bc635PCI
Solaris 5-8
Developer's Kit

8550-0007

User's Guide

Rev A

(April 2002)

bc635PCI
SOLARIS DEVELOPER'S KIT

TABLE OF CONTENTS

SECTION **PAGE**

CHAPTER ONE
INTRODUCTION

1.0 General 1-1
1.1 Features 1-1
1.2 Overview 1-1

CHAPTER TWO
INSTALLATION

2.0 Hardware Installation 2-1
2.1 Software Installation 2-1
2.2 Driver Compilation 2-1

CHAPTER THREE
DRIVER LIBRARY DEFINITIONS

3.0 General 3-1
3.1 Functions 3-1

CHAPTER FOUR
EXAMPLE PROGRAMS

4.0 General 4-1
4.1 Program Functions 4-1
4.2 Example 1: GPS Packet 46 – Health Packet Sample 4-1
4.3 Example 2: 1PPS Interrupt Sample 4-1

CHAPTER ONE
INTRODUCTION

1.0 GENERAL

The bc635PCI Developer's Kit is designed to provide a suite of tools useful in the development of applications which access features of the bc635PCI Time and Frequency Processor. This kit has been designed to provide an interface between the bc635PCI and applications developed for Solaris 5, 6, 7, and 8 environments. In addition to the interface driver library, an example program is provided, complete with source code, in order to provide a better understanding of the kit features and benefits.

1.1 FEATURES

The salient features of the Developer's Kit include:

- Driver interface library with access to all features of the bc635PCI.
- Example programs, with source, utilizing the interface library.
- User's Guide providing a library definition.

1.2 OVERVIEW

The Developer's Kit was designed to provide an interface to the bc635PCI Time and Frequency Processor in the 32/64 bit environments of Solaris 5, 6, 7, and 8. The example program provides sample code, which exercises the interface library as well as examples of converting many of the ASCII format data objects, passed to and from the device into a binary format suitable for operation and conversion. The example programs were developed using discrete functions for each operation, which allows the developer to clip any useful code and use it in their own applications.

This Page Intentionally Left Blank.

CHAPTER TWO
INSTALLATION

2.0 HARDWARE INSTALLATION

Installation of boards is quite a bit simpler than in most bus architectures due to two factors: geographical addressing, which eliminates the need for DIP switches and jumpers normally required to select a 'base address' or interrupt level for plug-in modules; and auto configuration, which allows the host computer to read the device ID and other configuration information directly from the Open Boot PROM located on the card itself so that the host can select the appropriate device driver automatically. The only thing the user has to do is pick a vacant slot and plug the bc635PCI into it and installs the device driver. Be sure to consult the user documentation that came with your particular workstation for any specific card installation instructions.

2.1 SOFTWARE INSTALLATION

The loadable device driver provides a simple interface to the bc635PCI Time and Frequency Processor (TFP) module. Two versions of the driver are available: compatible with Solaris 2.5.1, Solaris 2.6 32bit operating systems and SunOS 7/8 64bit operation systems.
To install the driver, execute the eisinstall script found with this README file. Alternatively, untar the package, which is the accompanying tar file EISbc635.tar and use the pkgadd command to install the resulting package. Both the eisinstall script and the pkgadd command must be installed as root. Reboot the computer after installing the driver in order to load the driver cleanly.
This driver should work correctly in its current binary form with the need for recompilation.

The install script is shown below:

```
#!/bin/sh

# Install easy script for EIS electronic delivered packages.
# Copyright (C)1997, 2000, EIS Computers, Datum Inc

#
# Run this simple script from the same directory as the EISbc635.tar file
# you have received

PATH=$PATH:/usr/bin:/usr/sbin
export PATH

EPKG=EISbc635

tar xf EISbc635.tar

pkgadd -d . $EPKG
```

2.2 DRIVER COMPILATION

The TFP module can generate interrupts, though they must be enabled by the user with an ioctl request.

