Netra™ CP3010 Board Programming Guide

For the Netra CT 900 Server

Sun Microsystems, Inc.
www.sun.com

Part No. 819-1185-10
January 2006, Revision A

Submit comments about this document at: http://www.sun.com/hwdocs/feedback

Contents

Tables

Code Examples

Preface

The Netra™ CP3010 board is a crucial building block that network equipment providers and carriers can use when scaling and improving the availability of next-generation, carrier-grade systems.

This *Netra CP3010 Board Programming Guide* is written for program developers and users who want to program this board to design original equipment manufacturer (OEM) systems, supply additional capability to an existing compatible system, or work in a laboratory environment for experimental purposes.

Note – You are required to have a basic knowledge of computers and digital logic programming to fully use the information in this document.

How This Book Is Organized

Chapter 1 describes environmental monitoring of the Netra CP3010 board.

Chapter 2 describes the user flash driver device for the Netra CP3010 onboard flash PROMs and how to use the device.

Using UNIX Commands

This document might not contain information about basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices. Refer to the following for this information:

- Software documentation that you received with your system
- Solaris™ Operating System documentation, which is at:

  `http://docs.sun.com`

Shell Prompts

Shell	Prompt
C shell	*machine-name*%
C shell superuser	*machine-name*#
Bourne shell and Korn shell	$
Bourne shell and Korn shell superuser	#

Typographic Conventions

Typeface*	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`
`AaBbCc123`	What you type, when contrasted with on-screen computer output	`% su` `Password:`
AaBbCc123	Book titles, new words or terms, words to be emphasized. Replace command-line variables with real names or values.	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* be superuser to do this. To delete a file, type `rm` *filename*.

* The settings on your browser might differ from these settings.

Related Documentation

The documents listed as online are available at:

http://www.sun.com/products-n-solutions/hardware/docs/

Title	Part Number
Netra CP3010 Board Product Notes	819-1181-*xx*
Netra CP3010 Board Getting Started Guide	819-1182-*xx*
Netra CP3010 Board User's Guide	819-1183-*xx*
Netra CP3010 Board Software Installation Guide	819-1184-*xx*
Netra CP3010 Board Transition Card Getting Started Guide	819-1186-*xx*
Netra CP3010 Board Transition Card User's Guide	819-1187-*xx*
Important Safety Information for Sun Hardware Systems	817-7190-10

Third-Party Web Sites

Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Documentation, Support, and Training

Sun Function	URL	Description
Documentation	http://www.sun.com/documentation/	Download PDF and HTML documents, and order printed documents
Support and Training	http://www.sun.com/supportraining/	Obtain technical support, download patches, and learn about Sun courses

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can submit your comments by going to:

http://www.sun.com/hwdocs/feedback

Please include the title and part number of your document with your feedback:

Netra CP3010 Board Programming Guide, part number 819-1185-10

CHAPTER 1

Environmental Monitoring

The Netra CP3010 board uses an intelligent fault detection environmental monitoring system that increases uptime and manageability of the board. Through an Intelligent Platform Management Controller (IPMC) and system management software, the Netra CP3010 board monitors voltage, temperature, and power on sensors.

This chapter contains the following topics:

Power Requirements

The onboard voltage controller allows power to the CPU of the Netra CP3010 board only when the following conditions are met:

- VDD core-1.7-volt supply voltage is greater than 1.53 volts (within 10 percent of nominal)
- 12-volt supply voltage is greater than 10.8 volts (within 10 percent of nominal)
- 5-volt supply voltage is greater than 4.5 volts (within 10 percent of nominal)
- 3.3-volt supply voltage is greater than 3.0 volts (within 10 percent of nominal)

The controller requires these conditions to be true for at least 100 milliseconds to help ensure the supply voltages are stable. If any of these conditions become untrue, the voltage monitoring circuit shuts down the CPU power of the board.

Inlet, Exhaust, and CPU Temperatures

The following table summarizes the functions on the Netra CP3010 board that monitor the hardware environment.

