Netra™ CP2300 cPSB Board Programming Guide

for Solaris Operating Environment

Sun Microsystems, Inc.
www.sun.com

Part No. 817-1331-12
March 2004, Revision A

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

---

Adobe PostScript

Contents

Figures

Tables

Code Samples

Preface

The Netra™ CP2300 CompactPCI Packet Switched Backplane (cPSB) board is a crucial building block that network equipment providers (NEPs) and carriers can use when scaling and improving the availability of next-generation, carrier-grade systems.

The *Netra CP2300 cPSB Board Programming Guide* is written for program developers and users who want to program this board in order 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, in order to fully use the information in this document.

How This Book Is Organized

Chapter 1 provides details on the Netra CP2300 watchdog timer driver and its operation.

Chapter 2 describes the specific environmental monitoring functions of the Netra CP2300.

Chapter 3 describes the user flash driver for the Netra CP2300 onboard flash PROMs and how to use it.

Chapter 4 describes how to program the User LED on the Netra CP2300.

Using UNIX Commands

This document may not contain information on basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices.

See one or more of the following for this information:

- Solaris Handbook for Sun Peripherals
- AnswerBook2™ online documentation for the Solaris™ Operating System
- Other software documentation that you received with your system

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.

Shell Prompts

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

Related Documentation

Title	Part Number
Netra CP2300 cPSB Board Product Note	816-7185-*xx*
Netra CP2300 cPSB Board Installation and Technical Reference	816-7186-*xx*
Netra CP2300 cPSB Board Programming Guide	817-1331-*xx*
Netra CP2300 cPSB Board Transition Card Product Note	816-7187-*xx*
Netra CP2300 cPSB Board Transition Card Installation and Technical Reference	816-7188-*xx*
Netra CP2300 cPSB Board Release Notes	817-1741-*xx*
Important Safety Information for Sun Hardware Systems	816-7190-*xx*

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.

Accessing Sun Documentation

You can view, print, or purchase a broad selection of Sun documentation, including localized versions, at:

`http://www.sun.com/documentation`

Contacting Sun Technical Support

If you have technical questions about this product that are not answered in this document, go to:

`http://www.sun.com/service/contacting`

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 CP2300 cPSB Board Programming Guide, part number 817-1331-12

CHAPTER 1

Watchdog Timer

The Netra CT system's watchdog service captures catastrophic faults in the Solaris operating environment (OE) running on the node card. The watchdog service reports such faults to the baseboard management controller (BMC) by means of either an IPMI message or by a de-assertion of the CPU's HEALTHY# signal.

This chapter contains the following sections:

Watchdog Timers

The Netra CT system management controller provides two watchdog timers: the watchdog level 2 (WD2) timer and the watchdog level 1 (WD1) timer. Systems management software starts and the Solaris OE periodically pats the timers before they expire. If the WD2 timer expires, the watchdog function of the WD2 timer forces the SPARC™ processor to optionally reset. The maximum range for WD2 is 255 seconds.

The WD1 timer is typically set to a shorter interval than the WD2 timer. User applications can examine the expiration status of the WD1 timer to get advance warning if the main timer, WD2, is about to expire. The system management software has to start WD1 before it can start WD2. If WD1 expires, then WD2 starts only if enabled. The maximum range for WD1 is 6553.5 seconds.

PICL Plug-in Module

The watchdog subsystem is managed by a platform information and control library (PICL) plug-in module. This PICL plug-in module provides a set of PICL properties to the system, which enables a Solaris PICL client to specify the attributes of the watchdog system.

To use the PICL API to set the watchdog properties, your application must follow the following sequence:

Note – The following instructions are not server-specific. Check your server documentation for additional software configuration that may be needed with the watchdog timer.

---

1. Before setting the watchdog to disable the primary HEALTHY# signal monitoring for the node card on which the watchdog timer is to be changed.
2. In your application, use the PICL API to disarm, set, and arm the active watchdog timer.

   Refer to the `picld`(1M), `libpicl`(3LIB), and `libpicl`(3PICL) man pages for a complete description of the PICL architecture and programming interface. Develop your application to use the PICL programming interface to do the following:

   - Disarm the active watchdog timer.
   - Change the watchdog timer PICL properties to the required values.
   - Re-arm the watchdog timer. The properties of watchdog-controller and watchdog-timer are defined in TABLE 1-1, TABLE 1-2, and TABLE 1-3.
3. Re-enable the primary HEALTHY# signal monitoring on the CPU card in the specified slot.

PICL interfaces for the watchdog plug-in module include the nodes watchdog-controller and watchdog-timer. See TABLE 1-1, TABLE 1-2 and TABLE 1-3 for descriptions of the properties of these nodes.

TABLE 1-1 Watchdog Plug-in Interfaces for Netra CP2300 Board Software

PICL Class	Property	Meaning
`watchdog-controller`	`WdOp`	Represents a watchdog subsystem.
`watchdog-timer`	`State`	Represents a watchdog timer hardware that belongs to its controller. Each timer depends on the status of its peers to be activated or deactivated.
	`WdTimeout`	Timeout for the watchdog timer
	`WdAction`	Action to be taken after the watchdog expires.

TABLE 1-2 Properties Under `watchdog-controller` Node

Property	Operations	Description
`WdOp`	`arm`	Activates all timers under the controller with values already set for `WdTimeout` and `WdAction`.
	`disarm`	All active timers under the controller will be stopped.

TABLE 1-3 Properties Under `watchdog-timer` Node

Property	Values	Description
`State`	`armed`	Indicates timer is armed or running. Cleared by `disarm`.
	`expired`	Indicates timer has expired. Cleared by `disarm`.
	`disarmed`	Default value set at startup time. Indicates timer is disarmed or stopped.
`WdTimeout`*	Varies by system and timer level	Indicates the timer initial countdown value. Should be set prior to arming the timer.
`WdAction`†	`none`	Default value. No action is taken.
	`alarm`	Send notifications to system alarm hardware by means of HEALTHY#.
	`reset`	Perform a soft or hard reset of the system (implementation specific).
	`reboot`	Reboot the system.

* A platform might not support a specified timeout resolution. For example, Netra CT systems only take -1, 0, and 100 to 6553500 msec in increments of 100 msec for level 1; and -1, 0, and 1000 to 255000 in increments of 1000 msec for level 2.

† A specific timer node might not support all action types. For example, Netra CT watchdog level 1 timer supports only `none`, `alarm`, and `reboot` actions. Watchdog level 2 timer supports only `none` and `reset`

To identify current settings of `watchdog-controller`, issue the command `prtpicl -v` as shown in CODE EXAMPLE 1-1.

CODE EXAMPLE 1-1 Example of `watchdog-controller` Settings

```
# prtpicl -v
         <snip>
        watchdog-controller1 (watchdog-controller,3600000729)
                :wd-op  disarm
                :_class watchdog-controller
                :name   watchdog-controller1
                   watchdog-level1 (watchdog-timer, 360000073f)
                        :WdAction     alarm
                        :WdTimeout    0x1f4
                        :State        armed
                        :_class       watchdog-timer
                        :name  watchdog-level1
                   watchdog-level2 (watchdog-timer, 3600000742)
                        :WdAction     none
                        :WdTimeout    0xffff
                        :State        disarmed
```

CODE EXAMPLE 1-1 Example of `watchdog-controller` Settings *(Continued)*

```
                              :_class       watchdog-timer
                              :name   watchdog-level2
```

Watchdog Node Management Code

CODE EXAMPLE 1-2 contains an example of the code used for managing the watchdog timer nodes. This code can be used to change watchdog timer action and timeout values and also to arm and disarm the watchdog controller.

CODE EXAMPLE 1-2 System Watchdog Node Management Code Example

```
/*
 * Copyright 2003 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "@(#)wdadm.c    1.6     03/10/16 SMI"

/*
 * This program is used to manage the system watchdog nodes.
 * Please refer to libpicl(3LIB) for information on picl APIs
 * To compile:
 *       cc -o wdadm -lpicl wdadm.c
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <strings.h>
#include <errno.h>
#include <alloca.h>
#include <libintl.h>
#include <locale.h>
#include <unistd.h>
#include <assert.h>
#include <inttypes.h>
#include <sys/termios.h>
#include <picl.h>

/*
 * Error codes
```

CODE EXAMPLE 1-2 System Watchdog Node Management Code Example *(Continued)*

```
 */
#define EM_USAGE                0
#define EM_INIT                 1
#define EM_GETROOT              2
#define EM_GETPVALBYNAME        3

#define USAGE_STR       "Usage:\n"\
                "wdadm -l [<controller_name:timer_name>...]\n"\
                "wdadm -m <controller_name:timer_name> [-t <timeout>]"\
                " [-a action]]\n"\
                "wdadm -c <controller_name> -o <op>\n"

#define DETAILED_HELP   "wdadm  - System Watchdog Controller Administration\n"\
"Description:\n"\
"The operations include displaying status (-l), modifying the values (-m)\n"\
"and executing commands on the watchdog controller (-c).\n"\
"This utility must be run with super user permissions.\n"\
"OPTIONS\n"\
"       -l   list all the watchdog timer nodes.\n"\
"            Each Timer node is denoted as controller:timer\n"\
"            Exmaple:\n"\
"            wdadm -l                  - lists all the nodes\n"\
"            wdadm -l c1:t1 c1:t2      - lists c:t1 and c:t2 nodes\n"\
"                     c1  - controller name\n"\
"                     t1  - timer name\n"\
"       -m   modify the timeout and action parameters for a timer node.\n"\
"            Example:\n"\
"            wdadm -m c1:t1 -t <timeout in ms> -a <action>\n"\
"            wdadm -m c1:t1 -t <timeout in ms>\n"\
"            wdadm -m c1:t1 -a <action>\n"\
"            Note: Before using this option, the controller must be\n"\
"                  disarmed (using -c option).\n"\
"       -c   Execute commands on the watchdog controller node\n"\
"            Commands supported are : arm, disarm\n"\
"            Example:\n"\
"            wdadm -c controller -o arm\n"\
"            arms the watchdog controller node called controller\n"

#define HEADER          "NAME (controller:timer)\t\tSTATUS"\
                        "\t\tACTION\t\tTIMEOUT\n"
#define PRINT_FORMAT    "\t%-10s\t%-10s\t%d"
#define ILLEGAL_TIMEOUT -999

/* watchdog properties */
#define WATCHDOG_ACTION                 "WdAction"
#define WATCHDOG_TIMEOUT                "WdTimeout"
#define WATCHDOG_STATUS                 "State"
```

CODE EXAMPLE 1-2 System Watchdog Node Management Code Example *(Continued)*

```
#define WATCHDOG_OP                         "WdOp"
#define PICL_WATCHDOG_CONTROLLER            "watchdog-controller"
#define WATCHDOG_DISARMED                   "disarmed"

/*
 * data structure that will be passed as argument to
 * picl_walk_tree_by_class callback function
 */
typedef struct {
        int start_index;
        int max_index;
        char    **list;
        char    *name;
        char    *action;
        char    *op;
        int32_t timeout;
        int     error_code;
} wdadm_args_t;

static  char            *prog;
static picl_nodehdl_t   rooth;
static          int count = 0;

/*
 * Error mesage texts
 */
static  char    *err_msg[] = {
        /* program usage */
        USAGE_STR,                                              /* 0 */
        /* picl call failed messages */
        "picl_initialize failed: %s\n",                         /*  1 */
        "picl_get_root failed: %s\n",                           /*  2 */
        "picl_get_propval_by_name failed: %s\n"                 /*  3 */
};

#define NUM_ERROR_CODES 7
/* mapping between picl error codes and errno */
static int error_map[][2] =  {
        {PICL_SUCCESS, 0}, { PICL_FAILURE, -1}, {PICL_VALUETOOBIG, E2BIG},
        {PICL_NODENOTFOUND, ENODEV}, {PICL_PERMDENIED, EPERM},
        {PICL_NOSPACE, ENOMEM}, {PICL_INVALIDARG, EINVAL} };

static int
picl2errno(int piclerr)
{
        int i;
        for (i = 0; i < NUM_ERROR_CODES; i++) {
```

CODE EXAMPLE 1-2 System Watchdog Node Management Code Example *(Continued)*

```
                if (error_map[i][0] == piclerr)
                        return (error_map[i][1]);
        }
        return (-1);
}

static void
print_errmsg(char *message, ...)
{
        va_list ap;

        va_start(ap, message);
        (void) fprintf(stderr, "%s: ", prog);
        (void) vfprintf(stderr, message, ap);
        va_end(ap);
}

/*
 * Print wdadm usage
 */
static void
usage(void)
{
        print_errmsg(gettext(err_msg[EM_USAGE]));
        exit(1);
}

/*
 * This function is used to read picl property. The value is copied
 * into vbuf.
 * memory allocated for vbuf must be free'd by caller
 */
static picl_errno_t
wdadm_get_picl_prop(picl_nodehdl_t nodeh, const char *prop_name, void **vbuf)
{
        picl_errno_t    err;
        picl_propinfo_t pinfo;
        picl_prophdl_t  proph;

        /* get the information about the property */
        if ((err = picl_get_propinfo_by_name(nodeh, prop_name,
                        &pinfo, &proph)) != PICL_SUCCESS) {
                return (err);
        }

        *vbuf = malloc(pinfo.size);
        if (vbuf == NULL)
```

