PCI / PCIe

Time and Frequency Processor

User's Guide

Products Included:

PCI

bc635PCI-V2

bc637PCI-V2

bc635PCI-V2-OCXO

bc637PCI-V2-OCXO

PCIe

bc635PCIe

bc637PCIe

bc635PCIe-OCXO

bc637PCIe-OCXO

Product CD, 098-00179-000

January 11, 2010

Revision B

Symmetricom Customer Assistance

Symmetricom's Customer Assistance Centers are a centralized resource to handle all of your customer needs.

Customer Assistance Center Telephone Numbers:

- Worldwide (Main Number): 1-408-428-7907
- USA, Canada, Latin America including Caribbean, Pacific Rim including Asia, Australia and New Zealand: 1-408-428-7907
- USA toll-free: 1-888-367-7966 (1-888-FOR-SYMM)
- Europe, Middle East & Africa: 49 700 32886435

Technical Support can be obtained either through the Online Support area of our website:

(http://www.symmetricom.com/support/online-support/ttm-product-support/), or by calling one of the above Customer Assistance Center numbers.

When calling the worldwide or USA-based number, select Option 1 at the first prompt. Telecom Solutions Division customers should then select Option 1; Timing, Test and Measurement Division customers should then select Option 2.

Technical Support personnel are available by phone:

- Between 7 a.m. to 5 p.m. Pacific Time, weekdays through the Main Customer Assistance Center number 1-408-428-7907.
- Between 8 a.m. to 5 p.m. Central European Time weekdays at the Europe, Middle East and Africa number 49 700 32886435.
- After hours support for emergencies only is handled through the worldwide (main) number 1-408-428-7907.

Customers who have purchased Technical Support Contracts may e-mail support requests to:

- support@symmetricom.com (Americas, Asia, Pacific Rim)
- emeasupport@symmetricom.com (Europe, Middle East, Africa)

Copyright

Due to continued product development this information may change without notice. If you find any errors in the documentation, please report them to us in writing. Symmetricom, Inc. does not warrant that this document is error-free.

Intellectual Property

The software contains proprietary information of Symmetricom, Inc.; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

Limited Product Warranty

Hardware and embedded software - For a period of one (1) year from date of shipment by Symmetricom, Symmetricom warrants that all Products shall be free from defects in design, material, and workmanship; shall conform to and perform in accordance with Symmetricom's published specifications, if any; shall be free and clear of any liens and encumbrances; and shall have good and valid title. This warranty will survive inspection, acceptance, and payment by Buyer. Symmetricom does not warrant that the operation of such Products will be uninterrupted or error free. This warranty does not cover failures caused by acts of God, electrical or environmental conditions; abuse, negligence, accident, loss or damage in transit; or improper site preparation.

This warranty shall be null and void in the event (i) Buyer or any third party attempts repair of the goods without Symmetricom's advance written authorization, or (ii) defects are the result of improper or inadequate maintenance by Buyer or third party; (iii) of damage to said goods by Buyer or third party-supplied software, interfacing or supplies; (iv) of improper use (including termination of non-certified third party equipment on Symmetricom's proprietary interfaces and operation outside of the product's specifications) by Buyer or third party; or (v) the goods are shipped to any country other than that originally specified in the Buyer's purchase order.

Goods not meeting the foregoing warranty will be repaired or replaced, at Symmetricom's option, upon return to Symmetricom's factory freight prepaid; provided, however that Buyer has first obtained a return materials authorization number ("RMA Number") from Symmetricom authorizing such return. The RMA Number shall be placed on the exterior packaging of all returns. Symmetricom will pay shipping costs to return repaired or replacement goods to Buyer.

Symmetricom reserves the right to disallow a warranty claim following an inspection of returned product. When a warranty claim is questioned or disallowed, Symmetricom will contact Buyer by telephone or in writing to resolve the problem.

Software - Symmetricom warrants that for a period of ninety (90) days from date of shipment by Symmetricom the accompanying media will be free from defects in materials and workmanship under normal use. The physical media warranty does not apply to defects arising from misuse, theft, vandalism, fire, water, acts of God or other similar perils. Symmetricom will not be liable for any damages caused by the Buyer's failure to fulfill its responsibilities as stated above.

THE FOREGOING WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF TITLE, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE HOWSOEVER ARISING.

Limitation of Liability - The remedies provided herein are the Buyer's sole and exclusive remedies. In no event or circumstances will Symmetricom be liable to Buyer for indirect, special, incidental or consequential damages, including without limitation, loss of revenues or profits, business interruption costs, loss of data or software restoration, or damages relating to Buyer's procurement of substitute products or services. Except for liability for personal injury or property damage arising from Symmetricom's negligence or willful misconduct, in no event will Symmetricom's total cumulative liability in connection with any order hereunder or Symmetricom's Goods, from all causes of action of any

kind, including tort, contract, negligence, strict liability and breach of warranty, exceed the total amount paid by Buyer hereunder. SOME JURISDICTIONS DO NOT ALLOW CERTAIN LIMITATIONS OR EXCLUSIONS OF LIABILITY, SO THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO ALL BUYERS.

Contact Information

Symmetricom, Inc.
Timing, Test & Measurement
3750 Westwind Blvd.
Santa Rosa, CA 95403
Main: +1 (707) 528-1230

For Sales, Technical Support, and Return Materials Authorization, please see

" Symmetricom Customer Assistance" on page ii.

Conventions

The conventions used in this manual are:

Note: Tips and clarifications

Warning:Actions to prevent equipment damage.

Bold:Used to show messages, menu items, etc., that appear on a computer screen. For example, click on **Submit Changes**.

Text: Used to indicate text you should enter with your keyboard, exactly as printed.

Errata

Errata are available on the CD ROM supplied with the equipment. The errata file name is “Errata.pdf”.

Table of Contents

This page intentionally left empty

1. PCI/PCIe TFP Hardware

1.1. Introduction

1.1.1. General Information

The Symmetricom model bc635PCI-V2, bc637PCI-V2, bc635PCIe, and bc637PCIe Time and Frequency Processors (TFP) are high performance plug-in cards used for precise time synchronization of the host computer over the PCI or PCIe bus.

Note that these Time and Frequency Processors will be referred as “TFP” or “TFPs” for the remainder of the document.

The PCI TFP cards operate at 33 MHz and are compatible with PCI Local Bus Specification Revision 2.3. The PCI products support both the 3.3V and 5V signaling environments defined by the PCI Local Bus Specification. They are considered Universal add-in cards that are capable of detecting the signaling environment and adapting themselves to that environment.

The PCIe TFP cards offer similar features to the PCI cards. The PCIe cards use a single lane and support full 2.5 Gbps in either direction.

The TFP products may be used in either Generator mode or Synchronized Generator mode, supplying precise time (100's nanoseconds through thousands of years) to the host computer. When the card is operating as a Synchronized Generator, the output signals are synchronized to the timing reference. The card phase locks to the timing reference and controls the on-board oscillator to remove frequency errors. If the timing reference is lost, the card continues to increment time and output timing signals based upon the card's 10 MHz oscillator frequency (flywheeling).

There are eight separate TFP products supported by this manual:

	Model	Description
1	bc635PCI-V2	PCI Time & Frequency Processor with TCXO
2	bc637PCI-V2	GPS Synchronized, PCI Time & Frequency Processor with TCXO
3	bc635PCI-V2-OCXO	PCI Time & Frequency Processor with OCXO
4	bc637PCI-V2-OCXO	GPS Synchronized, PCI Time & Frequency Processor with OCXO
5	bc635PCIe	PCI Express Time & Frequency Processor with TCXO
6	bc637PCIe	GPS Synchronized, PCI Express Time & Frequency Processor with TCXO
7	bc635PCIe-OCXO	PCI Express Time & Frequency Processor with OCXO
8	bc637PCIe-OCXO	GPS Synchronized, PCI Express Time & Frequency Processor with OCXO

Images of bc635PCI-V2 and bc637PCI-V2 follow.

Figure 1-1: Model bc635PCI-V2 Time and Frequency Processor

GPS Antenna

A: SMB Antenna Connector

B: J1 Module I/O 15 pin D-sub connector

Figure 1-2: Model bc637PCI-V2 (GPS option shown with GPS antenna)

All sections of this manual are applicable to all boards except where noted.

Images of bc635PCIe and bc637PCIe follow.

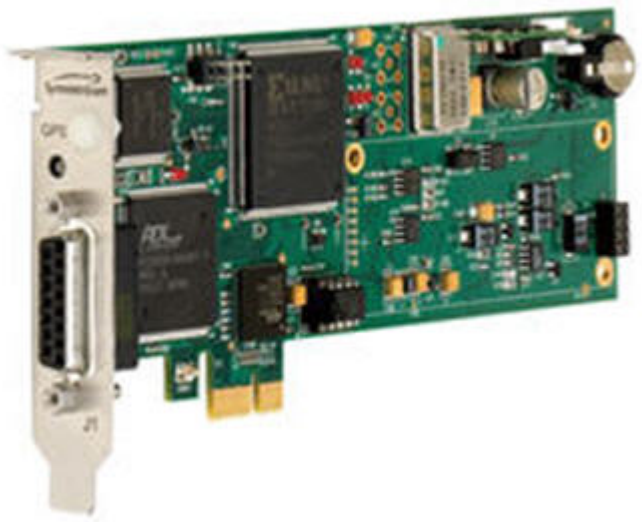
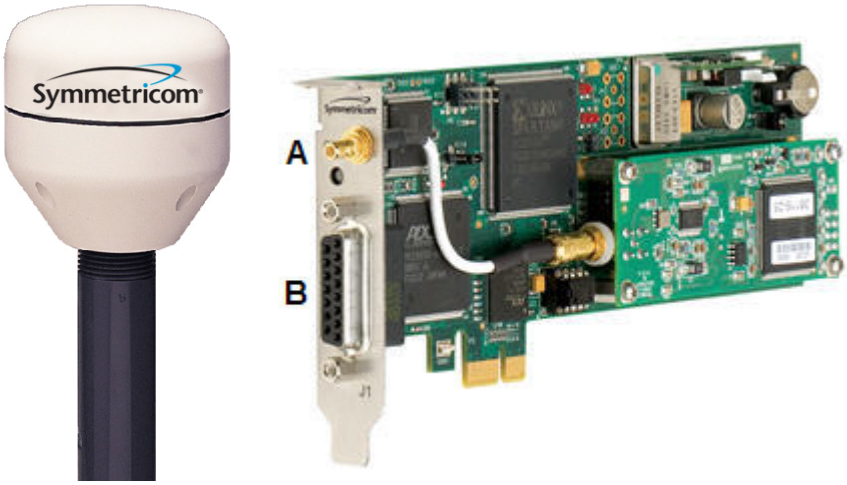


Figure 1-3: Model bc635PCIe Time and Frequency Processor



GPS Antenna

P/N 142-614-50

A: SMB Antenna Connector

B: J1 Module I/O 15 pin D-sub connector

Figure 1-4: Model bc637PCIe (GPS option shown with GPS antenna)

All sections of this manual are applicable to all boards except where noted.

1.1.2. Key Features

- All modes of operation are supplemented by flywheel operation. If the synchronization source is lost, the TFP will continue to function at the last known reference rate. The following operational modes are supported, and are distinguished by the reference source.

Mode	Source of Synchronization
0	Time Code - IRIG A, B, G, E, IEEE 1344, NASA 36, XR3 & 2137
1	Free Running (Generator Mode) - 10 MHz Selected Reference (Internal or External)
2	1 PPS - External One Pulse Per Second Input
3	RTC-Uses battery backed on-board real time clock I.C.
4-5	Reserved
6	GPS (bc637) - GPS Antenna/Receiver

- Time may be captured in four independent sets of time capture registers. The default time format is provided in binary form (UNIX seconds through 100 nanoseconds). The TFP Device Register Summary and register formats are outlined in "TFP Device Register Summary" on page 27.
- In synchronized generator mode, the TFP uses the selected reference source to discipline either the standard on-board TCXO (Temperature Compensated Crystal Oscillator), optional on-board OCXO (Oven Controlled Crystal Oscillator) or external 10 MHz oscillator. The 10 MHz oscillator drives all timing functions and outputs on the card. 10 MPPS and 1 PPS signals derived from the 10 MHz oscillator are provided as outputs, among others. The advantage of the OCXO is better holdover (should the synchronization source be lost).
- The TFP generates IRIG A, B, G, E, IEEE 1344, NASA 36, XR3 and 137 time codes. Amplitude modulated and DC level shift formats are produced simultaneously.
- A Programmable Periodic (also known as Heartbeat) output is provided. The output frequency is programmable and may be synchronized to the TFP 1 PPS signal. The Periodic output programming is discussed in"1.3.2. Heartbeat Output" on page 18. This signal may be internally connected to the Event Input to capture the time associated with the Programmable Periodic edge. The Event Input may be configured via the CONTROL Register as described in "CONTROL Register (0x10)" on page 28. Note that the Periodic square wave output and the DDS square wave output share the same output driver where either signal may be selected for output on J1 pin 15.
- A DDS (also known as frequency synthesizer) output may be selected in place of the periodic rate generator's output. The DDS offers a much wider frequency range than the Programmable Periodic. The DDS is discussed in more detail in "1.3.3. DDS Output" on page 18. Note that the DDS and the Programmable Periodic signals are capable of generating interrupts. Caution should be taken when using either of these sources with a rate that exceeds the computer's ability to service the interrupts.
- A Time Coincidence Strobe output is provided. The Strobe is programmable from days through microseconds. The strobe also has an each second mode (referred to in this manual as Minor Time Mode) programmable to microseconds. The strobe is discussed in more detail in "1.3.4. Time Coincidence Strobe Output" on page 20.

- One set of time capture registers is used for event time capture. Time is captured on the rising or falling edge (user programmable) of the Event Input signal provided to the TFP via the J1 I/O connector or from the Programmable Periodic Output signal. The Event Input configuration is manipulated via the CONTROL Register as described in "CONTROL Register (0x10)" on page 28.
- Two sets of event capture registers are available from dual-purposed input pins. J1 pin 10 may be used as the DCLS Time Code Input or as the Event2 input. J1 pin 14 may be used as the External 1PPS Input or as the Event3 input.
- Seven maskable interrupt sources are supported. All interrupt sources may be polled. Interrupts are discussed in more detail in "MASK Register (0x18)" on page 31.

Note: The bc635PCI-V2 and bc637PCI-V2 do not provide interrupts at system start-up and therefore do not support the PCI Local Bus Specification Revision 2.3 feature of software disable of interrupts at start-up.

1.1.3. Specifications and Settings

Time Code Inputs

Formats	IRIG A, B, G, E, IEEE 1344[1], NASA 36, XR3 and 2137 (AM/DCLS)
Carrier Range	± 5 PPM
Time Accuracy[2]	< 5 µsec. (AM with carrier frequencies 1 kHz or greater) < 1 µsec. (DCLS)
AM Modulation Ratio	2:1 to 4:1
AM Input Amplitude	1 to 8 Vp-p
AM Input Impedance	5 kΩ, AC Coupled
DCLS Input	5V HCMOS, >2V high, <0.8V low

[1] IEEE 1344 compliance - The translator processes the 27 control function bits of IRIG B time code as set forth in IEEE 1344.

[2] May require a calibration to attain this accuracy. See "1.3.7. AM Time Code Calibration" on page 21.

Time Code Outputs

Formats	IRIG A, B, G, E, IEEE 1344, NASA 36, XR3 & 2137 (Modulated/DCLS)
Modulation Ratio	3:1 ± 10%
Output Amplitude	3V p-p ±10% (fixed) into 50 Ω
DC Level Shift	5V HCMOS, >2V high, < 0.8V low into 50 Ω

PCIe Bus Characteristics

Specifications	Single Lane, 2.5 Gbps each direction
Size	Single-Width 6.6” x 2.7” (Low Profile)
Interrupts	Auto Configurable
Power (bc635PCIe)	+3.3V @ 400 mA, +12V @ 250 mA
Power (bc637PCIe)	+3.3V @ 400 mA, +12V @ 300 mA
Power (bc635PCIe-OCXO)	+3.3V @ 400 mA, +12V @ 350 mA, (0.5 A @ start up)
Power (bc637PCIe-OCXO)	+3.3V @ 400 mA, +12V @ 400 mA, (0.6 A @ start up)

PCI Bus Characteristics

Specifications	PCI local bus™ 2.2 compliant, 2.3 compatible, PCI-X compatible
Size	4.2” x 6.875” (Full Height)
Interrupts	Automatically Assigned (PnP)
Power (bc635PCI-V2)	+5V @ 700 mA, +12V @ 50 mA
Power (bc637PCI-V2)	+5V @ 800 mA, +12V @ 50 mA
Power (bc635PCI-V2-OCXO)	+5V @ 800 mA, (1.1 A @ start up), +12V @ 50 mA
Power (bc637PCI-V2-OCXO)	+5V @ 900 mA, (1.2 A @ start up), +12V @ 50 mA

Inputs

Event Inputs (1,2,3)	5V HCMOS,>2V high, < 0.8V, zero latency, Rising or Falling Edge Triggered, 20 nS min. width, 250 nS min. period
External 1 PPS	5V HCMOS, >2V high, < 0.8V low Rising Edge On Time, 20 nS minimum width
External 10 MHz	Digital 40% to 60% Duty Cycle (or) Sine wave, 0.5 to 8Vp-p, >10k Ω

Outputs

1 PPS	5V HCMOS, >2V high, < 0.8V low into 50 Ω , Rising Edge On Time, 60 µS Positive pulse
Periodic	5V HCMOS, >2V high, < 0.8V low into 50 Ω , Rising Edge On Time (selectable on time control), <1 Hz to 250 kHz, square wave
DDS	5V HCMOS, >2V high, < 0.8V low into 50 Ω , Rising Edge On Time (selectable on time control), 1/1e7 - 1e8 Hz, <2 nS jitter p-p, square wave
Strobe	5V HCMOS, >2V high, < 0.8V low into 50 Ω , 1 µS Positive pulse, variable delay
1, 5, 10 MHz clock	5V HCMOS, >2V high, < 0.8V low into 50 Ω (see "1.3. Functional Description" on page 16 for signal characteristics)
Time Code DCLS	5V HCMOS, >2V high, < 0.8V low into 50 Ω
External 10 MHz Oscillator DAC	Jumper selectable 0-5VDC or 0-10VDC into 1kΩ

bc635PCI-V2 and bc637PCI-V2 Jumpers

The following is a list of the hardware jumpers on the bc635PCI-V2 and bc637PCI-V2 boards:

- JP1 is a 2mm jumper that is the RTC battery connect switch. The RTC battery is a non-rechargeable lithium cell with 48 mAh capacity. When the card is not powered and the RTC battery has not been disabled, the RTC draws about 20 uA which will provide >100 days of RTC operation. Long-term storage of the card should be done only after issuing the Disconnect RTC Battery command or by the manual disconnection of the jumper on JP1. The factory configuration places the 2mm J1 jumper ON.
- JP2 is the 1, 5, 10 MPPS or 10 MHz oscillator select switch. The factory configuration places the 2mm JP2 jumper on pins 1-2 which will allow for the software selection of 1, 5 or 10 MPPS for the output on J1 pin 13. When the jumper is in the 2-3 position, the output on J1 pin 13 is a buffered signal from the 10 MHz oscillator.
- JP3 is the DAC voltage range switch. When the jumper is OFF, the DAC voltage is 0-5 VDC; when ON the voltage is 0-10 VDC. Both oscillators that are offered for this board have 0-5 VDC control voltage ranges, therefore the 2mm jumper is not factory installed. If an external oscillator requires a 0-10 VDC control voltage range, a 2mm jumper should be placed on JP3.
- JP4 is for factory use only.

bc635PCIe and bc637PCIe Jumpers

The following is a list of the hardware jumpers on the bc635PCIe and bc637PCIe boards:

- JP1 is a 2mm jumper that is the RTC battery connect switch. The RTC battery is a non-rechargeable lithium cell with 48 mAh capacity. When the card is not powered and the RTC battery has not been disabled, the RTC draws about 20 uA which will provide >100 days of RTC operation. Long-

term storage of the card should be done only after issuing the Disconnect RTC Battery command or by the manual disconnection of the jumper on JP1. The factory configuration places the 2mm J1 jumper ON.

- JP3 is the DAC voltage range switch. When the jumper is OFF, the DAC voltage is 0-5 VDC; when ON the voltage is 0-10 VDC. Both oscillators that are offered for this board have 0-5 VDC control voltage ranges, therefore the 2mm jumper is not factory installed. If an external oscillator requires a 0-10 VDC control voltage range, a 2mm jumper should be placed on JP3.
- JP4 is for factory use only.

Environmental Specifications

Temperature	Operating	0°C to +70°C (32°F to +158°F)
	Non-Operating	-30°C to +85°C (-22°F to +185 °F)
Relative Humidity	Operating/Non-Operating	To 95% RH, non-condensing

Front Panel LED

	LED indication	LED Definition
1	Off	No power
2	Red	No valid reference
3	Orange	Valid reference but not locked to it yet
4	Green	Freerun mode
5	Green Blinking	Locked to a timing reference

GPS Antenna

To operate in the GPS Synchronized Generator mode, the bc637 models use an external antenna. The standard antenna kit supplied with these models includes the antenna, 50 feet (15.24 meters) of coaxial cable and antenna mounting hardware.

Figure 1-5: Antenna parts

The antenna is housed in completely waterproof packaging designed to withstand the elements. When the four UNC 4-40 screws are loosened, the antenna module detaches as shown below, exposing the TNC connector.

Figure 1-6: Antenna with TNC connector

Warning: Models bc637PCI-V2 and bc637PCIe supply +5 VDC to the antenna. Connection to an alternate antenna may damage the board and/or antenna functionality.

General Specifications for the Antenna	
Operating Temperature	-40°C to +85°C (-40°F to +185°F)
Storage Temperature	-40°C to +100° C (-40°F to +212°F)
Humidity	100% condensing
Power	30 mA @ 5 V (supplied by card)

Cable Specifications for the Antenna Cable	
Type	RG-59 (Belden 9104)
Length	50 feet (15.24 meters)
Weight	1.2 lb. (0.545 kg)
Humidity	All weather, outdoors
Connectors	Type TNC male to BNC male

Cable lengths from 150 feet (45 meters) to 300 feet (90 meters) require an in-line GPS Signal Amplifier (P/N 150-200).

Cable lengths from 300 feet (90 meters) to 1,500 feet (457.2 meters) require the antenna Down/Up Converter option, part number 142-6150. Refer to the optional 142-6150 Down/Up Converter antenna manual for specifications.

Antenna and Down/Up Converter units are mounted on a 12-inch (30.48 cm) long PVC mast with 3/4-inch (1.9 cm) Male Pipe Thread (MPT) on both ends.

1.2. Installation

1.2.1. General

This section contains TFP installation instructions and information regarding operating modes and the use of registers to configure the card. Models bc637PCI-V2 and bc637PCIe have the additional feature of GPS mode that will automatically synchronize the card to UTC time.

Installation of PCI/PCIe boards is quite a bit simpler than in most bus architectures due to the following 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.
- Auto configuration that allows the host computer to read the device ID, and other configuration information directly from the PCI Configuration Registers.
- The TFP is shipped with software suitable for use with Linux, Solaris and Windows. The kit includes drivers for low-level access, as well as software programs for configuring and accessing the card.

Installation is as easy as choosing a vacant PCI or PCIe slot, plugging in the Symmetricom Time and Frequency Processor (TFP) and installing the device driver. Be sure to consult the user documentation that came with your particular workstation for any specific installation instructions. In addition, to protect the card, use good ESD protection practices when installing the card.

1.2.2. Installing the Card and Antenna

Installing the Card

- Unpack the card and carefully inspect it for shipping damage. Report any damage to the carrier immediately.
- Record the card's serial number. The serial number has eight numerals, for example 08190018. The first two are the last two numbers of the year. The second two is the week number of the year. The final four is a unique number for the card.
- With the computer's power turned OFF, if the TFP front panel height is correct install and secure the card in an empty card slot. If a low-profile front panel needs to be attached to the TFP, follow the instructions below.

Changing the TFP Card Front Panel

The bc63xPCI-V2 and bc63xPCIe TFP card is shipped with a full-height front panel attached to it. A low-profile front panel is included as an alternative for low-profile situations. If the low-profile front panel option is required use the following procedure. Note, the following three tools will be required to change the front panel:

- 3/16 inch wrench
- 1/4 inch wrench
- Phillip's #1 screwdriver

Standard ESD precautions should be followed when handling the TFP printed circuit board.

Procedure for Changing the TFP Card Front Panel

The TFP card is attached to the front panel in three places, the GPS RF connector, the 15 pin D-sub-connector and a single Phillip's #1 screw. To change the front panel, do the following:

1. Remove the two nuts and washers from the backs of the 15 pin D-sub connector (3/16 wrench).
2. Remove the two standoff screws from the front panel side of the 15 pin D-sub connector (3/16 wrench).
3. Remove the retaining nut and washer from the GPS RF connector (1/4 wrench).
4. Remove the Phillip's #1 screw that secures the TFP card directly to the front panel.
5. Set the full-height front panel aside and pick up the low-profile front panel.
6. Secure the GPS RF connector loosely to the front panel with its nut and washers.
7. Secure the TFP card loosely to the low-profile front oanel with the Phillip's #1 screw.
8. Secure the 15 pin D-sub connector to the front panel with its two standoff screws.

9. Secure the 15 pin D-sub connector to the front panel with its two retaining lock washers and nuts.
10. Tighten the GPS RF connector front panel retaining nut.
11. Tighten the Phillip's #1 screw connecting the TFP card directly to the front panel.

Antenna Location and Installation (bc637PCI-V2 and bc637PCIe)

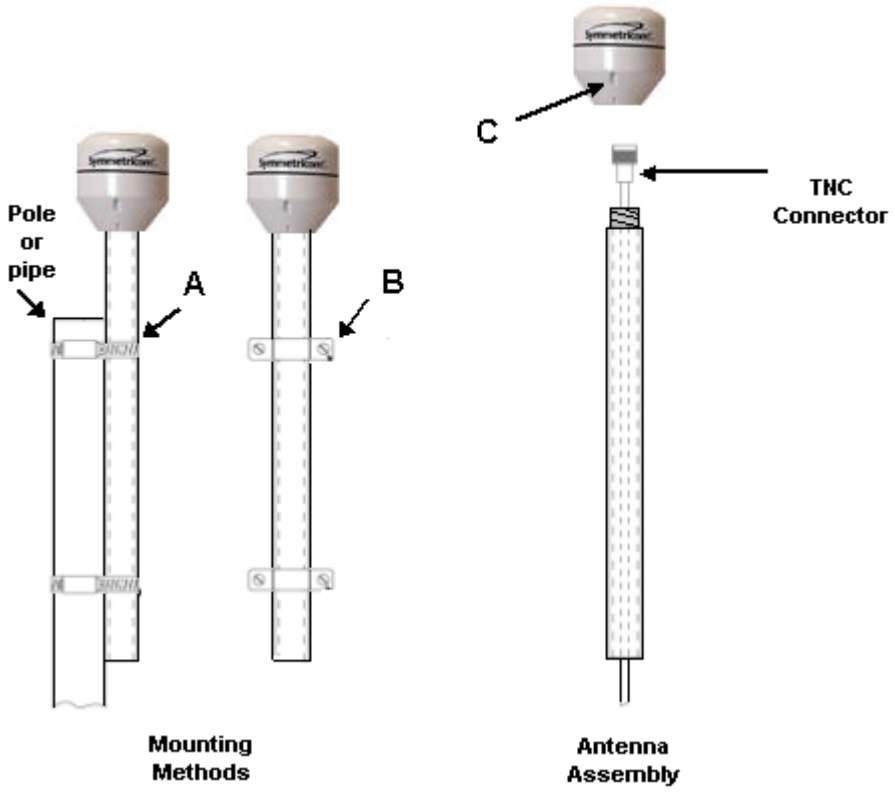
When selecting a site for the antenna, find an outdoor location that provides full 360-degree visibility of the horizon. In most cases, this means locating the antenna as high as possible. Any obstruction will degrade unit performance by blocking satellite signals. Blocked signals can significantly increase the time for satellite acquisition, or prevent acquisition all together.

The installation instructions are divided into “Quick Initial Setup” and “Permanent Installation” sections. This is designed to firstly get the GPS unit up and running as quickly as possible, and secondly to verify its operation and to become familiar with the equipment, We recommend that new users follow the “Quick Initial Setup” instructions first before proceeding to a permanent installation.

Quick Initial Setup

Connect the antenna cable to the unit and to the antenna. Simply run the antenna outside the building or set it on a window sill. Depending on the lead content or coating on the glass, it may be necessary to go outside. Turn on the unit and verify its operation.

Permanent Antenna Installation



A: Secure mast to pole or pipe

B: Secure mast to mounting surface with pipe straps.

- Wood mounting - Drill hole with #43 bit and install with a #10 wood screw.
- Drywall/Masonry - Drill hole with 1/4 “bit, Insert #10 screw anchor. Then install the #10 wood screws.

C: Shows the cable connector relative to the antenn. The antenna is shown separated from antenna mast. In reality, the cable is connected to the antenna and the mast is firmly attached to the antenna. Do not disconnect the pipe from the antenna assembly. To gain access to the TNC antenna input connector, untighten the four captive UNC screws that secure the two parts of the antenna.

Figure 1-7: Permanent Antenna Installation

Mast top mounting is the preferred mounting method. Special brackets are provided to mount the antenna to a pipe or the peak of a building. The antenna mounting mast should be 2-inch (5.08-cm) water pipe or conduit. The mast must be rigid and able to withstand high winds without flexing. Guy wires may be used to stabilize a mast longer than 10 ft. (3.048 m).

1.2.3. bc637PCI/PCIe Additional Hardware

The GPS antenna equipment included with the product and described in this manual consists of the following:

- One wide-range 5-12 VDC GPS L1 antenna
- One 50 ft. length of Belden 9104 coaxial cable with BNC(m) and TNC(m) connectors
- SMB to BNC adaptor is included

Optional cable lengths and accessories are available. Please note the following when setting up longer cable runs:

Note: Using Belden 9104, the maximum cable length without amplification is 150 feet . Using Belden 9104, the maximum cable length using the optional in-line amplifier is 300 feet. For cable runs longer than 300 feet, an optional GPS Down/Up Converter kit is available.

1.2.4. Minimum System Requirements

- PC with one free PCI or PCIe slot
- 25 MB disk space

1.2.5. Installation Under Windows

1. With the PC turned off, insert the Symmetricom TFP in an open slot. Use good ESD protection practices when installing the card.
2. Boot the PC. After login, Windows may prompt you to install newly found hardware. Disregard/cancel this dialog box.
3. Insert the bc63xPCIe-V2 product CD and select bc635PCIcfg or bc637PCIcfg to install depending on the type of TFP you inserted into your PC. The bc637PCIcfg.exe demonstrates the board's GPS functionality.Follow the instructions shown on the screen to finish the installation. The computer will reboot at the end of the installation.
4. If Auto-Run is disabled, manually install the software by browsing to the Windows directory on the product CD. Click on bc635PCIcfg_Setup.exe or bc637PCIcfg_Setup.exe depending on the type of TFP you inserted into your PC. Follow the instructions shown on the screen to finish the installation. The computer will reboot at the end of the installation.
5. After the computer reboots, launch the TFP configuration software bc635PCIcfg or bc637PCIcfg to begin communication with the card.
6. If no time is displayed, or if you are prompted

1.2.6. Windows Software Development Kit

To develop Windows based applications for Symmetricom TFP cards, you must install the bc63x Windows Software Developer's Kit included on the bc63xPCIe-V2 product CD. You can insert the product CD and use the SDK installation button to install the SDK or browse to the Windows directory and run bc635_637PCI_SDK_Setup.exe.