TABLE 1-1 Hardware Environmental Monitoring Functions

Function	Description
Board exhaust air temperature	Senses the air temperature at the trailing edge of the board. (Assumes air direction from the processor/heatsink toward the PCI mezzanine card (PMC) slots.)
CPU diode temperature	Senses a diode temperature in the processor junction.
Board inlet air temperature	Senses the air temperature at the leading edge of the board under the solder-side cover. (Assumes air direction from the processor/heatsink toward the PMC slots.)

The CPU diode sensor reading might vary from slot to slot and from board to board in a system, and is dependent primarily on system cooling. As an example, a system might have sensor readings for the CPU diode from 35˚C to 49˚C with an ambient inlet of 21˚C across many boards, with a variety of configurations and positions within a chassis. You must take care when setting the alarm and shutdown temperatures based on the CPU diode sensor value. This sensor typically is linear across the operating range of the board.

The exhaust sensor measures the local air temperature at the trailing edge of the board for systems with bottom-to-top airflow. This value depends on the character and volume of the airflow across the board. Typical values in a chassis might range from a delta over inlet ambient of 0˚C to 12˚C, depending on the power dissipation of the board configuration and the position in the chassis. The exhaust sensor is nonlinear with respect to ambient inlet temperature.

The inlet sensor measures the local air temperature at the leading edge of the board under the solder-side cover. This value typically can range from a reading of 0˚C to 13˚C above inlet system ambient in a chassis. You must understand the application and installation of the board to use this temperature sensor.

A sudden drop of all temperature sensors close to or near room ambient temperature can mean loss of power to one or more Netra CP3010 boards.

A gradual increase in the delta temperature from inlet to outlet can be due to dust clogging system filters.

The CPU diode temperature can be used to prevent damage to the board by shutting the board down if this sensor exceeds predetermined limits.

CHAPTER 2

Flash Device Driver

The Netra CP3010 board is equipped with flash memory. This chapter introduces the flash device driver for the onboard flash PROM and describes how to use the device.

This chapter contains the following topics:

Software Requirements

The following software releases support the flash driver:

- Solaris 9 (9/04) Operating System (Solaris OS) and newer releases
- Required OS patches
- Netra CP3010 board OpenBoot PROM

Storing Data and Applications

The Solaris OS *uflash* is the device driver for the flash device on the Netra CP3010 board. On the Netra CP3010 board, one driver is supported. Users can use this device driver for storing data and applications.

Multiple reads and writes can be submitted concurrently, however, they are serialized by the `uflash` device. For example, the driver blocks additional reads and writes to the device while a read or write is in progress.

The driver supports erase and lock features. Applications can use them through the IOCTL interface. The device is divided into logical blocks. Applications that issue these operations supply a block number or a range of blocks that are a target of these operations. Locks are preserved across reboots. Locking a block prevents an erase or write operation on that block.

Switch Settings

The flash modules on the Netra CP3010 board are write-enabled by default. The flash device is detected during OpenBoot™ PROM boot.

OpenBoot PROM Device Tree and Properties

The following information describes the OpenBoot PROM device node and its properties.

The OpenBoot PROM device node is as follows:

```
/pci@1e,600000/isa@7/flashprom@2,0
```

TABLE 2-1 lists the flash node properties.

TABLE 2-1 Flash Node Properties

Property	Description/Value
sunw,location	U55
model	SUNW,254-0078
boot-banks	00 00 00 00 00 00 00 02 00 00 00 04 00 00 00 06
system-banks	00 00 00 00 00 00 00 01 00 00 00 02 00 00 00 03
flash-banks	00 00 00 00 00 00 00 1f
write-window	00 08 00 00 00 08 00 00
boot-window	00 00 00 00 00 08 00 00
bank-size	00080000
version	OBP 4.21.0 2005/11/22 16:48 Netra CP3010 OBDIAG 4.x.0 2005/11/22 17:00 POST 4.50.48 2005/11/22 17:12
name	flashprom
compatible	isa-flashprom
reg	00000002 00000000 00100000 00000000 00000700 00000002

Flash Device Files

The flash device files are located in the following path:

```
/dev/uflash0
```

Interface (Header) File

The flash header file is located in the following path:

`/usr/`*platform*`/SUNW,Netra-CP3010/include/sys/uflash_if.h`

Flash Memory

The Netra CP3010 board has a 16-megabyte flash chip that is logically divided into two partitions:

- 2-megabyte system flash for storing copies of the OpenBoot PROM image. Users do not have access to this partition.
- 14-megabyte user flash for storing any user data and performing operations such as read and write. The user flash includes a flash PROM chip that can be programmed. The physical address of the user flash is 1ff.f000.0000.