Compiling 32bit driver:
The following command lines will compile the stfp.c driver source code.
venus# cc -D_ KERNEL -c stfp.c
venus# ld -r -o stfp stfp.o

Copy stfp and stfp.conf files into the /usr/ kernel/ drv directory (note that the stfp.conf file is necessary only if you want to use interrupt level different from the default). The driver is ready to be installed using following add_ drv(1M) function.

venus# add_ drv -m '* 0666 root sys' stfp
Use modunload(1M) to unload the driver from the system. Use modstat(1M) to determine the module-id.

venus# modunload -i module-id

Compiling 64bit driver:
The following command lines will compile the stfp.c driver source code.
venus# cc -D_ KERNEL –xarch=v9 –xcode=abs32 –xregs=no%appl –XO3 –c stfp.c
Copy stfp file into the /kernel/drv/sparcv9 directory

CHAPTER THREE
DRIVER LIBRARY DEFINITIONS

3.0 GENERAL

The interface library provides functions for each of the programming packets supported by the bc635PCI Time and Frequency Processor. In addition, functions are provided to both read and write individual registers and dual port RAM locations on the card. To understand the usage and effects of each of these functions, please refer to the User's Guides provided with the hardware.

3.1 FUNCTIONS

The 'STFP' device driver supports the bc635PCI Time and Frequency Processor (TFP) module, as well as the GPS version, the bc637PCI. The TFP supports time code decoding, synchronization to an external 1pps (Pulse Per Second) signal, a free running mode, a real time clock mode, and the GPS Satellite System. Several timing outputs, all synchronous with the timing source, are provided, including an IRIG B time code signal, a 1pps, programmable periodic, a time coincidence strobe, and a 1, 5, or 10 MHz clock.

The open (2), close (2), read (2), write (2), and ioctl (2) system calls are supported. Most TFP functions, including the reading of the time, are accesses through the ioctl (2) call.

Read/ Write Calls
The only purpose for the read (2) call is to read a GPS data packet that was previously requested with an ioctl (2) or write (2) call. These packets contain position, velocity, GPS system status, and other GPS information. One GPS packet is read for each read (2) call. The maximum GPS packet size is defined by STFP_ MAX_ READ found in 'stfpio. h'. Refer to the GPS documentation for GPS packet details.
The packet data contains floating-point types as well as various integer types, but these elements cannot be directly accessed when read into a char buffer because they are not properly aligned in memory. To obtain access to the various types of GPS data elements, union structures are generally used. For example, to extract a 4-byte float from the packet data, use the union shown below. Copy four consecutive bytes of packet data into the fconv. uc[] array, starting with fconv. uc[0] (since Sun workstations are big-endian machines,) then access the float data as fconv. f.

```
union {
        float f;
        u_ char uc[ 4];
} fconv;
```

Following a successful read(2) call, the read buffer will contain the packet length, ID, and data bytes of the requested GPS data packet as described in the GPS documentation section Communicating with the
GPS Receiver. Note that a successful read(2) call will return the number of bytes read which will equal the packet length plus 1 (one for the packet length byte itself.)

The write(2) call allows the user to send commands to the TFP. The TFP commands are used to set the timing mode, time code format, and other TFP functions. Refer to the bc635PCI User's Manual for TFP command details. The write buffer must contain the TFP command ID and zero or more command data bytes. As with GPS packets, command data consists of various data types that must be converted to a char array for the write(2) call. The maximum number of bytes used for a command is defined by STFP_ MAX_ WRITE found in "stfpio. h". Most commands are implemented with ioctl(2) calls, which are much simpler to use since they provide the conversion of data to an array of chars as required. Since most TFP commands can be executed with ioctl(2) calls, the only really useful function for the write(2) call is to execute the TFP commands that write data packets to the GPS receiver. In fact, the write(2) call is the only way to send GPS data packets to the GPS receiver. When write(2) is used to execute the Manually Request Packet from GPS Receiver command (command 0x32 described in the GPS documentation) and a response is expected (non-zero response packet ID), the write(2) call puts the calling process to sleep until the response arrives. The driver will not call sleep() if the user has directed the driver to send a signal on the occurrence of the INT_ PACKET (GPS packet available) interrupt. The response packet can take 10's or 100's of milliseconds to arrive. The read(2) call can then be used to read the response packet.