CODE EXAMPLE 1-2 System Watchdog Node Management Code Example *(Continued)*

```
                return (PICL_NOSPACE);

        /* read the property value */
        if ((err = picl_get_propval(proph, *vbuf, pinfo.size)) !=
                        PICL_SUCCESS) {
                return (err);
        }
        return (PICL_SUCCESS);
}

/*
 * This function is used to set the value of a picl property
 */
static picl_errno_t
wdadm_set_picl_prop(picl_nodehdl_t nodeh, const char *prop_name,
                                void *vbuf, int size)
{
        picl_errno_t    err;
        picl_propinfo_t pinfo;
        picl_prophdl_t  proph;
        void            *tmp_buf;

        if ((err = picl_get_propinfo_by_name(nodeh, prop_name,
                        &pinfo, &proph)) != PICL_SUCCESS) {
                return (err);
        }

        tmp_buf = alloca(pinfo.size);
        if (tmp_buf == NULL) {
                return (PICL_NOSPACE);
        }
        if (size > pinfo.size) {
                return (PICL_VALUETOOBIG);
        }

        bzero(tmp_buf, pinfo.size);
        (void) memcpy(tmp_buf, vbuf, size);

        /* set the property value */
        if ((err = picl_set_propval(proph, vbuf, pinfo.size)) !=
                        PICL_SUCCESS) {
                return (err);
        }
        return (PICL_SUCCESS);
}

/*
```

CODE EXAMPLE 1-2 System Watchdog Node Management Code Example *(Continued)*

```
 * This function prints the timeout, state, action of a
 * watchdog-timer node
 */
static picl_errno_t
print_watchdog_node_props(picl_nodehdl_t nodeh)
{
        int32_t *timeout = NULL;
        char    *action = NULL, *status = NULL;

        if (wdadm_get_picl_prop(nodeh, WATCHDOG_TIMEOUT,
                        (void **)&timeout) != PICL_SUCCESS) {
                free(timeout);
                return (PICL_FAILURE);
        }

        if (wdadm_get_picl_prop(nodeh, WATCHDOG_STATUS,
                                (void **)&status) != PICL_SUCCESS) {
                free(status);
                free(timeout);
                return (PICL_FAILURE);
        }

        if (wdadm_get_picl_prop(nodeh, WATCHDOG_ACTION,
                                (void **)&action) != PICL_SUCCESS) {
                free(status);
                free(timeout);
                free(action);
                return (PICL_FAILURE);
        }

        (void) printf(PRINT_FORMAT, status, action, *timeout);
        free(status);
        free(timeout);
        free(action);
        return (PICL_SUCCESS);
}

/*
 * This function is the callback function that gets called
 * due to picl_walk_tree_by_class call from print_wd_info function.
 * This function traveses all the watchdog-timer nodes under the given
 * controller and makes a call to print_watchdog_node_props to print
 * the watchdog properties
 */
static int
wd_printf_info(picl_nodehdl_t nodeh, void *args)
{
```

CODE EXAMPLE 1-2 System Watchdog Node Management Code Example *(Continued)*

```
        int err = PICL_SUCCESS;
        int print = 0, i = 0;
        wdadm_args_t    *wd_arg = NULL;
        picl_nodehdl_t childh, peerh;
        char cntrl_name[PICL_PROPNAMELEN_MAX];
        char wd_name[PICL_PROPNAMELEN_MAX];
        char name[2 * PICL_PROPNAMELEN_MAX];

        wd_arg = (wdadm_args_t *)args;

        /* get the controller name */
        err = picl_get_propval_by_name(nodeh, PICL_PROP_NAME,
                (void *)cntrl_name, PICL_PROPNAMELEN_MAX);
        if (err != PICL_SUCCESS) {
                print_errmsg(gettext(err_msg[EM_GETPVALBYNAME]),
                    picl_strerror(err));
                return (err);
        }

        /* get the first child of controller */
        err = picl_get_propval_by_name(nodeh, PICL_PROP_CHILD,
                &childh, sizeof (picl_nodehdl_t));
        if (err != PICL_SUCCESS) /* This controller has no childs */
                return (PICL_WALK_CONTINUE); /* move to next controller */

        peerh = childh;
        /* traverse thru all the timer nodes using peer property. */
        do
        {
                /* get the name of watchdog node */
                err = picl_get_propval_by_name(peerh, PICL_PROP_NAME,
                        (void *)wd_name, PICL_PROPNAMELEN_MAX);
                if (err != PICL_SUCCESS) {
                        print_errmsg(gettext(err_msg[EM_GETPVALBYNAME]),
                            picl_strerror(err));
                        return (err);
                }
                (void) sprintf(name, "%s:%s", cntrl_name, wd_name);

                if (wd_arg != NULL) {
                        /* check if the node is in the list  to print */
                        for (i = wd_arg->start_index; i < wd_arg->max_index;
                                                              i++) {
                                if (strcmp(wd_arg->list[i], name) == 0) {
                                        print = 1;
                                        break;
                                }
```

CODE EXAMPLE 1-2 System Watchdog Node Management Code Example *(Continued)*

```
                                }
                        }

                        if (wd_arg == NULL || print) {
                                if (count == 0) {
                                        (void) printf("%s", HEADER);
                                        count++;
                                }

                                (void) printf("%-30s", name);
                                (void) print_watchdog_node_props(peerh);
                                (void) printf("\n");
                                print = 0;
                        }
                        /* move to next timer node */
                        err = picl_get_propval_by_name(peerh, PICL_PROP_PEER,
                                &peerh, sizeof (picl_nodehdl_t));
                } while (err == PICL_SUCCESS);

                return (PICL_WALK_CONTINUE); /* move to next controller */
}

/*
 * This routine is used to print the information of watchdog nodes
 */
static int
print_wd_info(int argc, char **argv, int optind)
{
        int             err = PICL_SUCCESS;
        wdadm_args_t    *args = NULL;
        wdadm_args_t    wd_args;

        if (argc == optind) {
                /* print information of all the nodes */
                args  = NULL;
        } else {
                /* print information of only specified nodes */
                wd_args.list = argv;
                wd_args.start_index = optind;
                wd_args.max_index = argc;
                args  = &wd_args;
        }
        err = picl_walk_tree_by_class(rooth, PICL_WATCHDOG_CONTROLLER,
                (void  *)args, wd_printf_info);

        if (count == 0) {
                (void) fprintf(stderr, "%s:Node not found:%d\n",
```

CODE EXAMPLE 1-2 System Watchdog Node Management Code Example *(Continued)*

```
                        prog, picl2errno(PICL_NODENOTFOUND));
                return (PICL_NODENOTFOUND);
        }
        return (err);
}

/*
 * This function is the callback function that gets called
 * due to picl_walk_tree_by_class call from set_wd_params function.
 * This function checks if the given controller node has the watchdog-timer
 * of interest and then changes the timeout and action of that timer.
 */
static int
wd_set_params(picl_nodehdl_t nodeh, void *args)
{
        int err = PICL_SUCCESS;
        char     *ptr = NULL;
        char cntrl_name[PICL_PROPNAMELEN_MAX];
        char wd_name[PICL_PROPNAMELEN_MAX];
        picl_nodehdl_t childh, peerh;
        wdadm_args_t    *wd_arg = NULL;
        char            *status = NULL;

        wd_arg = (wdadm_args_t *)args;
        if (wd_arg == NULL || wd_arg->name == NULL)
                return (PICL_WALK_TERMINATE);

        /* get the name of the controller */
        err = picl_get_propval_by_name(nodeh, PICL_PROP_NAME,
                (void *)cntrl_name, PICL_PROPNAMELEN_MAX);
        if (err != PICL_SUCCESS) {
                print_errmsg(gettext(err_msg[EM_GETPVALBYNAME]),
                    picl_strerror(err));
                return (err);
        }

        /*
         * name is of cntrl:node_name format (user input)
         * do the parsing to extract controller name and watchdog-timer
         * name
         */
        ptr = strchr(wd_arg->name, ':');
        if (ptr == NULL) {
                (void) fprintf(stderr, "%s:Node not found:%d\n",
                        prog, picl2errno(PICL_NODENOTFOUND));
                return (PICL_NODENOTFOUND);
        }
```

CODE EXAMPLE 1-2 System Watchdog Node Management Code Example *(Continued)*

```
        /* check if the controller is of interest */
        if (strncmp(cntrl_name, wd_arg->name, (ptr - wd_arg->name)) != 0) {
                return (PICL_WALK_CONTINUE);
        }

        err = picl_get_propval_by_name(nodeh, PICL_PROP_CHILD,
                &childh, sizeof (picl_nodehdl_t));

        if (err != PICL_SUCCESS)
                return (PICL_WALK_TERMINATE);

        ptr++;  /* this points to watchdog node name */
        if (ptr == NULL) {
                (void) fprintf(stderr, "%s:Node not found:%d\n",
                        prog, picl2errno(PICL_NODENOTFOUND));
                return (PICL_WALK_TERMINATE);
        }

        /* traverse thru the list of timers under this controller */
        peerh = childh;
        do
        {
                /* get the name of watchdog node */
                err = picl_get_propval_by_name(peerh, PICL_PROP_NAME,
                        (void *)wd_name, PICL_PROPNAMELEN_MAX);
                if (err != PICL_SUCCESS) {
                        print_errmsg(gettext(err_msg[EM_GETPVALBYNAME]),
                            picl_strerror(err));
                        return (err);
                }

               /* This code segment changes the watchdog timeout and action */
                if (strcmp(ptr, wd_name) == 0) {
                        if ((err = wdadm_get_picl_prop(peerh, WATCHDOG_STATUS,
                                (void **)&status)) != PICL_SUCCESS) {
                                (void) free(status);
                                return (err);
                        }
                        if (strcmp(status, WATCHDOG_DISARMED) != 0) {
                                (void) fprintf(stderr, "%s: Timer is not "
                                        "disarmed, cannot change the "
                                        "parameters\n", prog);
                                (void) free(status);
                                return (PICL_PERMDENIED);
                        }
                        (void) free(status);
```

CODE EXAMPLE 1-2 System Watchdog Node Management Code Example *(Continued)*

```
                                /* set watchdog action */
                                if (wd_arg->action)
                                if ((err = wdadm_set_picl_prop(peerh, WATCHDOG_ACTION,
                                        wd_arg->action,
                                        strlen(wd_arg->action) + 1)) != PICL_SUCCESS) {
                                                (void) fprintf(stderr, "%s:Error in "
                                                "setting action:%d\n", prog,
                                                picl2errno(err));
                                        return (err);
                                }

                                /* set watchdog timeout */
                                if (wd_arg->timeout != ILLEGAL_TIMEOUT)
                                if ((err = wdadm_set_picl_prop(peerh, WATCHDOG_TIMEOUT,
                                                (void *)&wd_arg->timeout,
                                                sizeof (wd_arg->timeout))) !=
                                                                PICL_SUCCESS) {
                                                (void) fprintf(stderr, "%s:Error in "
                                                "setting timeout:%d\n", prog,
                                                picl2errno(err));
                                        return (err);
                                }
                                return (PICL_WALK_TERMINATE);
                        }
                        err = picl_get_propval_by_name(peerh, PICL_PROP_PEER,
                                &peerh, sizeof (picl_nodehdl_t));
        } while (err == PICL_SUCCESS);

        (void) fprintf(stderr, "%s:Node not found:%d\n",
                prog, picl2errno(PICL_NODENOTFOUND));
        return (PICL_NODENOTFOUND);
}

/*
 * This routine gets called to change the watchdog timeout and
 * action.
 * wd_name is of "controller:watchdog-timer" format
 */
static int
set_wd_params(char *wd_name, char *action, char *timeout)
{
        int             err = PICL_SUCCESS;
        char            *ptr = NULL;
        wdadm_args_t    wd_arg;

        if (wd_name == NULL) {
```

