                printk("byte swapping disabled.\n");
                break;
        }
        ++cnt;
}
```

A short compilation later, PCI probing was working! We got some beer and partied ;–)

4.3. Ethernet: our first PCI device

Our board uses an Intel ethernet chip, called i82559er, which has a module called *eepro100*. After compiling the module and booting, we discovered that the module isn't working, although an ethernet device was found. We guessed that it was an irq problem, and that the devices don't get the IRQs they need. We modified a function called `pmppc_map_irq()` to map our ethernet devices:

```
XXXX_map_irq(struct pci_dev *dev, unsigned char idsel, unsigned char pin) {
        static char pci_irq_table[][4] =
        /*
         *      PCI IDSEL/INTPIN->INTLINE
         *      A       B       C       D
         */
        {
                {22,    0,      0,      0},/* IDSEL 3 - Ethernet */
                {0,     0,      0,      0},/* IDSEL 4 - unused   */
                {0,     0,      0,      0},/* IDSEL 5 - unused   */
                {0,     0,      0,      0},/* IDSEL 6 - ????     */
                {0,     0,      0,      0},/* IDSEL 7 - unused   */
                {0,     0,      0,      0},/* IDSEL 8 - unused   */
                {0,     0,      0,      0},/* IDSEL 9 - unused   */
        };

        const long min_idsel = 3, max_idsel = 9, irqs_per_slot = 4;
        return PCI_IRQ_TABLE_LOOKUP;
}
```

The function maps IRQs according to IDselects, which means in the order on the PCI bus by which the devices are set. This structure is a bit tricky: *min_idsel* denotes the topleft corner of the array, and *max_idsel* is the bottomleft corner. *irqs_per_slot* is the number of IRQs per line. The structure is as follows:

```
each cell contains (IDSEL, SLOT#, IRQ)
                +---------------------------------------+
                | (3,0,22) | (3,1,0) | (3,2,0) | (3,3,0) |
                +---------------------------------------+
                | (4,0,0)  | (4,1,0) | (4,2,0) | (4,3,0) |
                +---------------------------------------+
                                                        ..........
                                                        ..........
                +---------------------------------------+
                | (9,0,0)  | (9,1,0) | (9,2,0) | (9,3,0) |
                +---------------------------------------+
```

As you can see, our i8559er needs IRQ 22, and is seated in IDselect 3. Of course, we didn't know that at the start, so we wrote a small piece of code that read all the vendor IDs in all the IDselects. Once done we compiled, but the ethernet device still didn't work.

The next problem was that the module couldn't decide on a MAC address for the device. The MAC address should be written on an EEPROM chip (connected to the device), but we discovered that the hardware guys decided that i82559 doesn't need the EEPROM, so they removed it. After hardcoding a MAC address inside `eepro100.c`, the ethernet device finally worked. The final solution was to make the module read the MAC address from NVRAM memory, and if no other choice was available, to fall back to a default MAC address.

The next step was to mount a NFS root filesystem. For details see the documentation in Documentation/nfsroot.txt

4.4. Some Miscellaneous Issues

We had new problems, some would say good problems. We didn't have a bootloader yet, however we needed to pass a command line to the kernel at boot time. We hard−coded the command line into the kernel inside the `parse_options()`. After that was finished, we made `console_init()` and `serial_console_setup()` work the way they should. They no longer ignored the command line, but still RTS and DTR stay low.

Another important issue was memory mapping. The file `arch/ppc/mm/init.c` contains a function called `MMU_init()`. This function is actually a big **switch** statment, divided by the machine type. Each machine maps its memory using the `setbat()` and `ioremap()` functions. The BAT mechanism is a way of translating virtual addresses into physical ones. Thus, `setbat()` is used by specifying a virtual address, a physical address and a page size. Not every size can be used here; you should use one of the finite set of sizes, ranging from 128KB to 256MB. We mapped our IO memory so that virtual equalled physical.

As mentioned, there is another way of mapping memory − `ioremap()`. `ioremap()` is used to map physical addresses into virtual ones, making them available to the kernel. The function *does not allocate any memory*, simply returns a virtual address by which one can access the memory region. The following is a snippet from `MMU_init()`:

```
case _MACH_mymachine:
        setbat(0, LOW_IO_VIRT_BASE, LOW_IO_PHYS_BASE, LOW_IO_SIZE, IO_PAGE);
        ioremap(UNIVERSE_BASE,UNIVERSE_SIZE);       /* Universe VME */
        ioremap(EEPRO100_BASE,EEPRO100_SIZE);       /* Ethernet EEPRO100 */
        break;
```

As you can see, we don't take the return value of `ioremap()`. We don't need it, since at this stage the kernel maps the addresses so that virtual address == physical address.

Chapter 5. Linux Is Booting ... What Now ?

5.1. The 64 bit barrier

The CPC700 has a "feature" which is supposed to make some memory access use 64 bit wide. This is a problem since some test–and–set registers on our board might get set unintentionally, because we were trying to read something 16 bits lower. In order to solve this situation, we set the memory controller to 64 bit wide intervals. If you try to access those areas in another manner (8 or 16 bit access), the CPC700 simply throws them away. We had to be able to read/write those areas, since important "discretes" (controlled by an Altera device) were mapped there.

In order to access those areas, we needed a function that does a 64 bit write. As far as I know, doing a 64 bit write on a PowerPC is possible in two ways: using cache lines and using a floating point register. The floating point register is a 64 bit sized register, so when we write it, the whole 64 bit get written. The problem is that you can't do floating point in the kernel. Since the kernel doesn't save the floating point registers during context switch, it doesn't allow FP, and will throw an exception if done in the kernel.

After messing with cache lines, we decided to go the FP way, and added the following function:

```
void out64(__u32 addr, long long *pVal) {
        __u32 flags, tmp_msr;

        save_flags(flags);
        cli();
        tmp_msr = __get_MSR();
        tmp_msr |= MSR_FP;
        tmp_msr &= ~(MSR_FE0 | MSR_FE1);
        __put_MSR(tmp_msr);

        sysOut64(addr, pVal);
        __put_MSR(flags & ~(MSR_EE));
        restore_flags(flags);
}
```

The function adds a floating point to the PowerPC MSR register, and makes sure that no exceptions will be generated as a result of doing FP. Once done, it uses an assembly code, described below in the `sysOut64()` to do the actual floating–point operation. Note that the function turns off interrupts, but this is acceptable here, since we use the function on rare occasion.

```
_GLOBAL(sysOut64)
stwu    r1, -DEPTH(r1)
mflr    r0
stw     r31, FP_LOC(r1)
stw     r0,  LR_LOC(r1)
mr              r31, r1
stfd    fr0, FPR_SAVE(r31)      /* save floating point reg contents */

lfd     fr0,0(r4)
stfd  fr0,0(r3)
eieio

lfd     fr0, FPR_SAVE(r31)      /* restore floating point value */
lwz     r4, 0(r1)               /* now restore the stack frame  */
lwz     r0, 4(r4)
mtlr    r0
```

```
lwz     r31, -4(r4)
mr      r1, r4
blr
```

5.2. Booting from flash

While Linux was booting using an NFS filesystem, this was not enough. For an actual field product, we needed Linux to boot from an independent device, without the need for a network at all. We decided to create a special kind of image, called *initrd*, which is basically a Linux kernel with a compressed file. The compressed file includes a Linux filesystem. The filesystem is unpacked to a ramdisk on boot, and mounted as the root filesystem.

During the boot process, the bootloader relocated the kernel image to address zero − which was fine, and the initrd part to a higher address. The area to which initrd was relocated was not mapped in our kernel's memory, and all we got was a kernel error (access to bad area). After modifying the bootloader to relocate initrd to a different address, all was fine and Linux booted succesfully.

If your board has some NVRAM memory, it would be a good idea to use it for bootloader purposes. After writing a module for our NVRAM memory (out of scope for this paper), we modified the bootloader, so that the kernel command−line, and MAC address were saved in NVRAM. When the bootloader starts, it checks NVRAM and if it is initialized (by a certain magic number), the bootloader uses the command line written there. Otherwise, the bootloader reverts to a default command line, allowing the user to edit it.

Appendix A. GNU Free Documentation License

Version 1.1, March 2000

A.1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

A.2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front−matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front−Cover Texts or Back−Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine−readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard−conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine−generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A.3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.4. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front−Cover Texts on the front cover, and Back−Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine−readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly−accessible computer−network location containing a complete Transparent copy of the Document, free of added material, which the general network−using public has access to download anonymously at no charge using public−standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front−matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties−−for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front−Cover Text, and a passage of up to 25 words as a Back−Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front−Cover Text and one of Back−Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

A.7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self–contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

A.9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

A.10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

A.11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.12. How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front−Cover Texts being LIST, and with the Back−Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front−Cover Texts, write "no Front−Cover Texts" instead of "Front−Cover Texts being LIST"; likewise for Back−Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix B. Trademarks

Linux® is a registered trademark of Linus Torvalds.

MontaVista" is a trademark of MontaVista Software Inc.

PowerPC® is a registered trademark of IBM Corporation.

Windows® is a registered trademark of Microsoft Corporation.

vxWorks" and Vision ICE" are trademarks of Wind River Systems Inc.

ProCOMM" is a trademark of Symantec Corporation.