The bc63x Windows SDK is a full featured software development kit containing functions necessary to control and read the time from the card. The example application programs were originally developed under Microsoft Visual C++ 6.0 and ported to support both Visual C++ 6.0 and Visual Studio 2008.This allows the user to create customized applications to use with the Symmetricom TFP cards with a minimum amount of time and effort. The SDK is an easy to integrate and highly reliable alternative to writing lower level code to access a TFP card's registers directly. The API functions make interfacing to a TFP card straight forward, keeping software development effort focused on the end user applications.

The Windows SDK supports both 32 and 64-bit Windows operating system from Windows XP through Windows 7. However, the SDK does not support Windows 2000, NT 4, or any Windows 95 based Windows operating system (95, 98 and Me). The target application development environment is Microsoft Visual Studio. The SDK includes .h, .lib, and .dll files for customer application development. The SDK also includes source code for the bc635PCIcfg, bc637PCIcfg and the TrayTime programs as well as other example programs.

Customers using the previous version of the SDK, which we will now refer to as the "Legacy bc635 SDK software", can continue to use that driver and any application software previously written using Legacy bc635 SDK software with the bc635PCIe.

For customers wanting to run previously written software with the new 32/64 bit driver, you will need to recompile your applications with the new driver and .dll to support operating systems from Windows XP through Windows 7. See "3.2. Release Notes" on page 102.

1.2.7. Linux Software Development Kit

To develop Linux based applications for Symmetricom TFP cards, you must install the bc63x Linux Software Developer's Kit included on the bc63xPCIe-V2 product CD. Browse to the Linux directory and follow instructions in the readme file to install the Linux SDK.

The Linux SDK is a full featured software development kit (SDK) containing functions necessary to control and read the time from the card. This SDK allows the user to create customized applications to use with the Symmetricom TFP cards with a minimum amount of time and effort. The SDK is an easy to integrate and highly reliable alternative to writing lower level code to access a TFP card's registers directly. The API functions make interfacing to a TFP card straight forward, keeping software development effort focused on the end user application.

The SDK supports both 32-bit and 64-bit Linux 2.4 and 2.6 kernels. The target application development environment is GNU gcc/g++. The SDK includes .h, .lib and .so files for customer application development. The SDK also includes source code for the bc63xPCIcfg example program. The example program uses discrete functions to access TFP features. This allows developers to copy and paste useful code into their own applications so that they can keep development effort focused on the end user applications.

1.2.8. Solaris Software Development Kit

To develop Solaris based applications for Symmetricom TFP cards, you must install the bc63x Solaris Software Developer's Kit included on the bc63xPCIe-V2 product CD. Browse to the Solaris directory and follow instructions in the readme file to install the Solaris SDK.

The SDK supports 64-bit Solaris 8 to 10 on both SPARC and x86_64 platform. The target application development environment is Solaris C/C++ compiler or GNU gcc/g++. The SDK includes source code for the driver stfp and the bc63xPCIcfg example program. The example program shows how to use discrete functions and ioctl code to access TFP features. This allows developers to copy and paste useful code into their own applications so that they can keep development effort focused on the end user applications.

1.2.9. Installation Under Other Operating Systems

Usage of the TFP under other operating systems may require a special driver for the device. Please contact Symmetricom for source code availability. With the wide variety of machines and operating systems that support the PCI(e) bus, it is not practical for Symmetricom to develop drivers for use in all of these environments.

1.3. Functional Description

1.3.1. General

The primary function of the TFP is to provide precise time to the host computer across the PCI or PCIe bus. The TFP can derive time from any one of the sources listed in "1.1.2. Key Features" on page 4.

In all but the Free Running mode of operation, the TFP synchronizes its 10 MHz oscillator to an input reference. The TFP achieves synchronization from the input reference and disciplines the 10 MHz oscillator such that the locally generated 1 PPS signal is matched in phase and frequency to the input reference.

Once synchronization is achieved, the TFP is able to maintain time even if the reference source is lost (though some timing drift will occur). This is referred to as flywheeling or holdover. If available, the TFP will obtain major time (days, hours, minutes, and seconds) from the input reference. In Time Code and GPS Mode, this major time is readily available, but in the Free Running and External 1 PPS Modes, major time is not available and must be set manually by the user. The Timing Mode is selected via the dual-port RAM interface as described in "Command 0x10: Set TFP Timing Mode" on page 41, using command 10.

Mode 0 (Time Code Mode)

In Time Code Mode, the TFP derives time from the currently selected input time code. The TFP will accept time code in either amplitude modulated (AM) or DC Level Shift (DCLS) form. AM time code is a sinusoidal analog signal that is amplitude modulated with the time data. DCLS is simply the envelope of the modulated time code and is a digital signal. Most time codes provide both major time (days,

hours, minutes, and seconds) and minor time (subseconds) to the card. Some IRIG time codes (e.g., IEEE1344) and GPS (if equipped) provide year information.

Mode 1 (Free Running Mode)

In Free Running Mode, no external timing source is used and the TFP oscillator is allowed to free-run. The user must set major time manually. The Major Time is selected via the dual port RAM interface as described in "Command 0x11: Set Time Register Format" on page 41, using command 11. Free Run mode allows the user to perform timing tests when an external timing source is unavailable.

Mode 2 (External 1 PPS Mode)

In External 1 PPS Mode, the TFP synchronizes its oscillator to a user-supplied 1 PPS signal. The user must set major time manually. The Major Time is selected via the dual-port RAM interface as described in "Command 0x11: Set Time Register Format" on page 41 Chapter 1.5, using command 11. The External 1PPS Mode will not indicate locked status to the External 1PPS reference without first being synchronized to a time reference that includes time of day information (Time Code or GPS). Please note that the TFM will always synchronize to the External 1PPS when selected, but if the card has not been previously synchronized by Time Code or GPS, the Tracking, Phase and Frequency Status information will not be valid in External 1PPS mode.

Mode 3 (RTC)

In the Real Time Clock (RTC) mode, the TFP receives its major time via the RTC and operates as a Generator (no input reference).

Mode 6 (GPS) - bc637PCI-V2 and bc637PCIe

In GPS Mode, like Time Code Mode, both major and minor times are derived from the timing source. In addition to time, other information is available from the GPS system such as position and velocity. This mode requires the use of a GPS antenna that has an unobstructed view of the sky. Note that the antenna location is important because the GPS receiver must initially acquire and track at least four satellites to obtain accurate time for the TFP card. If however, the user's position is accurately known, or has been previously determined, the position information can be sent to the TFP’s GPS receiver, enabling GPS lock from just one satellite.

Time Capture Registers

The TFP supports four independent sets of time capture registers. Each set consists of two 32-bit wide registers that hold both the major and minor time. One set of registers, called TIME0 and TIME1, support time on demand across the PCI bus. Time is captured in these registers whenever the user accesses a special time request register (TIMEREQ). The captured time is held until a subsequent access of the TIMEREQ register. Valid time can be read from the TIMEx registers immediately

following the access of the TIMEREQ register. "1.4.3. Device Register Description" on page 26 describes the available time formats used on the TFP.

The second set of time capture registers, called EVENT0 & EVENT1, are identical in format to the TIMEx registers. Time is captured in these registers whenever the user accesses the special time request register labeled EVENTREQ. Additionally, the EVENTx registers can be set up to capture time in response to either the Event Input or the Programmable Periodic Output (see next section). These device registers are described in more detail in "1.4.3. Device Register Description" on page 26.

The third and fourth set of time capture registers, EVENT2_0, EVENT 2_1, EVENT3_0, and EVENT3_1 are identical in format to the TIMEx registers, but they are only usable with external event sources. The inputs to the Event2 and Event3 time capture registers are on dual-purposed input pins, where usually one or other of the function is chosen.J1 pin 10 may be used as the DCLS Time Code Input or as the Event2 input. J1 pin 14 may be used as the External 1PPS Input or as the Event3 input.

1.3.2. Heartbeat Output

The Heartbeat output is a legacy function. The DDS output described in the next section is a superior frequrency synthesizer. The Heartbeat Output (also known as Programmable Periodic or PPO) allows the user to configure a repetitive digital output synchronized with the timing source. The PPO may optionally be synchronized to the TFP's 1 PPS signal when the Periodic Output frequency is an integer value. If PPO is not an integer value and Synchronous Mode is used, the last Periodic Output cycle before the 1 PPS edge will not be square.

The PPO signal is generated by dividing down a 1 MHz clock, synchronous to the 10 MHz oscillator. The periodic output frequency ranges from 250 kHz (n1 = n2 = 2) to less than 1 Hz. The frequency is determined by the relationship:

Frequency = 1,000,000 / (n1 * n2) Hz

Where:

n1 : divider 1 (range = 2-65535)

n2 : divider 2 (range = 2-65535)

Setting the periodic output frequency to less than 1 Hz and using Synchronous Mode will cause the periodic output to be held at a logic high level. When a rate below 1PPS is desired, Asynchronous Mode must be used. Values of 0 or 1 for either n1 or n2 will also cause the periodic output to be held at a logic high level in either Synchronous or Asynchronous Modes.

Note that the PPO signal is multiplexed with the DDS signal, where one or the other signal is selected for output on the 15-pin I/O connector pin 15.

1.3.3. DDS Output

This DDS circuit is a frequency synthesizer that provides a square wave output with a frequency resolution of 0.03125 (1/32) Hz. Refer to manual section 1.4.5 on Digital Outputs for specifications.

The formula for setting the desired frequency is:

- Frequency x 32 = DDS tuning word value.

The DDS circuit has two synchronization modes, Continuous and Fractional.

Continuous mode

Continuous mode synchronizes the DDS circuit each second, maintaining rising edge timing to the card's on-time 1PPS signal. Continuous mode may be used when the DDS is set for an integer rate.

Fractional mode

Fractional mode allows the DDS circuit to generate a non-integer frequency after first being synchronized to the card's on-time 1PPS signal. Note that the DDS circuit is always re-synchronized to the 1PPS rising edge when a new frequency is chosen.

DDS fractional frequency example:

- Desired frequency:10,491,426.56 Hz
- Desired frequency x 32: (example 10,491,426.56 x 32 = 335,725,649.92 Hz)
- Rounded to integer 335,725,650
- Actual DDS frequency10,491,426.5625 Hz

This is the closest the DDS can get to the desired frequency with 1/32 Hz resolution.

Divider Source

The DDS circuit also includes 7 decades of divider that may be used. The DDS frequency is passed through a divider circuit before being output. The divider's input source may be selected to be one of the following:

1. DDS
2. Multiplier (DDS x multiplier)
3. 100 MHz PLL source

Divider Mode

The divider can be used to generate low or fractional frequencies. The divide range is 1E0 through 1E7 in decades. The divider will also allow for fractional frequency outputs where the divider's input source is decimal shifted by up to seven places.

The divider has another selectable mode, Period Mode. This mode may be desirable when the DDS cannot be set exactly to the desired frequency but using a period value would be exact. When operating in this mode, the output is also a square wave with the period resolution equal to 2 times the period of the input (Divider Source). If the Divider Source of 100 MHz is selected, the period resolution is 20nS. When using 100 MHz as the Divider Source, a period register of 0 corresponds to a 40 nS period (25 MHz) which is Period Mode's upper limit. Use the following formula to set the Period Value Register.

Period Register = (Desired Period / (Divider Source period * 2)) - 2

Example of period mode calculation using 100 MHz Divider Source:

Desired period 59.3 ms= (59.3E-3 / 20E-9) - 2

Result= 2964998

Multiplier Mode

The DDS circuit also includes a frequency multiplier that may be used. The multiplier's input is the DDS and the output is DDS times the multiplication factor. This circuit can multiply the DDS frequency by 1, 2, 3, 4, 6, 8, 10 or 16. Note that the DDS frequency must be high or low enough for use (depends on multiplication factor - see following ranges).

	Input (MHz)
x1	22 - 150
x2	11 - 75
x3	8 - 56
x4	6 - 38
x6	5 - 23
x8	5 - 19
x10	5 - 15
x16	5 - 10

When using the Multiplier Mode, the resolution of the DDS is 1/32 Hz * the multiplier value. The output of this circuitry is capable of creating interrupts. This circuit can generate rates that far exceed a computer's ability to service the DDS interrupts. Note that the DDS signal is multiplexed with the PPO signal, where one or the other signals is selected for output on the 15-pin I/O connector pin 15. The TFP card will load previously set DDS configuration registers at power-on.

1.3.4. Time Coincidence Strobe Output

The TFP provides one Time Coincidence Strobe Output signal. The Strobe output is like an alarm that is activated at some preprogrammed time. The Strobe time can be set from days through microseconds. The duration of the Strobe pulse is one microsecond. Two modes of operation are supported. In one mode (STRMODE=0), both the major and minor times are used to generate the Strobe. In the other mode (STRMODE=1), only the minor time is used to generate the Strobe output,

producing an output pulse once each second. The Strobe is programmed using the CONTROL Register as described in "CONTROL Register (0x10)" on page 28.

1.3.5. PCI(e) Interrupts

The TFP supports the seven interrupt sources listed in Table 1. Each interrupt source can be individually masked off. Use the MASK register to mask on or off each interrupt source. Each interrupt source sets a corresponding bit in the INTSTAT register when the interrupt occurs. When servicing a TFP interrupt, the Interrupt Service Routine (ISR) in the driver reads the INTSTAT register in order to determine the interrupt source(s) requesting service.

Table 1: TFP Interrupt Sources

Int	Interrupt Source
0	Signal transition on Event Input has occurred (edge selected by EVSENSE)
1	PPO edge has occurred (edge selected by EVSENSE)
2	Time Coincident Strobe output rising edge has occurred
3	One second epoch (1 PPS output) rising edge has occurred
4	GPS data packet is available (bc637pci-V2 only)
5	Signal transition on Event2 Input has occurred
6	Signal transition on Event3 Input has occurred

1.3.6. Additional Timing Output Signals

The 1 PPS output is a 60 µsec wide pulse with the rising edge occurring at each 1 second epoch.

A Time Code output signal is available in both AM and DCLS forms simultaneously.

An output frequency of 1 MHz, 5 MHz, or 10 MHz (HCMOS) is selectable.

1.3.7. AM Time Code Calibration

The following AM Time Code calibration procedure is used to adjust the phase of the Time and Frequency Processor when operating in Time Code Mode and using an AM (amplitude modulated) input reference. Modulation ratio, carrier frequency, impedance loading, and other effects may degrade time code performance accuracy. Some applications require very high synchronization accuracy while using AM time codes as a reference. The following calibration procedure helps assure the best time code synchronization accuracy. Because of the inherent differences in the time codes, a different calibration factor exists for each time code type. The user should determine which time code to use in their system, and then perform the calibration for that code. The user can consult Symmetricom if unsure of which code type is best for their application.

1.3.8. Calibration Procedure

The following procedure uses the Windows based bc635PCIcfg.exe program to perform the adjustments. This procedure will synchronize the on-time mark of an incoming time code reference with the rising edge of an incoming 1 PPS signal. (The accuracy of this calibration is limited by the synchronization of the incoming 1 PPS and the on-time mark of the incoming time code).

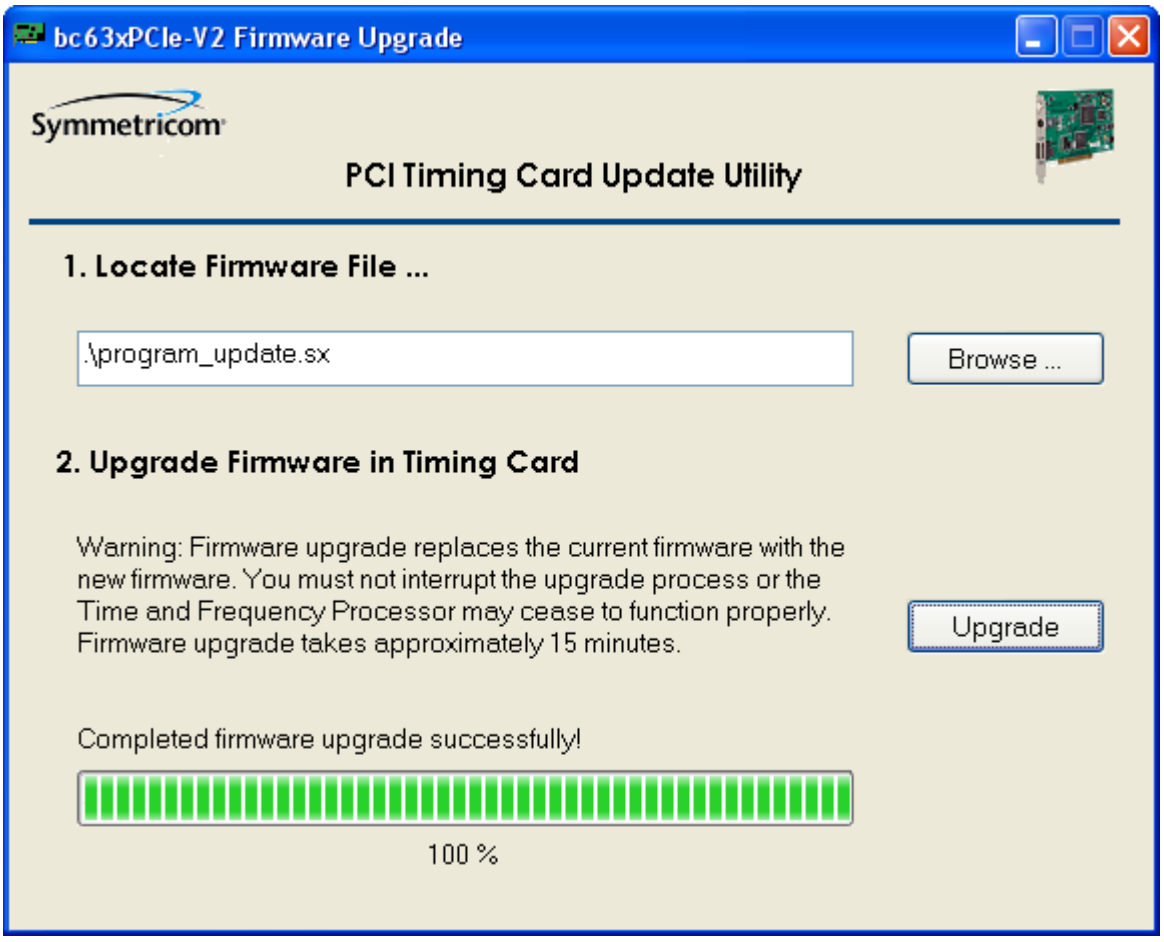
1. Connect the AM Time code reference to the Time code Input on J1 pin 7 of the TFP card.
2. Select the Format and Modulation type for the reference Time code (using the bc635PCIcfg.exe pull down menu “Time code > Decode”).
3. Connect the 1 PPS reference to the External Event Input on J1 pin 6 of the TFP card.
4. Enable the External Event Input on the Rising Edge (using the bc635PCIcfg.exe pull down menu “Signals > Events”, and select External Input and Rising Edge).
5. Read the minor event time (using the bc635PCIcfg.exe pull down menu “Time > Get Event Time”).
6. Repeat the previous step to obtain an average value. You may have to average two neighboring values (e.g., the average of x.000012 and x.000013 is x.0000125).
7. Convert the minor event time to a calibration factor. If the minor event time is close to rolling over (i.e., x.999950), subtract 1 from the minor event time to get the calibration factor in microseconds (e.g., -50 uS or 50E-6 sec); otherwise the minor event time is the calibration factor (e.g., X.000012 = 12 uS or 12E-6 sec).
8. Convert the units of the calibration factor measured in the previous step from microseconds to hundreds of nanoseconds by multiplying it by 10 (12 uS = 120 hundreds of ns). x hundreds of nanoseconds is equivalent to xE-7sec.
9. Set the Propagation delay on the card to the calibration factor in units of hundreds of nanoseconds as calculated in the previous step (using the bc635PCIcfg.exe pull down menu “Time > Set Prop Delay”).
10. Read the minor event time (using the bc635PCIcfg.exe pull down menu “Time > Get Event Time”), and verify that the minor time reading is .000000.

If the minor event time is greater than ±1 uS, adjust the calibration factor as necessary.

Note: Keep this calibration factor and the associated time code type (i.e., IRIG B AM CAL = +120E-7) in a safe place. If another time code type is to be used as a reference, a new calibration factor should be determined for that code.

1.3.9. Field Upgrade of Embedded Program

New versions of the bc63xPCI-V2 (identifiable by the status LED on the front panel) and the bc63xPCIe cards are upgraded over the host PC’s PCI or PCIe bus. Included on the product CD is the update program named bcUpgFmware.exe. This Windows based program is used to update the firmware in the TFP’s flash memory.

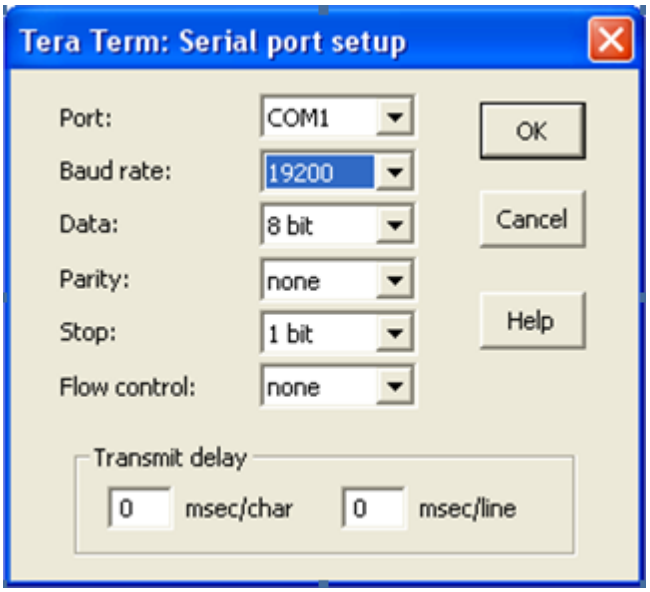


Older versions of the bc63xPCI-V2 cards (identifiable by the circular DIN connector on the front panel) use a serial port interface for firmware updates. TFP cards use flash memory, which is upgraded using an embedded boot loader program. The user transfers the program file that is in S Record format to the card via a serial port running a terminal program that supports a “text” file transfer format (e.g. TeraTerm or Hyperterminal). Insertion of the mating DIN connector is automatically detected which places the card in its boot load mode. The following procedure uses TeraTerm, an open source terminal emulator.

TeraTerm is configured as follows:

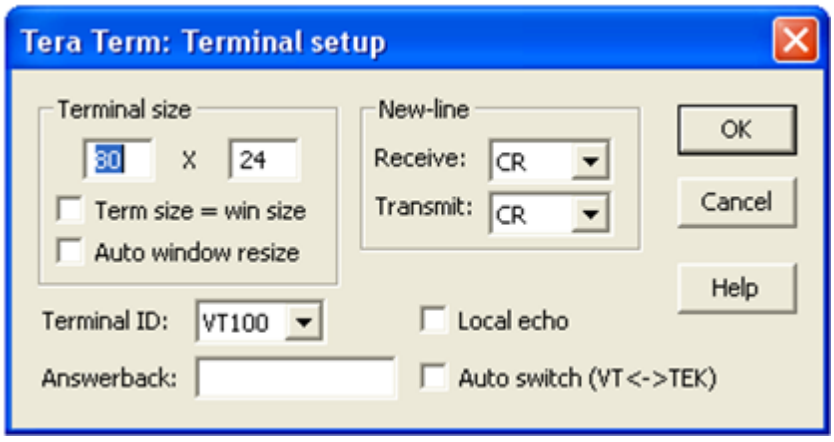
Setup/Serial port…

- Port:COMx
- Baud rate:19200
- Data:8 bit
- Parity:none
- Stop:1 bit
- Flow control:none
- Transmit delay:



Setup/Terminal...

- New-line
- Receive:CR
- Transmit:CR
- Local echo: not selected



After plugging in the DIN serial port adapter that is connected to a serial port running TeraTerm as configured above, the following string should be output to the terminal screen:

<E>rase , <P>rogram or <R>un Application:

1. Select <E> to erase the Flash, then select again after the message <Erased> has been returned.

2. Select <P>. Use File/Send file... (in Tera Term pulldown), a dialog box appears to select the file that is to be sent (e.g. bc635-v2.sx). The file should begin transferring with a string of asterisks being sent back to the terminal screen until the file transfer is complete (approximately 408KB). It takes approximately 5 minutes to download the new file.
3. Select <R> to run the application.

Note: The application is set up to run automatically. However, if the serial cable is connected to the card, but not to the host computer, the application will hang, awaiting response from the host. If the application does not run automatically, disconnect the serial cable from the card.

1.4. Device Registers

1.4.1. General

The Time & Frequency Processor (TFP) is controlled by a combination of hardware device registers and a dual-port RAM interface. This chapter describes the TFP device registers. "1.5. Dual-Port RAM Interface" on page 36 describes the dual-port RAM interface.

1.4.2. PCI Memory Map

The TFP is divided into two 2K memory spaces, Dual-Port RAM and Register. The following table lists the card's physical memory map.

TFP Physical Memory Map

Start	Type	Size	Register Field
PCI Auto	R/W	0x800	Dual Port RAM
PCI Auto	R/W	0x50 (0x800 reserved)	Device Registers

1.4.3. Device Register Description

The TFP device registers are 32-bits wide (PCI word size). For many of the registers, only a few of the bits have any significance while the rest of the bits are ignored during writes and are meaningless during reads. Registers may be:

- Read only (R)
- Write only (W)
- Read/write (R/W)
- Access (A)

Access type registers perform a function simply by being read or written without regard to the data contents. It is best to use a write operation with the access type registers because most optimizing compilers will remove statements that read a register but do nothing with the data returned. In some cases, a read/write register is structured to support dissimilar data in the read and write directions. The following table summarizes the type of register located at each offset and provides a brief description of the register function.

TFP Device Register Summary

Offset	Type	Reset	Label	Description
0x00	A	See Note	TIMEREQ	Time Request (TIME0-1)
0x04	A	See Note	EVENTREQ	Event Request (EVENT0-1)
0x08	A	See Note	UNLOCK1	Release Lockout Event1
0x0C	A	See Note	UNLOCK2	Release Lockout Event2
0x10	R/W	0	CONTROL	Control Register
0x14	R/W	See Note	ACK	Acknowledge Register
0x18	R/W	0	MASK	Interrupt Mask
0x1C	R/W	0	INTSTAT	Interrupt Status
0x20	R/W	See Note	MINSTRB	Minor Strobe Time
0x24	R/W	See Note	MAJSTRB	Major Strobe Time
0x28	R	See Note	EVENT2_0	Minor Event2 Register
0x2C	R	See Note	EVENT2_1	Major Event2 Register
0x30	R	See Note	TIME0	Minor Time Holding Register
0x34	R	See Note	TIME1	Major Time Holding Register
0x38	R	See Note	EVENT0	Minor Event1 Register
0x3C	R	See Note	EVENT1	Major Event1 Register
0x40	Reserved			
0x44	A	See Note	UNLOCK3	Release Lockout Event3
0x48	R	See Note	EVENT3_0	Minor Event3 Register
0x4C	R	See Note	EVENT3_1	Major Event3 Register

Note:Register contents are undefined at reset.

TIMEREQ Register (0x00)

Accessing this register (with a read or write operation) latches the current time and timing status in the TIME0 - TIME1 registers. The data value transferred is meaningless.

EVENTREQ Register (0x04)

Accessing this register (with a read or write operation) latches the current time and timing status in the EVENT0 - EVENT1 registers. The data value transferred is meaningless. Accessing the EVEN-TREQ register does not generate an Event Input interrupt.

UNLOCK1 Register (0x08)

Accessing this register (with a read or write operation) releases the EVENT1 time capture lockout function if it has been enabled, allowing the Event Input or Periodic/DDS Output to capture a new time. The Event Time Capture Lockout is enabled using bit 0 in the CONTROL Register (LOCKEN1).

UNLOCK2 Register (0x0C)

Accessing this register (with a read or write operation) releases the EVENT2 time capture lockout function if it has been enabled, allowing a new event to be captured. The Event2 Time Capture Lockout is enabled using bit 8 in the CONTROL Register (LOCKEN2).

CONTROL Register (0x10)

This register controls a variety of TFP hardware functions. The following table lists the function of each bit in this register.

CONTROL Register

Bit	Name	Function
0	LOCKEN 1	EVENT1 Capture Lockout Enable 0 = Disable Lockout 1 = Enable Lockout
1	EVSOURCE	EVENT1 Time Capture Register Source Select 0 = Event Input 1 = Periodic/DDS (Select Active Edge With EVSENSE)
2	EVSENSE1	Event1 Edge Select 0 = Rising 1 = Falling
3	EVENTEN1	Event1 Capture Register Enable 0 = Disable 1 = Enable (Use EVSOURCE to Select Event Source)
4	STREN	Time Coincidence Strobe Output Enable 0 = Disable (Strobe Output is Held Low) 1 = Enable
5	STRMODE	Time Coincidence Strobe Mode 0 = Use Major and Minor Time for Strobe Function 1 = Use Minor Time Only for Strobe Function If STRMODE = 1, an Output Strobe is Produced Each Second
6	FREQSEL0	Output Frequency Select 00 = 10MHz 01 = 5MHz 1X = 1MHz
7	FREQSEL1	Output Frequency Select 00 = 10MHz 01 = 5MHz 1X = 1MHz
8	LOCKEN2	EVENT2 Capture Lockout Enable 0 = Disable Lockout

Bit	Name	Function
		1 = Enable Lockout
9	EVSENSE2	Event2 Edge Select 0 = Rising 1 = Falling
10	EVENTEN2	Event2 Capture Enable 0 = Disable 1 = Enable
11	Reserved	
12	LOCKEN3	EVENT3 Capture Lockout Enable 0 = Disable Lockout 1 = Enable Lockout
13	EVSENSE3	Event3 Edge Select 0 = Rising 1 = Falling
14	EVENTEN3	Event3 Capture Enable 0 = Disable 1 = Enable
15 - 31	Reserved	

Note: Register contents are undefined at reset.

Examples: 0x08 enables event1, 0x400 enables event2, 0x4000 enables event3.

The EVSOURCE bit selects one of two signal sources for capturing time in the EVENT1 registers; either the Event Input signal from the Signal I/O connector or the PPO/DDS Output. When PPO/DDS is selected as the Event Source, the PPO/DDS and Event Input are internally connected, eliminating the need for an external physical connection.

The EVENTENx bit is used to enable capture of time into the respective EVENTx registers.

The EVSENSEx bit controls the active edge that is used to capture time into the respective EVENTx registers.

Enabling the Lockout function via the LOCKENx bit allows only the first instance of the selected signal source to latch time in the respective EVENTx registers, locking out any subsequent events. Use the UNLOCKx register (0x08, 0x0C, 0x44) to re-arm the respective circuits.

ACK Register (0x14)

The ACK register is used to prevent dual-port RAM data contention when the same address on both sides of a dual-port RAM is accessed simultaneously. See "Command 0x14: Set Periodic Output" on

page 43 for more information on the format and use of this register.

MASK Register (0x18)

The TFM supports the seven interrupt sources listed in the following table. Each interrupt source can be individually masked on or off using the MASK register (0x18). Each interrupt source sets a corresponding bit in the INTSTAT register (0x1C) when the interrupt occurs.

Bits 0-6 in the MASK register correspond to interrupt sources zero through six listed in the following table. An interrupt source is enabled (to generate a PCI interrupt) by writing a value of one to the corresponding MASK bit. Writing a zero to the interrupt MASK bit disables that interrupt source.

INTSTAT Register (0x1C)

The INTSTAT register has the same structure as the MASK register listed in the following table. Each interrupt source sets its corresponding bit in this register when it occurs. The INTSTAT register bits get set regardless of the state of the MASK bits. INTSTAT bits are cleared by writing to the INTSTAT register with the corresponding bit(s) set. For example, to clear INTSTAT bit zero, write 0x01 to the INTSTAT register. To clear all INTSTAT bits simultaneously, write 0x7F to the INTSTAT register. The corresponding INSTAT bit MUST be cleared in order to enable the next interrupt occurrence.

A PCI interrupt is generated anytime one or more INTSTAT bits [0 through 6] are set and the corresponding bit(s) are set in the MASK register and interrupts have been enabled (started).