CODE EXAMPLE 1-2 System Watchdog Node Management Code Example *(Continued)*

```
                return (PICL_INVALIDARG);
        }

        ptr = strchr(wd_name, ':');
        if (ptr == NULL) {      /* invalid format */
                (void) fprintf(stderr, "%s:Node not found:%d\n",
                        prog, picl2errno(PICL_NODENOTFOUND));
                return (PICL_NODENOTFOUND);
        }

        wd_arg.name = wd_name;
        wd_arg.action = action;
        wd_arg.error_code = 0;
        if (timeout) {
                errno = 0;
                wd_arg.timeout = strtol(timeout, NULL, 10);
                if (errno != 0) {
                        (void) fprintf(stderr, "%s:Illegal timeout value\n",
                                                            prog);
                        return (PICL_INVALIDARG);
                }
        } else {
               wd_arg.timeout = ILLEGAL_TIMEOUT; /* need not program timeout */
        }

        err = picl_walk_tree_by_class(rooth, PICL_WATCHDOG_CONTROLLER,
                (void  *)&wd_arg, wd_set_params);
        return (err);
}

/*
 * This is the callback function that gets called due to
 * picl_walk_tree_by_class function call from control_wd function.
 * This function is used to arm/disarm the watchdog controller.
 */
static int
wd_change_state(picl_nodehdl_t nodeh, void *arg)
{
        int err = PICL_SUCCESS;
        char cntrl_name[PICL_PROPNAMELEN_MAX];
        wdadm_args_t    *wd_arg = NULL;

        wd_arg = (wdadm_args_t *)arg;
        if (wd_arg == NULL || wd_arg->name == NULL)
                return (PICL_WALK_TERMINATE);

        err = picl_get_propval_by_name(nodeh, PICL_PROP_NAME,
```

CODE EXAMPLE 1-2 System Watchdog Node Management Code Example *(Continued)*

```
                (void *)cntrl_name, PICL_PROPNAMELEN_MAX);
        if (err != PICL_SUCCESS) {
                print_errmsg(gettext(err_msg[EM_GETPVALBYNAME]),
                    picl_strerror(err));
                return (err);
        }

        /*
         * check to see if the controller is of interest, otherwise
         * move to the next controller.
         */
        if (strcmp(cntrl_name, wd_arg->name) != 0) {
                return (PICL_WALK_CONTINUE);
        }

        count++;
        /* change the watchdog-controller's WdOp property */
        if ((err = wdadm_set_picl_prop(nodeh, WATCHDOG_OP,
                wd_arg->op, strlen(wd_arg->op) + 1)) != PICL_SUCCESS) {
                        (void) fprintf(stderr, "%s:Failed:%d\n", prog,
                                        picl2errno(err));
        }
        return (err);
}

/*
 * Function is used to disarm/arm the watchdog controller
 */
static int
control_wd(char *cntrl_name, char *op)
{
        wdadm_args_t    wd_arg;
        int err = PICL_SUCCESS;

        if (cntrl_name == NULL || op == NULL) {
                (void) fprintf(stderr, "%s:Invalid arguments\n", prog);
                return (PICL_INVALIDARG);
        }
        wd_arg.name = cntrl_name;
        wd_arg.op = op;
        wd_arg.error_code = 1;
        err = picl_walk_tree_by_class(rooth, PICL_WATCHDOG_CONTROLLER,
                (void  *)&wd_arg, wd_change_state);

        if (count == 0) {
                (void) fprintf(stderr, "%s:Invalid controller name\n",
                                                                prog);
```

CODE EXAMPLE 1-2 System Watchdog Node Management Code Example *(Continued)*

```
                return (PICL_NODENOTFOUND);
        }

        return (err);
}

int
main(int argc, char **argv)
{
        int             err;
        int             c, rc = 0;
        char            cntrl_name[PICL_CLASSNAMELEN_MAX];
        char            op[PICL_CLASSNAMELEN_MAX];
        char            wd_name[PICL_CLASSNAMELEN_MAX];
        char            timeout[PICL_CLASSNAMELEN_MAX];
        char            action[PICL_CLASSNAMELEN_MAX];
        int             cflg = 0, oflg = 0, lflg = 0;
        int             mflg = 0, tflg = 0, aflg = 0;

        (void) setlocale(LC_ALL, "");
        if ((prog = strrchr(argv[0], '/')) == NULL)
                prog = argv[0];
        else
                prog++;

        bzero(timeout, PICL_CLASSNAMELEN_MAX);
        bzero(action, PICL_CLASSNAMELEN_MAX);

        while ((c = getopt(argc, argv, "hlc:o:m:t:a:")) != EOF) {
                switch (c) {
                case 'l':
                        lflg = 1;
                        break;
                case 'c':
                        cflg = 1;
                        (void) strlcpy(cntrl_name, optarg,
                            PICL_CLASSNAMELEN_MAX);
                        break;
                case 'o':
                        oflg = 1;
                        (void) strlcpy(op, optarg,
                            PICL_CLASSNAMELEN_MAX);
                        break;
                case 'm':
                        mflg = 1;
                        (void) strlcpy(wd_name, optarg,
                            PICL_CLASSNAMELEN_MAX);
```

CODE EXAMPLE 1-2 System Watchdog Node Management Code Example *(Continued)*

```
                        break;
                case 't':
                        tflg = 1;
                        (void) strlcpy(timeout, optarg,
                            PICL_CLASSNAMELEN_MAX);
                        break;
                case 'a':
                        aflg = 1;
                        (void) strlcpy(action, optarg,
                            PICL_CLASSNAMELEN_MAX);
                        break;
                case 'h':
                        (void) printf("%s\n", USAGE_STR);
                        (void) printf("%s", DETAILED_HELP);
                        exit(0);
                case '?': /*FALLTHROUGH*/
                default:
                        usage();
                        /*NOTREACHED*/
                }
        }

        /* check if more than one action is specified */
        if ((lflg + cflg + mflg) > 1) {
                (void) printf("wdadm: more than one action "
                        "specified (-l,-m,-c)\n");
                usage();
        }

        if ((lflg + cflg + mflg) == 0) {
                /* if no args are specified, default action is listing */
                lflg++;
        }

        err = picl_initialize();
        if (err != PICL_SUCCESS) {
                print_errmsg(gettext(err_msg[EM_INIT]), picl_strerror(err));
                exit(1);
        }

        err = picl_get_root(&rooth);
        if (err != PICL_SUCCESS) {
                print_errmsg(gettext(err_msg[EM_GETROOT]),
                    picl_strerror(err));
                (void) picl_shutdown();
                exit(1);
        }
```

CODE EXAMPLE 1-2 System Watchdog Node Management Code Example *(Continued)*

```
        if (lflg) {
                rc = print_wd_info(argc, argv, optind);
                (void) picl_shutdown();
                return (picl2errno(rc));
        }

        if (argc != optind) {
                (void) picl_shutdown();
                usage();
        }

        if (mflg) {
                if ((aflg + tflg) < 1) {
                        /*
                         * m flag must be associated with atleast
                         * action or timeout
                         */
                        (void) printf("wdadm: timeout and action values "
                                "are missing\n");
                        (void) picl_shutdown();
                        usage();
                }
                rc = set_wd_params(wd_name, (aflg ? action : NULL),
                                (tflg ? timeout : NULL));
        }

        if (cflg) {
                if (oflg == 0) {
                        /* operation must be specified along with c option */
                       (void) printf("wdadm: operation argument is missing\n");
                        (void) picl_shutdown();
                        usage();
                }
                rc = control_wd(cntrl_name, op);
        }
        (void) picl_shutdown();
        return (picl2errno(rc));
}
```

OpenBoot PROM Interface

The OpenBoot™ PROM provides two environmental parameters, settable at the `ok` prompt, that control the behavior of the SMC watchdog timer.

These parameters are `watchdog-enable?` and `watchdog-timeout`. The `watchdog-enable?` parameter is a logical switch with two possible values: `true` or `false`.

If `watchdog-enable?` is set to `false,` the watchdog timer is disabled at boot time. Once the kernel is booted, applications have the option to open and start the watchdog timer.

If `watchdog-enable?` is set to `true`, the watchdog timer is enabled at boot time with its default actions, as follows. The WD1 timer is controlled by the value in the `watchdog-timeout` variable. The default value for `watchdog-timeout` is 65535 (in the unit of one-tenth of a second). When WD1 expires, it sends an asynchronous message to the SPARC CPU and starts the WD2 timer. The default value for WD2 is one second. If WD2 expires, it resets the system.

If the watchdog timer is enabled at boot time, it is your responsibility to ensure that an application program is run to periodically restart the WD1 timer. If you fail to do so, the watchdog timer may reset the SPARC CPU when the watchdog expires.

Watchdog Operation

The watchdog operation (the *local watchdog*) is the watchdog that works between the SPARC CPU and System Management Controller (SMC).

Commands at OpenBoot PROM Prompt

Commands for `smc` are available in the SMC controller device mode (`/pci@1f,0/pci@1,1/isa@7/sysmgmt@0,8010 alias hsc`). You need to go to the `sysmgmt` node before executing the `smc` commands and execute the following once:

```
ok dev hsc
```

TABLE 1-4 lists the commands at OpenBoot prompt.

TABLE 1-4 OpenBoot PROM Prompt Commands

Command	Description
`smc-get-wdt`	Gets the current timers values, and other watchdog state bits.
`smc-set-wdt`	Sets the timers values and other flags. This command is also used to stop watchdog operations.
`smc-reset-wdt`	Starts timer countdown and is often referred to as the "heartbeat".

Corner Cases

When watchdog reset occurs, the power module is toggled. Thus, the state of the CPU, except those stored in nonvolatile memory, will be lost. Once watchdog reset occurs after the SPARC CPU is restarted, the SPARC CPU must restart the watchdog timer.

The SPARC CPU must perform a corner case. After the SMC resets the SPARC CPU, the output buffer full (OBF) bit and OEM1 bit in the isa bus status register remain set. Since this is a read-only bit, the SMC cannot reset the bit. The SPARC CPU must ignore the status bits and clear the OBF bit by reading one byte of data from the isa bus. This action must be performed after watchdog reset. Otherwise, the SPARC CPU can inadvertently restart watchdog. For example, if the timer's values are set to very low numbers, the board can never boot to the Solaris operating system.

The SMC manages the race condition by putting interlock. The SMC does not start pre-timeout timer unless the warning is dispatched to the SPARC CPU. The code is set up on the SPARC CPU side after watchdog warning is issued. Use a Keyboard Controller Style (KCS) command to clear the watchdog interrupt. Using this command is the only way to avoid the selected pre-timeout action such as hard reset. This command rewinds the watchdog timer. The application program internally manages the warning, along with the command being sent to the SMC.

If `diag-switch?` is set to true, the timing for watchdog can be affected.

Setting the Watchdog Timer at OpenBoot PROM

The examples in this section are performed at the OpenBoot PROM level.

▼ To Set the Watchdog Timer Without Running the Pre-Timeout Timer

In this example, after level one expires, the CPU is reset.

1. **Set the timer to 10 minutes = 600 sec = 600,000/10 msec = 0x1770.**
2. **Set the reload values inside the SMC:**

```
ok 17 70 ff 0 31 4 smc-set-wdt
```

3. **Start the watchdog timer:**

```
ok smc-reset-wdt
```

▼ To Set the Watchdog Timer With Pre-Timeout Time

This procedure sets the reload values of countdown timer and pre-timeout timer. In this example, after level one expires, there are 80 seconds before the reset.

1. **Set the timer to 80 seconds = 0x50.**

   Set the countdown value to 10 minutes, as in the previous procedure, and set the pre-timeout timer to 80 seconds.

```
ok 17 70 ff 50 31 4 smc-set-wdt
```

2. **Start the watchdog timer:**

```
ok smc-reset-wdt
```

▼ To Stop the Watchdog Timer

```
ok ff ff ff 0 31 4 smc-set-wdt
```

CHAPTER 2

Environmental Monitoring

The Netra CP2300 board uses an intelligent fault detection environmental monitoring system that increases uptime and manageability of the board. The System Management Controller (SMC) module on the Netra CP2300 supports the temperature and voltage environmental monitoring functions. This chapter describes the specific environmental monitoring functions of the Netra CP2300.

Note – *Environmental monitoring* refers to the functionality that was previously called *Advanced System Monitoring (ASM)* in the Netra CPU board documentation.

This chapter includes the following sections:

Environmental Monitoring Component Compatibility

TABLE 2-1 lists the compatible environmental monitoring hardware, OpenBoot PROM, and Solaris operating environment for the Netra CP2300.

TABLE 2-1 Compatible Environmental Monitoring Components

Component	**Environmental Monitoring Compatibility**
Hardware	Board supports environmental monitoring
OpenBoot PROM	Environmental monitoring is supported by OpenBoot PROM.
Operating environment	Solaris 8 2/02 operating environment or subsequent compatible versions

Typical Environmental Monitoring System Application

FIGURE 2-1 illustrates the Netra CP2300 environmental monitoring application block diagram. For locations of the temperature sensors, see FIGURE 2-2 and FIGURE 2-3.

Note – In FIGURE 2-1, ASM refers to the environmental monitoring functionality. The ASM driver is no longer used on the Netra CP2300 board.