Accessing the Flash Device

You can access the flash device from the Solaris OS through an application or user C program. No command-line tool is available. Access to the driver is carried out through `open`, `read`, `write`, `pread`, `pwrite` and `ioctl` system interfaces. User programs open the device file, then issue `read`, `write`, or `ioctl` commands.

System calls are listed in the following table.

TABLE 2-2 System Calls

Call	Description
`read(), pread()`	Reads device
`write(), pwrite()`	Writes device
`ioctl()`	Erases device, queries device parameters

The following `ioctl` commands are supported:

```
#define UIOCIBLK (uflashIOC|0)      /* identify */
#define UIOCQBLK (uflashIOC|1)      /* query a block */
#define UIOCLBLK (uflashIOC|2)      /* lock a block */
#define UIOCCLCK (uflashIOC|4)      /* clear all locks */
#define UIOCEBLK (uflashIOC|5)      /* erase a block */
```

The following `ioctl` commands are *not* supported:

```
#define UIOCMLCK (uflashIOC|3)      /* master lock */
#define UIOCEALL (uflashIOC|6)     /* erase all unlocked blocks */
```

Using Structures in IOCTL Arguments

The following sections describe the PROM information structure and the flash user interface structure.

PROM Information Structure

The PROM information structure holds device information returned by the driver in response to an identify command.

CODE EXAMPLE 2-1 PROM Information Structure

```
/*
 * PROM info structure.
 */
typedef struct {
        uint16_t        mfr_id;            /* manufacturer id */
        uint16_t        dev_id;            /* device id */
        /* allow future expansion */
        int8_t          blk_status[256];   /* blks status filled
by driver */
       int32_t         blk_num;           /* total # of blocks */
      int32_t        blk_size;         /* # of bytes per block */
} uflash_info_t;
```

Flash User Interface Structure

The flash user interface structure holds user parameters to commands such as `erase`.

CODE EXAMPLE 2-2 Flash User Interface Structure

```
/*
 * uflash user interface structure.
 */
typedef struct {
        int             blk_num;
        int             num_of_blks;
       uflash_info_t   info;             /* to be filled by the
driver */
} uflash_if_t;
```

Resolving Structure Errors

The following table lists the error messages reported when structure errors occur.

TABLE 2-3 Structure Errors

Error Message	Description
`EINVAL`	An application passed one or more incorrect arguments to the system call.
`EACCESS`	A `Write` or `Erase` operation was attempted on a locked block.
`ECANCELLED`	A hardware malfunction has been detected. Normally, retrying the command should fix this problem. If the problem persists, power cycling the system might be necessary.
`ENXIO`	This error indicates problems with the driver state. Power cycle of the system or reinstallation of driver might be necessary.
`EFAULT`	An error was encountered when copying arguments between the application and driver (kernel) space.
`ENOMEM`	The system was low on memory when the driver attempted to acquire it.

Developing Programs

You can use the programs that follow as examples when you develop programs for the following actions on the flash device:

- Read
- Write
- Erase
- Block Erase

Example Read Program

CODE EXAMPLE 2-3 contains the Read Action on the flash device.

CODE EXAMPLE 2-3 Read Action on Flash Device

```
/*
* uflash_read.c
 * An example that shows how to read flash
 */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <uflash_if.h>
char *uflash0 = "/dev/uflash0";
int ufd0;
uflash_if_t ufif0;
char *buf0;
char *module;
static int
uflash_init() {
  char *buf0 = malloc(ufif0.info.blk_size);
if (!buf0) {
    printf("%s: cannot allocate memory\n", module);
    return(-1);
  }
/* open device */
  if ((ufd0 = open(uflash0, O_RDWR)) == -1 ) {
    perror("uflash0: ");
    exit(1);
```