INTSTAT Register

Bit	Function
0	Event Input has occurred
1	Periodic/DDS Output has occurred
2	Strobe (time coincidence) has occurred
3	1 PPS output has occurred
4	GPS Data Packet is available (bc637 models only)
5	Signal transition on Event2 Input has occurred
6	Signal transition on Event3 Input has occurred
7-31	Reserved

Note: Register contents are undefined at reset.

MINSTRB (0x20) – MAJSTRB (0x24) Registers

These registers hold the programmed Time Coincidence Strobe time. The contents of these registers depend on the time format selected. The Strobe time is programmable from hours through microseconds in the decimal time format. When the time format is set to binary, only the 22 least sig-

nificant bits of the major time are used (in addition to microseconds), this allows the user to program the Strobe to become activated up to 48 days beyond the current time.

Note: Disable the Strobe output (see CONTROL register) while programming the Strobe time to prevent spurious Strobe output pulses.

EVENT2_0 (0x28) – EVENT2_1 (0x2C) Registers

These read only registers hold the minor time (EVENT2_0) and major time (EVENT2_1) captured from an event on the EVENT2 input. The contents of these registers depend on the time format selected (see Time Format).

TIME0 (0x30) - TIME1 (0x34) Registers

These read only registers hold time captured by an access of the TIMEREQ register (0x00) providing a software time capture. The contents of these registers depend on the time format selected (see Time Format).

EVENT0 (0x38) - EVENT1 (0x3C) Registers

These read only registers hold time captured when the EVENTREQ register is accessed (0x04) by an Event Input (if enabled), or a PPO/DDS signal is generated (if enabled). The contents of these registers depend on the time format selected (see Time Format).

UNLOCK3 Register (0x44)

Accessing this register (with a read or write operation) releases the EVENT3 time capture lockout function if it has been enabled, allowing a new event to be captured. The Event3 Time Capture Lockout is enabled using bit 12 in the CONTROL Register (LOCKEN3).

EVENT3_0 (0x48) – EVENT3_1 (0x4C) Registers

These read only registers hold the minor time (EVENT3_0) and major time (EVENT3_1) captured from an event on the EVENT3 input. The contents of these registers depend on the time format selected (see Time Format).

1.4.4. TIME FORMAT

The TFP major time registers (TIME1, EVENT1, Event2_1, Event3_1, MAJSTRB) support binary time (Table 2) and decimal time (Table 3) formats. Sub-second time (minor time) is always represented in binary format. The 32-bit binary format represents time as the number of seconds since midnight, January 1, 1970 UTC (Universal Time Coordinated), which is the standard time format found on most UNIX systems. Note that the year field is stored in the dual-port RAM. The decimal time format is derived from the “struct-tm” format used on UNIX systems. The bottom numbers in

each cell in Table 2 and Table 3 define the bit positions for each data field. All undefined bit positions in Table 3 are N/A.

Table 2. TFP Binary Time Format

Register	Data Bits					
	31-28	27-24	23-20	19-16	15-8	7-0
TIME1 EVENT1 EVENT2_1 EVENT3_1	Major Time UNIX Seconds 31 - 0					
TIME0 EVENT0 EVENT2_0 EVENT3_0	N/A 31 - 28	Status 27 - 24	100 ns 23 - 20	Binary microseconds 19 - 0		
MAJSTRB	N/A 31 - 22		Major Time UNIX Sec (22 bits LSB) 21 - 0			
MINSTRB	N/A 31 - 28	Status 27 - 24	N/A 23 20	Binary microseconds 19 - 0		

Table 3. TFP Decimal Time Format

Register	Data Bits					
	31 - 28	27-24	23-20	19-16	15-8	7-0
TIME1_ EVENT1 EVENT2_1 EVENT3_1	Days (0-366) Bits 7 - 0 31 - 24		Hours (0 - 23) 20 - 16		Min (0 - 59) 12 - 8	Sec (0 - 59) 5 - 0
TIME0_ EVENT0 EVENT2_0 EVENT3_0	Days Bit 8 28	Status 27 - 24	100 nS 23 - 20	Binary microsecond 19 - 0		
MAJSTRB	N/A 31 - 24		hours (0 - 23) 20 - 16		Min (0 - 59) 12 - 8	Sec (0 - 59) 5 - 0
MINSTRB	N/A 31 - 28	Status 27 - 24	N/A 23 - 20	Binary microseconds 19 - 0		

The format of the minor time registers (TIME0, EVENT0, Event2_0, Event3_0, MINSTRB) is always binary, 20 bits of binary microseconds (0 - 999,999) in the lower part of the registers with an additional four bit field of hundreds of nanoseconds (0 - 9) located in bits 20 - 23. Most UNIX time functions use microseconds, but the TFP maintains time to hundreds of nanoseconds.

STATUS BITS

The TFP Status bits found in the minor time registers are summarized in the following table and are described below. Bits 24, 25 and 26, in the PCI Windows configuration program, are represented as LEDs labeled:

- Tracking (T)
- Phase (P)
- Frequency (F)

(0 = Green, 1 = Red).

Status Bits Summary

Bit	Description
24	Tracking (T) 0: Locked To Selected Reference 1: Flywheeling (Not Locked)
25	Phase (P) 0: < X Microseconds 1: > X Microseconds X = 5 (Mode 0) X = 2 (All Other Modes)
26	Frequency (F) 0: < 5 x 10-8 1: > 5 x 10-8
27	Reserved

STATUS: Tracking (Bit 24)

This bit indicates that the TFP is not tracking the reference time source, usually because the time source has been lost or has become unusable. When a timing Mode change occurs, this bit is set until the TFP locks to the new timing source.

STATUS: Phase (Bit 25)

This bit indicates the synchronization accuracy of the TFP relative to the timing source. This bit is updated approximately once per second. When the TFP's oscillator is synchronized to less than 5 microseconds with AM time code mode as a reference and less than 2 microseconds in other modes, this bit is cleared.

STATUS: Frequency Offset (Bit 26)

This bit is an indication of the TFP on-board oscillator frequency offset relative to the timing source. This bit reflects the short-term stability of the TFP's oscillato

1.5. Dual-Port RAM Interface

1.5.1. General

The byte-wide dual-port RAM (DPRAM) interface provides a communications pathway between the user and the Time & Frequency Processor (TFP) micro-controller (MPU). The RAM size is 2Kx8. The ACK register is used in conjunction with the DPRAM to avoid data contention when a memory location is accessed simultaneously from both sides of the DPRAM. Four areas within the DPRAM are available to the user:

Input Area

This area is used for sending commands to the TFP to set the timing mode, time code format, etc. This area is also used to send data packets to the optional GPS receiver.

Output Area

This area holds data that the user requests from the TFP.

GPS Area

This area holds packets of data from the optional GPS receiver such as position, velocity, GPS status, etc.

Year Area

This area holds the year number derived from the timing source (if available). The year value is restored after a power cycle.

DPRAM Address and Contents

Data	Size	Offset
Year Area	2 bytes	0x00
GPS Area	0x80 bytes	0x02
Output Area	0x80 bytes	0x82
Input Area	0x80 bytes	0x102

.5.2. ACK Register

This register is used to prevent dual-port RAM data contention when the same address on both sides of a dual-port RAM is accessed simultaneously. Only three bits in this register are used, and each bit operates independently. The function of each bit in this register is described below.

ACK Bit 0

Set by the TFP to acknowledge the receipt of a user command from the DPRAM Input Area. The user can clear this bit by writing to the ACK register with bit 0 set, but cannot set this bit.

ACK Bit 2

Set by the TFP to indicate that a GPS packet is available in the DPRAM GPS Packet Area. The user can clear this bit by writing to the ACK register with bit two set, but cannot set this bit. The transition of this bit from zero to one activates interrupt source four.

ACK Bit 7

The user writes to the ACK register with bit seven set to cause the TFP to read a command from the DPRAM Input Area. This bit has no meaning when read.

1.5.3. TFP DPRAM Commands

This section describes the TFP commands available through the DPRAM Interface. Commands consist of a command ID byte followed by zero or more data bytes. The command ID byte is written to the first location in the DPRAM Input Area, followed by the command data byte(s). The following command data types are used. Command data is loaded into the DPRAM in the Big-Endian fashion, most significant byte first. The following table summarizes the DPRAM commands.

DPRAM Commands

UINT8	Unsigned 8 Bit Integer (1 Byte)
INT8	Signed 8 Bit Integer (1 Byte)
UINT16	Unsigned 16-Bit Integer (2 Bytes)
INT16	Signed 16-Bit Integer (2 Bytes)
UINT32	Unsigned 32-Bit Integer (4 Bytes)
INT32	Signed 32-Bit Integer (4 Bytes)
FLOAT	ANSI / IEEE Std 754 Standard Floating-Point Format (4 Bytes)
DOUBLE	ANSI / IEEE Std 754 Standard Floating-Point Format (8 Bytes)

The following steps should be followed when sending commands to the TFP.

1. Write the command ID and data bytes to the DPRAM starting at the first location in the Input Area.
2. Clear bit zero of the ACK register by writing 0x01 to the ACK register.
3. Inform the TFP that a command is waiting by writing 0x80 to the ACK register.
4. Wait for the TFP to set bit 0 of the ACK register. Do not begin writing another command to the Input Area until this bit becomes set.

Windows example (uses bc635PCIcfg.exe GUI):

Set the Periodic rate output to 10KPPS and Synchronous Mode.

Select the Advanced Menu.

Use the Special Tab and the DP RAM selection from the menu list.

a. Offset 102 (input packet base address)

b. Value 14 (14 is the set periodic output command ID)

c. select the WRITE button

d. Offset 103

e. Value 1 (value of 1 here selects Synchronous Mode)

f. select the WRITE button

g. Offset 104

h. Value 0 (msb of n1 value)

i. select the WRITE button

j. Offset 105

k. Value a (lsb of n1 value set to decimal 10)

l. select the WRITE button

m. Offset 106

n. Value 0 (msb of n2 value)

o. select the WRITE button

p. Offset 107

q. Value a (lsb of n2 value set to decimal 10)

r. select the WRITE button

s. select the ACK button and the Periodic output will be set.

DPRAM Command Summary

ID	Reset	Command
0x10	NV	Set TFP Timing Mode
0x11	1	Set Time Register Format (1 = binary, 0 = bcd)
0x12	RTC	Set Major Time
0x13	RTC	Set Year
0x14	NV	Set Periodic Output
0x15	NV	Set Input Time Code Format
0x16	NV	Set Input Time Code Modulation
0x17	NV	Set Propagation Delay Compensation
0x18	NV	Request UTC Time Data (bc637 only)
0x19	N/A	Request TFP Data
0x1A	N/A	Software Reset
0x1B	NV	Set Time Code Output Format
0x1C	NV	Set Generator Time Offset
0x1D	NV	Set Local Time Offset
0x1E	0	Set Leap Second Event
0x1F	N/A	Request Firmware information
0x20	NV	Set Clock Source (0 = Internal, 1=External)
0x21	1	Control Jamsync
0x22	N/A	Force Jamsync
0x24	NV	Load DAC
0x25	N/A	Set Disciplining Gain
0x26	N/A	Battery Connection Status
0x27	N/A	Synchronize RTC to TFP Time
0x28	N/A	Battery Connection Control
0x30	N/A	Send Packet to GPS Receiver (bc637 only)
0x31	N/A	Request Packet from GPS Receiver (bc637 only)
0x32	N/A	Manual Request GPS Packet (bc637 only)
0x33	0	Select GPS Time Format (bc637 only)
0x40	NV	Observe Local Time Flag
0x41	NV	IEEE 1344 Daylight Saving and Local Time Flags
0x43	NV	Periodic/DDS Select (0=Periodic, 1=DDS)
0x44	NV	Periodic/DDS Enable (0 = OFF, 1=ON)
0x45	NV	DDS Divide Select
0x46	NV	DDS Divide Source
0x47	NV	DDS Sync Mode
0x48	NV	DDS Multiplier
0x49	NV	DDS Period Value
0x4A	NV	DDS Tuning Word
0x4F	N/A	Request PCI Firmware Part Number
0xF6	N/A	Request TFP Model Identification
0xFE	N/A	Request TFP Serial Number (Request only)

- RTC: Real Time Clock (Restored at Power-on)
- NV: Non Volatile (Restored at Power-on)

Command 0x10: Set TFP Timing Mode

This command selects the timing mode of the TFP.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x10
1	UINT8	Timing Mode	See Below

TFP Timing Mode:

- 0x00 Time Code (Selected Time Code AM or DCLS)
- 0x01 Free Running (Internal or External 10 MHz Reference Selected)
- 0x02 1 PPS (External One Pulse Per Second)
- 0x03 Real Time Clock (Battery backed RTC)
- 0x06 GPS (bc637 only)

Command 0x11: Set Time Register Format

This command allows the user to select the major time format. Available time formats are Binary Coded Decimal (BCD) and UNIX (Binary). The time format affects the TIMEx, and EVENTx registers and Command 0x12. See Table 1 and Table 2 for the UNIX and BCD time register definitions, respectively.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x11
1	UINT8	Data Format	See Below

Time Data Format:

- 0x00 BCD TIME FORMAT
- 0x01 UNIX TIME FORMAT (default)

Command 0x12: Set Major Time

This command allows the user to load the major time to the TFP Major Time Registers in binary (UNIX) or BCD format. The format is determined by Command 0x11 as referenced above. The default major time format is UNIX binary time. This command normally applies to the TFP while in time modes 1, 2 or 3. The TFP derives its major time from the selected external timing reference signal in time modes 0 and 6, and from the RTC in mode 3. If time mode 0 or 6 is used, any major time written by this command will be overwritten when the selected source is providing a valid time to the TFP.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x12

Case 1: UNIX Time Data Format = 0x01 (Command 0x11, format 0x01) default

Byte	Type	Item	Value or Range
1-4	UINT32	UNIX Time	0 to 0xffffffff

Case 2: BCD Time Data Format = 0x00 (Command 0x11, format 0x00)

Byte	Type	Item	Value or Range
0	UINT8	ID	0x12
1-2	UINT16	Year	1970 - 2036
3-4	UINT16	Days	0 to 0x16e (0 to 366)
5	UINT8	Hours	0 to 0x17 (0 to 23)
6	UINT8	Minutes	0 to 0x3b (0 to 59)
7	UINT8	Seconds	0 to 0x3b (0 to 59)

The time loaded by this command will not be readable until the one-second epoch following the load. There is a possibility the TFP will have incremented the time during the load. To prevent ambiguities in the time, the user must issue this command in advance of the 800-millisecond point within the one-second epoch, referencing the current epoch.

This command normally applies to the TFP modes 1 and 2. The TFP derives its major time from the timing reference signal in other modes. The format data of this command depends on the Command 0x11 time format selection.

Command 0x13: Set Year

This command allows the user to set the year.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x13
1	UINT16	Year	1970 - 2036

Command 0x14: Set Periodic Output

This command establishes the frequency of the TFP Programmable Periodic Output. "1.3.2. Heartbeat Output" on page 18 describes the relationship between the dividers n1, n2 and the Periodic Output frequency.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x14
1	UINT8	Sync Flag	0 = Don't Sync To 1PPS 1 = Sync To 1 PPS
2-3	UINT16	Divider n1	2 - 65535
4-5	UINT16	Divider n2	2 - 65535

Command 0x15: Set Input Time Code Format

This command selects the time code format for TFP Timing Mode “0” time code input. (See Command 0x10.) Use Command 0x16 to set the modulation type. Note that this selection is restored at power-up from NV memory.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x15
1	UINT8	Format1	See below
2	UINT8	Format2	See below

Format Choices:

Format1	Format2	ASCII	AM	Time Code	DCLS
0x41	0x0	'A'	A130,1,2,3	IRIG A no year	A000,1,2,3
0x41	0x59	'AY'	A134,5,6,7	IRIG A w/year	A004,5,6,7
0x42	0x0	'B'	B120,1,2,3	IRIG B no year	B000,1,2,3
0x42	0x59	'BY'	B124,5,6,7	IRIG B w/year	B004,5,6,7
0x45	0x0	'E'	E121,2	IRIG E 1K no year	E001,2
0x45	0x59	'EY'	E125,6	IRIG E 1K w/year	E005,6
0x65	0x0	'e'	E111,2	IRIG E 100 no year	E001,2
0x65	0x59	'eY'	E115,6	IRIG E 1K w/year	E005,6
0x47	0x0	'G'	E141,2	IRIG G no year	G001,2
0x47	0x59	'GY'	E145,6	IRIG G w/year	G005,6
0x42	0x54	'BT'	AM	IRIG B TrueTime	DCLS
0x49	0x0	'I'	AM	IRIG B IEEE 1344	DCLS
0x4e	0x0	'N'	AM	NASA 36	DCLS
0x58	0x0	'X'	AM	XR3 (250 Hz)	DCLS
0x32	0x0	'2'	AM	2137 (1 kHz)	DCLS

Note that AM time codes with carrier frequencies less than 1 KHz (IRIG E111, 2, 5, 6 @ 100 Hz and XR3 @ 250 Hz) will decode properly but will not necessarily maintain short term phase and frequency lock. Long term, the low frequency carrier time codes maintain phase and frequency with reduced accuracy. Phase and frequency accuracy is better when using DCLS time codes.

Note: that when Legacy TrueTime IRIG B is used, the TFP will decode the “Lock” bit that is encoded in the Control Function area and will not lock to the incoming code if the bit = 1 (an indication of unlock from the input reference).

Command 0x16: Set Input Time Code Modulation Type

This command selects the time code modulation type format for TFP Timing Mode “0” time code input (See Command 0x10). Use Command 0x15 to select the time code format.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x16
1	UINT8	modulation	See below

Modulation Choices:

- 0x4D ('M') amplitude modulated sine wave (AM)
- 0x44 ('D') DC level shift (DCLS)

Command 0x17: Set Propagation Delay Compensation

It is sometimes desired to program an offset into the basic TFP time keeping functions relative to the reference input. For example, if the reference input is an IRIG B time code, there may be significant cable delay between the IRIG B generator and the TFP location. This command allows this time difference to be removed by inserting the known amount of offset between the IRIG B reference and TFP location. The offset is programmable in units of 100 nanoseconds, and may be positive or negative.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x17
1-4	UINT32	offset	-4,000,000 to +4,000,000

Command 0x18: Request UTC Time Data (bc637 only)

This command queries current GPS and UTC time information derived from the GPS receiver. This command must be used in conjunction with Command 0x19.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x18
1	UINT8	GPS Time Format	See below
2	UINT8	Leap Second	0 to 0xff
3	INT8	Leap Second Flag	See below
4-7	UINT32	Leap Event UNIX Time	0 to 0xffffffff

GPS Time Format:

- 0x00 UTC Time (GPS time plus leap seconds)
- 0x01 GPS Time

Leap Second Flag:

- 0xff Deletion Event
- 0x00 No Event
- 0x01 Addition Event

Command 0x19: Request TFP Data

This command requests data from the TFP that is not available via the device registers. The TFP transfers the requested data to the DPRAM Output Area. The data is available to the user as soon as the TFP sets ACK bit 0.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x19
1	UINT8	Req. data type	(See Below)

Requested Data Type Choices

0x10 Timing Mode
0x11 Timing Format
0x13 Current Year
0x14 Periodic Output (Sync Flag only)
0x15 Time Code Format
0x16 Time Code Modulation
0x17 Propagation Delay
0x18 UTC Time Data (bc637)
0x1B Time Code Output Format

0x1C Generator Time Offset
0x1D Local Time Offset
0x1E Leap Second Setting
0x1F TFP Firmware Information
0x20 Clock Source (on-board/external)
0x21 Jamsync Control
0x24 DAC Value
0x26 Battery Connection Status
0x4F FW Revision
0xF4 Assembly Number
0xF5 Hardware Revision
0xF6 TFP Model
0xFE Serial Number

Command 0x1A: Software Reset

This command vectors the TFP MPU to its power-on reset point and contains no data.

Command 0x1B: Set Time Code Output Format

This command allows the user to select the time code format that is generated by the TFP on J1. Note that this selection is restored at power-up from NV memory.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x1B
1	UINT8	Format1	See below
2	UINT8	Format2	See below

Time Code Output Formats:

Format1	Format2	ASCII	AM	Time Code	DCLS
0x41	0x30	'A0'	A130	BCD, CF, SBS	A000
0x41	0x31	'A1'	A131	BCD, CF	A001
0x41	0x32	'A2'	A132	BCD	A002
0x41	0x33	'A3'	A133	BCD, SBS	A003
0x41	0x34	'A4'	A134	BCD, YR, CF, SBS	A004
0x41	0x35	'A5'	A135	BCD, YR, CF	A005
0x41	0x36	'A6'	A136	BCD, YR	A006
0x41	0x37	'A7'	A137	BCD, YR, SBS	A007
0x42	0x30	'B0'	B120	BCD, CF, SBS	B000
0x42	0x31	'B1'	B121	BCD, CF	B001
0x42	0x32	'B2'	B122	BCD	B002
0x42	0x33	'B3'	B123	BCD, SBS	B003

Format1	Format2	ASCII	AM	Time Code	DCLS
0x42	0x34	'B4'	B124	BCD, YR, CF, SBS	B004
0x42	0x35	'B5'	B125	BCD, YR, CF	B005
0x42	0x36	'B6'	B126	BCD, YR	B006
0x42	0x37	'B7'	B127	BCD, YR, SBS	B007
0x42	0x0	'B'	B122	BCD	B002
0x42	0x54	'BT'	AM	Legacy TrueTime	DCLS
0x49	0x0	'I'	AM	IEEE 1344	DCLS
0x45	0x31	'E1'	E121	BCD, CF	E001
0x45	0x32	'E2'	E122	BCD	E002
0x45	0x35	'E5'	E125	BCD, YR, CF	E005
0x45	0x36	'E6'	E126	BCD, YR	E006
0x65	0x31	'e1'	E111	BCD, CF	E001
0x65	0x32	'e2'	E112	BCD	E002
0x65	0x35	'e5'	E115	BCD, YR, CF	E005
0x65	0x36	'e6'	E116	BCD, YR	E006
0x47	0x35	'G5'	G145	BCD, YR, CF	G005
0x4E	0x0	'N'	AM	Nasa 36	DCLS
0x58	0x0	'X'	AM	XR3	DCLS
0x32	0x0	'2'	AM	2137	DCLS

IRIG Control Function (CF) Bits In the following tables, tq1 through tq4 are time quality bits. For time quality and unlock bits, "1" means active.

For IRIG output codes A, B, E or G with CF bits, the following CF bits are encoded:

Index Bit
count name
----- -----
70 (0)
71 (0)
72 (0)
73 unlock (1 = unit not locked to reference)
74 (0)
75 tq1 (1 = timing error estimate > 1us)
76 tq2 (1 = timing error estimate > 10us)
77 tq3 (1 = timing error estimate > 100us)
78 tq4 (1 = timing error estimate > 1ms)

For Legacy TrueTime IRIG B, the following CF bits are encoded:

Index Bit
count name
----- -----
50 (0)
51 (0)
52 (0)
53 unlock (1 = unit not locked to reference)
54 (0)
55 tq1 (1 = timing error estimate > 1us)
56 tq2 (1 = timing error estimate > 10us)
57 tq3 (1 = timing error estimate > 100us)
58 tq4 (1 = timing error estimate > 1ms)

Note that when Legacy TrueTime IRIG B is selected, CF bits are encoded. The unlock bit may be used to inform time code readers of the generators lock status.

Command 0x1C: Set Generator Time Offset

This command is used to add/subtract an offset to the time code generator output. This command affects the generator output only.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x1C
1-2	UINT16	Local Offset	0xfff0 to 0x0010 (-16 to +16)
3	UINT8	Half Hour	0 or 1

HALF HOUR:

- 0 = half hour not present (30 min)
- 1 = half hour present (30 min)

Command 0x1D: Set Local Time Offset

This command adds/subtracts local time offset to the TFP time. This command affects the TFP time only; the generator time is not affected. (See Command 0x1C.)

Byte	Type	Item	Value or Range
0	UINT8	ID	0x1D
1-2	UINT16	Local Offset	0xfff0 to 0x0010 (-16 to +16)
3	UINT8	Half Hour	0 or 1

Half Hour:

- 0 = half hour not present (30 min)
- 1 = half hour present (30 min)

Command 0x1E: Program Leap Second Event

Byte	Type	Item	Value or Range
0	UINT8	ID	0x1E
1	INT8	LS_Flag	-1 to +1

Leap Second Flag:

- 1 = Insertion
- -1= Deletion (0xff)
- 0 = Disable

Command 0x1F: Request Firmware Information

Byte	Type	Item	Value or Range
0	UINT8	ID	0x1F
1	INT8	Major SW version	0x1 to 0x63 (1- 99)
2	INT8	Minor identifier	(Model dependant) *
3	INT8	SW release Month	0x1 to 0xC (1-12)
4	INT8	SW release Date	0x1 to 0x1F (1-31)
5	INT8	SW release YR (MS)	Year
6	INT8	SW release YR (LS)	Year

* Model dependant (x=Major version)

x.20 = bc635

x .21= bc637

x.22 = bc635 + OCXO

x.23 = bc637 + OCXO

Year Example:

Byte5 = 0x07, Byte6 = 0xd9 => 0x7d9 = 2009

Command 0x20: Select Clock Source

This command selects the clock source for the TFP. The TFP uses a time base frequency of 10 MHz. The 10 MHz may be derived from the on-board oscillator or it may be supplied from an external oscillator via the J1 connector.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x20
1	UINT8	clock source	see below

Clock Source Choices:

- 0x49 ('I') Internal 10 MHz Oscillator
- 0x45 ('E') External 10 MHz Oscillator

Command 0x21: Control Jamsync

This command can be used to disable TFP jam-syncs that normally occur automatically. The default is jamsync enabled.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x21
1	UINT8	jamsync ctrl	0 = jamsync disabled 1 = jamsync enabled

Command 0x22: Force Jamsync

This command forces the TFP to perform a single jamsync operation and contains no data. (See Command 0x21).

Command 0x24: Load DAC

The TFP's 10MHz oscillator frequency is voltage controlled using the output from a 16-bit DAC. This command allows the user to directly load a 16-bit value to the DAC. This feature allows the user to fine tune the TFP time base when operating in the Free Running Mode. This voltage is also routed out of the TFP via the J1 connector (pin 9) allowing external oscillators to be disciplined. The DAC output voltage ranges nominally from 0 V (value = 0x0000) to 5 V (value = 0xFFFF).

JP3 is used to select the DAC voltage range. When the 2mm jumper is OFF, the DAC voltage is 0-5 VDC; when ON the voltage is 0-10 VDC. Both on-board oscillators that are offered for this card use the 0-5 VDC control voltage range. If an external oscillator requires a 0-10 VDC control voltage range, a 2mm jumper must be placed on JP3.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x24
1-2	UINT16	DAC value	0x0000 - 0xFFFF

Command 0x25: Set Disciplining Gain

This command allows the gain and sense of the disciplining process to be set by the user. A positive gain indicates that the voltage-controlled oscillator source frequency increases with increasing control voltage. This feature is valuable to anyone using the TFP to discipline an external oscillator.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x25
1-2	INT16	gain	-100 to +100

Note: Use this command with caution, as it will affect the TFP disciplining routine.

Gain Default:

- 0x02 = TCXO
- 0x20 = OCXO

Command 0x26: Request Battery Connection Status

This request (used with command 0x19) reads the current RTC battery connection status which is either connected (0x1) or disconnected (0x0).

Byte	Type	Item	Value or Range
0	UINT8	ID	0x26
1-2	INT16	battstat	0x0, 0x1

Command 0x27: Synchronize RTC to External Time Data

This command forces the TFP to synchronize the RTC time to the current time. Note that when the TFP is locked to a timing reference, the RTC is adjusted automatically.

Command 0x28: RTC Battery Connection Control

This command controls the connection/disconnection of the RTC battery. The disconnect control may be used to keep the battery from being discharged while in storage. The next time the board is powered up, the battery is automatically reconnected to resume RTC battery backup power when the board power is removed. Connect=0x1, Disconnect=0x0.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x26
1-2	INT16	battcon	0x0, 0x1

Command 0x30: Send Packet to GPS Receiver (bc637 only)

This command allows the user to send a GPS packet to the GPS receiver. The format and use of this command is described in GPS Receiver Interface.

Command 0x31: Request Packet from GPS Receiver (bc637 only)

This command allows the user to request a GPS packet (i.e., position, velocity, status, etc.) from the GPS receiver. The format and use of this command is described in GPS Receiver Interface.

Command 0x32: Manually Request Packet from GPS Receiver (bc637 only)

This command is similar in function to Command 0x31. Refer to GPS Receiver Interface.

Command 0x33: Set GPS Time Format (bc637 only)

This command allows the user to select between GPS time and UTC when using Timing Mode 6 (GPS). The relationship between UTC and GPS time is shown below. The default setting is UTC (UTC = GPS Time + Leap Seconds).

Byte	Type	Item	Value or Range
1	UINT8	ID	0x33
2-5	UINT8	GPS time format	flag

GPS time format flag:

- 0 = UTC (default)
- 1 = GPS Time

Command 0x40: Observe Local Time Flag

This command programs the local time observed flag. If the local time flag is enabled, the TFP adjusts its time by the local time offset. Note that the Generator Time is also affected by this setting. See Command 0x1d for programming local time offset.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x40
1	UINT8	Flag	0 or 1

Local Time Observe Flag:

- 0 = disable
- 1 = enable

Command 0x41: IEEE 1344 Daylight Saving and Local Time Flags

This command queries the daylight saving and local time observed flag. Additionally, this command is used to set the IEEE 1344 Time Code daylight saving observed flag. Use this command in conjunction with Command 0x19.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x41
1	UINT8	Flag	0x00 - 0xff

Flag:

- bit0 = reserve
- bit1 = reserve
- bit2 = reserve
- bit3 (0x8) = local time observe flag
- bit4 (0x10) = IEEE 1344 DST observe flag
- bit5 - bit7 = not used

Command 0x43: Select Periodic or DDS Output

This command selects the signal that is output on P1 pin 15. This output may be either the Periodic (heartbeat) signal or the DDS (frequency synthesizer) signal. Note that this selection is restored at power-up from NV memory.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x43
1	UINT8	Per/DDS	see below

- 0 = Periodic
- 1 = DDS

Command 0x44: Periodic or DDS Output Enable

This command controls the Periodic or DDS signal that is output on P1 pin 15. This output may be either on or off based on the selection.

Note: that this selection is restored at power-up from NV memory.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x44
1	UINT8	Dis/En	see below

- 0 = Disabled
- 1 = Enabled

Command 0x45: DDS Divide Select

The DDS frequency synthesizer's divider can be used to divide the selected input down to generate lower or fractional frequencies. Selectable decade divides that range from divide by 1E0 through divide by 1E7 are available. The divider will also allow for fractional frequency outputs where the divider's input source is decimal shifted by up to 7 places. The frequency synthesizer's divider is the output signal provided on P1 pin 15.