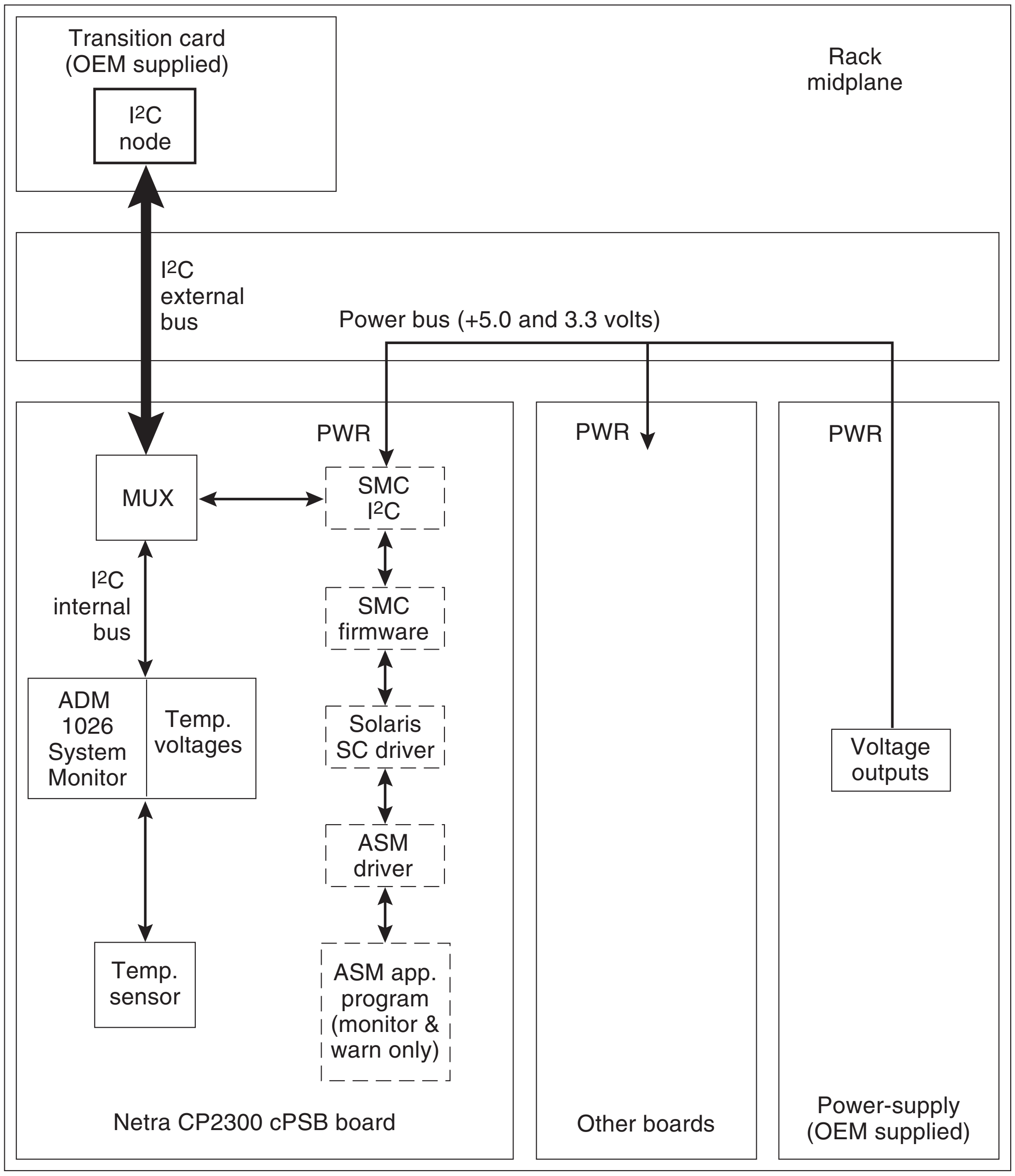


FIGURE 2-1 Typical Environmental Monitoring Application Block Diagram

The Netra CP2300 functions as a node board in a cPSB system rack. The Netra CP2300 monitors its CPU diode temperature and issues warnings at both the OpenBoot PROM and Solaris operating environment levels when these environmental readings are out of limits. At the Solaris operating environment level, the application program monitors and issues warnings for the board. At the OBP level, the CPU diode temperature is monitored if the NVRAM variable `env-monitor` is enabled.

Typical Cycle From Power Up to Shutdown

This section describes a typical environmental monitoring cycle from power up to shutdown.

Environmental Monitoring Protection at the OpenBoot PROM

The OpenBoot PROM monitors the CPU diode temperature at the fixed polling rate of 10 seconds and displays warning messages on the default output device whenever the measured temperature exceeds the pre-programmed NVRAM module configurable variable warning temperature (the `warning-temperature` parameter), the critical temperature (the `critical-temperature` parameter), or the shutdown temperature (the `shutdown-temperature` parameter). See "OpenBoot PROM Environmental Parameters" on page 37 for information on changing these pre-programmed parameters.

OpenBoot PROM-level protection takes place only when the `env-monitor` parameter is enabled (it is the default setting). If the NVRAM variable `env-monitor` is set to `enabled-with-shutdown` (`env-monitor=enabled-with-shutdown`), and if the board temperature exceeds the shutdown temperature, the OpenBoot PROM will shut down power to the Netra CP2300 CPU. If the NVRAM variable `env-monitor` is set to `enabled` (`env-monitor=enabled`), the OpenBoot PROM will send a warning, critical, or shutdown temperature message to the user that the Netra CP2300 is overheating.

Disabling `env-monitor` completely disables environmental monitoring protection at the OpenBoot PROM level but does not affect environmental monitoring protection at the Solaris operating environment level.

Note – To protect the system at OpenBoot PROM level, the `env-monitor` should be enabled at all times.

Environmental Monitoring Protection at the Operating Environment Level

Monitoring changes in the sensor temperatures can be a useful tool for determining problems with the room where the system is installed, functional problems with the system, or problems on the board. Establishing baseline temperatures early in deployment and operation could be used to trigger alarms if the temperatures from the sensors increase or decrease dramatically. If all the sensors go to room ambient, power has probably been lost to the host system. If one or more sensors rise in temperature substantially, there may be a system fan malfunction, the system cooling may have been compromised, or room air conditioning may have failed.

Protection at the operating environment level takes place when the PICL environmental monitoring program (`envmond`) is running. The environmental monitoring program is part of a Unix daemon that runs automatically when Solaris boots up.

In a typical environmental monitoring application program, the software reads the CPU, inlet, and exhaust temperature sensors once every polling cycle. The program then compares the measured CPU diode temperature with the warning temperature and displays a warning message on the default output device whenever the warning temperature is exceeded.

The program can also issue a shutdown message on the default output device whenever the measured CPU diode temperature exceeds the shutdown temperature. In addition, the `envmond` application program can be programmed to sync and shut down the Solaris operating environment when conditions warrant.

Refer to "Sample Application Program" on page 49 for an example of how a simple `envmond` program can be implemented.

The power module is controlled by the SMC subsystem (except for automatic controls such as overcurrent shutdown or voltage regulation). The functions controlled are core voltage output level and power sequencing/monitor.

Post Shutdown Recovery

The onboard voltage controller is a hardware function that is not controlled by either firmware or software. At the OpenBoot PROM level, if the NVRAM variable `env-monitor` is set to `enabled-with-shutdown` (`env-monitor=enabled-with-shutdown`), and if the board temperature exceeds the shutdown temperature, the OpenBoot PROM will shut down power to the Netra CP2300 CPU.

There is no mechanism for the Solaris operating environment to either recover or restore power to the Netra CP2300 when an unusual condition occurs (for example, if the CPU diode temperature exceeds its maximum recommended level). In either case, the end user must intervene and manually recover the Netra CP2300 as well as the cPSB system through hardware control. Once a shutdown has occurred, you can recover the board using a cold-reset IPMI command to SMC or by extracting and reinserting the board.

Hardware Environmental Monitoring Functions

This section summarizes the hardware environmental monitoring features on the Netra CP2300 board. TABLE 2-2 lists the environmental monitoring functions on a Netra CP2300 board.

TABLE 2-2 Typical Netra CP2300 cPSB Board Hardware Environmental Monitoring Functions

Function	Capability
Board Exhaust Air Temperature	Senses the air temperature at the trailing edge of the board. (Assumes air direction from the processor/heatsink toward the 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.)

TABLE 2-3 shows the I^2C components.

TABLE 2-3 I^2C Components

Component	Function
DS80CH11	SMC I^2C controller - IPMB
PCF8584	I^2C controller
PCF9545	4 channel I^2C multiplexor
AT24C64	I^2C EEPROM - motherboard FRUID
AT24C01	I^2C EEPROM - RTM FRUID + external I^2C header
ADM1026	System monitor/general purpose I/O
AT24C01	I^2C EEPROM - onboard memory SPD
DS1307	I^2C TOD
AT24C64	I^2C EEPROM - NVRAM/Ethernet MAC ID
LTC4300	I^2C hotswap isolator
AT24C*xx*	I^2C EEPROM - SO DIMM 1 SPD (add-on dependent)
AT24C*xx*	I^2C EEPROM - SO DIMM 0 SPD (add-on dependent)
AT24C*xx*	PMC/PTMC B (add-on card dependent)
AT24C*xx*	PMC/PTMC A (add-on card dependent)
87LPC764	"IMAX" configurable 4 channel I^2C multiplexor
ALi1535D+	Southbridge - SMBUS/I^2C controller

FIGURE 2-2 and FIGURE 2-3 show the location of the environmental monitoring hardware on the Netra CP2300.

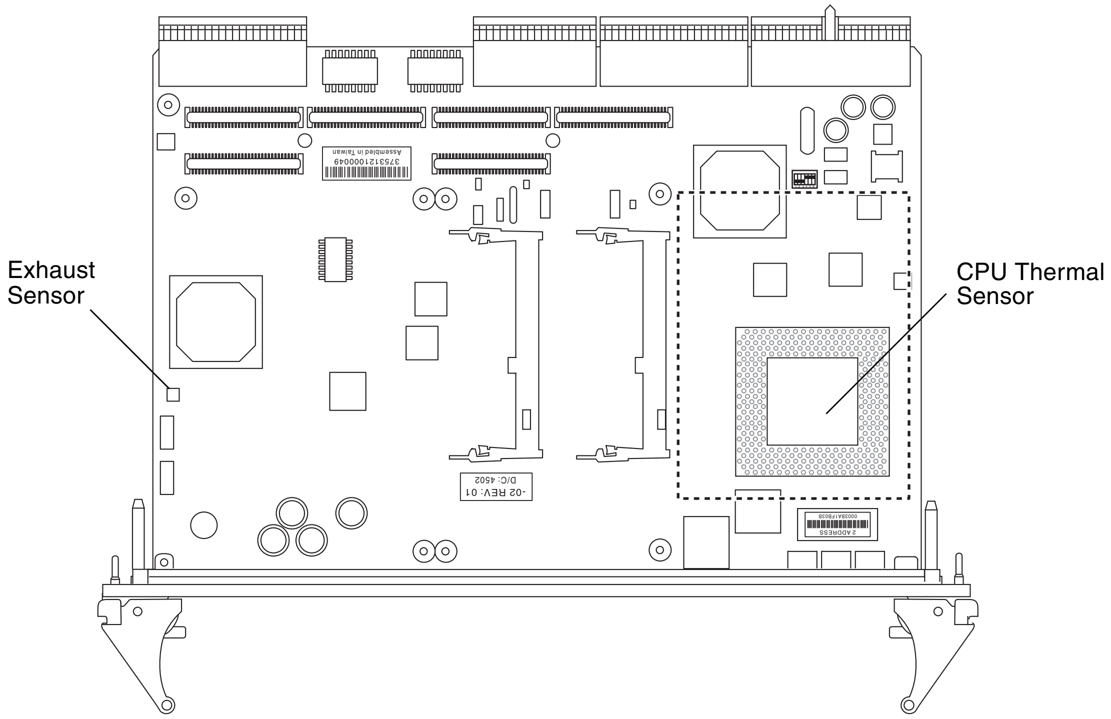


FIGURE 2-2 Location of Environmental Monitoring Hardware on the Netra CP2300 cPSB Board (Top Side)

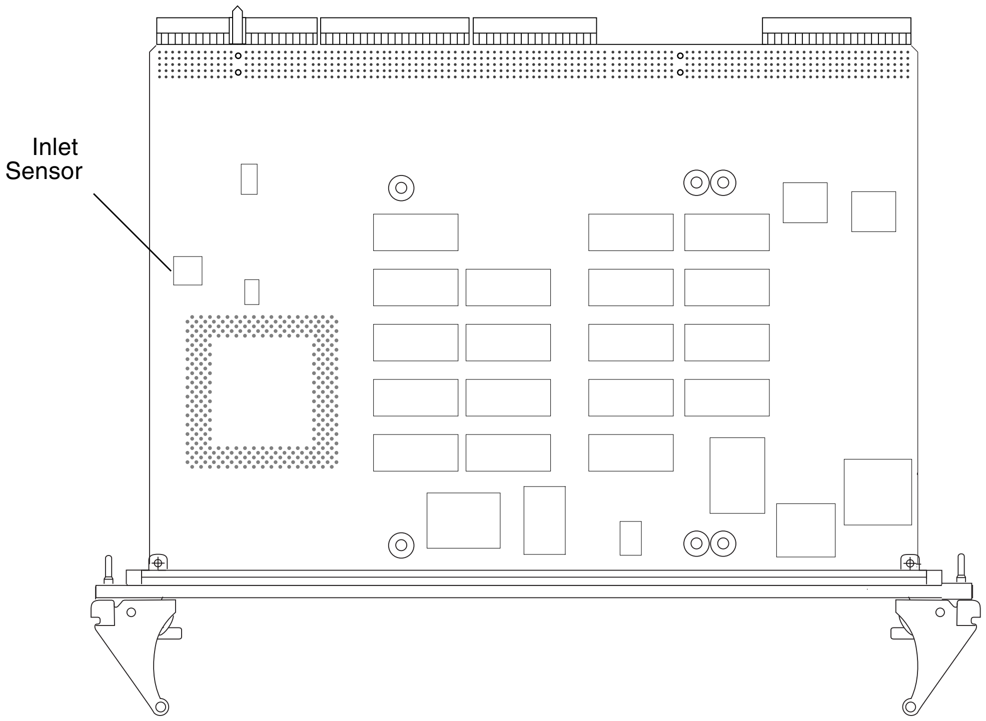


FIGURE 2-3 Location of Environmental Monitoring Hardware on the Netra CP2300 cPSB Board (Bottom Side)

FIGURE 2-4 is a block diagram of the environmental monitoring functions.