CODE EXAMPLE 2-3 Read Action on Flash Device *(Continued)*

```
  }
/* get uflash sizes */
  if (ioctl(ufd0, UIOCIBLK, &ufif0) == -1 ) {
    perror("ioctl(ufd0, UIOCIBLK): ");
    exit(1);
  }
if (ufd0) {
    printf("%s: \n", uflash0);
    printf("manfacturer id = 0x%p\n", ufif0.info.mfr_id);
    printf("device id = 0x%p\n", ufif0.info.dev_id);
    printf("number of blocks = 0x%p", ufif0.info.blk_num);
    printf("block size = 0x%p"  ufif0.info.blk_size);
  }
static int
uflash_uninit() {
  if (ufd0)
    close(ufd0);
cleanup:
  if (buf0)
    free(buf0);
}
static int
uflash_read() {
  /* read block 0 of user flash */
  if (pread(ufd0, buf0, ufif0.info.blk_size, 0) != ufif0.info.blk_size)
        perror("uflash0:read");
return(0);
}
main() {
  int ret;
module = argv[0];
ret = uflash_init();
if (!ret)
    uflash_read();
uflash_uninit();
}
```

Example Write Program

CODE EXAMPLE 2-4 contains the Write Action on the flash device.

CODE EXAMPLE 2-4 Write Action on Flash Device

```
/*
 * uflash_write.c
 * An example that shows how to write flash
 */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <uflash_if.h>
char *uflash0 = "/dev/uflash0";
int ufd0;
uflash_if_t ufif0;
char *buf0;
char *module;
static int
uflash_init() {
  char *buf0 = malloc(ufif0.info.blk_size);
if (!buf0) {
    printf("%s: cannot allocate memory\n", module);
    return(-1);
  }
/* open device */
  if ((ufd0 = open(uflash0, O_RDWR)) == -1 ) {
    perror("uflash0: ");
    exit(1);
  }
/* get uflash sizes */
  if (ioctl(ufd0, UIOCIBLK, &ufif0) == -1 ) {
    perror("ioctl(ufd0, UIOCIBLK): ");
    exit(1);
  }
if (ufd0) {
    printf("%s: \n", uflash0);
    printf("manfacturer id = 0x%p\n", ufif0.info.mfr_id);
    printf("device id = 0x%p\n", ufif0.info.dev_id);
    printf("number of blocks = 0x%p", ufif0.info.blk_num);
    printf("block size = 0x%p"  ufif0.info.blk_size);
  }
}
```

CODE EXAMPLE 2-4 Write Action on Flash Device *(Continued)*

```
static int
uflash_uninit() {
  if (ufd0)
    close(ufd0);
cleanup:
  if (buf0)
    free(buf0);
}
static int
uflash_write() {
  int i;
/* write some pattern to the buffers */
  for (i = 0; i < ufif0.info.blk_size; i += sizeof(int))
        *((int *) (buf0 + i)) = 0xDEADBEEF;
/* write block 0 of user flash */
  if (pwrite(ufd0, buf0, ufif0.info.blk_size, 0) != ufif0.info.blk_size)
       perror("uflash0:write");
return(0);
}
main() {
  int ret;
module = argv[0];
ret = uflash_init();
if (!ret)
    uflash_write();
uflash_uninit();
}
```

Example Block Erase Program

CODE EXAMPLE 2-5 contains the Block Erase Action on the flash device.

CODE EXAMPLE 2-5 Block Erase Action on Flash Device

```
/*
 * uflash_blockerase.c
 * An example that shows how to erase block(s) of flash
 */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <uflash_if.h>
char *uflash0 = "/dev/uflash0";
int ufd0;
uflash_if_t ufif0;
char *module;
static int
uflash_init() {
 /* open device */
  if ((ufd0 = open(uflash0, O_RDWR)) == -1 ) {
    perror("uflash0: ");
    exit(1);
  }
/* get uflash sizes */
  if (ioctl(ufd0, UIOCIBLK, &ufif0) == -1 ) {
    perror("ioctl(ufd0, UIOCIBLK): ");
    exit(1);
  }
if (ufd0) {
    printf("%s: \n", uflash0);
    printf("manfacturer id = 0x%p\n", ufif0.info.mfr_id);
    printf("device id = 0x%p\n", ufif0.info.dev_id);
    printf("number of blocks = 0x%p", ufif0.info.blk_num);
    printf("block size = 0x%p"  ufif0.info.blk_size);
  }
}
static int
uflash_uninit() {
  if (ufd0)
    close(ufd0);
}
static int
```

