AN2744
Rev. 1, 12/2004

PMON Module—An Example of Writing Kernel Module Code for Debian 2.6 on Genesi Pegasos II

by Maurie Ommerman
CPD Applications
Freescale Semiconductor, Inc.
Austin, TX

This application note is the seventh in a series describing the Genesi Pegasos II system and its various applications.

Contents

1 Introduction

This application note describes the PMON kernel module interface and how to write, compile and link a kernel module. As an example the PMON facility, which is an application written by the application team and is described and developed in the applications note *Software Analysis on Genesi Pegasos II Using PMON and AltiVec* (AN2743).

This paper assumes that the user will log in as `guest` with password `guest`, and all the examples discussed in this paper are in the /root/ppctools/pmon directory. However, all the code and commands must be done as the root user in the /root directory. Hence, the guest user will need to switch to the root user with this command:

```
su -
```

2 Terminology

The following terms are used in this document:

Linux OS	Linux operating system
PMON	Performance monitor facility
gcc	GNU compiler collections and GNU utilities
GNU	GNU is not Unix (a recursive acronym)
Performance monitors	The MPC74xx processors contain registers that can be used to monitor system activity.
Kernel	The part of the Linux system which runs in memory and controls the system operations. Linux is composed of two parts, the kernel and the root file system.
Root file system	The collection of directories and files that are used by the kernel to affect all operation.
Module	A dynamic facility for attaching kernel code to a running kernel, allowing the kernel to be expanded and contracted as needed.
POSIX	The Portable Operating System Interface (POSIX) standardization effort used to be run by the POSIX standards committee.

3 Introduction to Building a Kernel Module on Debian Linux 2.6 Kernels.

The process described here is for modules in the 2.6 Linux Kernel family. The process for the 2.4 Linux kernels is different and will not be discussed here. In particular, this process works on the Genesi Pegasos II Linux 2.6 Debian kernel.

3.1 Building the Module

A special form of the Makefile and make command are used to build 2.6 kernel modules.

```
root@debian:~/ppctools/maurie-pmon# cat Makefile

#linux 2.6 makefile

obj-m := pmon26.o

clean:

        rm -rf *.o *.ko *.mod.c

root@debian:~/ppctools/maurie-pmon#
```

The target must be obj-m and the dependency is the name of the source file with the '.o' appended. Thus, for this example, the source file is pmon26.c and the dependency then is pmon26.o.

The clean target is optional and is used here to clean up the directory by removing all objects and kernel objects, denoted by .o and .ko respectively and the interim *.mod.c file.

The make invocation is in the script, build.sh.

```
root@debian:~/ppctools/maurie-pmon# cat build.sh

make -C /usr/src/kernel-source-2.6.4 SUBDIRS=$PWD modules
```

This make invocation uses the flag -C to change to the directory containing the kernel source, in this case /usr/src/kernel-source-2.6.4. The macro SUBDIRS indicates the directory containing the module code and in this case it is the local directory indicated by the shell variable $PWD. Finally, the keyword modules indicates that we are building a kernel module.

The commands generated by this call to make are shown below.

```
root@debian:~/ppctools/maurie-pmon# ./build.sh
make: Entering directory `/usr/src/kernel-source-2.6.4'
*** Warning: Overriding SUBDIRS on the command line can cause
***          inconsistencies
make[1]: `arch/ppc/kernel/asm-offsets.s' is up to date.
  CC [M]  /root/ppctools/maurie-pmon/pmon26.o
  Building modules, stage 2.
  MODPOST
  CC      /root/ppctools/maurie-pmon/pmon26.mod.o
  LD [M]  /root/ppctools/maurie-pmon/pmon26.ko
make: Leaving directory `/usr/src/kernel-source-2.6.4'
```

The kernel script MODPOST, which is really an executable is used to generate the compile and link options to build the kernel module, in this case pmon26.ko from the source pmon26.c.The modpost code is also available in the `/usr/src/kernel-source-2.6.4/script directory.`