Note – In FIGURE 2-4, ASM refers to the environmental monitoring functionality. The ASM driver is no longer used on the Netra CP2300 board. The *ASM device driver* block in Figure 3-4 should be replaced with *SC device driver*.

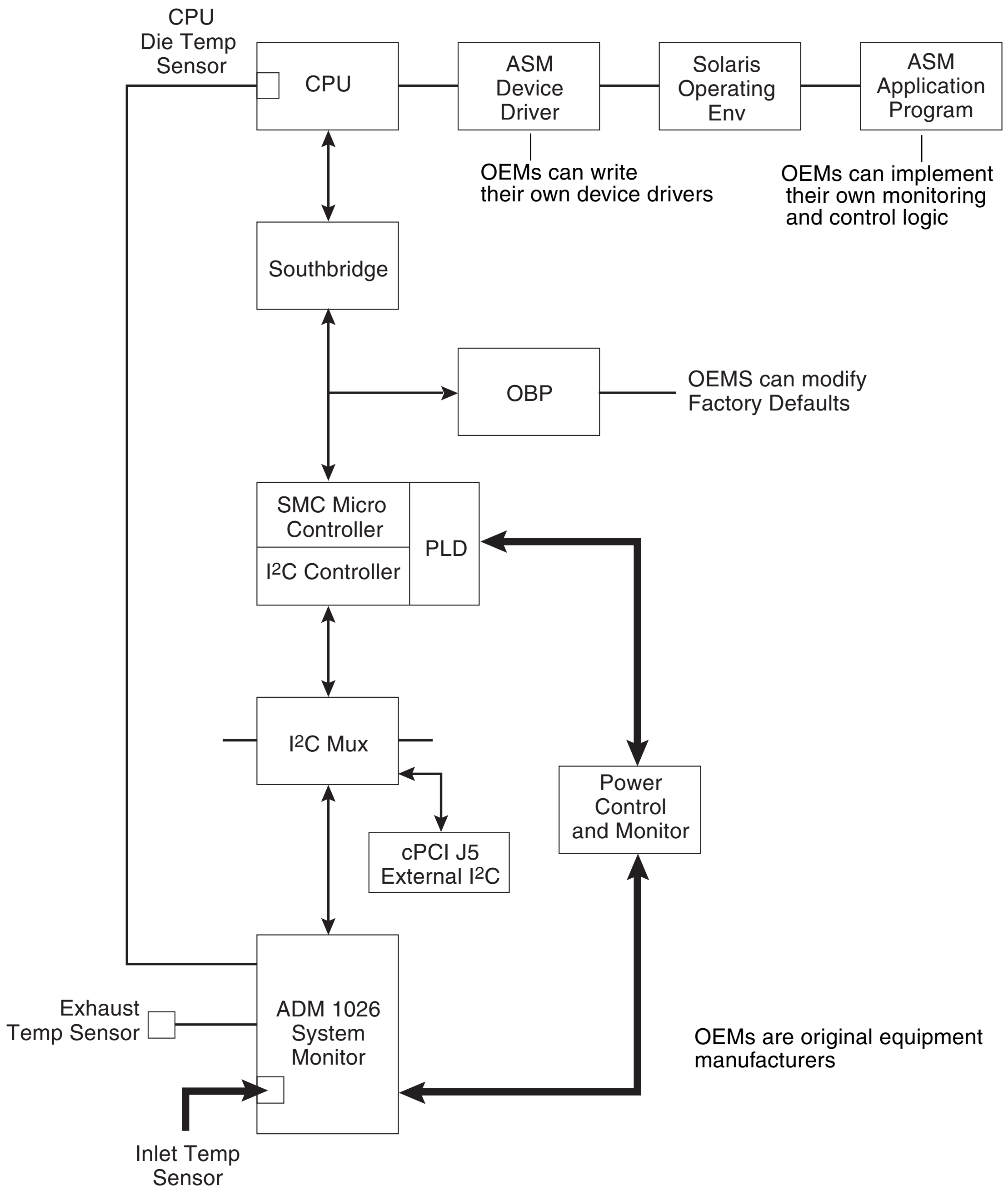


FIGURE 2-4 Netra CP2300 cPSB Board Environmental Monitoring Functional Block Diagram

Power On/Off Switching

The onboard voltage controller allows power to the CPU of the Netra CP2300 only when the following conditions are met:

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

The CPU diode sensor reading may 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 may 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. Care must be taken 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 may 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 on the solder-side 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; care must be taken to 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 CP2300s.

A gradual increase in the delta temperature from inlet to outlet can be due to dust clogging system filters. This feature can be used to set service levels for filter cleaning or changing.

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

Adjusting the Environmental Monitoring Warning, Critical, and Shutdown Parameter Settings on the Board

The Netra CP2300 uses the environmental monitoring detection system to monitor the temperature of the board. The environmental monitoring system will display messages if the board temperature exceeds the set warning, critical, and shutdown settings. Because the on-board sensors may report different temperature readings for different system configurations and airflows, you may want to adjust the warning, critical, and shutdown temperature parameter settings.

The Netra CP2300 determines the board temperature by retrieving temperature data from sensors located on the board. A board sensor reads the temperature of the immediate area around the sensor. Although the software may appear to report the temperature of a specific hardware component, the software is actually reporting the temperature of the area near the sensor. For example, the CPU diode sensor reads the temperature at the location of the sensor and not on the actual CPU heat sink. The board's OpenBoot PROM collects the temperature readings from each board sensor at regular intervals. You can display these temperature readings using the `show-sensors` OpenBoot PROM command. See "show-sensors Command at OpenBoot PROM" on page 41.

The temperature read by the CPU sensor will trigger OpenBoot PROM warning, critical, and shutdown messages. When the CPU sensor reads a temperature greater than the warning parameter setting, the OpenBoot PROM will display a warning message. Likewise, when the sensor reads a temperature greater than the shutdown setting, the OpenBoot PROM will display a shutdown message.

Many factors affect the temperature readings of the sensors, including the airflow through the system, the ambient temperature of the room, and the system configuration. These factors may contribute to the sensors reporting different temperature readings than expected.

TABLE 2-4 shows the sensor readings of a Netra CP2300 operating in a Sun server in a room with an ambient temperature of 21˚C. The temperature readings were reported using the `show-sensors` OpenBoot PROM command. Note that the reported temperatures are higher than the ambient room temperature.

TABLE 2-4 Reported Temperature Readings at an Ambient Room Temperature of 21˚C on a Typical Netra CP2300 cPSB Board

Board Sensor Location	Reported Temperatures (in Degrees Celsius)	Difference Between Reported and Ambient Room Temperature (in Degrees Celsius)
CPU	41	20
Inlet 1	31	10
Exhaust 1	29	8

Since the temperature reported by the CPU diode sensor might be different than the actual CPU temperature, you may want to adjust the settings for the `warning-temperature`, `critical-temperature`, and `shutdown-temperature` OpenBoot PROM parameters. The default values of these parameters have been conservatively set at 74˚C for the warning temperature, 79˚C for the critical temperature, and 91˚C for the shutdown temperature.

Note – If you have developed an application that uses the environmental monitoring software to monitor the temperature sensors, you may want to adjust your application's settings accordingly.

OpenBoot PROM Environmental Parameters

This section describes how to change the OpenBoot PROM environmental monitoring parameters. These global OpenBoot PROM parameters do not apply at the Solaris level. Instead, the environmental monitoring application program provides equivalent parameters that do not necessarily have to be set to the same values as their OpenBoot PROM counterparts. Refer to "Environmental Monitoring Application Programming" on page 47 for information about using environmental monitoring at the Solaris level. The OpenBoot PROM polling rate is at fixed intervals of 10 seconds.

OpenBoot PROM Warning Temperature Parameter

OBP programs SMC for temperature monitoring using the sensor commands. On a Netra CP2300, there are three NVRAM variables that provide different temperature levels. The critical-temperature limit lies between warning and shutdown thresholds. The default values of these temperature thresholds and corresponding action are shown in TABLE 2-5.

TABLE 2-5 Typical Netra CP2300 Board Temperature Thresholds and Firmware Action

Thresholds with Default	Firmware Action
warning-temperature = 74° C	OBP displays warning message
critical-temperature = 79° C	OBP displays warning message
shutdown-temperature = 91° C	OBP shuts down the CPU processor and the Netra CP2300 board if `env-monitor=enabled-with-shutdown`

Note that there is a lower limit of 50° C on `shutdown-temperature` value. If you try to set the temperature to a value lower than 50° C, OpenBoot PROM will not accept it. This safeguards a user from setting the `shutdown-temperature` lower than the room temperature and thereby causing the CPU processor and the Netra CP2300 to be powered off by SMC on the next reset.

The `warning-temperature` global OpenBoot PROM parameter determines the temperature at which a warning is displayed. The `shutdown-temperature` global OpenBoot PROM parameter determines the temperature at which the system is shut down. The temperature monitoring environment variables can be modified at the OpenBoot PROM command level as shown in examples below:

```
ok setenv warning-temperature 75
```

or:

```
ok setenv shutdown-temperature 90
```

The `critical-temperature` is a second-level warning temperature with a default value of 79° C. This variable can be modified using the OpenBoot PROM level `setenv` command as shown in example below:

```
ok setenv critical-temperature 80
```

OpenBoot PROM Environmental Monitoring

This section describes the OpenBoot PROM environmental monitoring functions.

CPU Monitoring

The following NVRAM module environmental monitoring variables are in OpenBoot PROM.

- NVRAM module variable name: `env-monitor`
  - Function: enables or disables environment monitoring at OpenBoot PROM
  - Data type: string
  - Valid values: disabled or enabled
  - Default value: enabled
  - OpenBoot PROM Usage:

```
ok setenv env-monitor disabled or enabled
```

- NVRAM module variable name: `warning-temperature`
  - Function: sets the CPU warning temperature threshold
  - Data type: byte
  - Unit: decimal
  - Default value:74
  - OpenBoot PROM Usage:

```
ok setenv warning-temperature temperature-value
```

- NVRAM module variable name: `critical-temperature`
  - Function: sets the CPU critical temperature threshold
  - Data type: byte
  - Unit: decimal
  - Default value: 79
  - OpenBoot PROM Usage:

```
ok setenv critical-temperature temperature-value
```

- NVRAM module variable name: `shutdown-temperature`
  - Function: sets the CPU shutdown temperature threshold
  - Data type: byte
  - Unit: decimal
  - Default value: 91
  - OpenBoot PROM Usage:

```
ok setenv shutdown-temperature temperature-value
```

Caution – Exercise caution while setting the above two parameters. Setting these values too high will leave the system unprotected against system over-heat. Setting these values too low will power down the system in an unpredictable manner.