CODE EXAMPLE 2-5 Block Erase Action on Flash Device *(Continued)*

```
uflash_blockerase() {
  /* erase 2 blocks starting from block 1 of user flash */
  uf0.blk_num = 1;
  uf0.num_of_blks = 2;
  if (ufd0 && ioctl(ufd0, UIOCEBLK, &ufif0) == -1 ) {
    perror("ioctl(ufd0, UIOCEBLK): ");
    return(-1);
  }
  printf("\nblockerase successful on %s\n", uflash0);
return(0);
}
main() {
  int ret;
module = argv[0];
ret = uflash_init();
if (!ret)
    uflash_blockerase();
uflash_uninit();
}
```

Example Flash Application Program

You can use the following program to test the flash device driver. This program demonstrates how the device can be used.

CODE EXAMPLE 2-6 Flash Application Program

```
/*
 *
 *        This application program demonstrates the user program
 *        interface to the Flash PROM driver.
 *
 *        One can read or write a number of bytes up to the size of
 *        the user PROM by means of pread() and pwrite() calls.
 *        All other functions of the PROM can be accessed by
 *        means of ioctl() calls such as:
 *         -) identify the chip,
 *         -) query block,
 *         -) lock block/unlock block,
 *         -) erase block
 *        Please note that not all of the above ioctl calls are
 *        available for all flash PROMs. It is the user's
 *        responsibility to find out the features of a given PROM.
 *        The type, block size, and number of blocks of the PROM
 *        are returned by "identify" ioctl().
 *
 *         The pwrite() erases the block[s] and then does the
 *         writing.
 *
 *        Use the following line to compile your custom application
 *        programs:
 *          make uflash_test
 */

#pragma ident   "@(#)uflash_test.c 1.0     03/04/30 SMI"

#include <stdio.h>
#include <sys/signal.h>
#include <stdio.h>
#include <sys/time.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/fcntl.h>
#include <sys/stream.h>
#include "uflash_if.h"
/*
      */
```

CODE EXAMPLE 2-6 Flash Application Program *(Continued)*

```
      #if 1
      #define PROM_SIZE 0x700000 /* 7 MBytes */
      #endif
static char *help[14] = {
       "0 -- read     user flash PROM",
       "1 -- write    user flash PROM",
       "2 -- identify user flash PROM",
       "3 -- query    blocks",
       "4 -- lock     blocks",
       "5 -- clear    all locks",
       "6 -- erase    blocks",
       "q -- quit",
       "?/h -- display this menu",
       ""
};

/*char            get_cmd(); */

static char
get_cmd()
{
           char    buf[10];
           gets(buf);
           return (buf[0]);
}

/*
 * Main
 */
main(int argc, char *argv[])
{
       int          n_byte;             /* returned from pread/pwrite */
       int          size, offset, pat;
       int          fd0, h, i;
       int          fd, prom_id;
       uflash_if_t  uflash_if;
       caddr_t      r_buf, w_buf;
       char         *devname0 = "/dev/uflash0";
       char         c;

       r_buf = (caddr_t)malloc(PROM_SIZE);
       w_buf = (caddr_t)malloc(PROM_SIZE);

       /*
        * Open the user flash PROM.
       */
```

CODE EXAMPLE 2-6 Flash Application Program *(Continued)*

```
        if ((fd0 = open(devname0, O_RDWR)) < 0) {
            fprintf(stderr, "couldn't open device: %s\n", devname0);
            exit(1);
        }

        /* set the default PROM */
        prom_id = 0;
        fd = fd0;

        /* let them know about the help menu */
         fprintf(stderr, "Enter <h> or <?> for help on commands\n");

        while (1) {
            fprintf(stderr, "[%d]command> ", prom_id);

            switch(get_cmd()) {
            case 'q':
               goto getout;

            case 'h':
            case '?':
               h = 0;
               while (*help[h]){
                   fprintf(stderr, "   %s\n", help[h]);
                   h++;
               }
               break;


            case '6':       /* erase flash PROM block */
               fprintf(stderr,
                                  "Enter PROM block number[0, 56]> ");
                         scanf ("%d", &uflash_if.blk_num);

               fprintf(stderr,
                   "Enter number of block> ");
               scanf ("%d", &uflash_if.num_of_blks);

               if (ioctl(fd, UIOCEBLK, &uflash_if) == -1)
                                  goto getout;
               break;

            case '5':       /* clear all locks */
               if (ioctl(fd, UIOCCLCK, &uflash_if) == -1)
                                  goto getout;
               break;
```