Note that this selection is restored at power-up from NV memory.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x45
1	UINT8	Div val	see below

- 0= divide by 1,
- 1= divide by 10,
- 2= divide by 100,
- 3= divide by 1000,
- 4= divide by 10000,
- 5= divide by 100000,
- 6= divide by 1000000,
- 7= divide by 10000000
- F = divide by value in Period Register (when selected, refer to command 49)

Command 0x46: DDS Divide Source

The frequency synthesizer's divide chain has 3 possible input sources.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x46
1	UINT8	Div sel	see below

- 0 =DDS
- 1 =Multiplier (DDS x multiplier)
- 2 =100 MHz (100 MHz PLL)

Command 0x47: DDS Synchronization Mode Select

The DDS frequency synthesizer's divider has 2 modes of operation, Fractional and Continuous. Fractional mode allows for fractional frequencies to be generated that are time synchronized only when a change is made to the DDS frequency but never again, allowing for non-integer rates. Continuous mode should be used for integer rates only where the frequency synthesizer and divider are synchronized each second. Note that integer frequency rates may use the fractional mode and it will remain on time if the unit does not adjust phase using a jamsync. Note that this selection is restored at power-up from NV memory.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x47
1	UINT8	sync sel	see below

- 0 = Fractional (synchronizes only once - allows fractional rates)
- 1 = Continuous (synchronizes every second - integer frequencies)

Command 0x48: DDS Multiplier Value

The DDS frequency synthesizer has the ability to multiply its output by 1, 2, 3, 4, 6, 8, 10 or 16. Note that the DDS frequency must be high or low enough for the multiplier to operate correctly (see Input Range in MHz below). Note that this selection is restored at power-up from NV memory.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x48
1	UINT8	mult sel	see below

Value		Input Range MHz
0x1 = DDS	x1	22 - 150
0x2 = DDS	x2	11 - 75
0x3 = DDS	x3	8 - 56
0x4 = DDS	x4	6 - 38
0x6 = DDS	x6	5 - 23
0x8 = DDS	x8	5 - 19
0xa = DDS	x10	5 - 15
0x10 = DDS	x16	5 - 10

Note that when using Multiplier Mode, the DDS resolution is reduced to 1/32 Hz x multiplication factor.

Command 0x49: DDS Period Value

The DDS divider has a selectable mode called Period Mode (command 0x45 value = F). This mode may be desirable when the DDS cannot be set to the exact frequency but a period value, based on the period of the DDS rate, will be exact. Note that this value is restored at power-up from NV memory.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x49
1-4	UINT32	Period val	0x0 to 0x00FFFFFF*

* (4 byte data, 3 lower bytes used)

Refer to section 1.3.3. Divider Mode for an example using the Period Mode divider.

Command 0x4A: DDS Tuning Word

The DDS frequency is set with this command. The desired frequency x32 = DDS Tuning Word. Frequencies higher than 22 MHz should be attained using Multiplier Mode (0x48, 0x46 commands). Note that this value is restored at power-up from NV memory.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x49
1-4	UINT32	DDS val	0x0 to 0x3FFFFFFF

Refer to section 1.3.3. DDS Output for details and an example on setting the DDS tuning word.

Note that the output of the DDS circuitry is capable of creating interrupts.

Command 0x4F: PCI Firmware Part Number (request only)

This command allows the user to request the TFP firmware part number. Use this command in conjunction with Command 0x19.

Byte	Type	Item	Value or Range
0	UINT8	ID	0x4f
1	UINT8	'P' 'P'	0x50
2	UINT8	'C' 'C'	0x43
3	UINT8	'I' 'I'	0x49
4	UINT8	'-' 'E'	0x2d or 0x45
5	UINT8	'v'	0x76
6	UINT8	'2'	0x32

Command 0xF6: TFP Model Identification (request only)

This command queries the PCI family TFP part number. Use this command in conjunction with Command 0x19. Both the PCI-V2 and PCIe cards are shown.

Byte	Type	Item	Value or Range
0	UINT8	ID	0xf6
1	UINT8	Model	'B'
2	UINT8	Model	'C'
3	UINT8	Model	'6'
4	UINT8	Model	'3'
5	UINT8	Model	'5' or '7'
6	UINT8	Model	'P'
7	UINT8	Model	'C'
8	UINT8	Model	'I'

Model:

- “BC635PCI” = Time Code (same for bc635PCIe)
- “BC637PCI” = Time Code and GPS (same for bc637PCIe)

Command 0xFE: TFP Serial Number (request only)

This command queries the TFP serial number. Use this command in conjunction with Command 0x19.

Byte	Type	Item	Value or Range
0	UINT8	ID	0xf7
1-4	UINT32	SN	0x00 - 0xffffffff

1.6. Inputs and Outputs

1.6.1. TFP I/O Connector Signals

The TFP products have the following signals connected to their 15 pin D-sub connectors. Refer to Table 4 for the pin out assignments. Note that a few of these pins are dual-purposed with the functions separated with a" /", see pins 10, 14, and 15 below.

Figure 1-10

Table 4: Signal I/O Connector

Pin	Direction	Signal
1	input	External 10 MHz input
2	n/a	Ground
3	output	Strobe output
4	output	1 PPS output
5	output	Time Code output (AM)
6	input	External Event input
7	input	Time Code input (AM)
8	n/a	Ground (Recommended Time Code return)
9	output	Oscillator Control Voltage output
10	input	Time Code input (DCLS) / **Event2 input**
11	output	Time Code output (DCLS)
12	n/a	Ground
13	output	1, 5, 10 MHz output
14	input	External 1 PPS input / **Event3 input**
15	output	Periodic / **DDS output**

1.6.2. bc635PCIe and bc637PCIe Accessories

Signal Breakout Kit

The Symmetricom model bc63x PCI/PCIe Signal Breakout Board is a tool for use with the bc635PCI-V2, bc637PCI-V2, bc635PCIe, and bc637PCIe Time and Frequency Processors (TFP) boards.

The Signal Breakout Boards provide a user-friendly interface by providing BNC connections to most of the I/O signals with the exceptions being the 10 MHz external oscillator input and the external oscillator voltage control output, which are available on a 3-pin Molex connector. The 3-pin connector on the breakout board is Molex part number 0022232031 (2.54mm (.100") Pitch KK® Solid Header, Vertical, with Friction Lock). This connector mates with Molex part number 0022012031 (2.54mm (.100") Pitch KK® Crimp Terminal Housing, 3 Circuits) using Molex part number 0008500113 pins (KK® Crimp Terminal, 22-30 AWG), not supplied.

The Breakout Kit consists of 2 PC boards where part number 089-00133-000 connects to the TFP via a male 15-pin D-sub connector. This board provides a separation between the TFP's high frequency signals and the lower frequency signals. The 089-00133-000 board has BNC connectors for the Periodic/DDS output and the 1, 5, 10 MPPS output as well as a 3-pin Molex interface for the 10 MHz external oscillator input and the external oscillator voltage control output. This board also provides an interface to the other 8 signals via a male 9-pin D-sub that connects to part number 089-00135-000 via a 9-pin extension cable.

The 089-00135-000 board provides 8 BNC connections to the lower repetition rate/frequency signals from the TFP. This board can be mounted in a 19" rack enclosure or may be cut to fit into a ½ rack system. Please note that the 9-pin extension cable provided with the kit (6 feet) has been selected as the optimum length interconnect between the 2 breakout boards. The fast edge rate from the digital drivers on the TFP board coupled through longer than 6 foot lengths may produce undesirable effects on the TFP's digital input lines.

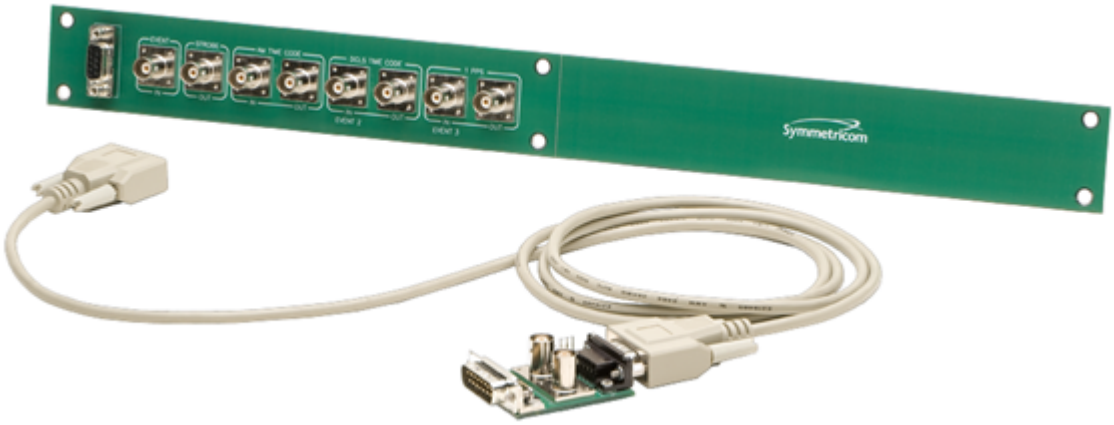


Figure 1-11 Large Breakout Board (top), and Small Breakout Board (bottom)

Note: the cable connecting the two breakout boards is six feet long.

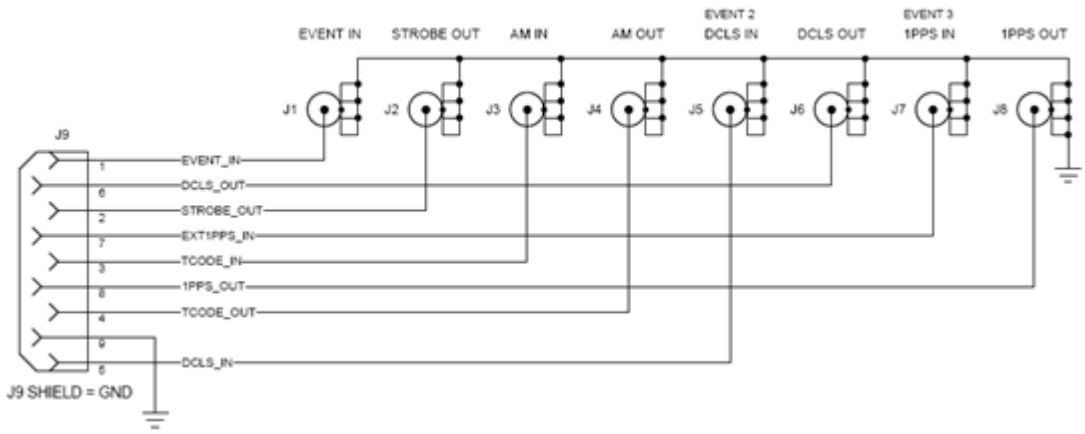


Figure 12: Large Breakout Board schematic

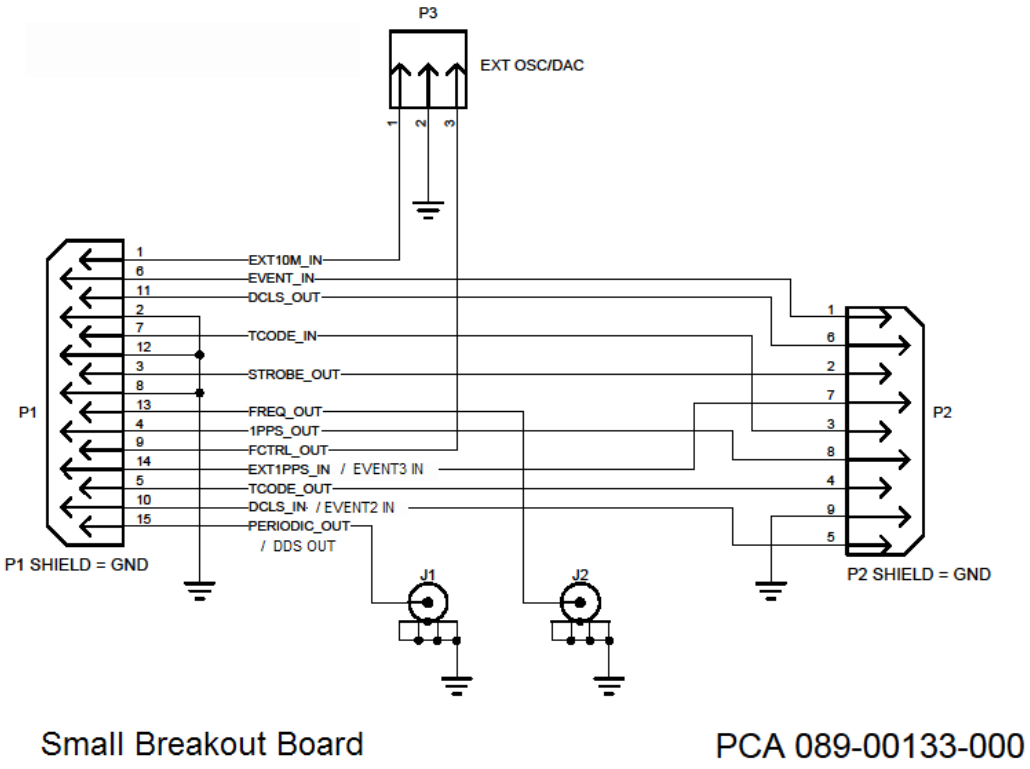


Figure 1-13 Small Breakout Board schematic

Breakout Cables

Breakout cables with connectors provide simple access to the in and out timing signals of the PCIe card. These labeled cables mitigate the need to create special cables during project develoment and assure the correct timing signals are being accessed. There are two forms of breakout cable as follows.

"D" to 6-BNC Breakout Cable, see cable (C) below for signal information

"D" to 5-BNC Breakoput Cable, see cable (A and B) below for signal information

Timing Input/Output Breakout cable and Patch Panel BNC Map

	"D" to 5-BNC (A)	"D" to 5-BNC (B)	"D" to 6-BNC (C)	Patch/ Breakout
Outputs				
Time Code (AM)	X	X	X	X
Time Code (DCLS)			X	X
1, 5, 10 MHz				X
Heartbeat/DDS	X			X
Strobe		X		X
1 PPS	X	X	X	X
Oscillator Control Voltage				X
Inputs				X
Time Code (AM)	X	X	X	X
Time Code (DCLS): Event2				X
External Event1	X	X	X	X
External 1 PPS; Event3			X	X
External 10 MHz				X

Breakout cable length

The following cable assemblies are 7 inches in overall length.

Breakout cable part numbers

(A) BC11576-1000

(B) BC11576-9860115

(C) PCI-BNC-CCS

1.7. GPS Receiver Interface

1.7.1. General

The most important aspects of using GPS equipment is antenna position. The GPS antenna must be located in an area that has a clear view of the sky. The GPS signals cannot penetrate foliage or structures. A good antenna position will provide optimal timing performance. See "1.2. Installation" on page 10 for detailed installation instructions.

1.7.2. GPS Timing Mode (Mode 6) Characteristics

- The 1 PPS signal generated by the GPS Receiver provides the timing reference for all timing functions.
- The 10 MHz oscillator is disciplined to the GPS 1 PPS signal whenever the receiver is tracking a sufficient number of satellites. If too few satellites are tracked then the TFP will flywheel.
- The TFP extracts major time and satellite tracking status information from data packets sent from the GPS receiver. By default, the TFP provides UTC time to the user. The user can select GPS time instead by issuing the DPRAM Command 0x33 (Select GPS Time Format) described in "Command 0x33: Set GPS Time Format (bc637 only)" on page 52.
- The TFP provides a communications pathway between the user and the GPS receiver. This pathway is most often used to receive GPS data packets for position, velocity, and GPS system status.

1.7.3. Communicating With the GPS Receiver

The dual-port RAM (DPRAM) interface, described in "1.5. Dual-Port RAM Interface" on page 36 provides the communications pathway between the user and the GPS receiver. Using DPRAM commands and the DPRAM GPS Packet Area, the user can send and receive GPS data packets. A GPS data packet consists of a packet length byte, a packet ID byte, and zero or more data bytes. A packet length of zero indicates that no valid packet ID and data bytes are present. The GPS data packet structure is summarized below:

Byte	Item
0	packet length = N = number of packet data bytes + 1 (for the packet ID byte)
1	packet ID
2 - N	packet data bytes

The GPS packet IDs and packet data formats are described later in this section and are taken from Trimble Navigation's manuals. The user can determine the packet length from the documentation. The documentation describes a packet structure that includes, in addition to the packet ID and packet

data bytes, header and trailer bytes and byte-stuffing/unstuffing. The TFP automatically adds (when sending packets) and removes (when receiving packets) the header and trailer bytes and handles all byte-stuffing/unstuffing operations. The TFP user should be concerned with the packet length, packet ID, and packet data bytes only.

Sending GPS Data Packets to the GPS Receiver

To send a GPS data packet to the receiver, use the DPRAM command “Send Packet to GPS Receiver” (command ID 0x30). The format of this command is shown below:

Byte	Type	Item	Value or Range
0	UINT8	command ID	0x30
1	UINT8	packet length	1 - 255
2	UINT8	packet ID	0x00 - 0xFF
3 - N	UINT8	packet data bytes	0 - 255

GPS packet data consists of various integer and floating-point data types. The user must convert these data types to an array of bytes.

Receiving GPS Data Packets from the GPS Receiver

The DPRAM GPS Packet Area holds packets received from the GPS receiver. The TFP writes GPS packets to this area upon user request. The format of the packets in the GPS Packet Area is shown below:

Byte	Type	Item	Value or Range
0	UINT8	packet length	1 - 255, 0 = no valid packet
1	UINT8	packet ID	0x00 - 0xFF
2 - N	UINT8	packet data bytes	0 - 255

There are two DPRAM commands that retrieve GPS Packets. These commands are described below. The TFP sets ACK register bit two whenever it writes a GPS packet to the GPS Packet Area. The transition of ACK register bit two from 0 to 1 is interrupt source four (GPS Packet Available). The user must clear ACK register bit two.

Retrieve Packet from GPS Receiver (Command 0x31)

This command allows the user to retrieve a GPS packet (i.e., position, velocity, status, etc.) from the GPS receiver by specifying the packet ID of the GPS packet of interest. Packets that can be

retrieved with this command are listed below. Packets not found on this list must be retrieved with Command 0x32. The format of Command 0x31 is shown below:

Byte	Type	Item	Value or Range
0	UINT8	command ID	0x31
1	UINT8	packet ID	0x00 - 0xFF

Some of the more commonly requested GPS packets are sent from the GPS receiver to the TFP either periodically (e.g., position fix) or whenever they change (e.g., satellite selection.) The TFP monitors these packets and stores them in on-board RAM so that they can be transferred to the user immediately upon request. The rest of the GPS packets must be retrieved from the GPS receiver by the TFP whenever the user requests them. Note that it can take 10's or 100's of milliseconds for the TFP to retrieve a packet from the GPS receiver. GPS packets that are monitored by the TFP are identified below.

Packet ID	Monitored	Packet Description
0x41	No	GPS Time
0x42	Yes	Single-Precision Position Fix, XYZ ECEF
0x43	Yes	Velocity Fix, XYZ ECEF
0x44	Yes	Satellite Selection
0x45	No	Firmware Version
0x46	Yes	Health of GPS receiver
0x47	No	Signal Level For All Satellites*
0x48	No	GPS System Message
0x49	No	Almanac Health Page For All Satellites
0x4A	Yes	Single-Precision Position Fix, Lat/Long/Alt
0x4B	Yes	Machine/Code ID And Additional Status
0x4D	No	Oscillator Offset
0x4F	No	UTC Parameters
0x55	No	I/O Options
0x56	Yes	Velocity fix, East-North-Up (ENU)
0x57	No	Information About Last Computed Fix
0x5E	No	Additional Fix Status
0x83	No	Double-Precision Position Fix, XYZ ECEF
0x84	No	Double-Precision Position Fix, Lat/Long/Alt

* Note: The first byte returned will be the length of the packet.

The retrieve packet command works as follows:

- If the requested packet is being monitored and a local copy exists, then the TFP transfers its local copy of the packet to the DPRAM GPS Packet Area, sets ACK bit 2, then sets ACK bit 0 to acknowledge the retrieve packet command.

- If the requested packet is not being monitored or if the TFP has not yet received a monitored packet, then the TFP must request the packet from the receiver by sending the appropriate request packet. Once the request has been sent, the TFP sets ACK bit 0 to acknowledge the retrieve packet command. Later, when the receiver responds with the retrieved packet, the TFP transfers the packet to the DPRAM GPS Packet Area and sets ACK bit 2. If the receiver does not respond to the request within a timeout period (typically 3 seconds), then the TFP sets the packet length byte in the DPRAM GPS Packet Area to zero and sets ACK bit 2.
- If the retrieved packet ID is not on the list above, the TFP sets the packet length byte in the DPRAM GPS Packet Area to zero, sets ACK bit 2, then sets ACK bit 0.

Manually Request Packet from GPS Receiver (Command 0x32)

This command is a hybrid of commands 0x30 and 0x31. With this command the user specifies the packet length and ID of a packet sent by the receiver (response packet) and specifies the packet length, ID, and data for the packet to be sent to the receiver (request packet.) The TPF sends the request packet to the receiver and transfers the response packet to the DPRAM GPS Packet Area when it arrives. If the response packet ID is 0x00 then the TFP will ignore the response, in which case, this command would be functionally identical to Command 0x30. The TFP sets ACK bit 0 once the request packet is sent to the receiver. Later, when the response packet has been transferred to the DPRAM GPS Packet Area, the TFP sets ACK bit 2. As with Command 0x31, if the receiver fails to respond within a timeout period, the bc637PCI-V2 sets the packet length in the DPRAM GPS Packet Area to zero and then sets ACK bit 2.

Note: A response packet length of 0 (ZERO) (Byte 1) will return any packet with the corresponding response packet ID (Byte 2). This is useful for packets, like 0x47, that return variable length responses.

Byte	Type	Item	Value or Range
0	UINT8	command ID	0x32
1	UINT8	response packet length	1 - 255
2	UINT8	response packet ID	0x00 - 0xFF
3	UINT8	request packet length	1 - 255
4	UINT8	request packet ID	0x00 - 0xFF
5 - N	UINT8	request packet data bytes	0 - 255

As an example of this command, let us suppose the user wants to retrieve packet 0x5B (satellite ephemeris status) for satellite number six. The receiver sends packet 0x5B in response to packet 0x3B (request satellite ephemeris status.) Packet 0x3B specifies the PRN number for the satellite of interest, in this case, satellite number six. The appropriate command structure for this example is shown below:

Byte	Item	Value
0	command ID	0x32
1	response packet length	17

Byte	Item	Value
2	response packet ID	0x5B
3	request packet length	2
4	request packet ID	0x3B
5	satellite PRN number	6

Byte	Item	Value
0	command ID	0x32
1	response packet length	0
2	response packet ID	0x47
3	request packet length	1
4	request packet ID	0x27

1.7.4. Position Fix Modes

An important aspect of GPS operation is the selection of the position fix mode (GPS packet 0x22.) The GPS receiver supports the following four GPS position fix modes.

Position Fix Mode 0

This mode uses as many satellites as are available to perform both position fixes and timing functions. Confusioncan arise because this mode selection interacts with the dynamics code selection (GPS packet 0x2C.) If a non-static dynamics code is selected then only three or four satellites will be used because the GPS sensor assumes that it is moving. If only three satellites are usable then altitude will be held constant. If a static dynamics code is entered then mode zero will use three or four satellites for a navigation solution as previously, however, if only one or two satellites are available the sensor will use the satellite with the highest elevation to continue calculating bias and bias rate (the timing functions will continue unimpaired). It is good therefore, to enter a static dynamics code if the sensor is static.

Symmetricom recommends using this static mode if the card is not on a moving installation.

Position Fix Mode 1

In this mode, a user-specified satellite is used for timing functions. If mode 1 is selected, only a single satellite will be used for timing, and the current position will be assumed accurate and static.

GPS packet 0x34 allows the satellite associated with mode one to be selected. This packet has one data byte that specifies the PRN of the desired satellite. If a data byte value of 0 is entered, then the sensor will always track the single satellite that has the highest elevation within the constellation in view.

It is good to operate in a single-satellite highest elevation mode for timing applications. The greatest contribution to timing error is the electron content variation in the path between the satellite and the receiver. Selecting the highest elevation satellite minimizes this variation.

Position Fix Mode 3 and 4

These modes are rarely used for timing applications unless the user operational platform is dynamic. Mode 3 is particularly useful in a marine environment where the sensor altitude is relatively constant.

1.7.5. GPS Default Parameters

The TFP sends the following GPS packets to the GPS receiver on reset or whenever the Timing Mode is changed to GPS Timing Mode 6.

Set Operating Parameters (GPS packet 0x2C)

Packet Data Item	Value
Packet ID	0x2c
Dynamics Code	4 (Static)
Elevation Angle Mask	0.1745 Radians (10 Degrees)
Signal Level Mask	6.0
PDOP Mask	12.0
PDOP Switch	8.0

Set High-8 / High-6 Mode (GPS packet 0x75)

Packet Data Item	Value
Packet ID	0x75
Mode	0 (high-8)

Set I/O Options (GPS packet 0x35)

Packet Data Item	Value
Packet ID	0x35
Position	0x03
Velocity	0x03
Timing	0x00
Auxiliary	0x00

To change any of the packet 0x35 options, keep the following in mind: The TFP monitors position and velocity packets so the “position” and “velocity” options should be sent with bits 0 and 1 set; the TFP extracts major time from time packets broadcast by the GPS receiver so the “timing” option should be sent with bits zero, one, and two cleared.

1.8. Legacy and New Generation Cards

This section covers differences between legacy bc635/637PCI-U (PCI-U) cards and the new generation bc635/637PCI-V2 (PCI-V2) and bc635/637PCIe (PCIe) cards.

Both the legacy and the new generation cards map both register and DPRAM into memory spaces, but there is a difference in the PCI base address register (BAR) mapping.

1.8.1. PCI Bar Mapping

PCI-U	PCI-V2 and PCIe
bar0 = 32-bit register space	bar0 = reserved
bar1 = 8-bit dpram space	bar1 = reserved
	bar2 = 32-bit register space
	bar3 = 8-bit dpram space

1.8.2. Differences Between Versions -U and New Generation Cards

	PCI-U	PCI-V2 and PCIe
Standard oscillator	VCXO	TCXO (better holdover)
1pps duty cycle	63 uS	60 uS
Heartbeat (periodic) duty cycle	Adjustable	square wave
Field program update	Replace flash	• Serial port download for early versions of bc635PCI-V2 and bc637PCI-V2. • Through the host PCI(e) bus for later models of bc635PCI-V2, bc637PCI-V2, and all models of bc635PCIe, and bc637PCIe.
OCXO option	+12V	+5V
OCXO option	sine wave	square wave
Time at power-on	elapsed	RTC
Time code input level (AM)	5V to 5V P-P	1V to 8V P-P
Time code output level (AM)	4V P-P 50 ohm	3V P-P 50 ohm
NV (Restored at Power-on)	not supported	many parameters restored
Strobe output tim-ing	1 uS late	on time
bc637 GPS receiver	Ace III	SKII (Lassen)
Battery disconnect jumper	No	Yes
DDS (frequency synthesizer) circuit	No	Yes
Time Code Inputs	No	IRIG G, E, XR3 & 2137
Time Code Outputs	No	IRIG A, G, E, NASA 36, XR3 & 2137

- The PCI-V2 and PCIe cards transition through end of year properly, version -U did not.
- SET CLOCK Command (10 uSec Advance Retard) is not implemented on PCI-V2 or PCIe design.
- PCI Special Boot mode is not supported on the PCI-v2 or PCIe design.

The user is no longer required to enter UNIX seconds of Leap Event when manually entering Leap Seconds settings. The PCI-V2 and PCIe cards use Year and Time of Year information to determine when Leap Events will occur.

2. Windows Application Programs

2.1. bc635PCIcfg.exe Windows Application Program

2.1.1. General

Configuration programs bc635PCIcfg.exe, bc637PCIcfg.exe and a System Clock Utility Tray-Time.exe are included with the TFP module.

When installing the SW from the supplied CD as described in "1.2.5. Installation Under Windows " on page 14, the installer will create a folder in 'program files' titled 'bc635PCI Configuration Software' containing bc635PCIcfg.exe, TrayTime.exe, and bc635pciReadEvent.exe. The bc635PCIcfg clock.exe icon is also copied to the system desktop.

When installing the SW, the installer will create a folder in 'program files' titled 'bc637PCI configuration Software' containing bc635PCIcfg.exe, TrayTime.exe, and bc635pciReadEvent.exe. Icons are copied to the system desktop.

The System Clock Utility, TrayTime.exe, (described later in this chapter) is a system tray utility that queries the TFP and periodically sets the computer's system clock at a user-defined interval. The utility may run either as a standard application or as a Windows Service.

The bc635PCIcfg.exe program allows the user to access the TFP card and demonstrates the card's functionality. This program is designed to operate under Microsoft Windows XP/Vista. This utility may be used to query current settings, modify settings, and retrieve or monitor data generated by the card. This program requires the runtime driver to be available in order to operate. The background window of the program provides time, as well as information regarding the clock status, interrupt bit status, and clock reference source type. A full menu system (described in the following paragraphs) has been designed to provide access to the card. Each associated pull-down menu provides a logical grouping of commands. Most of the pull-down menus also include a Current Settings selection that provides a review of the logical group.

The bc637cfg.exe program allows the user to access the GPS functions of the bc637PCI-V2 and bc637PCIe TFPs.

2.1.2. Quickstart Guide to Operating bc635PCIcfg.exe

Click on the desktop icon to execute the program bc635PCIcfg.exe that is an interface program for both the bc635 and bc637 cards. The bc637 may be set to GPS mode, and will initially display time based on the Real Time Clock until GPS lock is achieved. The bc637 unit is locked to GPS, and decoding time when the tracking LED in the GUI left-hand corner is green. The bc635 may be set to decode a time code. For testing, you may set the board time manually using the Time menu and selecting Set Time.

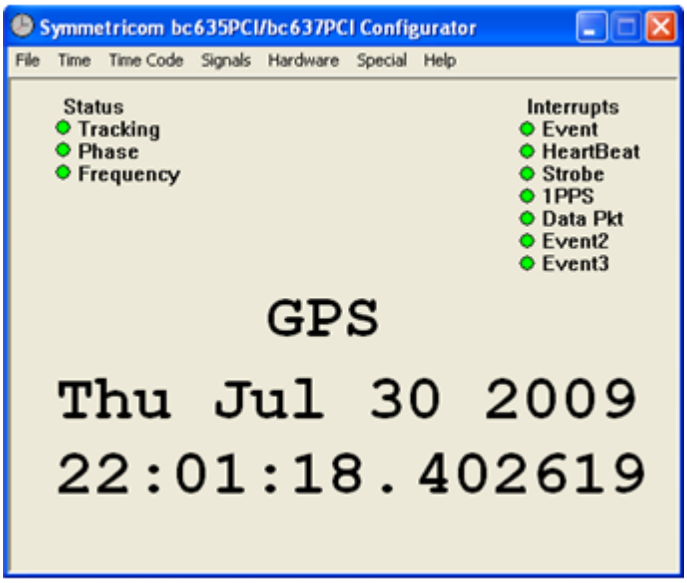


Figure 2-1: bc637 set to GPS mode

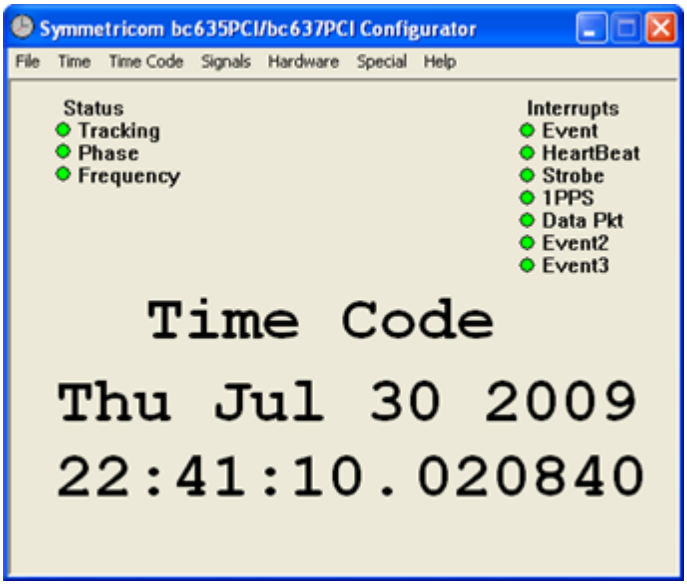


Figure 2-2: bc635 set to Time Code mode

2.1.3. bc637PCIcfg Program Menu Interface

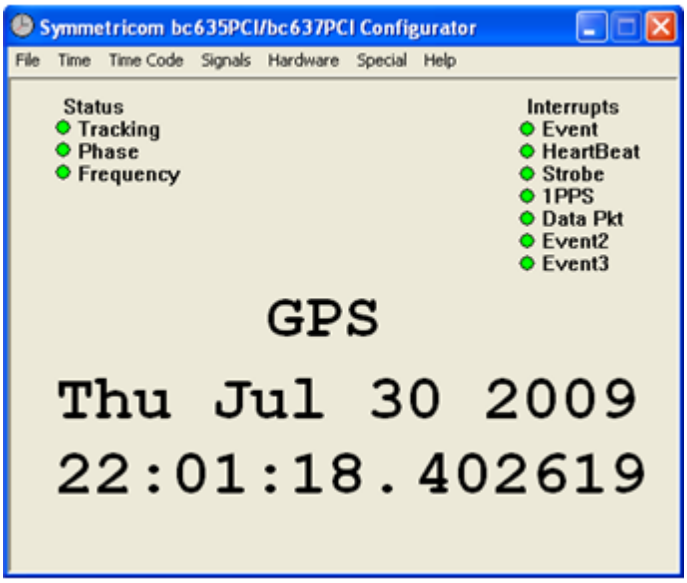


Figure 2-3: bc637PCIcfg Program

File Menu

The File menu group provides a few common functions associated with Windows applications.