Warning Temperature Response at OpenBoot PROM

When the CPU diode temperature reaches "warning-temperature," a similar message is displayed at the `ok` prompt at a regular interval:

```
Temperature sensor #2 has threshold event of
<<< WARNING!!! Upper Non-critical - going high >>>
The current threshold setting is : 74
The current temperature is : 75
```

Critical Temperature Response at OpenBoot PROM

When the CPU diode temperature reaches "critical-temperature," a similar message is displayed at the `ok` prompt at a regular interval:

```
Temperature sensor #2 has threshold event of
<<< !!! ALERT!!! Upper Critical - going high >>>
The current threshold setting is : 79
The current temperature is : 80
```

show-sensors Command at OpenBoot PROM

The `show-sensors` command at OpenBoot PROM displays the readings of all the temperature sensors on the board. A sample output for typical sensor readings for a Netra CP2300 is as follows:

```
ok show-sensors
Sensor#    Sensor Name                                Sensor Reading
=======    ====================================       ===================
   1       EP 5v                         Sensor       (d7)  5.112 volts
   2       EP 3.3v                       Sensor       (8e)  3.408 volts
   3       BP +12v                       Sensor       (d3)  12.048 volts
   4       BP -12v                       Sensor       (62)  -12.020 volts
   5       IPMB Power                    Sensor       (d7)  5.088 volts
   6       SMC Power                     Sensor       (d7)  5.088 volts
   7       VDD 3.3v                      Sensor       (ac)  3.3368 volts
   8       VCCP                          Sensor       (90)  1.6992 volts
   9       +12v                          Sensor       (c2)  12.1250 volts
   a       -12v                          Sensor       (37)  -11.968 volts
   b       +5v                           Sensor       (c4)  5.096 volts
   c       Standby 3.3v                  Sensor       (bf)  3.2852 volts
   d       Main 3.3v                     Sensor       (bf)  3.2852 volts
   e       External I  temp (CPU)        Sensor       (29)  41 degree C
   f       External II temp (Outlet)     Sensor       (1b)  31 degree C
  10       Internal    temp (Inlet)      Sensor       (1b)  29 degree C

Verifying Access to EEPROMs :

IPMI FRU EEPROM (EEPROM id 00) : Passed
SUN FRU EEPROM  (EEPROM id 20) : Passed
FRU EEPROM      (EEPROM id 21) : Passed
ADM chip EEPROM (EEPROM id 22) : Passed
ok
```

IPMI Command Examples at OpenBoot PROM

The Intelligent Platform Management Interface (IPMI) commands can be used to enable the sensors monitoring and subsequent event generation from other boards in the system.

The IPMI command examples provided in this section are based on the *IPMI Specification Version 1.0*. Please use the IPMI Specification for additional information on how to implement these IPMI commands.

Note – To execute an IPMI command, at the OpenBoot PROM `ok` prompt, type the packets *in reverse order* followed by the relevant information as shown in examples in "Examples of IPMI Command Packets" on page 43. Change the bytes in the example packet to accommodate different IPMI addresses, different threshold values or different sensor numbers. See also the *IPMI Specification Version 1.0.*

▼ Set or Change the Thresholds for a Sensor

The command `execute-smc-cmd` is available in SMC controller device mode (`/pci@1f,0/pci@1,1/isa@7/sysmgmt@0,8010 alias hsc`). You need to go to the `sysmgmt` node before executing the command `execute-smc-cmd` using the following:

```
ok dev hsc
```

1. **Set the thresholds for the sensors.**

   See "Set Sensor Threshold" on page 44. If no threshold is set, the default threshold operates:

```
ok packet bytes  number-of-bytes-in-packet  34 execute-smc-cmd
```

2. **Follow instructions in "Check Whether the IPMI Commands Are Executed Properly" on page 43 to check proper execution of the command.**

▼ Enable Events From a Sensor

1. **To execute a command to enable events from the sensor, type:**

```
ok packet bytes  number-of-bytes-in-packet  34 execute-smc-cmd
```

   See "Set Sensor Event Enable Command" on page 46 and "Get Sensor Event Enable" on page 47.

   There are supporting commands for any sensor and the corresponding packets at these commands: `get sensor threshold`, `get sensor reading`, and `get sensor event enable`.

2. **Follow instructions in "Check Whether the IPMI Commands Are Executed Properly" on page 43 to check proper execution of the command.**

▼ Check Whether the IPMI Commands Are Executed Properly

1. **Check whether the stack on the `ok` prompt displays `0` when the command is issued.**

   A 0 indicates that the command packet sent to the board was successful.

2. **Type `execute-smc-cmd (cmd 33)` command at the `ok` prompt as follows:**

   ```
   ok 0 33 execute-smc-cmd
   ```

   This command verifies that the target satellite board received and executed the command and sent a response.

3. **Check the completion code which is the seventh byte from left.**

   If the completion code is 0, then the target board successfully executed the command. Otherwise the command was not successfully executed by the board.

4. **Check that rsSA and rqSA are swapped in the response packet.**

   The rsSA is the responder slave address and the rqSA is the requestor slave address.

5. **(Optional) If command not correctly executed, resend the IPMI command.**

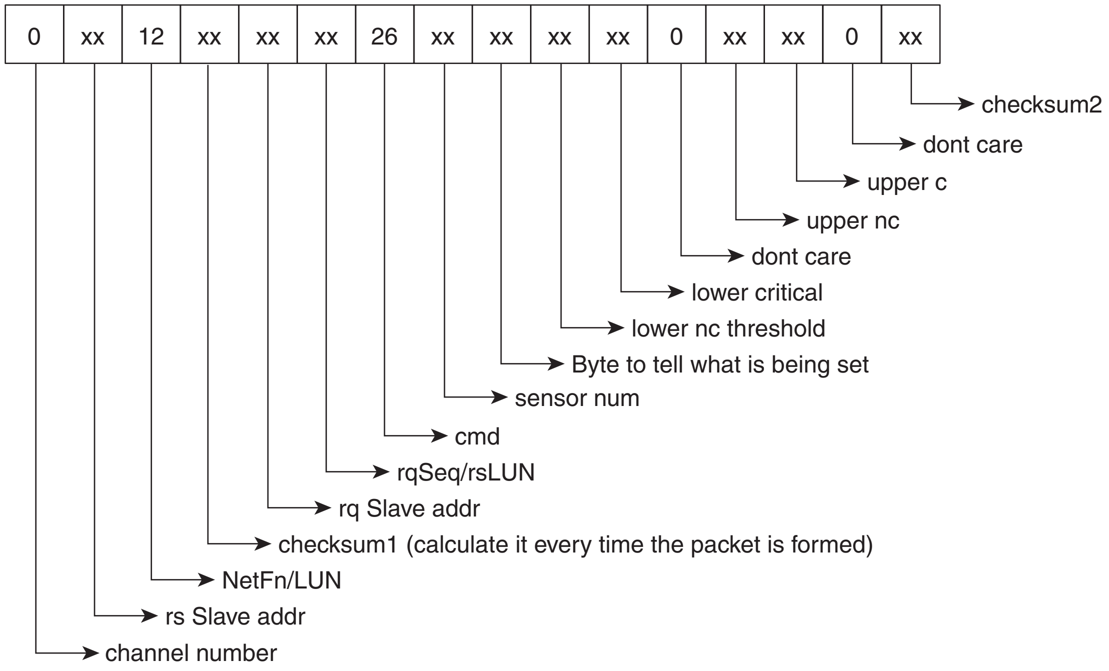
Examples of IPMI Command Packets

The following packets are IPMI command packets that can be sent from the OpenBoot PROM `ok` prompt:

Set Sensor Threshold

A typical example of the sensor command is as follows:

```
37 0 41 10 0 0 3 1b 0 26 12 20 34 12 ba 0 10 34 execute-smc-cmd
```

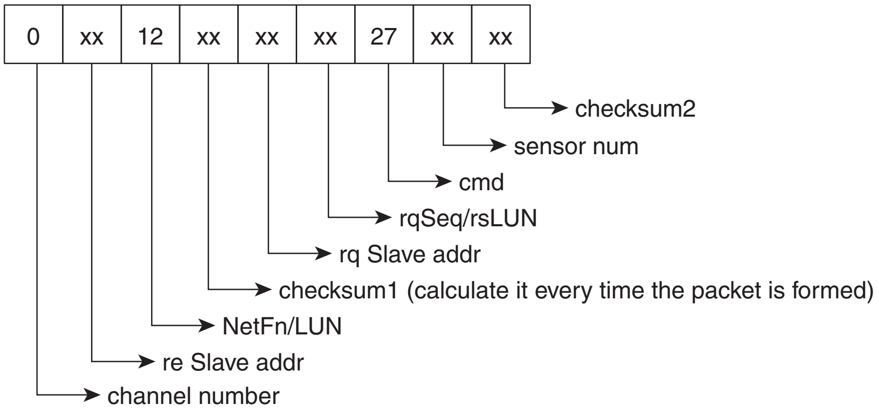


Note – In byte number 9, if the bit for a corresponding threshold is set to 1, then that threshold is set. If the bit is 0, the System Management Controller ignores that threshold. But if an attempt is made to set a threshold that is not supported, an error is returned in the command response.

Get Sensor Threshold

A typical example of the sensor command is as follows

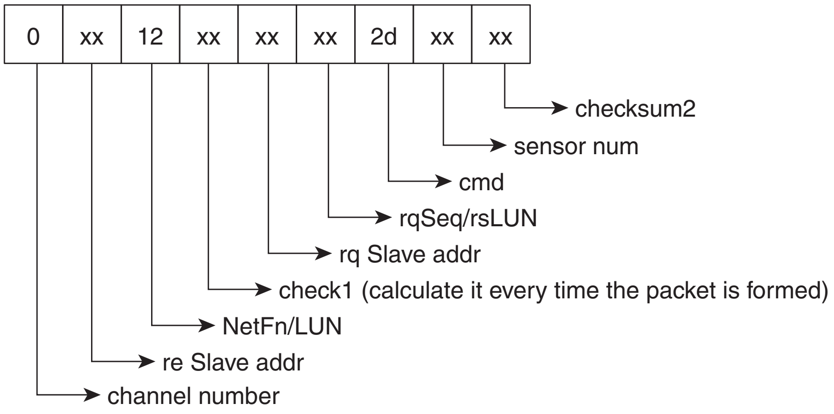
```
a5 0 27 12 20 34 12 ba 0 9 34 execute-smc-cmd
```



Get Sensor Reading

A typical example of the sensor command is as follows:

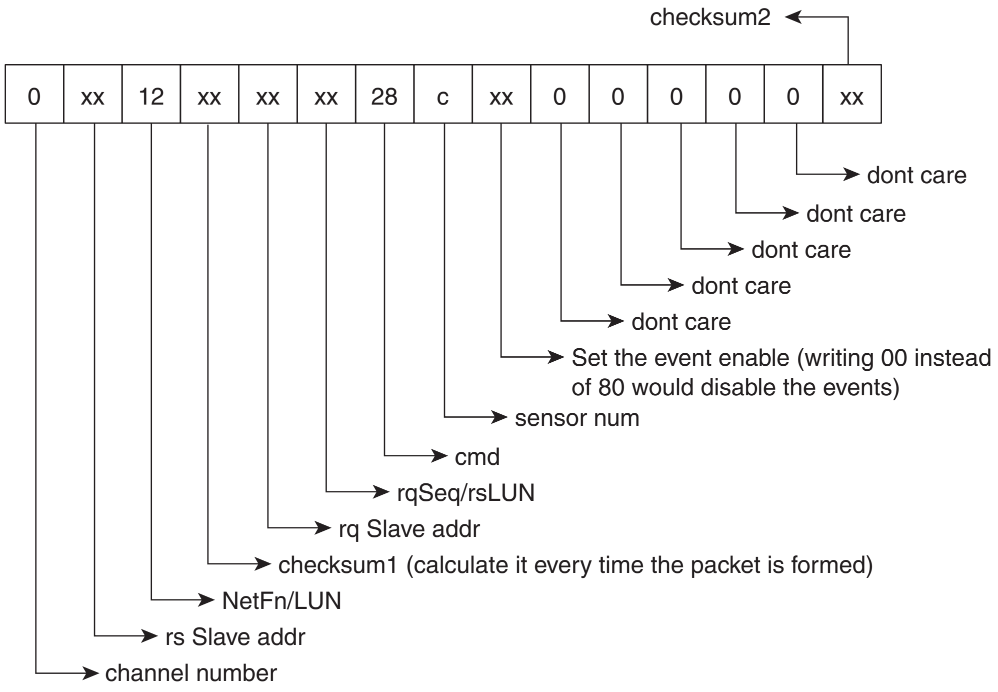
```
93 e 2d 12 20 34 12 ba 0 9 34 execute-smc-cmd
```



Set Sensor Event Enable Command

A typical example of the sensor command is as follows:

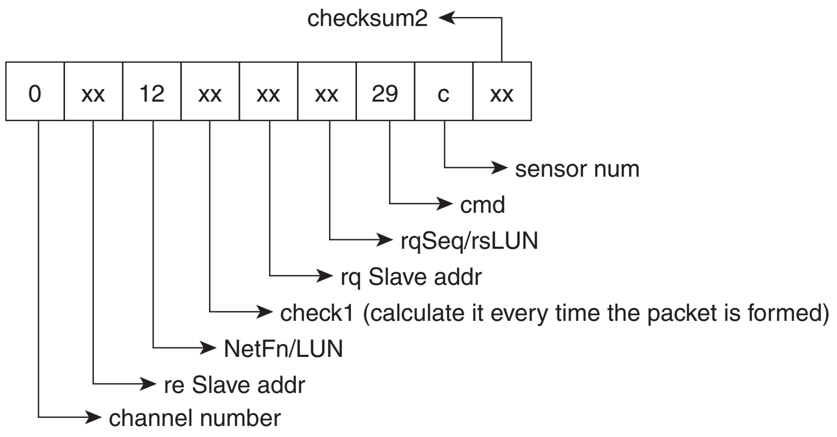
```
24 0 0 0 0 80 2 28 12 20 34 12 ba 0 e 34 execute-smc-cmd
```



Get Sensor Event Enable

A typical example of the sensor command is as follows:

```
a3 2 29 12 20 34 12 ba 0 9 34 execute-smc-cmd
```



Note – The NetFN/LUN for all sensor IPMI commands is 12, which implies that the netFn is 0x04 lun= 0x2.