At this point, the 2.6 kernel module, pmon26.ko, is available. It can be instantiated with the `insmod` command and destroyed with the `rmmod` command.The `modinfo` command gives some info about the module and only looks in the current directory or the /lib/modules directory.

```
root@debian:~/ppctools/maurie-pmon# insmod pmon26.ko
root@debian:~/ppctools/maurie-pmon# rmmod pmon26.ko
root@debian:~/ppctools/maurie-pmon# modinfo pmon26.ko
license:        GPL
vermagic:       2.6.4 gcc-3.3
depends:
```

3.2 Instantiating the Module

The module is instantiated by the `insmod` command and removed by the `rmmod` command. When this PMON module is started it writes a message to the kernel log and when removed it also writes a message to the kernel log. This log is accessible by the `dmesg` command. These messages are not part of the kernel or the instantiation, but are written by the code, thus messages for starting and stopping the module are optional. Section 4, "Writing the Module," explains these messages.

The command string below shows instantiating and removing the module and using `dmesg` to see the module start and stop messages.

```
root@debian:~/ppctools/maurie-pmon# insmod pmon26.ko

root@debian:~/ppctools/maurie-pmon# rmmod pmon26.ko


root@debian:~/ppctools/pmon# dmesg

Total memory = 256MB; using 512kB for hash table (at c0400000)

Linux version 2.6.4-pegasos (luther@pegasos2) (version gcc 3.3.3 (Debian 20040306)) #1 Mon Mar
22 12:47:08 CET 2004

..................

created proc entry for l2cr, major =254

removed pmon module

..................
```

3.3 Creating the Interface

For the case of PMON, we are using the interface /dev/pmon, which is created as a char device with all users have all permissions, i.e. 777. The user by accessing this device can interface to PMON with POSIX calls, like open, read, write, close, etc. The command to create this char device is `mknod`, specify the type, c, which is char type, the major number, which in this case is 254, and the minor number, which is 0. To verify that `mknod` completed successfully, do an `ls -l` on /dev/pmon.

Specifically, the major number is assigned by the `insmod pmon26.ko` command. The /proc/devices files shows which major number has been assigned. Thus after the `insmod`, list the /proc/devices file, determine the PMON major number, then perform the `mknod` command with that major number.

```
root@debian:~/ppctools/pmon# insmod pmon26.ko

root@debian:~/ppctools/pmon# cat /proc/devices

Character devices:

  1 mem

  4 /dev/vc/0

.... intervening items removed for this example

180 usb

254 pmon


Block devices:

  1 ramdisk

  3 ide0

  ... remaining items removed for this example
```

In this case the major number associated with PMON is 254 as shown above in the /proc/devices listing.

```
root@debian:~/ppctools/pmon
root@debian:~/ppctools/pmon# mknod /dev/pmon c 254 0
root@debian:~/ppctools/pmon# chmod 777 /dev/pmon
root@debian:~/ppctools/pmon# ls -l /dev/pmon
crwxrwxrwx    1 root     root     254,   0 Jul 12 16:28 /dev/pmon
```

4 Writing the Module

This example of the PMON facility consists of a single header file, pmon.h, and a single source file, pmon26.c.

Since PMON is implemented as a char device, the user can interface to it with POSIX function calls, like open, read, write, close, etc. Therefore, the code has function entry points with IO type functionality, all preceded with the pmon_ character string. Function names are therefore, pmon_open, pmon_read, pmon_write, etc. When the module is instantiated by the `insmod` command, the initialization function, pmon_init_module will be run, and when removed by the `rmmod` command, the clean up module, pmon_cleanup_module will be called.