Refer to Figure 2-3

File > Open Device

Each instance of the bc635PCIcfg.exe program communicates with only one device at a time. Open Device allows the user to open and operate any of up to four installed TFP devices. By default, the program opens and operates using the first device in the system (Device 0). By selecting a new device to open, the program will close the currently selected device before opening the newly requested device. This command will also clear the interrupt mask.

Select Device 0, 1, 2 or 3 and click OK. Use Device 0 if only 1 board is installed.

File > Interrupt Start

This command allows the user to start an interrupt service routine capable of handling the selected hardware interrupts created by the TFP module. After starting the interrupt service routine, the user may initiate any interrupt source located under “Signals > Interrupts”. For more information on interrupts, see the MASK and INSTAT registers described in "MASK Register (0x18)" on page 31.

File > Exit

This command allows the user to close the device and exit the program. This command will also clear the interrupt mask.

Time Menu

The Time menu group, see Figure 2-4, provides access to functions that control how the TFP maintains time data. These functions allow the user to select where to obtain time data, whether or not to manipulate the time data and how to present the time data.

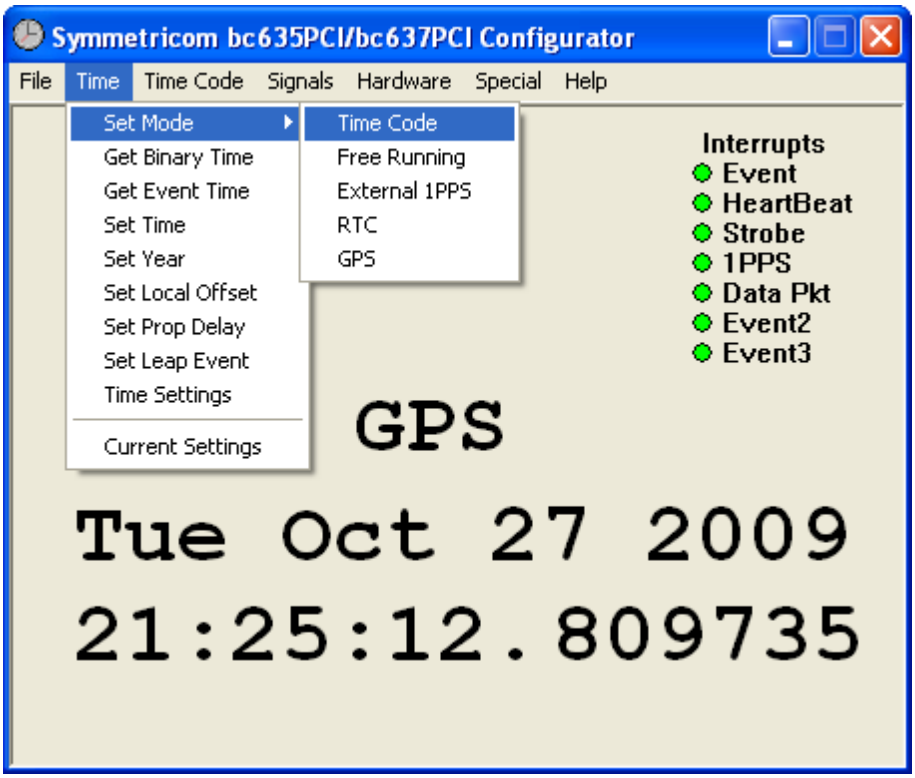


Figure 2-4: Time Menu

Time > Set Mode

The Set Mode menu selection allows the user to change the operating mode of the installed card. Selecting this option reveals a secondary menu listing the available operating modes of the TFP. The available mode selections are:

- Time Code
- Free Running
- External 1PPS
- RTC
- GPS

Note 1: The card automatically increments the Year value, in every operating mode. For more information on setting the card synchronization mode, refer to "CONTROL Register (0x10)" on page 28.

Note 2: The bc635 cards do not support GPS mode.

Time > Get Binary Time

The Get Binary Time menu selection exercises the device time capture registers. Get Binary Time requests the binary time 25 times, retrieving 25 consecutive timestamps as fast as the system will allow, and displays them. This function is designed to display binary data only. This command is provided as a demonstration of the binary time request. For more information on the Time Registers, refer to "1.4.3. Device Register Description" on page 26.

Time > Get Event Time

The Get Event Time menu selection exercises the event capture registers of the device. The function is similar to Get Binary Time where 25 consecutive requests are made to the event register. This function is designed to display binary data. The bc635PCI-V2 & bc637PCI-V2 card should be set to the binary time format when executing this function. If the decimal Time Register Format is selected, the major time (in front of the decimal point) will be garbled. Minor time will display correctly.

Time > Set Time

The Set Time menu selection will set the time on the TFP. Set Time displays the current time, years through seconds in a decimal format. The user may change any or all of these values and select the OK button. This command will load the time properly regardless of the currently selected time format. This function is typically used when operating in either the Free Running or External 1PPS modes. While the function may be used when operating in Time code or GPS modes, subsequent time data received from the selected reference source will overwrite the manually entered time. For information on Set Major Time, see "Command 0x12: Set Major Time" on page 42.

Time > Set Year

The Set Year menu selection will set the year data without affecting the other time data. Many time code formats do not include year information in the data. The supported range is 1970 - 2036. Set the year and click OK. See "Command 0x13: Set Year" on page 42.

Time > Set Local Offset

The Set Local Offset menu selection allows the user to program a local time offset into the TFP. If the local offset value is nonzero, the device will adjust any reference timing information in order to maintain a local time in TFP clock. Use of this function only affects the time data in the TIME registers described in Chapter1.4. Allowed values are -16 through +16, and can include half hour offsets. See "Command 0x1D: Set Local Time Offset" on page 49.

Enter Local Offset in hours (-16 to +16), check Half Hour if necessary and click OK.

Time > Set Prop Delay

The Set Prop Delay menu selection allows the user to compensate for propagation delays introduced by the currently selected reference source. For example, when the unit is operating in Time code

decoding mode, a long cable run may result in the input time code having a propagation delay. The delay value is programmable in units of 100 ns and has an allowed range from -4000000 (advance by 400 ms) through +4000000 (retard by 400 ms). See "Command 0x17: Set Propagation Delay Compensation" on page 44.

Enter the Propagation Delay and click OK

Time > Set Leap Event

The Set Leap Event menu selection allows the user to program a future leap second event. The user is not required to enter UNIX seconds of Leap Event when manually entering Leap Seconds settings. The card uses Year and Time of Year information to determine when a Leap Event will occur. In operating modes that do not have a time source that provides leap second information, the card will calculate the leap event for UTC midnight of the current month. GPS time and IRIG B 1344 operating modes provide automatic leap second adjustments, which normally occur on June 30th or December 31st at UTC midnight.

When the leap second insertion transition takes place (add 1 second) the Time Code output produces a second 60 followed by second 0. The application program will show 2 second 59s followed by second 0.

When the leap second deletion transition takes place (subtract 1 second) the Time Code output produces a second 58 followed by second 0. The application program will show second 58 followed by second 0.

The TFP card uses only the Flag values of Disable, Insertion or Deletion; the Leap Event Time value is not supported.

See "Command 0x1E: Program Leap Second Event" on page 49.

Time > Time Settings

The Time Settings function allows the user to modify other timing operations. The UTC Corrections may be enabled or disabled. Enabling UTC Corrections commands the device to include any leap second corrections provided by the reference source and act on any leap event data that is present. The default operation is to use UTC corrections. This function is also used to enable or disable the following options: IEEE Daylight Saving (strips DST bit from IEEE 1344 Time Code input and output) and Local Time Offset (controls the use of Local Offset value). The board time format (Binary or Decimal) is also selected using this function.

Note: the Year Auto Increment cannot be disabled; the year is always incremented when operating with sources that do not include year information.

See "1.5. Dual-Port RAM Interface" on page 36, commands 0x11, 0x1D, 0x33 and 0x41.

Time > Current Settings

The Current Settings function provides a summary of all the time data relevant to the devices current settings. In addition to the programmable values, the values of some of the card's timing data are also

presented as information points via the UTC Data button. These values include UTC Control, leap second count, leap second event data and leap second event time.

Settings are:

- Timing Mode
- Time Format
- Current Year
- Local Offset
- Propagation Delay
- Daylight Saving Flag
- Local time Flag

Time Code Menu

The Time Code menu group, see Figure 2-5, provides access to functions that control how the TFP card operates while decoding time code. These functions allow the user to control both the time code decoding, and the time code generating circuits.

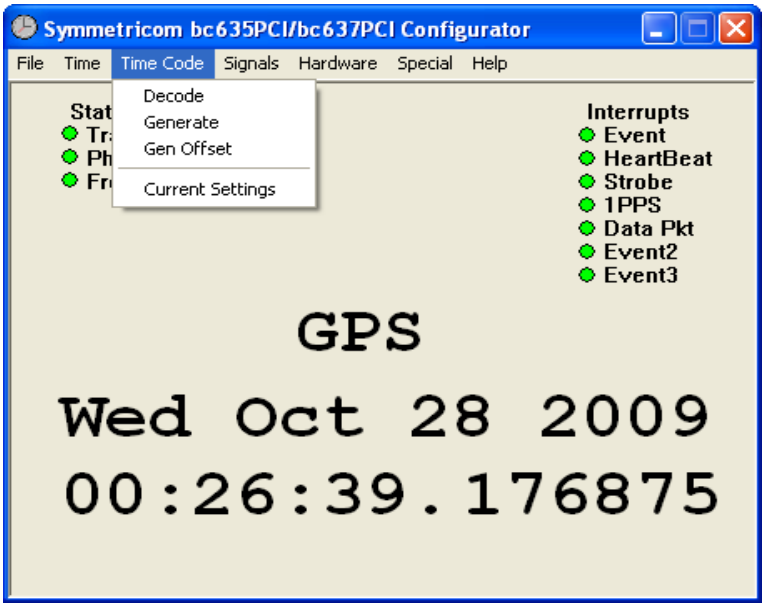


Figure 2-5: Time Code

Time Code > Decode

The Decode menu selection allows the user to select the format and modulation types associated with an input timing signal. These values control how the device attempts to decode the input time code. These values may be set regardless of the mode but will only be used in time code decoding mode. The format defines the type of the time code data. The modulation defines the envelope for the signal and which input pin the signal will be extracted from. See "Command 0x15: Set Input Time Code Format" on page 43: Set Input Time Code Format for supported time codes and command 0x16, which is used to select the modulation type.

Time Code > Generate

The Generate menu selection allows the user to select the time code format that will be generated by the TFP. See "Command 0x1B: Set Time Code Output Format" on page 46 for supported time codes. Detailed performance specifications are outlined in "1. PCI/PCIe TFP Hardware" on page 1

Time code > Gen Offset

The Gen Offset menu selection allows the user to add an offset to the time code signal produced by the TFP. The generator offset only affects the time code generation. This functionality is useful for driving time code display units to display local time. Allowed values are -16 through +16, and may include half hour offsets. See "Command 0x1C: Set Generator Time Offset" on page 48.

Time code > Current Settings

The Current Settings menu selection provides time code output data summary for:

- Decoding
- Modulation Type
- Generating
- Generator Offset

Signals Menu

This menu group provides access to functions that control various hardware-timing signals.

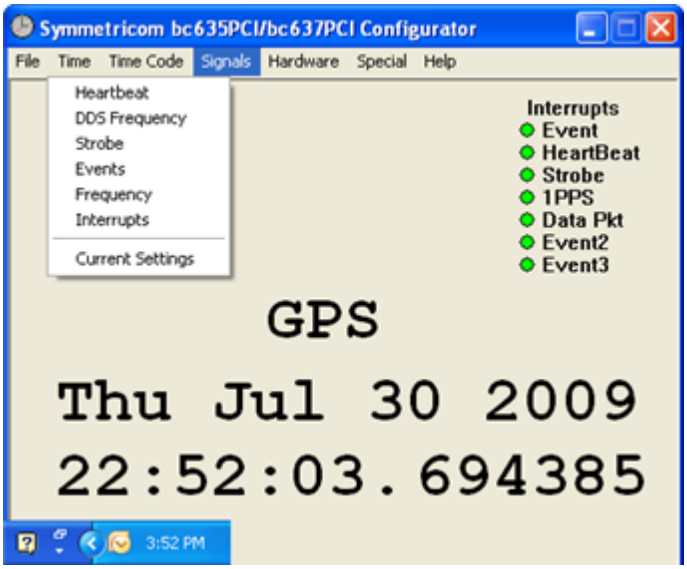


Figure 2-6: Signals Menu

Signals > Heartbeat

The Heartbeat function allows the user to command the TFP to produce a clock signal at a specified frequency. The heartbeat signal, also referred to as a periodic pulse output (PPO), periodic or programmable periodic, may be either synchronous or asynchronous to the TFP's 1PPS epoch.

The formula for determining the heartbeat frequency is f = 1,000,000/n1*n2, where n1 and n2 are greater than or equal to 2 and less than 65536. Synchronous mode aligns the periodic output's rising edge to rising edge of 1PPS.

Periodic output fractional frequencies (non-integer) should use Asynchronous mode.

Signals > DDS (aka frequency synthesizer)

The DDS output may be selected in place of the periodic rate generator's output. This circuit provides a square wave output with a frequency resolution of 0.03125 (1/32) Hz. Direct frequency entry of 7 significant digits with decimal adjustment up to 7 places or entry of 8 digits with 1 Hz resolution is supported with the entry in this dialog box. Value range is 1/1e7 (.0000001) to 1e8 (100000000).

Signals > Strobe

The Strobe function allows the user to command the TFP to produce a hardware signal at a particular time, or at a particular point during a 1 second interval. When major/minor mode is selected, a hardware signal (1 uS pulse) will be produced when the internal time of the TFP matches the values entered for the major and minor strobe registers. Up to 22 bits of binary major time may be supplied in addition to the microseconds loaded in the minor strobe register. This allows strobe signals to be programmed up to 48 days in advance. This function is designed to operate with the timing format in binary mode. When minor mode is selected, a strobe signal is produced every second when the internal microsecond count in the TFP matches the value entered in the minor strobe register. The output of this circuitry is capable of creating a PCI bus interrupt. For details on Strobe programming, see "1.3.4. Time Coincidence Strobe Output" on page 20.

Signals > Events

The Events function allows the user to command the TFP to monitor a hardware-timing signal. The source for the signal can be either the External Event input or the Periodic/DDS output. The capture may be set to occur on either the rising or falling edge. When the selected signal occurs, the time at which the signal occurred is loaded into the event time registers. The capture lockout checkbox can be used to control whether or not subsequent signals will overwrite the data in the event time registers. The output of this circuitry is capable of creating a PCI bus interrupt. See "CONTROL Register (0x10)" on page 28 for event programming details.

Note: The capture rate is dependent on the rate that the Software extracts the event times which is dependent on the operating system and speed of the host computer.

Signals > Frequency

The Frequency function allows the user to control the frequency signal output by the TFP. The available frequencies are 1, 5 and 10 MHz. See the CONTROL Register described in "CONTROL Register (0x10)" on page 28 for more detail.

Signals > Interrupts

The Interrupts function allows the user to control the generation of interrupts by the TFP. The detection of an interrupt will be displayed in the background of the main window by seven LEDs which are displayed in the upper-right corner. When an interrupt occurs, the program queries the interrupt source and the associated LED is displayed in red. In order to display consecutive interrupts, the LEDs are changed back to green once per second. This may result in LEDs only remaining red for a short period of time.

The default state of the interrupts is OFF. Interrupt programming is described in detail in "MASK Register (0x18)" on page 31.

Signals > Current Settings

The Current Settings function provides a summary of all the signal data. In addition to the programmable values, other values may be presented as information points.

Hardware Menu

See Figure 2-7. This group provides additional access to functions that control the oscillator and its associated disciplining circuits. These functions modify the actual oscillator control function used to slave the oscillator to the selected reference signal. This function is not modified during standard operation.

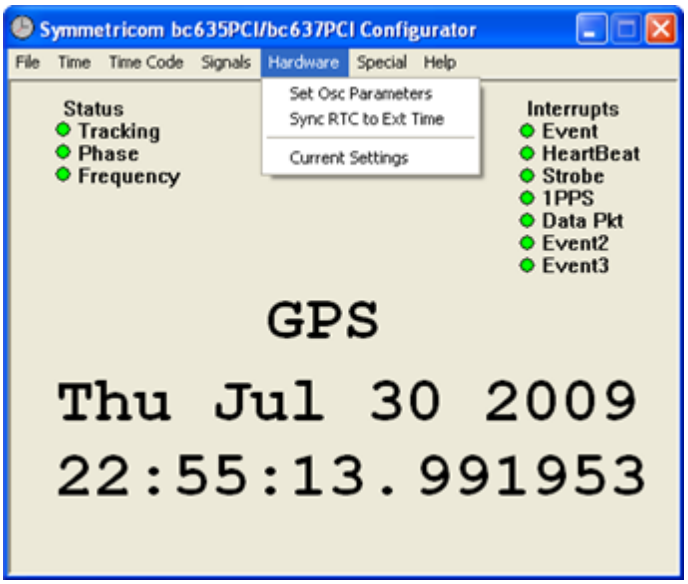


Figure 2-7: Hardware Menu

Hardware > Set Osc Parameters

The Set Osc Parameters function allows the user to select an external oscillator or the on-board oscillator, in addition to enabling/disabling disciplining and jamsyncing. Oscillator selection and control is described in more detail in "1.5. Dual-Port RAM Interface" on page 36; commands 20, 21, and 24.

Hardware > Sync RTC to Ext Time

The Sync RTC to Ext Time function updates the RTC circuit time to the current time on the board. The board contains a separate battery-backed Real Time Clock Circuit (RTC) that may be used to keep time while the device is powered down.

Hardware > Current Settings

The Current Settings function provides a summary of all the oscillator data. In addition to the programmable values, other values may be presented as information points.

Special Menu

The Special Menu group provides access to those functions that do not fit in any particular category. See Figure 2-8. Most of these functions are not used during normal operation.

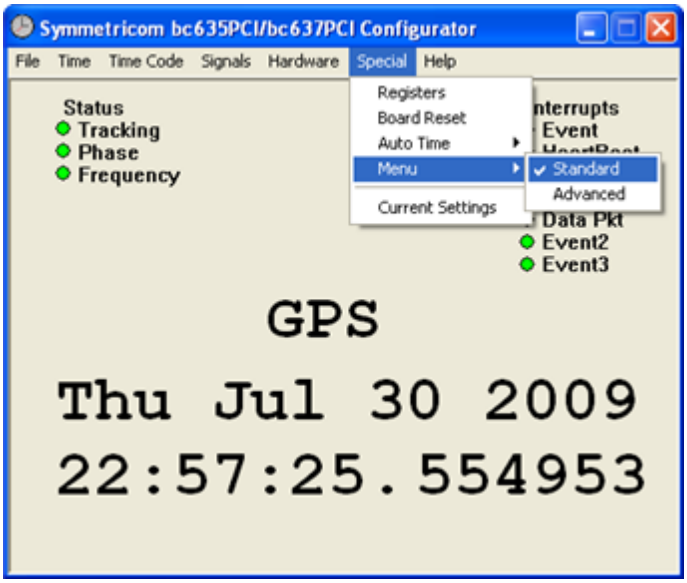


Figure 2-8: Special Menu

Special > Registers

The Registers function is provided to perform direct reads and writes to the TFP registers. While most of the functionality available through the registers can be controlled via other aspects of the configurator program, this function may be useful for debugging purposes.

Special > Board Reset

The Board Reset function allows the user to reset the TFP device. This command is useful when starting a test or in the case that unexpected behavior is observed from the card. This function is not used during normal operation. See "Command 0x1A: Software Reset" on page 46.

Special > Autotime

The Autotime function allows the user to control the data display in the background of the main program window. If this function is turned off, the display will stop updating but will continue to show the reference source type.

Special > Menu

The Menu command allows the user to switch to an advanced version of the menu. See Figure 2-9. The advanced menu selection adds Special menu options, as well as the PCI menu selection.

Warning: The advanced menu contains operations that may disable the function of the TFP.

Special > Menu > Advanced

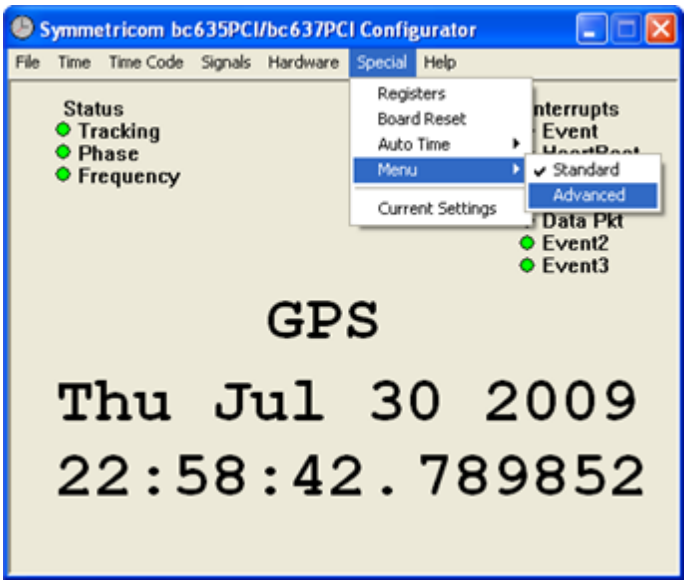


Figure 2-9: Special Menu

Selecting the advanced Special Menu, opens several new selections:

Special > DP RAM

DP RAM allows reading, writing hex offsets, and values to the dual port RAM.

Special > Packet Offset

Displays the following packet information

Special > Debug

Debug can be turned off or on. Turning debug on, causes the three status lights, (tracking, Phase, and Frequency), to provide debugging information.

Special > Emulator

The emulator can be turned off or on.

Special > Current Settings

The Special > Current Settings function provides information related to the PCI interface to the board. This command is useful for determining whether or not the driver has obtained access to the device. It may also be used to review the PCI mapping of the device onto the bus. The interrupt level selected during PCI configuration may also be reviewed.

PCI Menu

In addition to the additional Special Menu selections, a new menu choice PCI is available when the Advanced Menu is selected.

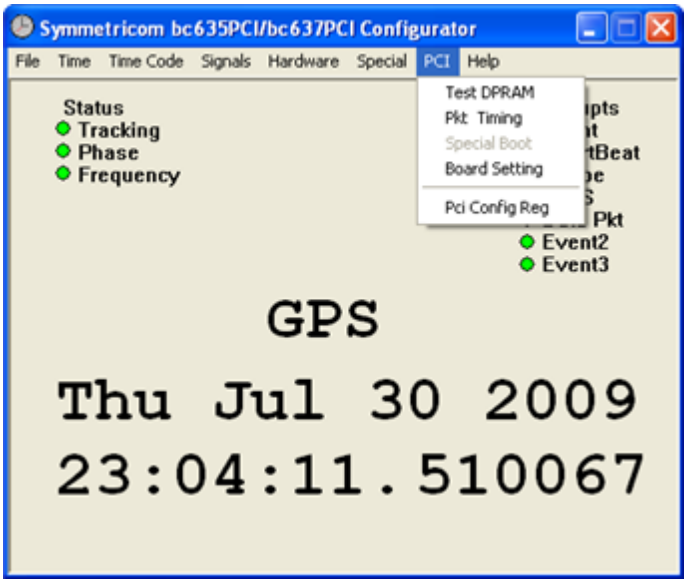


Figure 2-10

The PCI menu has the following selections:

- Test DPRAM
- Packet Timing
- Special Boot
- Board Setting
- PCI Config Reg

Help Menu

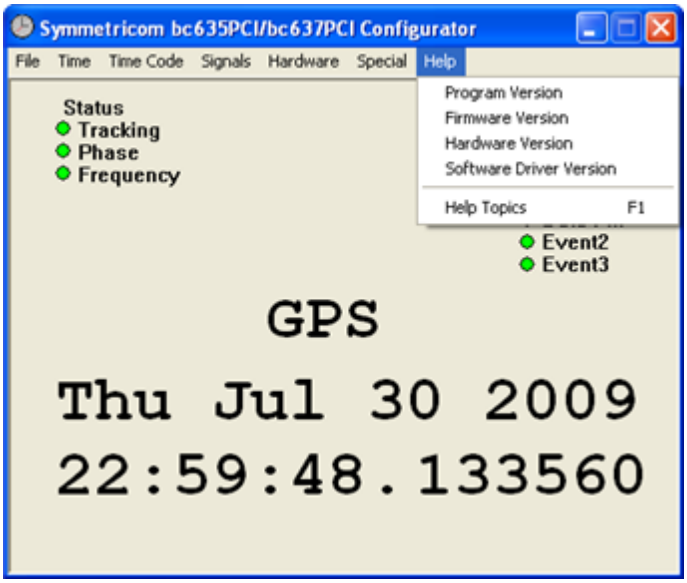


Figure 2-11

- Program Version: shows the versions of the running program.
- Firmware Version: retrieves the firmware version from the TFP.
- Hardware Version: retrieves the hardware version from the TFP.
- Software Driver Version:shows the version opf the driver that is loaded.

2.2. bc637PCIcfg.exe Windows Application Program

2.2.1. General

In addition to the configuration program (bc635PCIcfg.exe), and the System Clock Utility (Tray-Time.exe), the TFP includes a GPS configuration program (bc637PCIcfg.exe). When installing the SW from the supplied CD, as described in "1.2.5. Installation Under Windows " on page 14, the installer will create a folder in 'program files' titled 'bc637PCI configuration Software' containing bc635PCIcfg.exe, bc637PCIcfg.exe, TrayTime.exe, and bc635pciReadEventexe. The bc635PCIcfg.exe and bc637PCIcfg.exe clock icons are copied to the system desktop.

The bc637PCIcfg.exe program allows the user to access the TFP and demonstrates the boards GPS functionality. This program is designed to operate under Microsoft Windows XP/Vista. This utility may be used to query current settings, modify settings, and retrieve or monitor data generated by the card and/or the GPS receiver. This program requires that the runtime driver be available in order to operate. The background window of the program provides current time, as well as information regarding the clock status and clock reference source type. A full menu system (described in the following paragraphs) has been designed to provide access to the card and the GPS receiver. Each associated pull-down menu provides a logical grouping of commands.

2.2.2. Quickstart Guide to Operating bc637PCIcfg.exe

1. Verify that the antenna is connected to the SMB connector on the rear of the card.
2. Click on the bc637PCIcfg.exe desktop icon to execute the program. The card will start counting using the RTC value, until lock is achieved. The TFP unit is locked to GPS, and decoding UTC time when the tracking LED, indicated by the letter “T” in the GUI shown in Figure 2-13, is green.

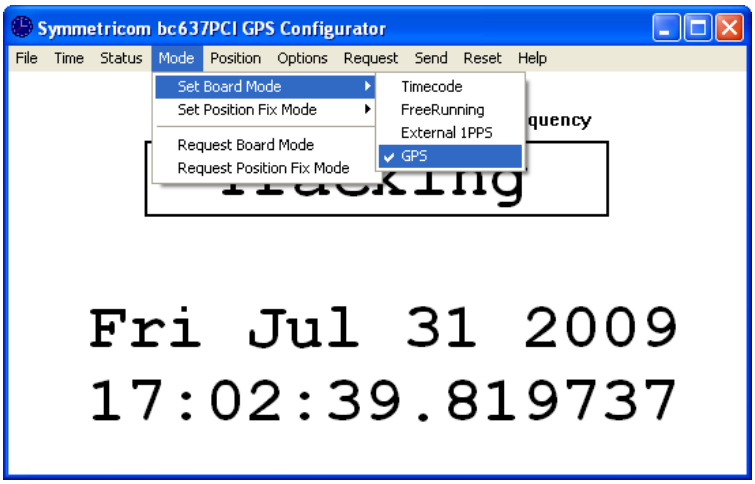


Figure 2-13: bc637 Menus

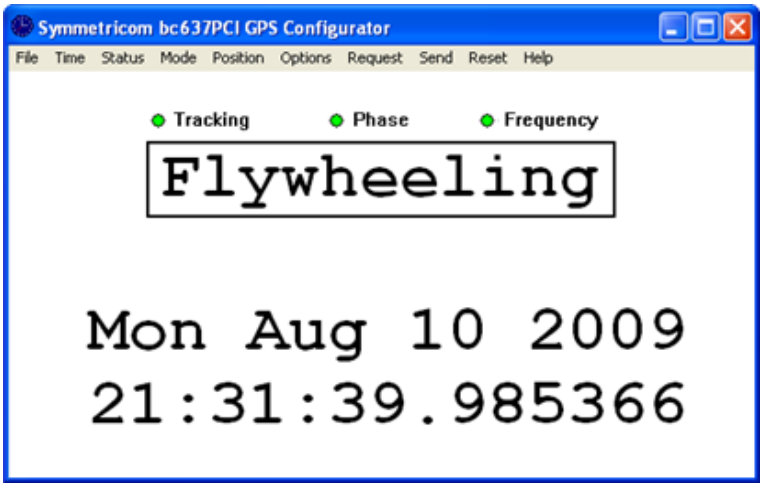


Figure 2-14: bc637 Counting Time in Flywheel State

2.2.3. bc637PCIcfg.exe Program Menu Interface

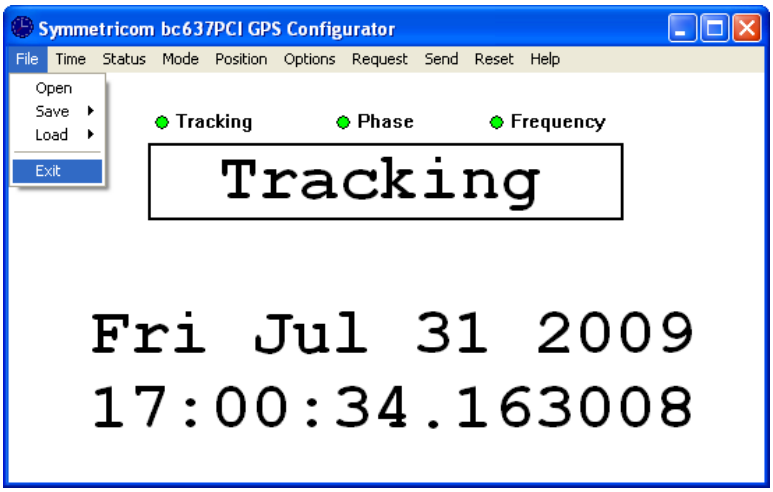


Figure 2-15: File

File Menu

The File menu group provides a few common functions associated with Windows applications.

File > Open

Each instance of bc637PCIcfg is designed to communicate with only one device at a time. Open allows the user to open and operate any of up to four installed TFP cards. By default, the program opens and operates using the first device in the system (Device 0). By selecting a new device to open, the program will close the currently selected device before opening the newly requested device. This command will also clear the interrupt mask.

File > Save

This command allows the user to save the values to the GPS receiver located under “Save > Save variable”. For more information on the GPS variables, see: GPS Receiver Interface.