Ioctl Calls ioctl (fd, request[, arg])

The ioctl(2) request codes, as well as all the other defined constants listed below, are contained in 'stfpio. h'. For most ioctl() functions, arg is a pointer to data either used by or returned by the function.

Other functions either ignore arg or use it directly as an int value. In many functions, most of which have request labels of the form SELXXX or CONTROLXXX, the int value selects some option from a list of options defined in 'stfpio'.

Following each request code below is the arg type expected by the driver.

SELTIMINGMODE, int

Selects the TFP timing mode specified in the int arg.

SELTIMEFORMAT, int

Selects between the decimal and binary time formats. The decimal time format is characterized by the TFP_ time structure. The binary time format is characterized by the TFP_ timeval structure. These structures are declared in 'stfpio. h'.

TIMEREQUEST, int

EVENTREQUEST, int

The driver writes to the TIMEREQ or EVENTREQ register (the int value is ignored) which causes time to be captured and held in the TIMEx or EVENTx registers. No time data is transferred.

RDTIME, *struct stfp_time

RDEVENT, *struct stfp_time

Reads time from the TFP TIMEx or EVENTx registers assuming the time format is decimal. Time is not captured with these requests.

RDTIMETV, *struct stfp_timeval
RDEVENTTV, *struct stfp_timeval
Reads time from the TFP TIMEx or EVENTx registers assuming the time format is binary. Time is not captured with these requests.

RDTIMEREQ, *struct stfp_time
RDEVENTREQ, *struct stfp_time
These requests capture and read time from the TFP TIMEx or EVENTx registers assuming the time format is decimal.

RDTIMETVREQ, *struct stfp_timeval
RDEVENTTVREQ, *struct stfp_timeval
These requests capture and read time from the TFP TIMEx or EVENTx registers assuming the time format is binary.

WRSTROBE, * struct stfp_time
Writes time to the STROBEx registers assuming the time format is decimal. This request disables the Strobe output while the STROBEx registers are written.

WRSTROBETV, *struct stfp_timeval
Writes time to the STROBEx registers assuming the time format is binary. This request disables the Strobe output while the STROBEx registers are written.

SELTCFORMAT, int
Selects the time code input format.

SELTCMOD, int
Selects the time code input modulation type.

SETTIME, int
Manually sets the TFP major time assuming the time format is binary. The minor time is not affected.

SETDECTIME, struct stfp_dec_tm
Manually sets the TFP major time assuming the time format is decimal. The minor time is not affected

SETYEAR, int
Manually sets the TFP year.

SETPERIODIC, *struct periodic

This request sets the Programmable Periodic output frequency and enables the 1pps synchronous mode.

SETTIMINGOFFSET, int
Sets the TFP timing offset with the int arg value.

SELFREQUENCYOUT, int
Selects output frequency (1, 5, or 10 MHz).

CONTROLEVENT, int
This request performs a variety of functions relevant to the Event Time Capture feature.

CONTROLSTROBE, int
This request performs a variety of functions relevant to the Time Coincidence Strobe feature.

CAPUNLOCK, int
This request writes to the TFP UNLOCK register to release the Event Capture Lockout feature (if enabled via CONTROLEVENT).

SETINTSIGNAL, int
Setup one or more interrupt sources to generate a signal (SIGUSR1) to the process making this ioctl(2) call. The int arg is comprised of one or more interrupt source bits (defined in 'stfpio. h') OR'ed together. The following ioctl(2) call would cause the driver to send a signal on the occurrence of the Event Input and/ or Strobe Output interrupt.
The signal handler can use the RDINTSIGNAL request to find out which interrupt source(s) caused the signal. An arg value of 0 will disable signals.
ioctl (fd, SETINTSIGNAL, INT_ EVENT | INT_ STROBE);

RDINTSIGNAL, *int
Use this request to find out which interrupt source(s) generated the last signal.
Use the SETINTSIGNAL request to enable signals. The driver automatically clears the INTSTAT bits during its interrupt service routine.