A listing of the code is shown below, the numbers are here to allow for describing the code and should not be included when coding this program. The numbers are generated by the `cat -n` shell command.

```
root@debian:~/ppctools/pmon# cat -n pmon26.c
/*******************************************************************************
 * Kernel module for MPC74xx performance monitoring counters
 * Filename: pmon.c
 * Feature:  This kernel module is used to read and control all performance
 *           monitoring counters on MPC7410 and MPC744X processors
 * Note:     Dependent on /dev/pmon char device which is created after insmod
 *           is issued and a major number is shown in /proc/devices. Use
 *           "mknod /dev/pmon c 254 0"
 *           Once inode is created, user code can then access pmon counters via POSIX
 *           system calls, e.g. open, release, ioctl, read, write.
 *
 * ToDo:
 * Author: Jacob Pan
 *
 * History:
 *
 ******************************************************************************/
#include <linux/module.h>
#include <linux/config.h>
#include <linux/init.h>
```

```
MODULE_LICENSE("GPL");
#include <asm/uaccess.h>
#include <asm/ioctl.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/string.h>
#include <linux/errno.h>
#include <linux/devfs_fs_kernel.h>
#include <linux/proc_fs.h>
#include "pmon.h"
static int pmon_read(struct file *file, char* buf, size_t count, loff_t *ppos);
static int pmon_write(struct file *file, const char* buf, size_t count, loff_t *ppos);
static int pmon_open(struct inode *inode, struct file *file);
static int pmon_release(struct inode *inode, struct file *file);
static int pmon_ioctl(struct inode* inode, struct file *file, \
                      unsigned int cmd, unsigned long arg);
static int pmon_proc_read(char* buf, char** start, off_t offset,\
                   int count, int* eof, void* data);
static int major;
/* the ordinary device operations */
static struct file_operations pmon_fops =
{
  owner:   THIS_MODULE,
  read:    pmon_read,
```

```
  write:   pmon_write,
  open:    pmon_open,
  release: pmon_release,
  ioctl:   pmon_ioctl
};
/* device open method */
int pmon_open(struct inode *inode, struct file *file)
{

  return(0);
}

/* device close method */
int pmon_release(struct inode *inode, struct file *file)
{

  return(0);
}
static int pmon_read(struct file *file, char* buf, size_t count, loff_t *ppos)
{
  unsigned int cycles;
  asm volatile ("mfspr %0, 937" : "=r" (cycles));
  __copy_to_user(buf, &cycles, sizeof(cycles));

  return(4);//four bytes

}
/*******************************************************************************
 * Function: pmon_write()
*          this function is used to write performance monitoring mode control registers
 * Arguments: data passed from application/usr code include selections of PMC events
 *             in char size pointed by char* buf, the user code identifies CPU info to
 *            determine how many PMC counters are available in the CPU. Total PMC number
 *             is passed in count.
 * TODO:      It might be a good idea to check PVR and verify the total PMC counts are
```

```
 *            correct.
 *
 *
 */
static int pmon_write(struct file *file, const char* buf, size_t count, loff_t *ppos)
{
  unsigned int mmcr0=0, mmcr1=0, pmc_default=0;
  unsigned int* pmcSel= (unsigned int*)buf;
  printk("INFO: calling pmon_write %d bytes\n", count);
  mmcr0 |= ((pmcSel[0] <<(31-25))|(pmcSel[1]));
  mmcr1 |= ((pmcSel[2] <<(31-4)) |(pmcSel[3] <<(31-9)));
  /* clear all counters */
  asm volatile ("mtspr 953, %0" : : "r" (pmc_default));
  asm volatile ("mtspr 954, %0" : : "r" (pmc_default));
  asm volatile ("mtspr 957, %0" : : "r" (pmc_default));
  asm volatile ("mtspr 958, %0" : : "r" (pmc_default));
//  if(count == 6)
    {
        mmcr1 |= ((pmcSel[4] <<(31-14))|
                   (pmcSel[5] <<(31-20)));
        asm volatile ("mtspr 945, %0" : : "r" (pmc_default));
        asm volatile ("mtspr 946, %0" : : "r" (pmc_default));

    }
  /* freeze all counters, let usr use ioctl to control start and stop */
  //  mmcr0 |= 0x80000000;
  asm volatile ("mtspr 952, %0" : : "r" (mmcr0));
  asm volatile ("mtspr 956, %0" : : "r" (mmcr1));

  return(0);
}


static int pmon_ioctl(struct inode* inode, struct file *file, \
                       unsigned int cmd, unsigned long arg)
```

```
{

  return(0);

}
static int pmon_proc_read(char* buf, char** start, off_t offset,\
                          int count, int* eof, void* data)
{
  unsigned int upmc1;
  int len=0;
  asm volatile ("mfspr %0, 1017" : "=r" (upmc1));
  len += sprintf(buf+len, "L2CR 0x%08x\n", upmc1);
  *eof =1;
  return (len);
}

/* load the module */
static int __init pmon_init_module(void)
{
  struct proc_dir_entry* entry;
  mode_t mode =0;
  if ((major=register_chrdev(0, "pmon", &pmon_fops))<0) {
    printk("pmon: unable to register character device\n");
    return (-EIO);
  }
  /* Example showing adding SPR read entry in proc dir */
  entry = create_proc_read_entry("l2cr",mode ,NULL, pmon_proc_read, NULL);
  if (entry)
    printk(KERN_INFO "created proc entry for l2cr, major =%d\n",major);
  else
    printk(KERN_INFO "failed proc entry for l2cr\n");
   return (0);
}

/* remove the module */
```

```
161  static void __exit pmon_cleanup_module(void)
162  {
163
164    /* unregister the device */
165    unregister_chrdev(major, "pmon");
166    remove_proc_entry("l2cr", NULL);
167    printk("removed pmon module\n");
168    return;
169  }
170
171  module_init (pmon_init_module);
172  module_exit (pmon_cleanup_module);
root@debian:~/ppctools/pmon#
```