CODE EXAMPLE 2-6 Flash Application Program *(Continued)*

```
            case '4':       /* lock flash PROM block */
               /* on certain PROMs */
               fprintf(stderr,
                                 "Enter PROM block number[0, 56]> ");
                         scanf ("%d", &uflash_if.blk_num);

               fprintf(stderr,
                   "Enter number of block> ");
               scanf ("%d", &uflash_if.num_of_blks);

               if (ioctl(fd, UIOCLBLK, &uflash_if) == -1)
                                 goto getout;
               break;

            case '3':       /* query flash PROM */
               /* on certain PROMs */
               fprintf(stderr,
                                 "Enter PROM block number[0, 56]> ");
                         scanf ("%d", &uflash_if.blk_num);

               fprintf(stderr,
                   "Enter number of block> ");
               scanf ("%d", &uflash_if.num_of_blks);

               if (ioctl(fd, UIOCQBLK, &uflash_if) == -1)
                                 goto getout;
               for (i = uflash_if.blk_num;
                   i < (uflash_if.blk_num+uflash_if.num_of_blks);
                   i++)
               {
                   fprintf(stderr, "block[%d] status = %x\n",
                      i, uflash_if.info.blk_status[i] & 0x1);
               }
               break;

            case '2':       /* identify flash PROM */
               if (ioctl(fd, UIOCIBLK, &uflash_if) == -1)
                                 goto getout;
               fprintf(stderr, "manufacturer id = 0x%x, device id =\
                           0x%x\n# of blks = %d, blk size = 0x%x\n",
                        uflash_if.info.mfr_id & 0xFF,
                        uflash_if.info.dev_id & 0xFF,
                        uflash_if.info.blk_num,
                        uflash_if.info.blk_size);
               break;
```

CODE EXAMPLE 2-6 Flash Application Program *(Continued)*

```
            case '1':       /* write to user flash PROM */
               fprintf(stderr,
                                "Enter PROM offset[0, 0xXX,XXXX]> ");
                         scanf ("%x", &offset);

               fprintf(stderr,
                   "Enter number of bytes[hex]> ");
               scanf ("%x", &size);

               fprintf(stderr,
                                "Enter data pattern[0, 0xFF]> ");

                         scanf ("%x", &pat);

               /*
                * init write buffer.
                */
               for (i = 0; i < size; i++) {
                   w_buf[i] = pat;
               }

               n_byte = pwrite (fd, w_buf, size, offset);
               if (n_byte != size) {
                   /* the write failed */
                   printf ("Write process was failed at byte 0x%x \n",
                       n_byte);
               }
               break;

            case '0':   /* read from user flash PROM */
               fprintf(stderr,
                                "Enter PROM offset[0, 0xXX,XXXX]> ");
                         scanf ("%x", &offset);

               fprintf(stderr,
                   "Enter number of bytes[hex]> ");
               scanf ("%x", &size);

               getchar();   /* clean up the char buf */

               n_byte = pread (fd, r_buf, size, offset);
               if (n_byte != size) {
                                /* the read failed */
                                printf ("Read process was failed at \
                              byte 0x%x \n",
                                          n_byte);
                   continue;
```

CODE EXAMPLE 2-6 Flash Application Program *(Continued)*

```
                        }

                printf ("\nuser data buffer:\n");
                for (i = 0; i < size; i++) {
                    printf("%2x ", r_buf[i] & 0xff);
                }
            printf("\n");

            default:
                continue;
            }
        }

        /* exit */
getout:
        close(fd0);
        return;

} /* end of main() */
```

Index