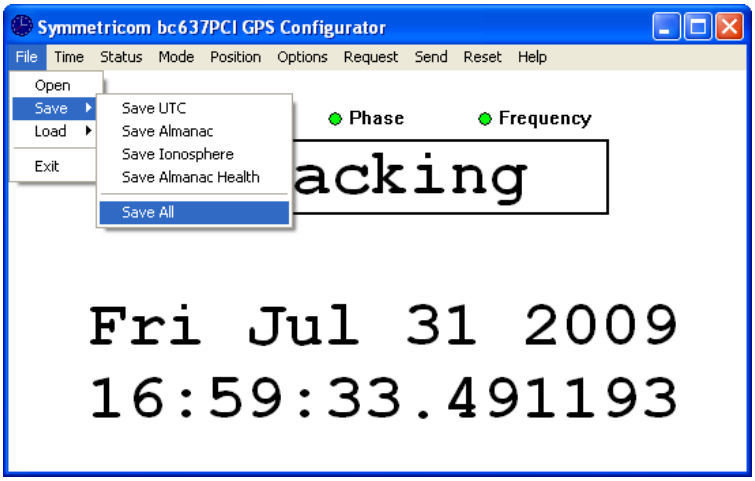


Figure 2-16: File Save

File > Load

This command allows the user to save the values to the GPS receiver located under “Save > Load variable”. For more information on the GPS variables, see: GPS Receiver Interface.

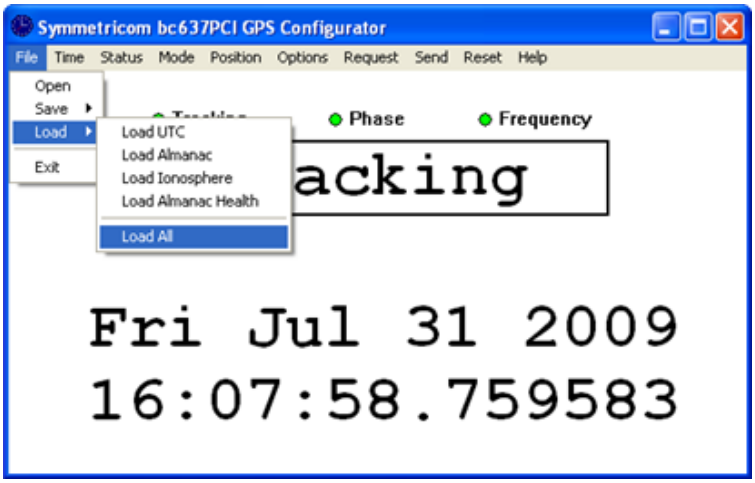


Figure 2-17: File Load

File > Exit

This command allows the user to close the device and exit the program.

Time Menu

The Time menu group, see Figure 2-18, provides access to functions that control how the TFP card maintains time data. These functions allow the user to select where to obtain time data, whether or not to manipulate the time data, and how to present the time data.

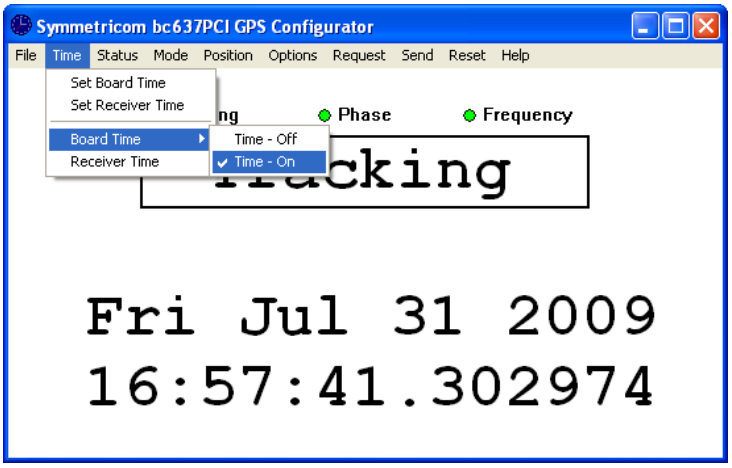


Figure 2-18: Time Menu

Time > Set Board Time

The Set Board Time menu selection will set the time on the TFP. The Set Board Time interface GUI displays the current time, years through seconds in a decimal format. The user may change any or all of these values and select the OK button. This function is typically used when operating in either the Free Running or External 1PPS modes. While the function may be used when operating in Time code or GPS modes, subsequent time data received from the selected reference source will overwrite the loaded time when lock is achieved. This function accesses the DPRAM command 0x12, Set Major Time. For information on Set Major Time, see "Command 0x12: Set Major Time" on page 42.

Time > Set Receiver Time

The Set Receiver Time menu selection will acquire time from the TFP device and set the time on the GPS receiver. This will improve the initial time required to track satellites. This command accesses the set GPS time packet 0x2E as described in GPS Receiver Interface.

Warning: Be sure to set an accurate “GMT” time on the TFP module before issuing this command.

Time > Receiver Time

The Receiver Time menu selection will return time from the GPS receiver. This command requests current time via packet 0x21, and returns packet 0x41, as described in GPS Receiver Interface.

Status Menu

The Status menu group, (see Figure 2-19), provides access to the GPS data packets that provide the GPS receiver status data. These commands access the data packets that return packets 46, 47 and 4F (as described in GPS Receiver Interface).

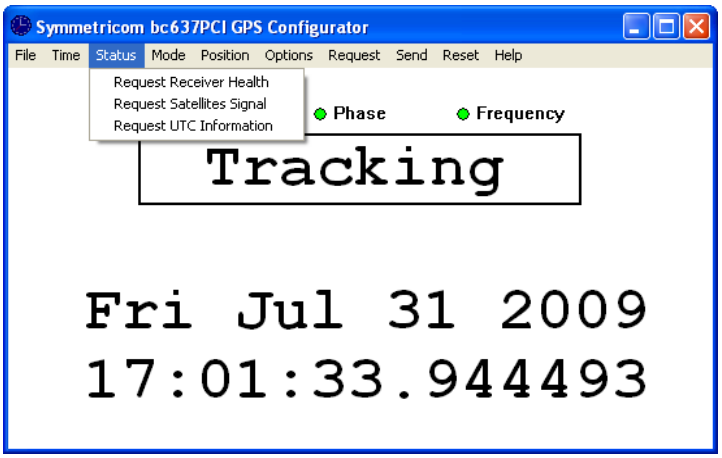


Figure 2-19: Status Menu

Mode Menu

The Mode menu group, (see Figure 2-20) provide access to the various functional modes of the TFP and the GPS receiver.

The Mode > Set Board Mode menu selection allows the user to change the operating mode of the installed TFP. Selecting this option reveals a secondary menu, listing the available operating modes of the TFP. The available mode selections are:

- Time code
- Free Running
- External 1PPS
- GPS

For more information on setting the card synchronization mode, refer to "CONTROL Register (0x10)" on page 28. To verify the mode, select Mode > Request Board Mode.

The Mode > Set Position Fix Mode menu selection allows the user to change the GPS receiver mode. The GPS receiver modes are explained in more detail in GPS Receiver Interface. To verify the mode, select Mode > Request Position Fix Mode.

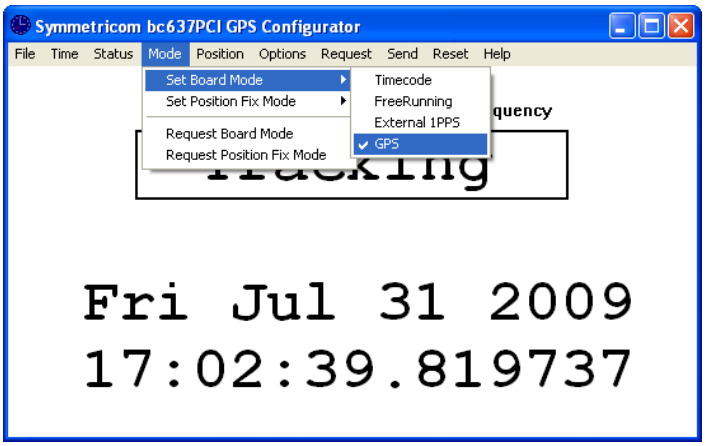


Figure 2-20: Mode Menu: Set Board Mode

Position Menu

The Position menu group, (see Figure 2-21) provide access to position data from the GPS receiver. Both LLA and XYZ position formats are supported. These packets are addressed in more detail in: GPS Receiver Interface, packets 0x2B, 0x31, 0x42, and 0x4A.

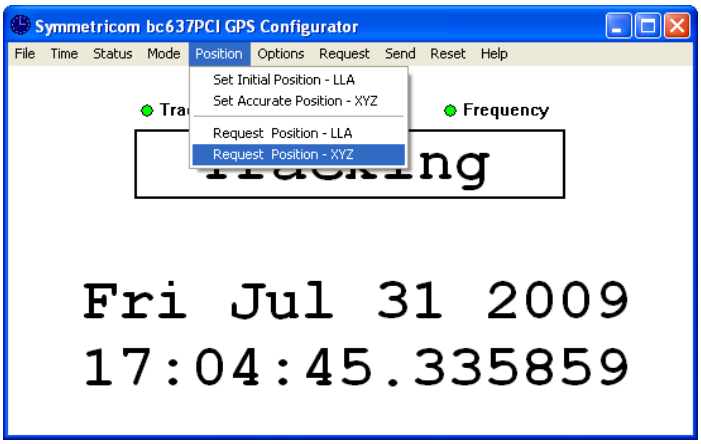


Figure 2-21

Options Menu

The Options menu group, (see Figure 2-22) provide access to set or request the input and output options for the GPS receiver. These commands are described in more detail in GPS Receiver Interface, packets 0x35, 0x43, and 0x56.

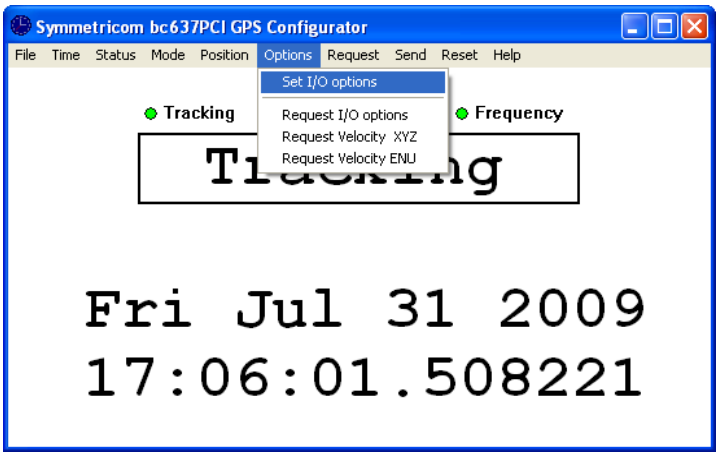


Figure 2-22: Options Menu

Request Menu

The Request menu group, (see Figure 2-23) provide access to data from the GPS receiver. These commands are described in more detail in GPS Receiver Interface, packets 0x40, 0x4C, and 0x4D.

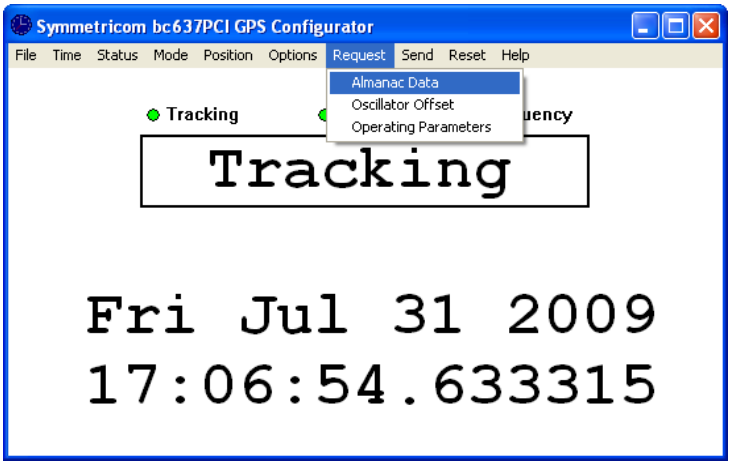


Figure 2-23: Request Menu

Request > Almanac Data

This command provides almanac data for a single satellite.

Request > Oscillator Offset

This packet provides the receiver oscillator offset in Hertz at the carrier. The packet format is described in GPS Receiver Interface, packet 0x4D.

Request > Operating Parameter

This packet provides several operating parameters for the GPS receiver, and includes the dynamics code, elevation angle mask, signal level mask, and PDOP mask. These parameter formats are described in GPS Receiver Interface, packet 0x4C.

Send Menu

The Send menu group, (see Figure 2-24), allows the user to set the 2-D altitude and receiver operating parameters on the GPS receiver. These commands are described in more detail in GPS Receiver Interface, packets 0x2A and 0x2C.

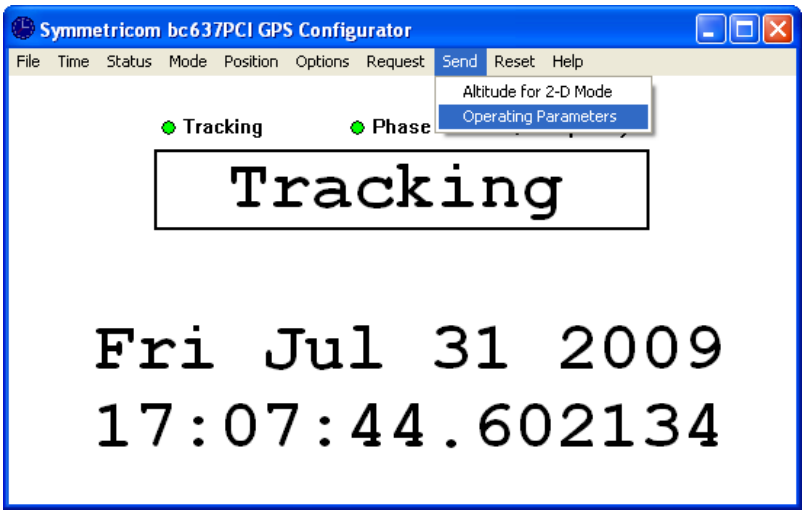


Figure 2-24: Send Menu

Send > Altitude for 2-D Mode

This packet provides the altitude used for 2-D (3-satellite) mode, and is used until a 3-D fix is completed. See GPS Receiver Interface, packet 0x2A for more detail.

Send > Operating Parameters

This packet is used to optionally set the GPS receiver operating parameters, requesting the current values after they are set. See GPS Receiver Interface, packet 0x2C for more detail.

Reset Menu

The Send menu group commands, (see Figure 2-25) are used to reset various components on the TFP card and the GPS receiver.

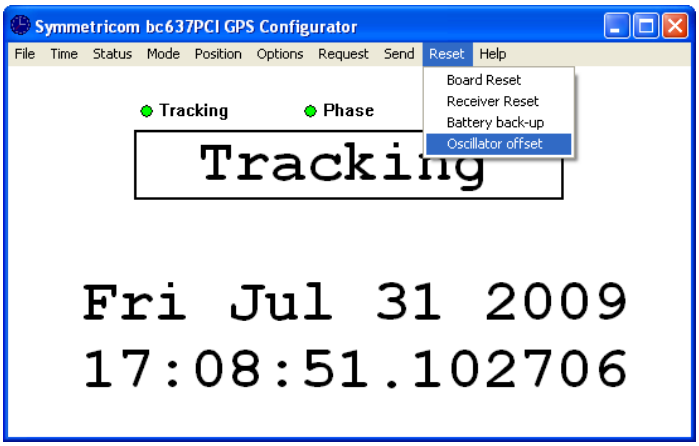


Figure 2-25: Reset menu

Reset > Board Reset

This command will reset the TFP card per the DPRAM command 0x1A, and is described in "Command 0x1A: Software Reset" on page 46.

Reset > Receiver Reset

This command performs a GPS receiver software reset, which is equivalent to cycling power. The self-test function is performed as part of the reset operation. This command is described in more detail in: GPS Receiver Interface, packet 0x25.

Reset > Battery back-up

This packet commands the GPS receiver to clear data and perform a software reset. This command is described in more detail in GPS Receiver Interface, packet 0x1E.

Reset > Oscillator offset

This packet commands the GPS receiver to clear the stored oscillator offset. This command is described in more detail in GPS Receiver Interface, packet 0x1D.

Help Menu

The Help menu group provide access to data from the TFP hardware and GPS receiver.

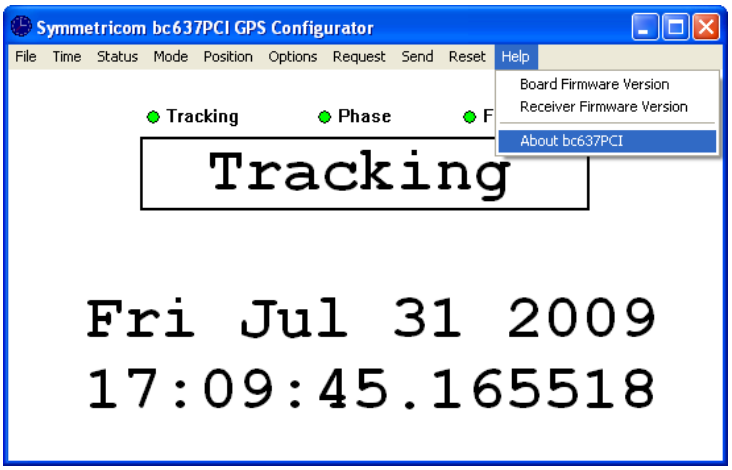


Figure 2-26: Help Menu

Help > Board Firmware Version

The Board Firmware Version command returns information from the TFP firmware. For the firmware data format, see "Command 0x4F: PCI Firmware Part Number (request only)" on page 57.

Help > Receiver Firmware Version

The Receiver Firmware Version returns firmware version information for the GPS receiver. The data format is covered in GPS Receiver Interface, packet 0x45.

Help > About bc637PCI

The About bc637PCI selection, returns information on the bc637PCIcfg.exe program.

2.3. Traytime Windows Application Program

This utility is designed to operate under Microsoft Windows XP/Vista. This system tray utility will query the TFP and set the system clock on a periodic basis. Double click on the “TrayTimeCPP.exe” to install. TrayTimeCPP.exe is copied to the same folder as the other configuration software. A small world icon will show up on the lower right portion of the desktop (where the desktop clock appears in the taskbar tray). See Figure 2-27. Click on that icon to display the TrayTimeCPP interface window. Click Setup under Selected Source on the Status tab, and click on Hardware if it has not been selected as the source for time.

Figure 2-27: Tray Time Icon

If the Current Status is: “Waiting for the board to acquire time,” then the time on the host computer has not synchronized to the TFP time yet. If the status is: “Set Clock OK” or “Captured Board Time,” then the synchronizing process is taking effect. Change the Update Interval to the desired value and press OK.
To allow the program to continue running in the background, and synchronize the System Time, minimize the window or click OK.

Pressing “Quit” will terminate the program.
Drag the program into your startup group to have it run automatically at boot time.

Caution: Using both TrayTime and either the bc635PCIcfg, or the bc637PCIcfg applications with the Local Time Offset value set to anything other than 00 (UTC) will corrupt TrayTime's time value.

2.3.1. Installation

TrayTime is now installed as integrated with the bc635 and bc637 Demo software and so it is installed along with those programs.

2.3.2. Functionality

Once the TrayTime installation is complete, a small, world icon will appear in the lower right portion of the desktop (where the clock appears), referred to as the “tray”. Click on the world icon to activate the main program window. The “Reference Status” control in the GUI indicates if the selected hardware source is “Locked” or “Unlocked” to the selected input source. If “Reference Status” shows “Unlocked”, the card is not tracking its reference source (GPS, Timecode, 1PPS etc). It is important to note that the TrayTime utility will not update the system's time until the selected hardware locks to its reference source. Once the hardware locks, TrayTime will update the system time at a periodic rate, selectable by the user.

Note: TrayTime assumes that the time read from the hardware is Universal Time Coordinate (UTC) based, and will adjust the hardware time with the machine's local time-zone offset.

2.3.3. TrayTime Dialog Windows

Main Window

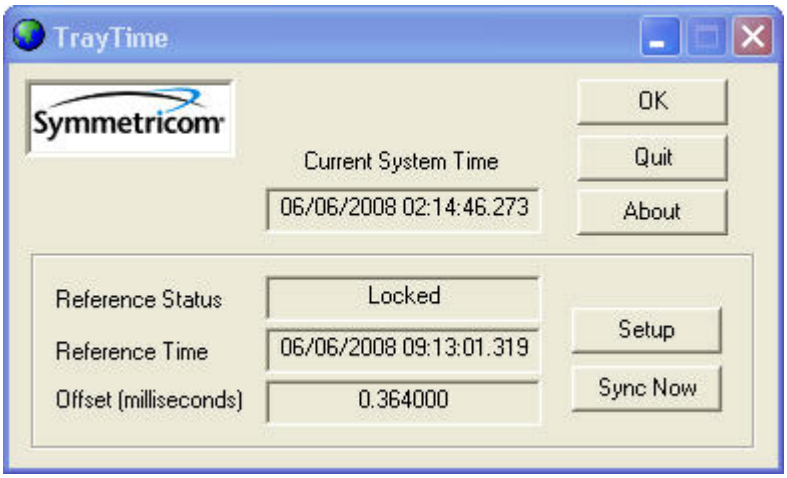


Figure 2-28: TrayTime main window

OK: Minimizes the utility into the Windows System Tray.

Quit: Exits the utility.

About: Shows utility version number.

Setup: Launches the Setup Windows,with Status and Configuration tabs. See below.

Sync Now: Commands the utility to capture time from the board and set the system's time.

Reference Status: Shows “Locked” or “Unlocked” depending on whether the reference hardware is “tracking” or “not tracking” an external time reference, respectively.

Reference Time: Shows the reference time, UTC-Based time zone.

Offset (milliseconds): Shows the time offset between the reference hardware and the system clock in milliseconds.

TrayTime Setup - Status Window

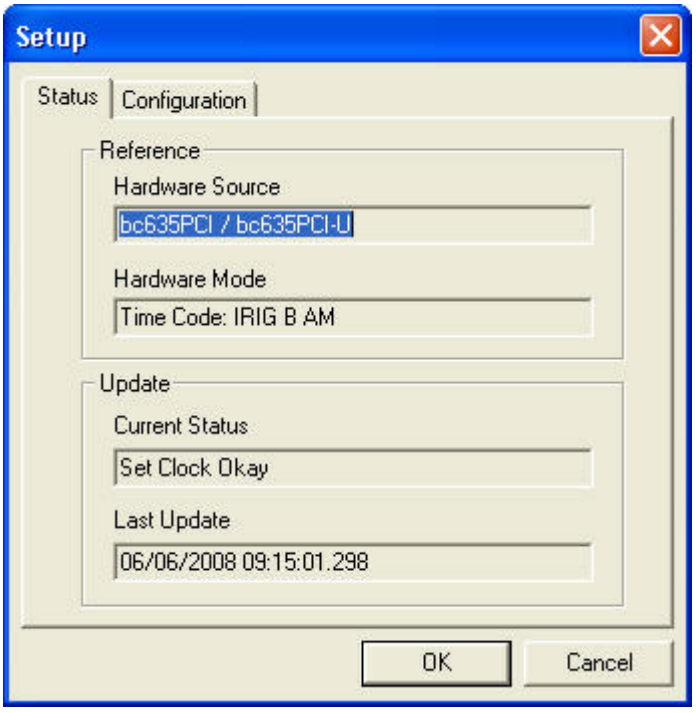


Figure 2-29: Status Window

Hardware Source: Shows the selected Hardware type used for time synchronization.

Hardware Mode: Shows the selected Hardware mode.

Current Status: Shows the current update status. If status is: “Waiting for the board to acquire time” then machine's time is not synchronized to the reference hardware since the reference hardware is not tracking an external time source. If status is: “Set Clock Okay”, then the machine's time was updated to reference hardware time.

Last Update: Time of the last update, UTC based.

TrayTime Setup - Configuration Window

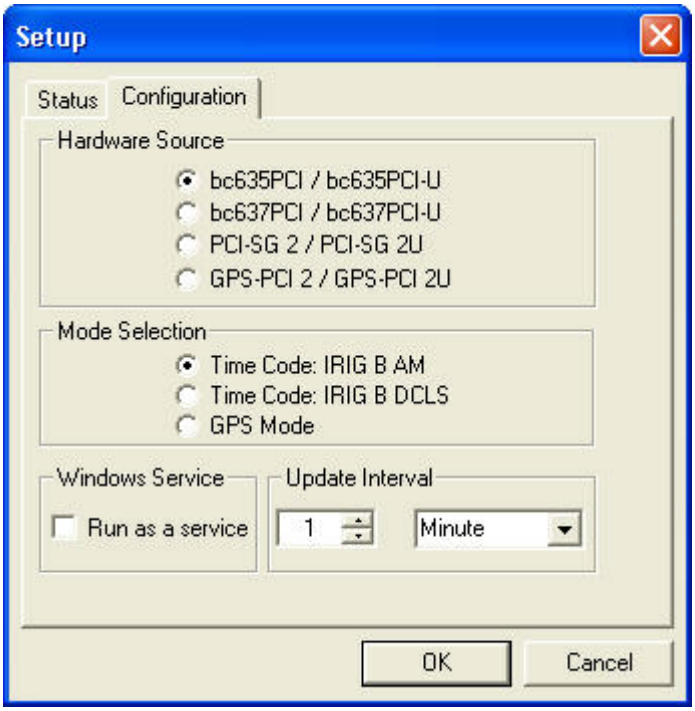


Figure 2-30: Configuration Window

Hardware Source: The TrayTime utility supports the listed 8- types of hardware sources.

Mode Selection: The TrayTime utility supports setting the mode on the selected hardware.

Note: “GPS Mode” is only available on GPS capable devices.

Windows Service: Check this option to run the TrayTime utility as a Windows service. If this option is checked, the TrayTime utility will automatically run and update the system time without a user login to the system.

Update Interval: This option sets the interval of how often the system clock time should be updated. The default interval is 1 minute.

3. Windows SDK

3.1. Introduction

3.1.1. General

The bc63xPCI-V2 and bc63xPCIe Software Developer's Kit is designed to provide a suite of tools useful in the development of applications which access features of the bc63xPCI-V2 and bc63xPCIe Time and Frequency Processor. This kit has been designed to provide an interface between the bc63xPCI-V2 and bc63xPCIe Time and Frequency Processor and applications developed for 32 and 64 bit Windows XP or newer Windows operating systems. In addition to the interface library, example programs are provided, complete with source code, in order to provide a better understanding of the kit features and benefits.

3.1.2. Features

The main features of the Software Developer's Kit include:

- API interface library with access to all features of the bc635PCI-V2, bc637PCI-V2, bc635PCIe, and bc637PCIe Time and Frequency Processor.
- Example application programs, with source code, utilizing the API interface library.
- This User's Guide providing the API library definition.

3.1.3. Overview

The Windows Software Developer's Kit was designed to provide an interface to the bc635PCI-V2, bc637PCI-V2, bc635PCIe, and bc635PCIe Time and Frequency Processor in the 32 and 64 bit Windows environments from Windows XP to the newest Windows 7. Specifically, the supported Windows operating systems include 32 and 64 bit Windows XP, Vista, Server 2003, Server 2008 and 7. The example application programs were originally developed under Microsoft Visual C++ 6.0 and ported to support both Visual C++ 6.0 and Visual Studio 2008. The project files for both Visual C++ 6.0 and Visual Studio 2008 are provided. The Visual Studio 2008 project files include 64 bit debug and release target. The example programs provide sample code that 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 that allow the developer to clip any useful code and use it in their own applications. The configurator example program provides interface dialogs to allow the operator to set any configurable parameters for operating the bc635PCI-V2, bc637PCI-V2, bc635PCIe, and bc637PCIe Time and Frequency Processor. Application programs developed using the interface library run on any 32 or 64 bit Windows XP or newer Windows operating system.

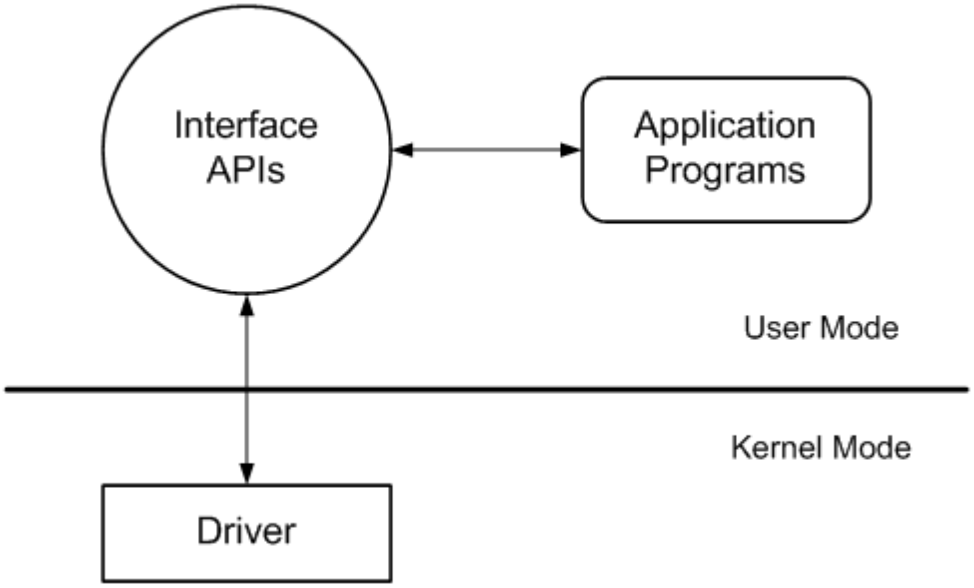


Figure 3-1

The interface API library provides an abstraction layer between the application programs and the device driver. This allows Symmetricom to advance the Time and Frequency Processor hardware features while protecting your investment in application development at the same time.

3.2. Release Notes

Driver

This preferred device driver is called 'SymmBCPCI.sys', which replaces the old WinRT device driver published originally by the Bluewater Systems. The WinRT driver was used in previous releases. The 'SymmBCPCI.sys' driver supports all types of the Windows operating systems, both 32 and 64-bit, from Windows XP to Windows 7. If your target platform is Windows 2000 or earlier, you must use a WinRT based software release (release 8.0.0 or earlier).

Installation

The InstallShield created setup program uses Microsoft WinDDK tool 'DPInst.exe' to install the target Windows driver package. 'DPInst.exe' sometimes takes a long time to install the driver package. This is because 'DPInst.exe' tries to run the 'Found New Hardware' wizard silently when installing the driver for found bc63xPCI-V2 and bc63xPCIe bus card(s). The setup program displays a DOS window with the message "Install SymmBCPCI driver package. This may take a few minutes …" Please do not interrupt the setup program while the DOS window is still shown and wait for the 'DPInst.exe' to finish.

Both the bc635PCIcfg and the bc637PCIcfg setup programs include the TrayTime.exe. The installation of the bc637PCIcfg program also installs the bc635PCIcfg.exe program.

Driver Packages

Driver packages for all supported Windows operating systems are available in the 'Driver' directory under the program installation folder. You can use the 'Add Hardware' control panel applet later to reinstall the driver package. Make sure you instruct the 'Add Hardware' wizard to search for .inf file in the driver package directory that matches your running Windows operating system.

64-Bit Applications

The installed bc635PCIcfg.exe and bc637PCIcfg.exe programs are 32-bit application programs. The 64-bit version of both programs and the required DLL are installed in the 'x64 Program Files' directory under the program installation folder.

The 'SymmBCPCI.sys' device driver supports both 32 and 64-bit application programs.

DLL File

The DLL file exporting the APIs is BC637PCI.dll. This DLL combines the APIs that were exported from both BC637PCI.dll and BC_INT.dll from the previous releases. The BC637PCI.dll has a 32 and a 64-bit version. By default, the 32 bit version is installed into the Windows\System32 directory. The 64-bit version is in the 'x64 Program Files' directory as mentioned above.