The performance monitor SPRs are described in Chapter 11 of the *MPC7450 RISC Microprocessor Family User's Manual* (MPC7450UM/D)

Kernel module relevant line numbers will be described here, other standard c lines will not be described.

The functions described here are used by the pmon_test.c code, which is described in the application note *Software Analysis on Genesi Pegasos II Using PMON and AltiVec* (AN2743).

NOTE

> All these functions have file arguments because that is the standard argument list for IO functions, however, the arguments are ignored, because PMON is only using these IO interfaces as a methodology for the user to call the functions. There is no IO to be performed, rather, these functions just read and write the SPRs associated with performance monitor registers.

8 This comment explains how to make the /dev/pmon device, which is used by the PMON interface.

18 though 31 with the exception of 21 are include files. These include files are not extracted from the application include files in /usr/include. They are extracted from the kernel source at /usr/src/kernel-source-2.6.4 because the make command uses the -C flag to redirect us to the kernel sources. these include files are then relative to /usr/src/kernel-source-2.6.4/include

33 #include "pmon.h" includes the header file pmon.h. However, it is not used, and in fact, if this line is commented out, it still builds and runs correctly. Hence, the listing of this header file is not included here.

35 through 48 are standard prototypes for functions that will be described later in this code.

50 is a variable to extract the /dev/pmon major number value, which will always be 254 as long as we use the `mknod` described in line 8.

52 through 60 is the structure that contains the pointers to the functions that are called when this modules IO functions are called from a user program.

61 through 66 is a required function, since a user can request an open on any device in the /dev directory. It just returns a zero.

67 through 73 is a required function, since a user can request a close on any device in the /dev directory. It just returns a zero.

74 through 82 is the function that reads the performance monitor SPR937, UPMC1 counter. These counters can be set to count a variety of CPU activities, however, PMON expects to find the count of cycles, since it sets up the MMCR0 register, MMCR0[PMC1SEL] to do so, as we will see later in the code.

74 is the call for pmon_read, note that we ignore the file and size_t arguments, only using the second argument char* buf.

78 This line writes the number of cycles to the memory buffer from the caller.

81 through 141 is the function, pmon_write, which sets the appropriate performance monitor registers to count what we have selected, which is cycles and instructions.