Environmental Monitoring Application Programming

The following sections describe how to use the environmental monitoring functions in an application program.

For the environmental monitoring application program (`envmond`) to monitor the hardware environment, the following conditions must be met:

- The system controller device driver must be installed.
- The environmental monitoring application program (`envmond`) must be installed and running.

The environmental monitoring parameter values in the application program apply when the system is running at the Solaris level and do not necessarily have to be the same as the corresponding to the parameter settings in the OpenBoot PROM.

To change the environmental monitoring parameter setting at the OpenBoot PROM level, see "OpenBoot PROM Environmental Parameters" on page 37 for the procedure. The OpenBoot PROM environmental monitoring parameter values only apply when the system is running at the OpenBoot PROM level.

Reading Temperature Sensor States Using the PICL API

Temperature sensor states may be read using the `libpicl` API. The following properties are supported in a PICL temperature sensor class node:

TABLE 2-6 PICL Temperature Sensor Class Node Properties

Property	Type	Description
LowPowerOffThreshold	INT	Low threshold for power off
LowWarningThreshold	INT	Low threshold for warning
LowShutdownThreshold	INT	Low threshold for shutdown
HighPowerOffThreshold	INT	High threshold for power off
HighWarningThreshold	INT	High threshold for warning
HighShutdownThreshold	INT	High threshold for shutdown

The PICL plug-in receives these sensor events and updates the State property based on the information extracted from the IPMI message. It then posts a PICL event.

Threshold levels of the PICL node class *temperature sensor* are:

- Warning
- Critical
- Shutdown

To obtain a reading of temperature sensor states, use the `prtpicl -v` command:

```
# prtpicl -c temperature-sensor -v
```

PICL output of temperature sensors on a Netra CT system is shown in CODE EXAMPLE 2-1.

CODE EXAMPLE 2-1 Example Output of PICL Temperature Sensors

```
# prtpicl -c temperature-sensor -v
 CPU-sensor (temperature-sensor, 450000039e)
            :State         ok
            :LowPowerOffThreshold  -20
            :HighWarningThreshold  74
            :HighShutdownThreshold         79
            :HighPowerOffThreshold         91
            :LowWarningThreshold   -10
            :LowShutdownThreshold  -13
            :Temperature            59
            :GeoAddr       0xe
            :Label         Ambient
            :_class        temperature-sensor
            :name          CPU-sensor
```

Solaris Driver Interface

The PICL envmond plug-in opens a SMC driver stream and requests sensor events. The SMC monitors the sensors and generates an event when it detects a change at a particular sensor which meets one of the specified thresholds and generates an event to local Solaris software. This event is captured by the SMC driver (as an IPMI message) and is sent on an open STREAM that has requested sensor events. The sensor events are received by the PICL plug-in. The PICL plug-in updates the State property based on the information it extracts from the IPMI message and posts a PICL event.

Sample Application Program

This section presents a sample environmental monitoring (envmond) application that monitors the CPU diode temperature.

CODE EXAMPLE 2-2 Sample envmond Application Program

```
/*
 * sensor_readwrite.c
 *
 * compile: cc sensor_readwrite.c -lthread -lpicl -o sensor_readwrite
 */
#include <stdio.h>
#include <picl.h>
```

CODE EXAMPLE 2-2 Sample envmond Application Program *(Continued)*

```
#define HI_POWEROFF_THRESHOLD   "HighPowerOffThreshold"
#define HI_SHUTDOWN_THRESHOLD   "HighShutdownThreshold"
#define HI_WARNING_THRESHOLD    "HighWarningThreshold"
#define LO_POWEROFF_THRESHOLD   "LowPowerOffThreshold"
#define LO_SHUTDOWN_THRESHOLD   "LowShutdownThreshold"
#define LO_WARNING_THRESHOLD    "LowWarningThreshold"
#define CURRENT_TEMPERATURE     "Temperature"

static int
get_child_by_name(picl_nodehdl_t nodeh, char *name, picl_nodehdl_t *resulth)
{
        picl_nodehdl_t  childh;
        picl_nodehdl_t  nexth;
        char            propname[PICL_PROPNAMELEN_MAX];
        picl_errno_t    rc;

        /* look up first child node */
        rc = picl_get_propval_by_name(nodeh, PICL_PROP_CHILD, &childh,
                                        sizeof (picl_nodehdl_t));
        if (rc != PICL_SUCCESS) {
                return (rc);
        }

        /* step through child nodes looking for named node */
        while (rc == PICL_SUCCESS) {
                rc = picl_get_propval_by_name(childh, PICL_PROP_NAME,
                                                propname, sizeof (propname));
                if (rc != PICL_SUCCESS) {
                        return (rc);
                }

                if (name && strcmp(propname, name) == 0) {
                        /* yes - got it */
                        *resulth = childh;
                        return (PICL_SUCCESS);
                }

               if (get_child_by_name(childh, name, resulth) == PICL_SUCCESS) {
                        return (PICL_SUCCESS);
                }

                /* get next child node */
                rc = picl_get_propval_by_name(childh, PICL_PROP_PEER,
                                        &nexth, sizeof (picl_nodehdl_t));
                if (rc != PICL_SUCCESS) {
                        return (rc);
                }
```

CODE EXAMPLE 2-2 Sample envmond Application Program *(Continued)*

```
                childh = nexth;
        }
        return (rc);
}

void
get_sensor_thresholds(picl_nodehdl_t nodeh)
{
        int8_t  threshold;

        if (picl_get_propval_by_name(nodeh, HI_POWEROFF_THRESHOLD,
                &threshold, sizeof (threshold)) != PICL_SUCCESS) {
                fprintf(stderr, "Failed to read high power-off threshold.");
        } else
                fprintf(stdout, "High power-off threshold = %d\n", threshold);

        if (picl_get_propval_by_name(nodeh, HI_SHUTDOWN_THRESHOLD,
                &threshold, sizeof (threshold)) != PICL_SUCCESS) {
                fprintf(stderr, "Failed to read high shutdown threshold.");
        } else
                fprintf(stdout, "High shutdown threshold = %d\n", threshold);

        if (picl_get_propval_by_name(nodeh, HI_WARNING_THRESHOLD,
                &threshold, sizeof (threshold)) != PICL_SUCCESS) {
                fprintf(stderr, "Failed to read high warning threshold.");
        } else
                fprintf(stdout, "High warning threshold = %d\n", threshold);

        if (picl_get_propval_by_name(nodeh, LO_POWEROFF_THRESHOLD,
                &threshold, sizeof (threshold)) != PICL_SUCCESS) {
                fprintf(stderr, "Failed to read low power-off threshold.");
        } else
                fprintf(stdout, "Low shutdown threshold = %d\n", threshold);

        if (picl_get_propval_by_name(nodeh, LO_SHUTDOWN_THRESHOLD,
                &threshold, sizeof (threshold)) != PICL_SUCCESS) {
                fprintf(stderr, "Failed to read low shutdown threshold.");
        } else
                fprintf(stdout, "Low shutdown threshold = %d\n", threshold);

        if (picl_get_propval_by_name(nodeh, LO_WARNING_THRESHOLD,
                &threshold, sizeof (threshold)) != PICL_SUCCESS) {
                fprintf(stderr, "Failed to read low warning threshold.");
        } else
                fprintf(stderr, "Low warning threshold = %d\n", threshold);
}
```

CODE EXAMPLE 2-2 Sample envmond Application Program *(Continued)*

```
void
set_sensor_thresholds(picl_nodehdl_t nodeh, char *threshold, int8_t value)
{
        int8_t  new_value = value;

        if (picl_set_propval_by_name(nodeh, threshold, &new_value,
                                sizeof (new_value)) != PICL_SUCCESS)
                fprintf(stderr, "Failed to set *s\n", threshold);
}

int
main(void)
{
        int     warning_temp;
        int8_t  temp;
        char    *sensor = "CPU-sensor";

        picl_nodehdl_t  rooth;
        picl_nodehdl_t  platformh;
        picl_nodehdl_t  childh;

        if (picl_initialize() != PICL_SUCCESS) {
                fprintf(stderr, "Failed to initialise picl\n");
                return (1);
        }
        if (picl_get_root(&rooth) != PICL_SUCCESS) {
                fprintf(stderr, "Failed to get root node\n");
                picl_shutdown();
                return (1);
        }
        if (get_child_by_name(rooth, "platform", &platformh) != PICL_SUCCESS) {
                fprintf(stderr, "Failed to get platform node\n");
                picl_shutdown();
                return (1);
        }

        if (get_child_by_name(platformh, sensor, &childh) != PICL_SUCCESS) {
                fprintf(stderr, "Failed to get %s sensor.", sensor);
                picl_shutdown();
                return (1);
        }

        get_sensor_thresholds(childh);

        /* Read current sensor temperature */
        if (picl_get_propval_by_name(childh, CURRENT_TEMPERATURE,
                &temp, sizeof (temp)) != PICL_SUCCESS) {
```

CODE EXAMPLE 2-2 Sample envmond Application Program *(Continued)*

```
                fprintf(stderr, "Failed to read current temperature\n");
        } else
                fprintf(stdout, "Current temperature = %d\n", temp);

        set_sensor_threshold(childh, HI_WARNING_THRESHOLD, temp+5);

        picl_shutdown();
        return (0);
}
```

Reading the CPU Temperature and OpenBoot PROM Temperature Limits

You can access the CPU temperature sensor current readings and environmental monitoring settings from the Solaris prompt by typing the following commands. Sample output is listed after each command.

`prtpicl` command example:

```
# prtpicl -v -c temperature-sensor
CPU-sensor (temperature-sensor, 36000005ce)
  :State         ok
  :HighWarningThreshold   74
  :HighShutdownThreshold  79
  :HighPowerOffThreshold  91
  :LowWarningThreshold   -10
  :LowShutdownThreshold  -13
  :LowPowerOffThreshold  -20
  :Temperature   53
  :GeoAddr       0xe
  :Label         Ambient
  :_class        temperature-sensor
  :name  CPU-sensor
```

`prtdiag` command example:

```
# prtdiag -v

CPU Node Temperature Information
--------------------------------

Temperature Reading: 53
Critical Threshold Information
------------------------------
High Power-Off Threshold           91
High Shutdown Threshold            79
High Warning Threshold             74
Low Power Off Threshold           -20
Low Shutdown Threshold            -13
Low Warning Threshold             -10
```

eeprom command example:

```
# eeprom | grep temp
shutdown-temperature=91
critical-temperature=79
warning-temperature=74
```

TABLE 2-7 shows which Solaris commands correspond to the environmental monitoring warning that runs when the CPU temperature exceeds the set limit.

TABLE 2-7 Description of Values Displayed by Solaris Commands

Environmental Monitoring Warning	`prtpicl`	`prtdiag`	`eeprom`
The first-level temperature warning is displayed.	`HighWarning Threshold`	`High Warning Threshold`	`warning-temperature`
The second-level temperature warning is displayed.	`HighShutdown Threshold`	`High Shutdown Threshold`	`critical-temperature`
The CPU shutdown message is displayed and the CPU is shut off.	`HighPowerOff Threshold`	`High Power-Off Threshold`	`shutdown-temperature`

CHAPTER 3

User Flash

This chapter describes the user flash driver for the onboard flash PROM and how to use it. The Netra CP2300 is equipped with user flash memory. This chapter includes the following sections:

User Flash Usage and Implementation

The customer can use the flash memory for various purposes such as storage for RTOS, user data storage, OpenBoot PROM information or to store *dropins*. Dropins simplify customizing a system for the user.

When OpenBoot PROM in system flash is corrupted, and if a backup copy of OpenBoot PROM is stored in user flash, you can switch the SMC switch to boot the OpenBoot PROM from the user flash and then use flash update to get a good OpenBoot PROM image back into the system flash.

The user flash includes a flash PROM chip that can be programmed. The Netra CP2300 has an 8MB flash that is logically divided into two parts: 1MB for the system/boot flash and 7MB for the user flash. The physical address for the flash is 1ff.f000.0000.

System Compatibility

The following releases support the user flash driver:

- Solaris 8 2/02 operating environment or other compatible versions that support this feature
- Netra CP2300 OpenBoot PROM
  - Firmware version 1.0.1
  - Firmware CORE Release 1.0.3
  - Release 4.0 Version 16
  - SMCFW FLASH Code Version 4.0.12
  - SMCFW BOOT Code Version 4.15.1
  - PLD Revision 1.1
  - CORE 1.0.3
  - All the above versions or other compatible versions that support this feature

User Flash Driver

The *uflash* is the device driver for the flash PROM device on the Netra CP2300. Access to the driver is carried out through `open`, `read`, `write`, `pread`, `pwrite` and `ioctl` system interfaces.