Software Developers Kit

As with previous releases, the SDK installation does not install the device driver. You must run either the bc635PCIcfg.exe or the bc637PCIcfg.exe setup program or use 'Add Hardware' control panel applet to install the device driver.

The source code for the sample programs, including both bc635PCIcfg.exe and bc637PCIcfg.exe, is in the 'Example Programs' directory under the SDK installation folder. The project files are provided in both Visual C++ 6.0 and Visual Studio 2008 formats. There are two workspace files in the 'Example Programs' directory. The 'BC635PCI SDK Examples.dsw' is for Visual C++ 6.0, and the 'BC635PCI SDK Examples_VS2008.sln' is for Visual Studio 2008. Each example program uses two directories. The main directory contains the source files and the Visual C++ 6.0 project file. The directory ending with '_VS2008' has the Visual Studio 2008 project files only. The Visual Studio 2008 project files refer to source files in the main directory. Each Visual C++ 6.0 project file has Win32 debug and release targets. Each Visual Studio 2008 project files have both Win32 and x64 debug and release targets.

The 32-bit import library file BC637PCI.lib is in the 'Lib\Win32' directory under the 'Example Programs' folder. Similarly, the same 64-bit version file is in the 'Lib\x64' directory under the 'Example Programs' folder.

TrayTime.exe

The TrayTime.exe links with APIs exported by the 'TrueTimeSDK.dll' file. The 'TrueTimeSDK.dll' supports Win32 debug and release targets only. Therefore, there is no 64-bit TrayTime.exe. There are compiler warnings when compiling TrayTime.exe in Visual Studio 2008. These warnings are due to TrayTime.exe using deprecated CRT functions. Compilation of other projects is free from compiler warnings since those source files are modified to use different CRT functions depending on the Microsoft compiler version.

API Calling Convention

In previous releases, the API functions use __cdecl calling convention. In this release, the API functions use __stdcall calling convention. The reason for the change is to allow application development using Visual Basic. Visual Basic programs can only access exported DLL functions using __stdcall calling convention. Because of the calling convention change, you must recompile your existing applications in order to use the new SDK. The current release provides all the API functions that are available in previous releases with the exact function signature. In other word, only new API functions are added but existing API functions are never taken away. Hence, the recompilation of your application should be successful without any code change.

If you cannot recompile or you don't have the source code to recompile, you must use a WinRT based software release (release 8.0.0 or earlier). Please do not install the driver and new SDK.

NoSync Read Time Functions

The major and minor time are held in two 32-bit registers. In previous releases, the read time API functions bcReadBinTime() and bcReadDecTime() do not use synchronization primitives to protect the two separate 32-bit register reads. This caused occasional problems when two quick consecutive calls of read time functions returned out of sync time values. In this release, the above two functions and the two newly added functions bcReadBinTimeEx() and bcReadDecTimeEx() use a shared critical section to protect the two 32-bit register reads. This ensures the two successive calls of read time functions return correct time values.

However, the addition of critical section introduces a slight per call delay. Sometimes, the added delay is not desirable for applications that manage the time value synchronization problem themselves. For this reason, four additional API functions bcReadBinTimeNoSync(), bcReadBinTimeExNoSync(), bcReadDecTimeNoSync() and bcReadDecTimeExNoSync() are added that do not use any critical section object. The bcReadBinTimeNoSync() and bcReadDecTimeNoSync() are the same as bcReadBinTime() and bcReadDecTime() respectively from the previous releases. You now have choices to decide which function to use in your application per your need.

3.3. Installation

Hardware and driver installation

You must install the TFP hardware and the appropriate driver software before you proceed to the Software Developer's Kit Installation. Please refer to the Section 1 of this manual for the hardware and driver software installation instruction.

Software developer's kit installation

Installation of the Software Developer's Kit is handled by executing the installer program "bc635_637PCI_SDK_Setup.exe". You must accept the license agreement and follow the instruction on the screen to install the Software Developer's Kit.

Configuration

Directory structures are created in the specified installation location. These structures contain all required files to develop Windows XP and newer Windows platform based user applications.

Test installation

You can use either Microsoft Visual C++ 6.0 or Microsoft Visual Studio 2008 to test the installation. Use Microsoft Visual C++ 6.0 to open the workspace “BC635PCI SDK EXAMPLES.dsw” in the “Example Programs” directory under the installation location. Select “Rebuild All” in the Visual C++ 6.0 IDE to build all the sample programs. You can select the program you want to run from the Visual C++ 6.0 IDE by making that project as the active project. The bc635cpp project contains all source code for the bc635PCIcfg.exe program. The bc637pci project contains all the source code for the bc637PCIcfg.exe program. The TrayTimeCpp project contains all the source code the TrayTime.exe program.

Use Microsoft Visual Studio 2008 to open the solution “BC635PCI SDK EXAMPLES_2008.sln” in the “Example Programs” directory under the installation location. You can select to build Win32 or x64 sample programs. Make your choice and select “Rebuild All” in the IDE to build all the sample programs. Please note that 64-bit TrayTime.exe will not link due to the lack of 64-bit ‘TrueTimeSDK.lib’. If you want to test 64-bit sample programs, you also must make sure the 64-bit BC637PCI.DLL is copied to the respective Windows system directory.

If a device open error is returned, the hardware interface was not installed or configured properly. Verify that the correct driver was installed according to the guidelines in the “bc635PCI-V2, bc637PCI-V2 Users Guide”.

Project creation

Microsoft Visual C++ 6.0

If you want to use BC637PCI.dll in your own project, you may follow the instructions below:

1. Update your project setting - C++ preprocessors “Additional include directories” to include the directory where bc637pci.h and bc_int.h can be found. They are in the “Include” directory under the SDK installation directory.

2. Update your project setting - Link's “Additional library path” to include the directory where BC637PCI.lib can be found. They are in the “Lib\Win32” directory under the SDK installation directory. Add BC637PCI.lib to the “Object/library modules” list.

3. If building a new project similar to TrayTimeCpp, you may need to change the project settings:

   a. For both debug version and release version, go to "C/C++" tab, select "Precompiled Headers" category and then check "Not using precompiled headers" button. Next, go to the Link tab, select "General category" and add "ws2_32.lib" to "Object/Library Module" edit box.
   b. For release version, Link tab, select "Customize" category and then check "Force File Output" box.

4. Copy the shared DLL BC637PCI.DLL to the system directory and make sure install the “SymmBCPCI.sys” driver.Microsoft Visual Studio 2008.

Microsoft Visual Studio 2008

You can create Visual Studio 2008 project following the similar steps as those outlined using Visual C++ 6.0. Using Visual Studio 2008 you can create 64-bit applications. The 64-bit import library BC637PCI.lib is in the “Lib\x64” directory under the SDK installation directory. The 64-bit DLL BC637PCI.DLL is in the “Bin\x64” directory under the SDK installation directory.

3.4. Library definitions

General

The interface library provides functions for each of the programming packets supported by the bc635PCI-V2, bc637PCI-V2, bc635PCIe, and bc637PCIe 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 earlier sections of this Users Guide.

Windows SDK Command Finder

[caption]

Windows SDK Functional Command Summary	
"bcStartPCI" on page 109	Opens the underlying hardware layer.
"bcStopPCI" on page 109	Closes the underlying hardware layer.
"bcGetReg" on page 110	Returns the contents of the requested register.
"bcSetReg" on page 110	Sets the contents of the requested register.
"bcGetDPReg" on page 110	Returns the contents of the requested register.
"bcSetDPReg" on page 110	Sets the contents of the requested register.
"ReadBinTime" on page 111	Latches and returns time captured from the time registers.
"bcReadDecTime" on page 111	Latches and returns time captured from the time registers.
"bcReadBinTimeEx" on page 111	Latches and returns time captured from the time registers.
"bcReadDecTimeEx" on page 112	Latches and returns time captured from the time registers.
"ReadBinTimeNoSync" on page 112	Latches and returns time captured from the time registers.
"bcReadBinTimeExNoSync" on page 113	Latches and returns time captured from the time registers.
"bcReadEventTime" on page 114	Latches and returns time captured from the time registers.
"bcReadEventTimeEx" on page 114	Latches and returns time captured caused by an external event..
"bcReadDecTimeNoSync" on page 113	
"ReadDecTimeExNoSync" on page 113	
"bcSetBinTime" on page 114	Description: Sets the major time buffer.
"bcSetBinTime" on page 114	
"bcSetBCDTime" on page 114	Sets the major time buffer.
"bcSetYear" on page 115	Sets the current year value.
"bcSetMode" on page 115	Sets the operating mode of the board.
"bcCommand" on page 115	Sends reset command to the board.
"bcSetDac" on page 116	Sets new dac value.
"bcSetHbt" on page 116	Programs a periodic output (synchronous or asynchronous to 1pps).
"bcSetPDelay" on page 116	Programs a propagation delay into the timing engine to account for delays introduced by long cable runs.
"bcSetTcIn" on page 117	Sets time code type and format for operating mode 0 (time code mode).
"bcSetClkSrc" on page 117	Sets the 10MHz-clock source for the board.
"bcSetGenCode" on page 117	Sets the time code generator format.
"bcSetLocOff" on page 118	Programs the board to operate at an offset from UTC.
"bcReqOscData" on page 118	Returns oscillator data from the board.
"bcReqTimeData" on page 118	Returns time data from the board.
"bcReqTimeCodeData" on page 118	Returns timecode data from the board.
"bcReqOtherData" on page 119	Returns other data from the board.
"bcReqVerData" on page 119	Returns version data from the board.
"bcReqManufData" on page 119	Returns manufacture data from the board.
"bcSetGain" on page 120	Modifies the internal oscillator frequency control algorithm.

Windows SDK Functional Command Summary	
"bcSetGenOff" on page 121	Programs the board time code generator to operate at an offset from UTC.
"bcSetJam" on page 121	Modifies the internal oscillator frequency control algorithm.
"bcSetDis" on page 121	Modifies the internal oscillator frequency control algorithm.
"bcSetLocalFlag" on page 121	Enables or disables the local time offset entered using bcSetLocOff( ) function.
"bcSetDayLightFlag" on page 122	This command only applies Decoding IEEE 1344 Time Code. If the dlight_flag is enabled, the TFP adjusts its time by one hour.
"bcYearAutoInc" on page 122	This function is deprecated. It is provided for backwards compatibility.
"bcSetTmFmt" on page 122	Modifies the format of the major time data returned by the board.
"bcSetUtcCtl" on page 123	Modifies the time base in GPS mode.
"bcSetLeapEvent" on page 123	This command can be used in modes other than GPS mode for inserting or deletion of one leap second.
"bcAdjustClock" on page 124	This command advances or retards the TFP internal clock.
"bcSpecialBoot" on page 124	The Special Boot is no longer supported in the V2 hardware.
"bcSetPciCard" on page 124	This command sets the manufacture settings of the module.
"bcForceJam" on page 125	This command forces the TFP to Jam-Sync on the next rising edge of the 1PPS output.
"bcSyncRtc" on page 125	This command forces the TFP to Synchronize the RTC time to current time.
"bcDisRtcBatt" on page 125	This command disconnects the RTC IC from the Battery after power is turned off.
"bcGPSReq" on page 126	Retrieves a data packet from the GPS receiver.
"bcGPSSnd" on page 126	Sends a data packet from the GPS receiver.
"bcGPSMan" on page 126	Manually sends and retrieves data packets from the GPS receiver.
"bcGPSOperMode" on page 127	This function should only be used when the TFP is in GPS Mode of Operation.
"bcStartInt" on page 127	Starts the interrupt thread to signal interrupts.
"bcStopInt" on page 127	Stops the interrupt thread and releases any used resources.
"bcSetInts" on page 127	Enables one interrupt source.
"bcReqInts" on page 128	Query the currently enabled interrupt.
"bcGetLastInts" on page 128	Query the last signaled interrupt(s).
"bcSetMultInts" on page 129	Starts the interrupt thread to signal interrupts.
"bcReqRevisionID" on page 129	Returns the revision ID of the hardware.
"bcReqTimeCodeDataEx" on page 129	Returns timecode data from the board.
"bcSetPeriodicDDSSelect" on page 130	This command selects periodic output or DDS output.
"bcSetPeriodicDDSEnable" on page 130	This command enables or disables periodic or DDS output.

Windows SDK Functional Command Summary	
"bcSetDDSDivider" on page 130	This command sets the DDS divider for DDS output.
"bcSetDDSDividerSource" on page 131	This command sets the DDS divider source for DDS output.
"bcSetDDSSyncMode" on page 131	This command sets the DDS synchronization mode for DDS output.
"bcSetDDSMultiplier" on page 132	This command sets the DDS multiplier for DDS output.
"bcSetDDSPeriodValue" on page 132	This command sets the DDS period value for DDS output.
"bcSetDDSTuning Word" on page 132	This command sets the DDS tuning word for DDS output.
"bcSetDDSFrequency" on page 133	This function sets the DDS output frequency.
"bcSetTcInEx" on page 133	Sets time code type, subtype and modulation type for operating mode 0 (time code mode).
"bcSetGenCodeEx" on page 135	Sets the time code generator format and its subtype.
"bcReadEvent2TimeEx" on page 137	Latches and returns time captured caused by a external event2
"bcReadEvent3TimeEx" on page 137	Latches and returns time captured caused by a external event3
"bcReqOtherDataEx" on page 138	Returns other data, which are extended now, from the board.
"bcReqEventsData" on page 138	Returns event, event2 and event3 data.
"bcSetEventsData" on page 138	Sets event, event2 and event3 data.

Functions

bcStartPCI	
Prototype	int bcStartPCI (INT devno);
Packet	N/A
Input Parameter	Device Number (>= 0)
Returns	RC_OK On Success RC_ERROR On Failure
Description: This opens the underlying hardware layer. ‘devno’ is the card number. It is 0 based. Use 0 if only one card in the system. There is no limit on the number cards you can access in your system.	

bcStopPCI	
Prototype	int bcStopPCI (void);
Packet	N/A
Input Parameter	None
Returns	RC_OK On Success RC_ERROR On Failure
Description: Closes the underlying hardware layer.	

bcGetReg	
Prototype	int bcGetReg (UINT offset, ULONG *data);
Packet	N/A
Input Parameter	Offset = 0 based offset of requested register.. Data = pointer to unsigned long to return value requested.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Returns the contents of the requested register.	

Note: This command operates on the LCA registers (status/config registers).

bcSetReg	
Prototype	int bcSetReg (UINT offset, ULONG *data)
Packet	N/A
Input Parameter	Offset = 0 based offset of requested register. Data = pointer to unsigned long value to be set.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Sets the contents of the requested register.	

Note: This command operates on the LCA registers (status/config registers).

bcGetDPReg	
Prototype	int bcGetDPReg (UINT offset, UCHAR *data);
Packet	N/A
Input Parameter	Offset = 0 based offset of requested register. Data = pointer to unsigned char to return value requested.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Returns the contents of the requested register.	

Note: This command operates on the Dual Port Memory (packet interface).

bcSetDPReg	
Prototype	int bcSetDPReg (UINT offset, UCHAR *data)
Packet	N/A
Input Parameter	Offset = 0 based offset of requested register. Data = pointer to unsigned char value to be set.
Returns	RC_OK On Success RC_ERROR On Failure

bcSetDPReg
Description: Sets the contents of the requested register.

Note: This command operates on the Dual Port Memory (packet interface).

ReadBinTime	
Prototype	int bcReadBinTime (ULONG *major, ULONG *min, UCHAR * stat);
Packet	N/A
Input Parameter	major = unsigned long pointer to store major time (Unix format). min = unsigned long pointer to store microseconds. stat = unsigned char to store status bits.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Latches and returns time captured from the time registers.	

Note:The function uses the shared critical section object to synchronize the two 32 bit major and minor register reads.

bcReadBinTimeEx	
Prototype	int bcReadBinTimeEx (ULONG *major, ULONG *min, USHORT *nano, UCHAR *stat);
Packet	N/A
Input Parameter	major = unsigned long pointer to store major time (Unix format). min = unsigned long pointer to store microseconds. nano = pointer to unsigned short to store 100 nano seconds count. stat = unsigned char to store status bits.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Latches and returns time captured from the time registers.	

Note: The function uses the shared critical section object to synchronize the two 32 bit major and minor register reads.

bcReadDecTime	
Prototype	int bcReadDecTime (struct tm *ptm, ULONG *min, UCHAR *stat);
Packet	N/A

bcReadDecTime	
Input Parameter	ptm = pointer to tm struct to store major time (calendar format). min = pointer to unsigned long to store microseconds. stat = pointer to unsigned char to store status bits.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Latches and returns time captured from the time registers.	

Note: The function use the shared critical section object to synchronize the two 32 bit major and minor register reads.

bcReadDecTimeEx	
Prototype	int bcReadDecTimeEx (struct tm *ptm, ULONG *min, USHORT *nano, UCHAR *stat);
Packet	N/A
Input Parameter	ptm = pointer to tm struct to store major time (calendar format). min = pointer to unsigned long to store microseconds. nano = pointer to unsigned short to store 100 nano seconds count. stat = pointer to unsigned char to store status bits.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Latches and returns time captured from the time registers.	

Note: The function uses the shared critical section object to synchronize the two 32 bit major and minor register reads.

ReadBinTimeNoSync	
Prototype	int bcReadBinTimeNoSync (ULONG *major, ULONG *min, UCHAR * stat);
Packet	N/A
Input Parameter	major = unsigned long pointer to store major time (Unix format). min = unsigned long pointer to store microseconds. stat = unsigned char to store status bits.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Latches and returns time captured from the time registers.	

Note:The function does NOT use any critical section object to synchronize the two 32 bit major and minor register reads.

bcReadBinTimeExNoSync	
Prototype	int bcReadBinTimeExNoSync (ULONG *major, ULONG *min, USHORT *nano, UCHAR *stat);
Packet	N/A
Input Parameter	major = unsigned long pointer to store major time (Unix format). min = unsigned long pointer to store microseconds. nano = pointer to unsigned short to store 100 nano seconds count. stat = unsigned char to store status bits.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Latches and returns time captured from the time registers.	

Note: The function does NOT use any critical section object to synchronize the two 32 bit major and minor register reads.

bcReadDecTimeNoSync	
Prototype	int bcReadDecTimeNoSync (struct tm *ptm, ULONG *min, UCHAR *stat);
Packet	N/A
Input Parameter	ptm = pointer to tm struct to store major time (calendar format). min = pointer to unsigned long to store microseconds. stat = pointer to unsigned char to store status bits.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Latches and returns time captured from the time registers.	

Note: The function does NOT use any critical section object to synchronize the two 32 bit major and minor register reads.

ReadDecTimeExNoSync	
Prototype	int bcReadDecTimeExNoSync (struct tm *ptm, ULONG *min, USHORT *nano, UCHAR *stat);
Packet	N/A
Input Parameter	ptm = pointer to tm struct to store major time (calendar format). min = pointer to unsigned long to store microseconds. nano = pointer to unsigned short to store 100 nano seconds count. stat = pointer to unsigned char to store status bits.

ReadDecTimeExNoSync	
Returns	RC_OK On Success RC_ERROR On Failure
Description: Latches and returns time captured from the time registers.	

Note:The function does NOT use any critical section object to synchronize the two 32 bit major and minor register reads.

bcReadEventTime	
Prototype	int bcReadEventTime (ULONG *maj, ULONG *min);
Packet	N/A
Input Parameter	maj = pointer to unsigned long to store major time (Unix format). min = pointer to unsigned long to store microseconds.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Latches and returns time captured caused by an external event.	

bcReadEventTimeEx	
Prototype	int bcReadEventTime (ULONG *maj, ULONG *min, USHORT *nano);
Packet	N/A
Input Parameter	maj = pointer to unsigned long to store major time (Unix format). min = pointer to unsigned long to store microseconds. nano = pointer to unsigned short to store 100 nano seconds count.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Latches and returns time captured caused by an external event.	

bcSetBinTime	
Prototype	int bcSetBinTime (ULONG newtime);
Packet	0x12
Input Parameter	newtime = unsigned long time value to set (Unix format).
Returns	RC_OK On Success RC_ERROR On Failure
Description: Sets the major time buffer.	

bcSetBCDTime	
Prototype	int bcSetBCDTime (struct tm stm);

bcSetBCDTime	
Packet	0x12
Input Parameter	stm = tm struct containing new time values to set.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Sets the major time buffer.	

bcSetYear	
Prototype	int bcSetYear (INT year);
Packet	0x13
Input Parameter	year = int value of new year (1990-2036).
Returns	RC_OK On Success RC_ERROR On Failure
Description: Sets the current year value.	

bcSetMode	
Prototype	int bcSetMode (UCHAR mode);
Packet	0x10
Input Parameter	UCHAR mode: Sets the TFP operating mode. Note: The following are defined in the "Bc637pci.h" header file; #define MODE _ IRIG 0x00 #define MODE _FREE 0x01 #define MODE_1pps 0x02 #define MODE _RTC 0x03 #define MODE _GPS 0x06
Returns	RC_OK On Success RC_ERROR On Failure
Description: Sets the operating mode of the board.	

bcCommand	
Prototype	int bcCommand (INT command);
Packet	0x1A
Input Parameter	int command = requested command action Note: The following are defined in the "Bc637pci.h" header file. #define CMD_ WARMSTART 0x01
Returns	RC_OK On Success RC_ERROR On Failure
Description: Sends reset command to the board.	

bcSetDac	
Prototype	int bcSetDac (INT dacval);
Packet	0x24
Input Parameter	int dacval = new d/a value to modify frequency of internal oscillator. Allowed values 0x0000 - 0xffff.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Sets new dac value.	

Note: This command is not required for standard operation of the device. Be sure to understand the effects of this operation before utilizing this command.

bcSetHbt	
Prototype	int bcSetHbt (CHAR mode, INT cnt1, INT cnt2);
Packet	0x14
Input Parameter	char mode = requested mode int cnt1 = divisor 1 int cnt2 = divisor 2
Returns	RC_OK On Success RC_ERROR On Failure
Description: Programs a periodic output (synchronous or asynchronous to 1pps).	

bcSetPDelay	
Prototype	int bcSetPDelay (LONG delay);
Packet	0x17
Input Parameter	long int delay = propagation delay (-9999999 to +9999999 1 00ns steps)
Returns	RC_OK On Success RC_ERROR On Failure
Description: Programs a propagation delay into the timing engine to account for delays introduced by long cable runs.	

Note: Usage of a propagation delay value with an absolute value larger than 1 millisecond (or 10000 steps) requires first that the user disable jamsynchs. Refer to Chapter 1 for more information.

bcSetTcIn	
Prototype	int bcSetTcIn (UCHAR format, UCHAR type);
Packet	0x15, 0x16
Input Parameter	Unsigned char format = time code format. Unsigned char type = modulation type of time code. Note: The following are defined in the "Bc637pci.h" header file; format #define TCODE _IRIG _A 'A' #define TCODE _IRIG _B 'B' #define TCODE _IEEE 'I' type #define TCODE _MOD _AM 'M' #define TCODE _MOD _DC 'D'
Returns	RC_OK On Success RC_ERROR On Failure
Description: Sets time code type and format for operating mode 0 (time code mode).	

bcSetClkSrc	
Prototype	int bcSetClkSrc (UCHAR which);
Packet	0x20
Input Parameter	Unsigned char which = which clock source (internal | external). Note: The following are defined in the "Bc637pci.h" header file; #define CLK_INT 'I'/* Use on-board clock */ #define CLK_EXT 'E'/* Use external clock */
Returns	RC_OK On Success RC_ERROR On Failure
Description: Sets the 10MHz-clock source for the board.	

Note: This command is not required for standard operation of the device. Be sure to understand the effects of this operation before utilizing this command.

bcSetGenCode	
Prototype	int bcSetGenCode (UCHAR format);
Packet	0x1B
Input Parameter	Unsigned char format = time code format. Note: The following are defined in the "Bc637pci.h" header file; #define TCODE_GEN_B 'B' #define TCODE _GEN _I 'I'
Returns	RC_OK On Success

bcSetGenCode	
	RC_ERROR On Failure
Description: Sets the time code generator format.	

bcSetLocOff	
Prototype	int bcSetLocOff (INT Offset, UCHAR half)
Packet	0x1D
Input Parameter	INT Offset = hours from input time source. (-16 - +16). UCHAR half = half hour increment 0: No Half Hour increment 1: Add Half Hour increment
Returns	RC_OK On Success RC_ERROR On Failure
Description: Programs the board to operate at an offset from UTC.	

bcReqOscData	
Prototype	int bcReqOscData (OscData *pdata);
Packet	0x19
Input Parameter	pdata = pointer to OscData structure. The structure is defined in the "Bc637pci.h" header file.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Returns oscillator data from the board.	

bcReqTimeData	
Prototype	int bcReqTimeData (TimeData *pdata);
Packet	0x19
Input Parameter	pdata = pointer to TimeData structure. The structure is defined in the "Bc637pci.h" header file.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Returns time data from the board.	

Note: See the header file for TimeData structure to understand values being returned.

bcReqTimeCodeData	
Prototype	int bcReqTimeCodeData (TimeCodeData *pdata);

bcReqTimeCodeData	
Packet	0x19
Input Parameter	pdata = pointer to TimeCodeData structure. The structure is defined in the "Bc637pci.h" header file.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Returns timecode data from the board.	

Note: See the header file for TimeCodeData structure to understand values being returned.

bcReqOtherData	
Prototype	int bcReqOtherData (OtherData *pdata);
Packet	0x19
Input Parameter	pdata = pointer to OtherData structure. The structure is defined in the "Bc637pci.h" header file.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Returns other data from the board.	

Note: See the header file for OtherData structure to understand values being returned.

bcReqVerData	
Prototype	int bcReqVerData (VerData *pdata);
Packet	0x19
Input Parameter	pdata = pointer to VerData structure. The structure is defined in the "Bc637pci.h" header file.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Returns version data from the board.	

bcReqManufData	
Prototype	int bcReqManufData (ManufData *pdata);
Packet	0x19
Input Parameter	pdata = pointer to ManufData structure. The structure is defined in the "Bc637pci.h" header file.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Returns manufacture data from the board.	

Note: See the header file for ManufData structure to understand values being returned.

bcSetGain	
Prototype	int bcSetGain (INT gain);
Packet	0x25
Input Parameter	int gain = oscillator control filter gain value.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Modifies the internal oscillator frequency control algorithm.	

Note: This command is not required for standard operation of the device. Be sure to understand the effects of this operation before utilizing this command.

bcSetGenOff	
Prototype	int bcSetGenOff (INT Offset, UCHAR half)
Packet	0x1C
Input Parameter	INT offset = hours from input time source. (-16 - +16). UCHAR half = half hour increment 0: No Half Hour increment 1: Add Half Hour increment
Returns	RC_OK On Success RC_ERROR On Failure
Description: Programs the board time code generator to operate at an offset from UTC.	

bcSetJam	
Prototype	int bcSetJam (INT jam_ctl);
Packet	0x21
Input Parameter	INT jam_ctl = 0 or 1 0 = Jam-Synchs Disabled 1 = Jam-Synchs Enabled
Returns	RC_OK On Success RC_ERROR On Failure
Description: Modifies the internal oscillator frequency control algorithm.	

Note: This command is not required for standard operation of the device. Be sure to understand the effects of this operation before utilizing this command.

bcSetDis	
Prototype	int bcSetDis (INT dis_ctl);
Packet	0x23
Input Parameter	INT dis_ctl = 0 or 1 0 = Oscillator Disciplining Disabled 1 = Oscillator Disciplining Enabled(default)
Returns	RC_OK On Success RC_ERROR On Failure
Description: Modifies the internal oscillator frequency control algorithm.	

Note: This command is not required for standard operation of the device. Be sure to understand the effects of this operation before utilizing this command.

bcSetLocalFlag	
Prototype	int bcSetLocalFlag (UCHAR local_flag);

bcSetLocalFlag	
Packet	0x40
Input Parameter	UCHAR local_flag = 0 or 1 Note: The following are defined in the "Bc637pci.h" header file; #define LOCAL _FLAG _DIS(0)(default) #define LOCAL _FLAG _ENA (1)
Returns	RC_OK On Success RC_ERROR On Failure
Description: Enables or disables the local time offset entered using bcSetLocOff( ) function.	

bcSetDayLightFlag	
Prototype	int bcSetDayLightFlag (UCHAR dlight_flag);
Packet	0x41
Input Parameter	UCHAR dlight_flag = 0 or 1 Note: The following are defined in the "Bc637pci.h" header file; #define DAY _LIGHT _DIS (0)(default) #define DAY _LIGHT _ENA (1)
Returns	RC_OK On Success RC_ERROR On Failure
Description: This command only applies Decoding IEEE 1344 Time Code. If the dlight_flag is enabled, the TFP adjusts its time by one hour.	

bcYearAutoInc	
Prototype	int bcYearInc (UCHAR year_inc);
Packet	0x42
Input Parameter	UCHAR year_inc = 0 or 1 Note: The following are defined in the "Bc637pci.h" header file; #define YEAR _AUTO _DIS (0) #define YEAR _AUTO _ENA (1) (default)
Returns	RC_OK On Success RC_ERROR On Failure
Description: This function is deprecated. It is provided for backwards compatibility. The V2 hardware automatically increments year. For the V1 hardware, 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.	

bcSetTmFmt	
Prototype	int bcSetTmFmt (INT tm_fmt);

bcSetTmFmt	
Packet	0x11
Input Parameter	INT tm_fmt = 0 or 1 1 = Binary Time Format (Default) 0 = Decimal Time Format
Returns	RC_OK On Success RC_ERROR On Failure
Description: Modifies the format of the major time data returned by the board.	

Note: This command is not required for standard operation of the device. Be sure to understand the effects of this operation before utilizing this command.

bcSetUtcCtl	
Prototype	int bcSetUtcCtl (INT utc_ctl);
Packet	0x33
Input Parameter	INT utc_ctl = 0 or 1 0 = UTC Format (Default) 1 = GPS Format
Returns	RC_OK On Success RC_ERROR On Failure
Description: Modifies the time base in GPS mode. This command determines whether the board will correct the received GPS time for leap second offset and events.	

Note: This command is not required for standard operation of the device. Be sure to understand the effects of this operation before utilizing this command.

bcSetLeapEvent	
Prototype	int bcSetLeapEvent (CHAR flag, ULONG levent);
Packet	0x1E
Input Parameter	CHAR flag = -1, 0, 1 -1 = Deletion of 1 second 0 = Disable 1 = Insertion of 1 second ULONG levent = Unix time since January, 1st 1970
Returns	RC_OK On Success RC_ERROR On Failure
Description: This command can be used in modes other than GPS mode for inserting or deletion of one leap second.	

Note: This command is not required when the TFP is in GPS mode since the TFP automatically handles Leap seconds insertion or deletion.

bcAdjustClock	
Prototype	int bcAdjustClock (LONG clkvalue);
Packet	0x29
Input Parameter	LONG clkvalue = 0x80000000 to 0x7FFFFFFF
Returns	RC_OK On Success RC_ERROR On Failure
Description: This command advances or retards the TFP internal clock. The TFP can adjust its clock up to 100 milliseconds per each second. Each count is equal to 10 microseconds.	

Note: This command is not required for standard operation of the device. Be sure to understand the effects of this operation before utilizing this command.

bcSpecialBoot	
Prototype	int bcSpecialBoot (INT sp_boot);
Packet	0xFB
Input Parameter	INT sp_boot; Note: The following are defined in the "Bc637pci.h" header file; #define NORMAL _ BOOT 0x0000 #define SPECIAL _BOOT 0x0 100
Returns	RC_OK On Success RC_ERROR On Failure
Description: The Special Boot is no longer supported in the V2 hardware. This function is provided for backwards compatibility. For the V1 hardware, in Special Boot Configuration, the TFP ignores additional resets after power-on.	