95 is the call for pmon_write, note that we ignore the file and size_t argument, we only use the const char* buf argument, which is the array of pmc_selections.

98 Sets pmcSel array to the second argument, const char* buf, which is the group of pmc selections sent to this function.

100 and 101 set the appropriate bits in the MMCR0, SPR952 and MMCR1, SPR956 to count the selected CPU activities.

103 through 106 clears the counter registers, PMC1, SPR953, PMC2, SPR954, PMC3, SPR957, and PMC4, SPR958. These registers are set by the asm construct, which will insert the mtspr instruction with the appropriate arguments, in this case mtspr 953,0, etc. The %0 is replaced by the “r” (value) construct.

107 is commented out, so we always assume 6 performance counter registers.

109 and 110 is used to set the extra two counters designated by MMCR1 by setting the appropriate bits.

111 and 112 set these two counters PMC5, SPR945 and PMC6, SPR946 to zero. In essence, setting the counter to zero before we begin counting again.

115 and 116 are commented out, so user can’t control start and stop, we just set MMCR0, SPR952 and MMCR1, SPR956. We set up these bit patterns in 100, 101, 109, and 110.

124 through 130 is the function to allow the user to set start and stop or any other parameters via the ioctl call, however, this version does nothing it just returns 0.

131 through 140 reads the L2CR L2 cache control register, SPR1017 value in line 136. The value is stored in the variable upmc1, which is a misleading name, but we are just going to print it out, so the name is irrelevant.

142 through 159 is the initialization code for the module, which is called by the `insmod pmon26.ko` command. Actually, this command will call the module_init function at line 171, which will call this function, pmon_init_module.

147 registers the character device, /dev/pmon with the register_chrdev function. There are three arguments to this function, the first one is 0, the second one is the device, “pmon”, which is /dev/pmon, and the last argument &pmon_fops, created in lines52 through 60, define the local function names for the POSIX System function call names.

151 through 158 is an example of reading the L2CR, SPR 1017, register. During the initialization of PMON, in line 154, the L2CR register is read and printed along with the major number for the /dev/pmon device, 254, to the kernel log, see Section 3.2, “Instantiating the Module.”

160 through 169 is called by the `rmmod pmon26.ko` command. Actually, the command will call module_exit, which will then call this function, pmon_cleanup_module. In line 167, the clean up message is printed to the kernel log.

165 unregisters the /dev/pmon device and removes the L2CR proc entry.

5 Instantiating any Module

Generally, any time a user needs to use a privileged facility, they must have kernel support. This support is always invoked through some type of call, such as a file manipulation, IO, calls. These privileged facilities can be either compiled into the kernel itself, a static operation that increases the size of the kernel permanently, or invoked via a module. Modules are linked in as part of the kernel dynamically, i.e. when needed. Thus the kernel can be kept small and only enlarged for those facilities that are needed.

The PMON facility, used here as an example, is used to access the performance registers, which are privileged registers. Only the Linux root user, or kernel can change them. However, a user can read the resultant performance counts, since they are not privileged. Therefore, it is necessary for a normal user to call a kernel support function to set these registers. Linux does not supply such a facility, however, the PMON facility included in the Pegasos II system does contain such a facility, which was written by the Freescale CPD applications team. This facility, called PMON, is supplied as a kernel module in the root directory at /root/ppctools/pmon. Using PMON as an example, we can describe this interaction.

Figure 1 shows that the User program, which runs as a normal user, interfaces to the kernel via a call to the PMON module, which can in turn perform root activities for the user program.