RDINTSTAT, *int
CLRINTSTAT, int
These requests allow the user to read and clear bits in the TFP INTSTAT register. All INTSTAT bits can be read, but only those bits that are not setup to generate a signal can be cleared. Use these requests to poll for the occurrence of one or more interrupt source(s) instead of using signals.

CONTROLTIMEBASE, int
This request performs a variety of time base control functions, such as oscillator disciplining and jam-sync control, clock selection, etc.

SETDAC, int

Loads the TFP D/ A Converter with the int arg value.

RDDAC, *int
Reads the TFP D/ A Converter value.

SETDISCGAIN, int
Loads the TFP discipline gain.

REQGPSPACKET, int
This request is for users of the bc637PCI, the GPS version of the bc635PCI. The int arg contains one of the GPS packet ID's supported with the Retrieve Packet from GPS Receiver command (command 0x31). The TFP monitors and stores several commonly requested packets that the GPS receiver broadcasts periodically to the TFP. These packets are available to be read immediately. GPS packets that are not monitored by the TFP are requested from the GPS receiver by the TFP. Since this task can take 10's or 100's of milliseconds, the driver puts the calling process to sleep until the GPS packet becomes available. The driver will not call sleep() if the user has directed the driver to send a signal on the occurrence of the INT_ PACKET (GPS packet available) interrupt. The requested packet is read using the read(2) call.

GETDATA, struct getdata_t
Gets various data packets from the board. See ‘stfpio.h’ for the list of commands you can request using this command.

SOFTWARERESET, int
Issues a software reset on the card.

SETTCOUTFMT, int
Sets the time code output format

SETGENTMOFFSET, struct tcgenoffset
Sets the generator time code offset

SETLOCTMOFFSET, struct loctmoffset
Sets the local time offset

SETLEAPSECEVENT, struct leapseconds
This command can be used in modes other than GPS mode for inserting or deletion of one leap second.

SETCLKVAL, int
This command advance/retard the TFP internal clock. The TFP can adjust its clock up to 100 milliseconds per each second. Each count is equal to 10 microseconds.

SETGPSTMFMT, int

Modify the time base in GPS mode. This command determines whether the board will correct the received GPS time for leap second offset and events

SETGPSMDFLG, int
By default, the TFP directs the GPS receiver to Static Mode of Operation after the TFP is tracking to GPS. This Command allows the user to disable this feature. See Packet 2C in the GPS Manual for detail description on this feature.
This function should only be used when the TFP is in GPS Mode of Operation.

SETLOCTMFLG, int
Enables or disables the local time offset

SETYRINCFLG, int
This commands the TFP to enable or disable the auto incrementing of the Year at the beginning of each year. The Year variable is stored into the EEPROM for reference.

SYNCRTC,
This command forces the TFP to Synchronize the RTC time to current time.

DISCBATT,
This command disconnects the RTC IC from the Battery after power is turned off. Upon power on, the TFP automatically connects the RTC IC to the battery.

RDCONTROL, *int
Reads the Control register

SETCONTROL, int
Sets the Control register

CHAPTER FOUR
EXAMPLE PROGRAM

4.0 GENERAL

The bc635pci.c is an example program that provides sample code, which exercises the interface driver library as well as an example of converting many of the ASCII format data objects passed to and from the device into a binary format suitable for operation and conversion. The example program was developed using discrete functions for each operation, which allows the developer to clip any useful code and use it in their own applications.