On the Netra CP2300, one of these devices is supported. There is one logical device file for each physical device that can be accessed from applications. Users can use these devices for storing applications and data.

An instance of the driver is loaded for the device. The driver blocks any reads to the device while a write is in progress. Multiple, concurrent reads can go through to the same device at the same time. Writes to a device occur one at a time. All read and write operations are supported at this time. Access to the device normally happens a byte at a time.

The device also 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 also 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 user flash modules on the Netra CP2300 are write enabled by default. The user flash is detected during OpenBoot PROM boot by default.

OpenBoot PROM Device Tree and Properties

This section provides information on the user flash OpenBoot PROM device node and its properties.

User flash OpenBoot PROM device node is:

```
/pci@1f,0/pci@1,1/isa@7/flashprom@1f,100000
```

See TABLE 3-1 for the user flash node properties.

TABLE 3-1 User Flash Node Properties

Property	**Description/Value**
dcode-offset	00000002
blocks-per-bank	00000038
block-size	00020000
model	SUNW,yyy-yyyy
compatible	nct-uflash
reg	0000001f 00100000 00700000

User Flash Device Files

The user flash device files are as follows:

- `/dev/uflash0`

Interface (Header) File

The user flash header file is located in the following path:

`/usr/`*platform*`/SUNW,Netra-CP2300/include/sys/uflash_if.h`

Application Programming Interface

Access to the user flash device from the Solaris operating environment is through an application or user C program. No command-line tool is available. User programs open this device file and then issue `read`, `write`, or `ioctl` commands to use the user flash device.

The system calls are listed below in TABLE 3-2.

TABLE 3-2 System Calls

Call	Description
read(), pread()	reads device
pwrite()	writes device
ioctl()	erases device, queries device parameters

The `ioctl` supported commands are listed below:

```
#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 */
```

Note that these `ioctl` commands are *not* supported:

```
#define UIOCMLCK (uflashIOC|3)      /* master lock */
#define UIOCEALL (uflashIOC|6)      /* erase all unlocked blocks */
#define UIOCEFUL (uflashIOC|7)      /* erase full chip */
```

Structures to Use in IOCTL Arguments

PROM Information Structure

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

CODE EXAMPLE 3-1 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;
```

User Flash User Interface Structure

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

CODE EXAMPLE 3-2 User Flash 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;
```

Errors

EINVAL	Application passed one or more incorrect arguments to the system call.
EACCESS	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 may be necessary.
ENXIO	This error indicates problems with the driver state. Power cycle of the system or reinstallation of driver may be necessary.
EFAULT	An error was encountered when copying arguments between the application and driver (kernel) space.
ENOMEM	System was low on memory when the driver attempted to acquire it.

Example Programs

Example programs are provided in this section for the following actions on user flash device:

- Read
- Write
- Erase
- Block Erase

Read Example Program

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

CODE EXAMPLE 3-3 Read Action on User Flash Device

```
/*
* uflash_read.c
 * An example that shows how to read user flash
 */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

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

```
#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);
  }
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() {
```

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

```
  int ret;
module = argv[0];
ret = uflash_init();
if (!ret)
    uflash_read();
uflash_uninit();
}
```

Write Example Program

CODE EXAMPLE 3-4 contains the Write Action on the user flash device.

CODE EXAMPLE 3-4 Write Action on User Flash Device

```
/*
 * uflash_write.c
 * An example that shows how to write user 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): ");
```

CODE EXAMPLE 3-4 Write Action on User Flash Device *(Continued)*

```
    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_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();
}
```

Erase Example Program

CODE EXAMPLE 3-5 contains the Erase Action on the user flash device.

CODE EXAMPLE 3-5 Erase Action on User Flash Device

```
/*
 * uflash_erase.c
 * An example that shows how to erase user 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 (ufd0 && 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
uflash_erase() {
```

CODE EXAMPLE 3-5 Erase Action on User Flash Device *(Continued)*

```
  if (ufd0 && ioctl(ufd0, UIOCEFUL, &ufif0) == -1 ) {
    perror("ioctl(ufd0, UIOCEFUL): ");
    return(-1);
  }
  printf("\nerase successful on %s\n", uflash0);
return(0);
}
main() {
  int ret;
module = argv[0];
ret = uflash_init();
if (!ret)
    uflash_erase();
uflash_uninit();
}
```

Block Erase Example Program

CODE EXAMPLE 3-6 contains the Block Erase Action on the user flash device.

CODE EXAMPLE 3-6 Block Erase Action on User Flash Device

```
/*
 * uflash_blockerase.c
 * An example that shows how to erase block(s) of user 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 */
```

CODE EXAMPLE 3-6 Block Erase Action on User Flash Device *(Continued)*

```
  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
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();
}
```

Sample User Flash Application Program

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

CODE EXAMPLE 3-7 Sample User Flash Application Program

```
/*
 *
 *        This application program demonstrates the user program
 *        interface to the User 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,
 *         -) master lock,
 *         -) erase block, erase all unlocked blocks, and
 *            erase whole PROM
 *        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 3-7 Sample User 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 -- master   lock",
       "6 -- clear    all locks",
       "7 -- erase    blocks",
       "8 -- erase    all unlocked blocks",
       "9 -- erase    whole PROM",
       "a -- switch   PROMs",
       "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);
```

CODE EXAMPLE 3-7 Sample User Flash Application Program *(Continued)*

```
        /*
         * Open the user flash PROM.
        */
        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 ’9’:       /* erase the whole flash PROM */
               fprintf(stderr,
                             "Are you sure?[y/n]");
                              scanf ("%c", &c);

               if (c != ’y’)
                   continue;
               if (ioctl(fd, UIOCEFUL, &uflash_if) == -1)
                                   goto getout;
               break;

           case ’8’:       /* erase all unlocked flash PROM blocks */
               /*
                * This ioctl is valid only for those
                * chips that have query command.
```

CODE EXAMPLE 3-7 Sample User Flash Application Program *(Continued)*

```
                 */
                 if (ioctl(fd, UIOCEALL, &uflash_if) == -1)
                              goto getout;
                break;

            case '7':       /* erase flash PROM block */
                fprintf(stderr,
                                  "Enter PROM block number[0, 31]> ");
                          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 '6':       /* clear all locks */
                /* on certain PROMs */
                if (ioctl(fd, UIOCCLCK, &uflash_if) == -1)
                                  goto getout;
                break;

            case '5':       /* master lock */
                /* on certain PROMs */
                if (ioctl(fd, UIOCMLCK, &uflash_if) == -1)
                                  goto getout;
                break;

            case '4':       /* lock flash PROM block */
                /* on certain PROMs */
                fprintf(stderr,
                                  "Enter PROM block number[0, 31]> ");
                          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,
```

CODE EXAMPLE 3-7 Sample User Flash Application Program *(Continued)*

```
                            "Enter PROM block number[0, 31]> ");
                    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] & 0xF);
          }
          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;

       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++) {
```

CODE EXAMPLE 3-7 Sample User Flash Application Program *(Continued)*

```
                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;
                      }

            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;
```

CODE EXAMPLE 3-7 Sample User Flash Application Program *(Continued)*

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

CHAPTER 4

Programming the User LED

This chapter describes how to use the Alarm/User LED. The Alarm/User LED is located on the front panel of the Netra CP2300. The bi-colored LED is red and green in color (see FIGURE 4-1 for the location of the Alarm/User LED on the board front panel).

In order to use the LED function, a SPARC V9 64-bit C library and the `led.h` file are required. The library and the file are available in the `SUNWcp23u` package. The Application Programming Interface (API) for the user is documented in the `led.h` file. See "Files and Packages Required to Support the Alarm/User LED" on page 79 for more information.

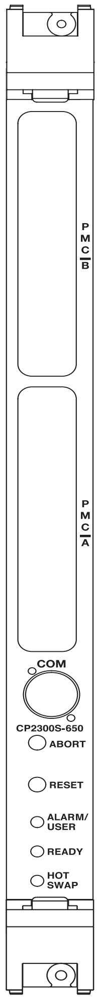


FIGURE 4-1 Illustration of a Typical Netra CP2300 cPSB Board Front Panel Showing the Alarm/User LED

Files and Packages Required to Support the Alarm/User LED

To use the Alarm/User LED feature, the user should update the firmware with the appropriate firmware version that supports this feature on the Netra board.

Note – To check the current firmware version and for instructions on how to update the firmware, refer to the technical reference manual of the Netra board that you are using.

The list of packages that are required are as follows:

- `SUNWcp23u:` SPARC V9 64-bit C library `libcp2300.so.1` available at:

  `/usr/platform/${PLATFORM}/lib`

- `SUNWcp23u:` LED `include` file available at:

  `/usr/platform/${PLATFORM}/include/sys/`

Ensure that the following driver is also there, as needed:

- `SUNWcp23x.u`: 64-bit sc_nct driver available at:

  `/platform/${PLATFORM}/kernel/drv/sparcv9/sc_nct`

A typical example of ${PLATFORM} is `SUNW,Netra-CP2300` for the Netra CP2300 board. An example for the library directory is:

```
/usr/platform/SUNW,Netra-CP2300 /lib
```

Applications

This section provides the application programming interface (API) to control the command combination of the Alarm/User LED, and instructions on how to compile and link the information.

Note – Since the LED interface installs and then removes the sc_nct streams module, an error can occur when multiple applications attempt to use this interface at the same time. If the user desires more than one application to use this interface, application software should incorporate a synchronization method such that only one access to the interface exists at any time.

Application Programming Interface (API)

CODE EXAMPLE 4-1 Application Programming Interface for the Netra CP2300 Board

```
extern   int   led(int led, int cmd);

/* LEDS */
#define BLUE_LED           0x0
#define HOTSWAP_LED        BLUE_LED
#define PLD_GREEN_LED      0x04
#define GREEN_LED          PLD_GREEN_LED
#define AMBER_LED          0x08

/* LED COMMANDS */
#define LED_OFF         0x00
#define LED_ON          0x01
#define LED_BLINK_SLOW  0x02
#define LED_BLINK_FAST  0x03

/* ERROR CODES */
#define ESEQUENCE       200  /* portnum mismatch */
#define ECMDCOMP        201  /* non-zero command completion */
#define ECMDCODE        202  /* smc command mismatch */
```

The supported LED and command combinations are shown in TABLE 4-1.

TABLE 4-1 Supported LED and Command Combinations for the Netra CP2300 Board

Color of LED	LED_OFF	LED_ON	LED_BLINK_SLOW	LED_BLINK_FAST
BLUE_LED	Yes	Yes	No	No
GREEN_LED	Yes	Yes	Yes	Yes
AMBER_LED	Yes	Yes	No	No

Compile

As you compile your application, you need to use the compiler command (`cc`) flag **`-I`**, to include the `sys/led.h` file named in "Files and Packages Required to Support the Alarm/User LED" on page 79. Specify 64-bit binaries by setting the `-xarch=v9` and `-D__sparcv9` compiler flags.

For example:

```
-xCC -xarch=v9 -D__sparcv9 -I/usr/platform/SUNW,Netra-CP2300/include/
```

Note – Type the above command all on one line.

Link

To create a link to the library named (`libcp2300.so.1`) listed in "Files and Packages Required to Support the Alarm/User LED" on page 79, use the linker flag **`-L`** command.

For example:

```
-L /usr/platform/SUNW,Netra-CP2300/lib
```

Sample Application Program

This section presents a sample test.c application to turn the LED on, off, and blink.

CODE EXAMPLE 4-2 Sample LED Application Program

```
#include       <stdio.h>
#include       <sys/led.h>

main()
{
      /* blue on, rest off */
      printf("\n\nTesting Blue led ON, rest off\n");
      fflush(stdout);
      printf("BLUE_LED on returned %d\n", led(BLUE_LED, LED_ON));
      fflush(stdout);
      sleep(4);
      printf("GREEN_LED off returned %d\n", led(GREEN_LED, LED_OFF));
      fflush(stdout);
      sleep(4);
      printf("AMBER_LED off returned %d\n", led(AMBER_LED, LED_OFF));
      fflush(stdout);
      sleep(4);

      /* all lights on, and green blinking fast */
      printf("\n\nTesting all led's on and green blinking fast\n");
      fflush(std out);
      printf("BLUE_LED on returned %d\n", led(BLUE_LED, LED_ON));
      fflush(stdout);
      sleep(4);
      printf("AMBER_LED on returned %d\n", led(AMBER_LED, LED_ON));
      fflush(stdout);
      sleep(4);
      printf("GREEN_LED blink returned %d\n", led(GREEN_LED, LED_BLINK_FAST));
      fflush(stdout);
      sleep(4);
}

cc -xCC -xarch=v9 -D__sparcv9 \
        -I /usr/platform/SUNW,Netra-CP2300/include \
        -L /usr/platform/SUNW,Netra-CP2300/lib
        -l cp2300 \
        -o test \
        test.c
```

Index