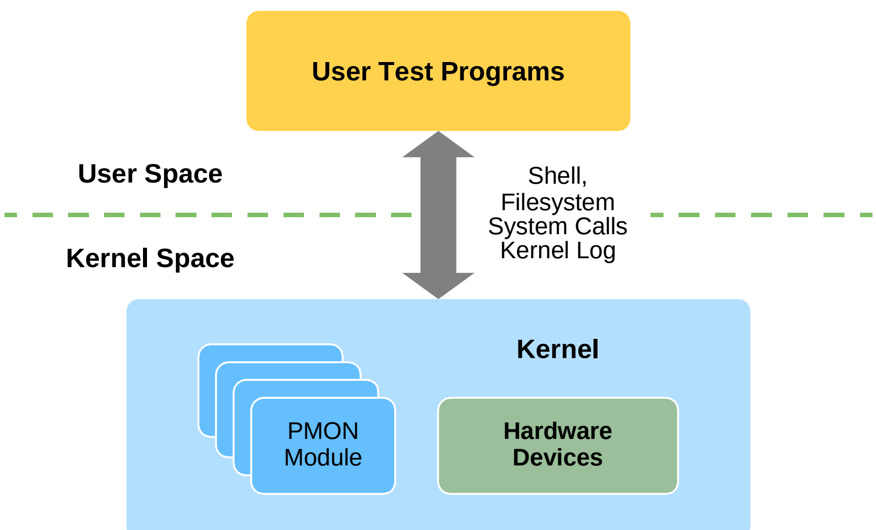

Figure 1. User Interaction with Kernel Module PMON

Figure 2 shows that a user cannot instantiate a kernel module. Hence the module must be instantiated at boot time or by a root user with the `insmod` command. The `insmod` command, which means, instantiate module, will install the module in the kernel, then call the init_module, the module instantiation function of PMON, which will initialize itself and wait for user calls to the PMON module. The `rmmod` command will de-instantiate it, that is, call the module_exit, cleanup_module to clean up any memory or other resources it is using and then remove it from the kernel module list

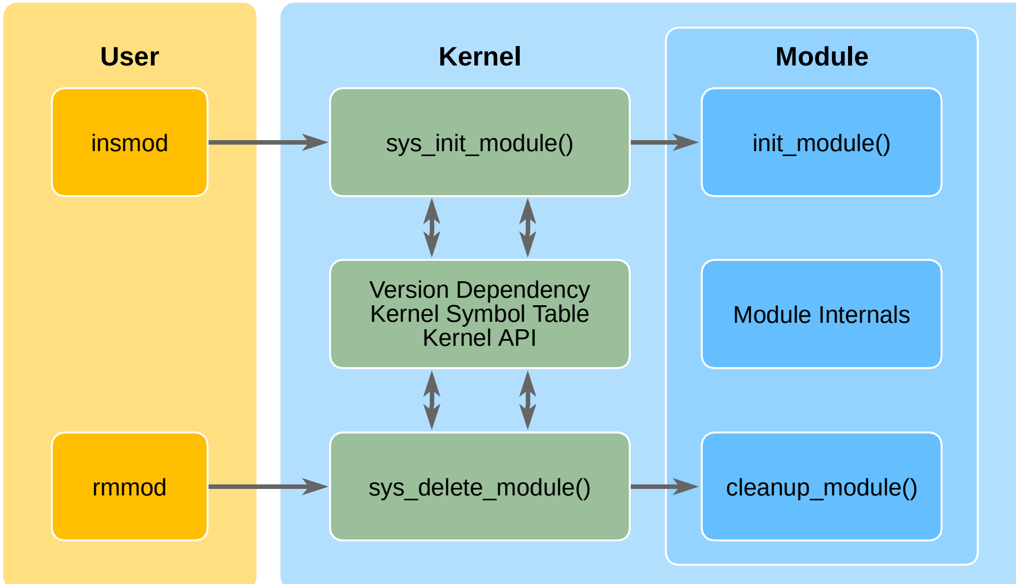

Figure 2. Instantiating the PMON Module

Figure 3 shows that once running, normal users can interface to the module with the standard POSIX API, using file commands, like `open` and `read` and `write`. This is because PMON instantiates itself as a char device and will get a device entry in /dev of /dev/pmon.