4.1 PROGRAM FUNCTIONS

Function	open
Description	Opens an instance of the device driver

Example	int fd;
	if ((fd = open ("/dev/stfp0", O_RDWR)) < 0)
	{
	printf("Error opening Device Driver ..... Exiting");
	_exit(1);
	}

Function	close
Description	Closes the device driver
Example	close(fd);

Registers	pci_read_time
Description	Reads the time in binary or decimal time format
Example	struct stfp_time stm; struct stfp_timeval tvTime; /* Decimal Time Format*/ ioctl (fd, RDTIMEREQ, &stm); printf (" Julian Time: %03d %d %02d:%02d:%02d.%06d Status: %x\n", stm.tm.tm_yday+1, stm.tm.tm_year+1900, stm.tm.tm_hour, stm.tm.tm_min, stm.tm.tm_sec, stm.usec, stm.status); /* Binary Time Format */ ioctl (fd, RDTIMETVREQ, &tvTime); printf ("Binary Time: %lu.%06ld status: %x ", tvTime.tv.tv_sec, tvTime.tv.tv_usec, tvTime.status);

	printf ("%s", ctime (&tvTime.tv.tv_sec));

Registers	pci_read_event_time
Description	Reads the event time in binary time format
Example	ioctl (fd, RDEVENTTV, &tvTime);

Registers	pci_set_strobe
Description	Sets a strobe mode and time
Example	ioctl (fd, CONTROLSTROBE, STROBE_SECUS); ioctl (fd, CONTROLSTROBE, STROBE_USONLY);

Registers	pci_set_control
Description	Reads and sets the control register
Example	ioctl(fd, RDCONTROL, &ctlreg); ioctl(fd, SETCONTROL, ctlreg);

Command 0x10	pci_mode
Description	Sets the timing mode
Example	ioctl (fd, SELTIMINGMODE, MODE_TIMECODE);

Command 0x15, 0x16	pci_time_code
Description	Sets the time code input format and modulation
Example	ioctl (fd, SELTCFORMAT, TC_IRIGB); ioctl (fd, SELTCMOD, MOD_AM);

Registers	pci_out_freq
Description	Sets the frequency output 1, 5, 10 MHz
Example	ioctl (fd, SELFREQUENCYOUT, FREQ_10MHZ);

Command 0x11	pci_time_format
Description	Sets the time format, binary or decimal
Example	ioctl (fd, SELTIMEFORMAT, TIME_BINARY);

Command 0x14	pci_heartbeat
Description	Sets the heartbeat mode and frequency
Example	ioctl (fd, SETPERIODIC, &sper);

Command 0x13	pci_set_year
Description	Sets the TFP year
Example	ioctl (fd, SETYEAR, year);

Command 0x12	pci_set_time
Description	Sets the TFP time in either binary or decimal format
Example	ioctl (fd, SETDECTIME, dec); ioctl (fd, SETTIME, tm_sec);

Command 0x17	pci_set_prop_delay
Description	Set the time code propagation delay
Example	ioctl (fd, SETTIMINGOFFSET, prop_delay);

Command 0x1A	pci_sw_rest
Description	Issues a software reset on the TFP
Example	ioctl (fd, SOFTWARERESET, 1);

Command 0x1B	pci_tc_out_format
Description	Sets the time code output format
Example	ioctl (fd, SETTCOUTFMT, TC_IRIGB);

Command 0x20	pci_set_clock_src
Description	Sets the clock source, internal or external
Example	ioctl (fd, CONTROLTIMEBASE, CLOCK_INTERNAL);

Command 0x1C	pci_set_gen_off
Description	Sets the generator time offset
Example	ioctl (fd, SETGENTMOFFSET, &gen);

Command 0x1D	pci_set_loc_off
Description	Sets the local time offset
Example	ioctl (fd, SETLOCTMOFFSET, &loc);

Command 0x1E	pci_set_leap_sec
Description	Program the leap seconds into the TFP
Example	ioctl (fd, SETLEAPSECEVENT, &leap);

Command 0x21	pci_ctl_jam_sync
Description	Enable or disable the jam sync control
Example	ioctl (fd, CONTROLTIMEBASE, JAMSYNC_DISABLE);