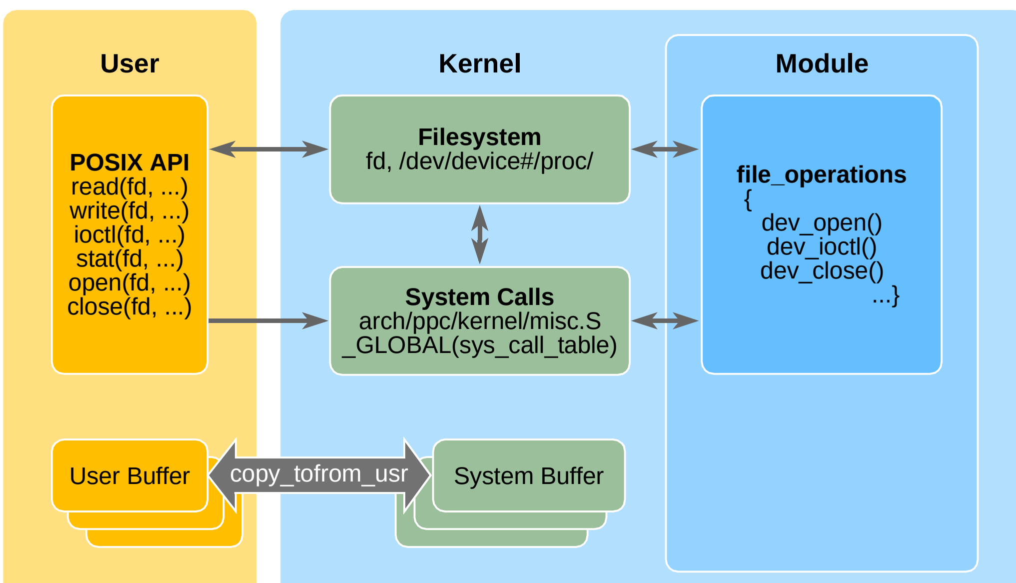

Figure 3. User Interaction to the module

In summary, a root user must start PMON with the `insmod pmon26.ko` command. A shell script is available for this action in /root/ppcttools/pmon/install.sh If the system is rebooted, then PMON must be reinstalled.

6 References

The following documents published by Freescale Semiconductor, Inc. describe the various applications of the Genesi Pegasos II system or are references for the systems:

1. Application note AN2666, *Genesi Pegasos II Setup*
2. Application note AN2736, *Genesi Pegasos II Boot Options*
3. Application note AN2738, *Genesi Pegasos II Firmware*
4. Application note AN2739, *Genesi Pegasos II Debian Linux*
5. Application note AN2751, *Genesi Pegasos II Yellow Dog Linux 3*
6. Application note AN2743, *Software Analysis on Genesi Pegasos II Using PMON and AltiVec*
7. Application note AN2748, *Genesi Pegasos II Kernel and NFS facility*
8. Application note AN2749, *Genesi Pegasos II Using sim_G4plus*
9. Application note AN2750, *Genesi Pegasos II Analysis and Optimization of Code with sim_G4plus*
10. Application note AN2801, *Upgrade or Restore Firmware and Hard Drive on Genesi Pegasos II*
11. Application note AN2802, *Genesi Pegasos II Yellow Dog Linux 4*
12. *AltiVec Technology Programming Environments Manual* (ALTIVECPEM/D Rev. 1)
13. *AltiVec Programming Interface Manual* (ALTIVECPIM/D Rev. 1)
14. *RISC Microprocessor Family User's Manual* (MPC7450)

For assistance or answers to any question on the information that is presented in this document, send an e-mail to risc10@freescale.com.

7 Document Revision History

Table 1 provides a revision history for this application note.

Table 1. Document Revision History

Revision Number	Date	Change(s)
1	12/22/2004	Made minor edits and added AN2801 and AN2802 to Section 6, "References."
0	07/14/2004	Initial release

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

email:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-0047 Japan
0120 191014
+81 3 3440 3569
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
(800) 441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN2744
Rev. 1
12/2004