Command 0x22	pci_frc_jam_sync
Description	Forces a jam sync on the TFP
Example	ioctl (fd, CONTROLTIMEBASE, FORCE_JAMSYNC);

Command 0x24	pci_set_da_con
Description	Loads the D/A converter
Example	ioctl (fd, SETDAC, da_con);

Command 0x25	pci_set_gain
Description	Sets the disciplining gain
Example	ioctl (fd, SETDISCGAIN, dis_gain);

Command 0x27	pci_sync_rtc
Description	Sync's the RTC to current time
Example	ioctl (fd, SYNCRTC);

Command 0x28	pci_dis_rtc
Description	Disconnect battery from RTC
Example	ioctl (fd, DISCBATT);

Command 0x29	pci_set_clk_val
Description	Sets the clock value of the TFP
Example	ioctl (fd, SETCLKVAL, clk_val);

Command 0x33	pci_set_gps_tm_fmt
Description	Sets GPS or UTC time format
Example	ioctl (fd, SETGPSTMFMT, UTC_FMT);

Command 0x34	pci_set_gps_mode_flg
Description	Enable or disable the GPS mode flag
Example	ioctl (fd, SETGPSTMFMT, GPS_FLG_ENA);

Command 0x40	pci_set_local_off_flg
Description	Enable or disable the local time offset
Example	ioctl (fd, SETLOCTMFLG, LOC_OFF_DIS);

Command 0x42	pci_set_yr_auto_inc_flg
Description	Enable or disable the year automatic increment
Example	ioctl (fd, SETYRINCFLG, YR_INC_DIS);

Command 0x19	pci_req_time_settings
Description	Request time setttings

Example	/* Get Timing Mode */
	get.arg = GETDATA_MODE;
	if (!((ioctl (fd, GETDATA, &get)) < 0))
	mode = (int)get.data.tmode;

Command 0x19	pci_req_clock_settings
Description	Requests clock settings

Example	/* Get Clock Source */
	get.arg = GETDATA_CLKSRC;
	if (!((ioctl (fd, GETDATA, &get)) < 0))
	clk_scr = (u_char)get.data.clksrc;

Command 0x19	pci_req_offsets_settings
Description	Requests offsets settings

Example	/* Get Local Time Offset */
	get.arg = GETDATA_LOCTMOFF;
	if (!((ioctl (fd, GETDATA, &get)) < 0))
	{
	loc_off = (float)get.data.locoff.locoff;
	loc_off_flg = (int)get.data.locoff.locflg;
	}

Command 0x19	pci_req_utc_info
Description	Request UTC Information
Example	/* Get UTC Info */ get.arg = GETDATA_UTCINFO; ioctl (fd, GETDATA, &get);

Command 0xF4,0xF5, 0xF6, 0xFE	pci_req_assembly
Description	Request Model, Serial Number, Assembly Number and Hardware FAB
Example	/* Get TFP Model */

	get.arg = GETDATA_TFPMODEL; ioctl (fd, GETDATA, &get);

Command 0x4F	pci_req_fw_ver
Description	Request firmware version
Example	/* Get Firmware Version */ get.arg = GETDATA_DTFW; ioctl (fd, GETDATA, &get);

4.2 EXAMPLE 1: GPS PACKET 46 – HEALTH PACKAT SAMPLE

```
int i;
char rbuf[STFP_MAX_READ];

printf ("\n\nGPS PACKET 46 - GPS HEALTH PACKET\n\n");
ioctl (fd, REQGPSPACKET, 0x46);
read (fd, rbuf, STFP_MAX_READ);

printf ("Raw Data: ");
for (i = 0; i < 18; i++)
        printf ("%02X ", rbuf[i] & 0xff);

printf ("\nID: \t%02X \nStatus: 0x%02X \nError: \t0x%02X\n",
        rbuf[1] & 0xff, rbuf[2] & 0xff, rbuf[3] & 0xff);
```