Note: This command is not required for standard operation of the device. Be sure to understand the effects of this operation before utilizing this command.

bcSetPciCard	
Prototype	int bcSetPciCard (UCHAR setting, ULONG password, INT data);
Packet	0xFE
Input Parameter	Note: The following are defined in the "Bc637pci.h" header file; UCHAR setting = module settings #define MODEL _ID 0x04/* IRIG or GPS Model */ #define CRYSTAL _ID 0x03/* Standard or Oven crystal */ ULONG password; Symmetricom will supply the password INT data;

bcSetPciCard	
	For Model ID #define SET_BC635 0x0635 #define SET_BC637 0x0637 For Crystal ID #define STD _CRYSTAL 0x0002 #define MTI _CRYSTAL 0x0014
Returns	RC_OK On Success RC_ERROR On Failure
Description: This command sets the manufacture settings of the module. To change any of these settings, a password is required. This command is mainly used for field upgrades.	

bcForceJam	
Prototype	int bcForceJam (void);
Packet	0x22
Input Parameter	None
Returns	RC_OK On Success RC_ERROR On Failure
Description: This command forces the TFP to Jam-Sync on the next rising edge of the 1PPS output. The Jam-Sync bit must be enabled before using this command.	

bcSyncRtc	
Prototype	int bcSyncRtc (void);
Packet	0x27
Input Parameter	None
Returns	RC_OK On Success RC_ERROR On Failure
Description: This command forces the TFP to Synchronize the RTC time to current time.	

bcDisRtcBatt	
Prototype	int bcDisRtcBatt (void);
Packet	0x28
Input Parameter	None
Returns	RC_OK On Success RC_ERROR On Failure
Description: 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.	

bcGPSReq	
Prototype	int bcGPSReq (GpsPkt *ptr);
Packet	0x31
Input Parameter	GpsPkt *ptr This structure commands information detailing the packet to retrieve and the buffer where the data will be stored.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Retrieves a data packet from the GPS receiver. (See PACKET 0x3 1 definition.)	

bcGPSSnd	
Prototype	int bcGPSSnd (GpsPkt *ptr);
Packet	0x30
Input Parameter	GpsPkt *ptr This structure commands information detailing the packet to send and the buffer where the data is stored.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Sends a data packet from the GPS receiver. (See PACKET 0x30 definition.)	

bcGPSMan	
Prototype	int bcGPSMan (GpsPkt *ptrIn, GpsPkt *ptrOut);
Packet	0x32
Input Parameter	GpsPkt *ptrIn This structure commands information detailing the packet to send and the buffer where the data is stored. .GpsPkt *ptrOut This structure commands information detailing the packet to retrieve and the buffer where the data will be stored.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Manually sends and retrieves data packets from the GPS receiver. (See PACKET 0x32 definition.)	

bcGPSOperMode	
Prototype	int bcGPSReq (UCHAR static);
Packet	0x34
Input Parameter	UCHAR static = 0 or 1 Note: The following are defined in the "Bc637pci.h" header file; #define GPS _NONE _STATIC (0) #define GPS _STATIC (1) (default)
Returns	RC_OK On Success RC_ERROR On Failure
Description: 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 for detail description on this feature. This function should only be used when the TFP is in GPS Mode of Operation.	

bcStartInt	
Prototype	HANDLE bcStartInt (INT dev_no);
Packet	N/A
Input Parameter	INT dev_no = bc635PCI device to open (same used in bcStartPCI).
Returns	Handle to the event object On Success RC_ERROR On Failure
Description: Starts the interrupt thread to signal interrupts.	
Notes: See IntrSamp.c that shows how to use interrupts.	

bcStopInt	
Prototype	int bcStopInt (void);
Packet	N/A
Input Parameter	None
Returns	RC_OK On Success RC_ERROR On Failure
Description: Stops the interrupt thread and releases any used resources.	
Notes: * See IntrSamp.c that shows how to use interrupts.	

bcSetInts	
Prototype	int bcSetInts (ULONG *mask, INT *latch);
Packet	N/A
Input Parameter	ULONG *mask = pointer to mask to load into INTERRUPT MASK register. Note: The following are defined in the header file; #define INT _EVENT (1<<0) #define INT _HBEAT (1<<1) #define INT _STROBE (1<<2) #define INT_1pps (1<<3) #define INT _DPA (1<<4) INT *latch = 0 or 1

bcSetInts	
	0 = do not latch time in event registers when interrupt detected. 1 = latch time in event registers when interrupt detected.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Enables one interrupt source.	
Notes: See IntrSamp.c that shows how to use interrupts.	

Note: Refer to the Chapter 1 for more information regarding allowed values for the INTERRUPT MASK.

bcReqInts	
Prototype	int bcReqInts (ULONG *mask, INT *latch);
Packet	None
Input Parameter	ULONG *mask = pointer to mask to load from INTERRUPT MASK register. INT *latch = value of current latch setting.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Query the currently enabled interrupt.	
Notes: See IntrSamp.c that shows how to use interrupts.	

Note: Refer to Chapter 1 for more information regarding allowed values for the INTERRUPT MASK.

bcGetLastInts	
Prototype	int bcGetLastInts (ULONG *lastInts);
Packet	None
Input Parameter	ULONG * lastInts = pointer to last signal interrupt.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Query the last signaled interrupt(s).	
Notes: See MultSamp.c for demonstration of this function. Multiple interrupts support is enabled using bcSetMultInts	

Note: Refer to Chaper 1 for more information regarding allowed values for the INTERRUPT MASK.

bcSetMultInts	
Prototype	HANDLE bcSetMultInts (INT enable);
Packet	N/A
Input Parameter	INT enable = flag to enable or disable multiple interrupt use. The default is single interrupt use. 0: disable multiple interrupts (default). Single interrupt only. 1: enable multiple interrupts
Returns	RC_OK On Success RC_ERROR On Failure
Description: Starts the interrupt thread to signal interrupts.	
Notes: * See MultSamp.c for demonstration of this function.	

bcReqRevisionID	
Prototype	int bcReqRevisionID (UCHAR *pID);
Packet	N/A
Input Parameter	*pID = revision ID of the hardware.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Returns the revision ID of the hardware. The ID for the V2 hardware is in the range of [0x20, 0x2F].	

Note: The revision ID is stored in the 8 bit Revision ID field of the PCI Configure space. You can retrieve the revision ID from reading the PCI Configure space.

bcReqTimeCodeDataEx	
Prototype	int bcReqTimeCodeDataEx (TimeCodeDataEx *pdata);
Packet	0x19
Input Parameter	pdata = pointer to TimeCodeDataEx structure. The structure is defined in the "Bc637pci.h" header file.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Returns timecode data from the board.	

Note: The time code has been extended in the V2 hardware. It now consists of the time code and its sub type. The original bcReqTimeCode() function is still supported. But it does not return sub type information. See the header file for TimeCodeDataEx structure to understand values being returned.

bcSetPeriodicDDSSelect	
Prototype	int bcSetPeriodicDDSSelect (UCHAR sel);
Packet	0x43
Input Parameter	UCHAR sel = 0 or 1 Note: The following are defined in the "Bc637pci.h" header file; #define SELECT _PERIODIC _OUT (0x0) #define SELECT _DDS _OUT (0x1)
Returns	RC_OK On Success RC_ERROR On Failure
Description: This command selects periodic output or DDS output.	

Note: This function only makes the output choice. You have to call bcSetPeriodicDDSEnable() (see below) to enable the output unless the output has already been enabled.

bcSetPeriodicDDSEnable	
Prototype	int bcSetPeriodicDDSEnable (UCHAR enable);
Packet	0x44
Input Parameter	INT enable = flag to enable or disable periodic or DDS output 0: disable periodic or DDS output. 1: enable periodic or DDS output
Returns	RC_OK On Success RC_ERROR On Failure
Description: This command enables or disables periodic or DDS output.	

Note: This function only enables the output. You have to call bcSetPeriodicDDSSelect() (see above) to select the output type.

bcSetDDSDivider	
Prototype	int bcSetDDSDivider (UCHAR div);
Packet	0x45
Input Parameter	UCHAR div = an enum value Note: The following are defined in the "Bc637pci.h" header file; #define DDS_DIVIDE_BY_1E00x0 #define DDS_DIVIDE_BY_1E10x1 #define DDS_DIVIDE_BY_1E20x2 #define DDS_DIVIDE_BY_1E30x3 #define DDS_DIVIDE_BY_1E40x4 #define DDS_DIVIDE_BY_1E50x5 #define DDS_DIVIDE_BY_1E60x6

bcSetDDSDivider	
	#define DDS_DIVIDE_BY_1E70x7 #define DDS_DIVIDE_BY_PREG 0xF
Returns	RC_OK On Success RC_ERROR On Failure

Description: This command sets the DDS divider for DDS output. For information on the detailed DDS description, please refer Chapter 1 for more information.

bcSetDDSDividerSource	
Prototype	int bcSetDDSDividerSource (UCHAR src);
Packet	0x46
Input Parameter	UCHAR src = 0, 1 or 2. Note: The following are defined in the "Bc637pci.h" header file; #define DDS_DIVIDED_SRC_DDS0x0 #define DDS_DIVIDER_SRC_MULT0x1 #define DDS _DIVIDER _SRC _100MHZ 0x2
Returns	RC_OK On Success RC_ERROR On Failure
Description: This command sets the DDS divider source for DDS output. For information on the detailed DDS description, please refer Chapter 1 for more information.	

bcSetDDSSyncMode	
Prototype	int bcSetDDSSyncMode (UCHAR mode);
Packet	0x47
Input Parameter	UCHAR src = 0 or 1 Note: The following are defined in the "Bc637pci.h" header file; #define DDS _SYNC _MODE _FRAC0x0 #define DDS _SYNC _MODE _CONT0x1
Returns	RC_OK On Success RC_ERROR On Failure
Description: This command sets the DDS synchronization mode for DDS output. For information on the detailed DDS description, please refer Chapter 1 for more information.	

bcSetDDSMultiplier	
Prototype	int bcSetDDSMultiplier (UCHAR mul);
Packet	0x48
Input Parameter	UCHAR mul = an enum value Note: The following are defined in the "Bc637pci.h" header file; #define DDS_MULTIPLY_BY_10x1 #define DDS_MULTIPLY_BY_20x2 #define DDS_MULTIPLY_BY_30x3 #define DDS_MULTIPLY_BY_40x4 #define DDS_MULTIPLY_BY_60x6 #define DDS_MULTIPLY_BY_80x8 #define DDS_MULTIPLY_BY_100xA #define DDS_MULTIPLY_BY_160x10
Returns	RC_OK On Success RC_ERROR On Failure
Description: This command sets the DDS multiplier for DDS output. For information on the detailed DDS description, please refer Chapter 1 for more information.	

bcSetDDSPeriodValue	
Prototype	int bcSetDDSPeriodValue (DWORD period);
Packet	0x49
Input Parameter	DWORD period = unsigned long value in the range of [0, 0xFFFFFF] for the period value.
Returns	RC_OK On Success RC_ERROR On Failure
Description: This command sets the DDS period value for DDS output. For information on the detailed DDS description, please refer Chapter 1 for more information.	

bcSetDDSTuning Word	
Prototype	int bcSetDDSTuningWord (DWORD tune);
Packet	0x4A
Input Parameter	DWORD tune = unsigned long value for the tuning word.
Returns	RC_OK On Success RC_ERROR On Failure
Description: This command sets the DDS tuning word for DDS output. For information on the detailed DDS description, please refer Chapter 1 for more information.	

bcSetDDSFrequency	
Prototype	int bcSetDDSFrequency (double freq);
Packet	N/A
Input Parameter	DWORD freq = double value to specify DDS frequency. The DDS frequency can have fractional value.
Returns	RC_OK On Success RC_ERROR On Failure
Description: This function sets the DDS output frequency. For information on the detailed DDS description, please refer Chapter 1 for more information.	

Note: This function automatically selects the periodic/DDS output to the DDS output and the synchronization mode to DDS_SYNC_MODE_FRAC.

bcSetTcInEx	
Prototype	int bcSetTcInEx (UCHAR format, UCHAR subtype, UCHAR mod);
Packet	0x15, 0x16
Input Parameter	Unsigned char format = time code format. Unsigned char subtype = time code sub type. Unsigned char mod = modulation type of time code. Note: The following are defined in the "Bc637pci.h" header file; format #define TCODE _IRIG _A 'A' #define TCODE _IRIG _B 'B' #define TCODE _IEEE 'I' #define TCODE_IRIG_E'E' #define TCODE_IRIG_e'e' #define TCODE_IRIG_G'G' #define TCODE_NASA36'N' #define TCODE_XR3'X' #define TCODE_2 137'2' subtype #define TCODE _IRIG _SUBTYPE _NONE 0 #define TCODE_IRIG_SUBTYPE_Y'Y' #define TCODE_IRIG_SUBTYPE_T'T'

bcSetTcInEx	
	mod #define TCODE _MOD _AM 'M' #define TCODE _MOD _DC 'D'
Returns	RC_OK On Success RC_ERROR On Failure
Description: Sets time code type, subtype and modulation type for operating mode 0 (time code mode).	

Note: This function extends the function bcSetTcIn() by allowing more time code type and a subtype with certain time code type. The bcSetTcIn() is still supported for backwards compatibility. The supported combinations of time code type and subtype are as follows:

format	subtype	Note
TCODE_IRIG_A	TCODE_IRIG_SUBTYPE_NONE	'A' - IRIG A no year
TCODE_IRIG_A	TCODE_IRIG_SUBTYPE_Y	'AY' - IRIG A with year
TCODE_IRIG_B	TCODE_IRIG_SUBTYPE_NONE	'B' - IRIG B no year
TCODE_IRIG_B	TCODE_IRIG_SUBTYPE_Y	'BY' - IRIG B with year
TCODE_IRIG_B	TCODE_IRIG_SUBTYPE_T	'BT' - IRIG B Legacy TrueTime
TCODE_IEEE	TCODE _IRIG _SUBTYPE _NONE	'I' - IRIG B IEEE 1344
TCODE_IRIG_E	TCODE_IRIG_SUBTYPE_NONE	'E' - IRIG E 1000Hz no year
TCODE_IRIG_E	TCODE_IRIG_SUBTYPE_Y	'EY' - IRIG E 1000Hz with year
TCODE_IRIG_e	TCODE_IRIG_SUBTYPE_NONE	'e' - IRIG E 100Hz no year
TCODE_IRIG_e	TCODE_IRIG_SUBTYPE_Y	'eY' - IRIG E 100Hz with year
TCODE_IRIG_G	TCODE_IRIG_SUBTYPE_NONE	'G' - IRIG G no year
TCODE_IRIG_G	TCODE_IRIG_SUBTYPE_Y	'GY' - IRIG G with year
TCODE_NASA	TCODE_IRIG_SUBTYPE_NONE	'N' - NASA 36
TCODE_XR3	TCODE _IRIG _SUBTYPE _NONE	'X' - XR3
TCODE_2 137	TCODE_IRIG_SUBTYPE_NONE	'2' - 2137

bcSetGenCodeEx	
Prototype	int bcSetGenCodeEx (UCHAR format, UCHAR subtype);
Packet	0x1B
Input Parameter	Unsigned char format = time code format. Note: The following are defined in the "Bc637pci.h" header file; format #define TCODE _IRIG _A 'A' #define TCODE _IRIG _B 'B' #define TCODE _ IEEE 'I' #define TCODE_IRIG_E'E' #define TCODE_IRIG_e'e' #define TCODE_IRIG_G'G' #define TCODE_NASA36 'N' #define TCODE_XR3'X' #define TCODE_2 137'2' subtype #define TCODE _IRIG _SUBTYPE _NONE 0 #define TCODE_IRIG_SUBTYPE_0'0' #define TCODE_IRIG_SUBTYPE_1'1' #define TCODE_IRIG_SUBTYPE_2'2' #define TCODE_IRIG_SUBTYPE_3'3' #define TCODE_IRIG_SUBTYPE_4'4' #define TCODE_IRIG_SUBTYPE_5'5' #define TCODE_IRIG_SUBTYPE_6'6' #define TCODE_IRIG_SUBTYPE_7'7'
Returns	RC_OK On Success RC_ERROR On Failure

Description: Sets the time code generator format and its subtype.

Note: This function extends the function bcSetGenCode() by allowing more time code type and a sub-type with certain time code type. The bcSetGenCode() is still supported for backwards compatibility. The supported combinations of time code type and subtype are as follows:

format	subtype	Note
TCODE_IRIG_A	TCODE_IRIG_SUBTYPE_0	'A0' - IRIG A BCD,CF,SBS
TCODE_IRIG_A	TCODE_IRIG_SUBTYPE_1	'A1' 1. - IRIG A BCD,CF
TCODE_IRIG_A	TCODE_IRIG_SUBTYPE_2	'A2' 1. - IRIG A BCD
TCODE_IRIG_A	TCODE_IRIG_SUBTYPE_3	'A3' 1. - IRIG A BCD,SBS
TCODE_IRIG_A	TCODE_IRIG_SUBTYPE_4	'A4' 1. - IRIG A BCD,YR,CF,SBS
TCODE_IRIG_A	TCODE_IRIG_SUBTYPE_5	'A5' 1. - IRIG A BCD,YR,CF
TCODE_IRIG_A	TCODE_IRIG_SUBTYPE_6	'A6' 1. - IRIG A BCD,YR
TCODE_IRIG_A	TCODE_IRIG_SUBTYPE_7	'A7' 1. - IRIG A BCD,YR,SBS
TCODE_IRIG_B	TCODE_IRIG_SUBTYPE_0	'B0' - IRIG B BCD,CF,SBS
TCODE_IRIG_B	TCODE_IRIG_SUBTYPE_1	'B1' 1. - IRIG B BCD,CF
TCODE_IRIG_B	TCODE_IRIG_SUBTYPE_2	'B2' 1. - IRIG B BCD
TCODE_IRIG_B	TCODE_IRIG_SUBTYPE_3	'B3' 1. - IRIG B BCD,SBS
TCODE_IRIG_B	TCODE_IRIG_SUBTYPE_4	'B4' 1. - IRIG B BCD,YR,CF,SBS
TCODE_IRIG_B	TCODE_IRIG_SUBTYPE_5	'B5' 1. - IRIG B BCD,YR,CF
TCODE_IRIG_B	TCODE_IRIG_SUBTYPE_6	'B6' 1. - IRIG B BCD,YR
TCODE_IRIG_B	TCODE_IRIG_SUBTYPE_7	'B7' 1. - IRIG B BCD,YR,SBS
TCODE_IRIG_B	TCODE_IRIG_SUBTYPE_T	'BT' - IRIG B BCD,CF,SBS - Legacy
TCODE_IEEE	TCODE _IRIG _SUBTYPE _NONE	'I' - IRIG B IEEE 1344
TCODE_IRIG_E	TCODE_IRIG_SUBTYPE_1	'E1' 1. - IRIG E 1000Hz BCD,CF
TCODE_IRIG_E	TCODE_IRIG_SUBTYPE_2	'E2' 1. - IRIG E 1000Hz BCD

format	subtype	Note
TCODE_IRIG_E	TCODE_IRIG_SUBTYPE_5	'E5' 1. - IRIG E 1000Hz BCD,YR,CF
TCODE_IRIG_E	TCODE_IRIG_SUBTYPE_6	'E6' 1. - IRIG E 1000Hz BCD,YR
TCODE_IRIG_e	TCODE_IRIG_SUBTYPE_1	'e1' 1. - IRIG E 100Hz BCD,CF
TCODE_IRIG_e	TCODE_IRIG_SUBTYPE_2	'e2' 1. - IRIG E 100Hz BCD
TCODE_IRIG_e	TCODE_IRIG_SUBTYPE_5	'e5' 1. - IRIG E 100Hz BCD,YR,CF
TCODE_IRIG_e	TCODE_IRIG_SUBTYPE_6	'e6' 1. - IRIG E 100Hz BCD,YR
TCODE_IRIG_G	TCODE_IRIG_SUBTYPE_5	'G5' - IRIG G BCD,YR,CF
TCODE_NASA	TCODE_IRIG_SUBTYPE_NONE	'N' - NASA 36
TCODE_XR3	TCODE_IRIG_SUBTYPE_NONE	'X' - XR3
TCODE_2 137	TCODE _IRIG _SUBTYPE _NONE	'2' - 2137

bcReadEvent2TimeEx	
Prototype	int bcReadEvent2Time (ULONG *maj, ULONG *min, USHORT *nano);
Packet	N/A
Input Parameter	maj = pointer to unsigned long to store major time (Unix format). min = pointer to unsigned long to store microseconds. nano = pointer to unsigned short to store 100 nano seconds count.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Latches and returns time captured caused by a external event2	

bcReadEvent3TimeEx	
Prototype	int bcReadEvent3Time (ULONG *maj, ULONG *min, USHORT *nano);
Packet	N/A
Input Parameter	maj = pointer to unsigned long to store major time (Unix format). min = pointer to unsigned long to store microseconds. nano = pointer to unsigned short to store 100 nano seconds count.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Latches and returns time captured caused by a external event3	

bcReqOtherDataEx	
Prototype	int bcReqOtherDataEx (OtherDataEx *pdata);
Packet	0x19
Input Parameter	pdata = pointer to OtherDataEx structure. The structure is defined in the "Bc637pci.h" header file.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Returns other data, which are extended now, from the board.	

bcReqEventsData	
Prototype	int bcReqEventsData (EventsData *pdata);
Packet	N/A
Input Parameter	pdata = pointer to EventsData structure. The structure is defined in the "Bc637pci.h" header file.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Returns event, event2 and event3 data. The information for each event includes enabled, sense and capture lock.	

bcSetEventsData	
Prototype	int bcSetEventsData (EventsData *pdata);
Packet	N/A
Input Parameter	pdata = pointer to EventsData structure. The structure is defined in the "Bc637pci.h" header file.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Sets event, event2 and event3 data. The information for each event includes enabled, sense and capture lock.	

4. Linux SDK

4.1. Introduction

4.1.1. General

The PCI/PCIe TFP Linux Developer's Kit is designed to provide a suite of tools useful in the development of applications which access features of the bc635PCI-V2, bc637PCI-V2, bc635PCIe, and bc637PCIe Time and Frequency Processor. This kit is designed to provide an interface between the PCI/PCIe TFPs and applications developed for Linux environments. In addition to the interface library, an example program is provided, complete with source code, in order to provide a better understanding of the kit features and benefits.

4.1.2. Features

The main features of the Developer's Kit include:

- An interface library with access to all features of the PCI/PCIe TFPs.
- Example programs, with source, utilizing the interface library.
- User's Guide providing the API interface library definition.

4.1.3. Overview

The Developer's Kit is designed to provide an interface to the bc635PCI-V2, bc637PCI-V2, bc635PCIe, and bc637PCIe Time and Frequency Processor in the Linux OS environment. The example program provides sample code to exercise the interface library and examples to convert 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, allowing the developer to clip any useful code.

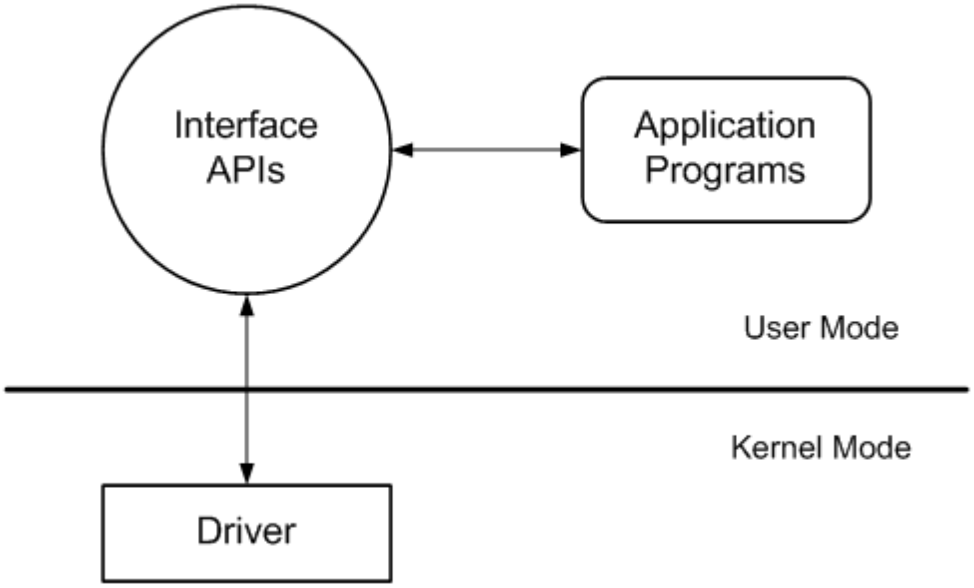


Figure 4-1

The interface API library provides an abstraction layer between the application programs and the device driver. This allows Symmetricom to advance the Time and Frequency Processor hardware features while protecting your investment in application development. Symmetricom will maintain interface API compatibilities between the current product release and future product releases.

4.2. Installation

4.2.1. Hardware installation

PCI/PCIe HW installation is facilitated due to two factors:

- Geographical addressing, eliminating the need for DIP switches and jumpers normally required to select a “base address” or interrupt level for plug-in modules.
- Auto configuration, allowing the host computer to read the device ID and other configuration information directly from the Configuration Registers.

Choose a vacant PCI/PCIe slot and insert the bc635PCI-V2, bc637PCI-V2, bc635PCIe, and bc637PCIe Time and Frequency Processor (TFP) and install the software. Be sure to consult the user documentation that came with your particular workstation for any specific PCI/PCIe card installation instructions.

4.2.2. Software installation

Because the bc635PCI-V2 and 637PCI-V2 driver is a KLM (kernel loadable module), the Linux source code including version.h must be installed on the system for the driver to install correctly.

Building modules for Linux kernel 2.6.x requires that you have a configured and built kernel tree on your system. This requirement is a change from previous versions of the kernel, where a current set of header files was sufficient. 2.6 modules are linked against object files found in the kernel source tree; the result is a more robust module loader, but also the requirements that those object files are available.

We have done our best to make our example modules safe and correct, but the possibility of bugs is always present. Faults in kernel code can bring about the demise of a user process or, occasionally, the entire system. They do not normally create more serious problems, such as disk corruption. Nonetheless, it is advisable to do your kernel experimentation on a system that does not contain data that you cannot afford to lose, and that does not perform essential services. Kernel developers typically keep a “sacrificial” system around for the purpose of testing new code.

NOTE: To verify the Linux source code is installed on your machine, check the /usr/src directory. Usually, there is a symbolic link 'linux' that points to the actual directory where Linux source code was installed. To check if the version.h is installed on the machine, check /usr/src/linux/include/linux to see if you have this file.

To install the Software driver and the sample program:

1. Create a directory /home/user/bc635pci (user is your login, for example)

```
/home/user> mkdir bc635pci
```

2. Make bc635pci your active directory

```
/home/user> cd bc635pci
```

3. If you are building on x86_32 platform, extract the file BCPCI-V<aaa>-x86_32-build<bbb>.tgz (replace the <aaa> and <bbb> with the actual numbers). If you are building on x86_64 platform, extract the file BCPCI-V<aaa>-x86_64-build<bbb>.tgz (replace the <aaa> and <bbb> with numbers).

```
/home/user/bc635pci> tar xvzf BCPCI-V<aaa>-x86_32-build<bbb>.tgz
```

4. Compile the Driver

```
/home/user/bc635pci> make
```

5. Become super user

```
/home/user/bc635pci> su
```

6. Install the Driver

/home/user/bc635pci# make install

7. Configure the Driver

Change the user and group IDs and give read/write permissions to the device file /dev/windrvr6 depending on how you want to allow users to access the device.

NOTE: The kernel module (windrvr6.o/windrvr6.ko) can be reloaded upon every boot. It will only be loaded after executing the following command upon boot:

```
/sbin/insmod windrvr6
```

It is recommended that you add this call to your rc.local file.

NOTE: By default, /dev/windrvr6 is created with permissions only for the root user. To enable other user access, change the write permissions of /dev/windrvr6

4.2.3. Linux kernel versions supported

The bc635/637PCI API uses Jungo's WinDriver to control and access the PCI card. The version of the Linux kernel supported is determined by the WinDriver Linux Device Driver. Symmetricom integrates the latest WinDriver release (WinDriver 10.0.2) that supports Linux kernel up to 2.6.29. Since there are many Linux distributions available, it is not possible to test Symmetricom drivers and APIs in all installations. Symmetricom installed and tested its software (bc63xPCIcfg) on Fedora Core 8 (x86_32) and Fedora Core 7 (x86_64).

4.2.4. Test Installation

The bc635/637PCI API is provided in both a static library (bcsdklib.a) and a shared library (libbcsdk.so). The 'sample' directory has a prebuilt test program bc63xPCIcfg (linked with bcsdklib.a) and bc63xPCIcfg-so (linked with libbcsdk.so). To use the bc63xPCIcfg-so program, the libbcsdk.so has to be put in a directory that is on the library path.

Rebuild the sample test program to verify that the software installation was successful.

1. Make sample your active directory.

```
/home/user/bc635pci> cd sample
```

2. Rebuild the sample code.

```
/home/user/bc635pci/sample> make clean
```

```
/home/user/bc635pci/sample> make
```

3. Run the sample program.

```
/usr/bin/bc635pci/sample> ./bc63xPCIcfg
```

NOTE: If a device open error is received, do the following:

1. Restart the computer.

2. Make drvr your active directory.

```
/home/user/bc635pci> cd drvr
```

3. Become super user.

```
/home/user/bc635pci/drvr> su
```

4. Run this command.

The installer should have created a folder called "LINUX.x.x.x.x" under the bc635pci path.

Substitute the x.x.x.x with the actual folder name, and run the command below:

/home/user/bc635pci/drvr# ./`wdreg` LINUX.x.x.x.x/windrvr6.o FALSE

NOTE: This command needs to be run every time the machine is powered up. You may use the “/sbin/insmod windrvr6” command upon every boot. 'wdreg' is a shell script supplied by Jungo as part of theWinDriver toolkit. Read the script to understand what it does.

5. Run the sample program.

Switch to normal user and run the sample program.

```
/home/user/bc635pci/sample> ./bc63xPCIcfg
```

4.2.5. Using the bc63xPCIcfg.exe program

The bc63xPCIcfg program may be used to test the PCI/PCIe TFP hardware. See the bc63xPCIcfg menu below:

```
--------------------------------------------------------------------------

                              Symmetricom - TT & M

                         bc635/637PCIe-V2 Configurator                Ver-
sion 3.0

--------------------------------------------------------------------------

 1. Read Current Time (Press enter to return to menu)
 2. Read Event Time   (Press enter to return to menu)
 3. Set Current Time
 4. Set Current Year
 5. Set Strobe Time
 6. Program Control Register
 7. Program Leap Event Seconds
 8. Select Time Format
 9. Select Operational Mode
 10. Select Decoding TimeCode Format
 11. Select TimeCode Output Format
 12. Select Clock Source
 13. Select Output Frequency
 14. Program Heartbeat counters
 15. Set Local Tffset                16. Set Generator Time Offsetime O
 17. Set Propogation Delay           18. Set Local Time Offset Flag
 19. Set Year Auto Increment Flag    20. Sync RTC to External Time Data
 21. Set GPS Time Format             22. Set GPS Mode Flag
 23. Software Reset                  24. Request Time Settings
 25. Request Time Code Settings      26. Request Clock Settings
 27. Request Control Settings        28. Request Model Information
 29. Request Firmware Version        30. Advanced Menu
```

```
31. Interrupts Menu                 32. GPS Menu - bc637PCI Only
33. PCI Revision ID                 34. Read from Dual Port RAM
35. Write to Dual Port RAM          36. Write ACK - Send DP Command
37. DDS Menu - V2 Hardware Only
38. Select TimeCode Input Format - V2 Hardware
39. Select TimeCode Output Format - V2 Hardware
40. Read from Register              41. Write to Register
42. Select events settings
43. Read Event2 Time  (Press enter to return to menu)
44. Read Event3 Time  (Press enter to return to menu)
0.  Exit the Pro-
gram.....................................................
```

Select Option:

The words “Select Option:“ will appear at the bottom of the menu driven program to allow the user to select the desired functionality.

Following are some examples demonstrating how to use the bc63xPCIcfg.

Select Operational Mode

There are several operational modes for the PCI-V2/PCIe card. The default Timing Mode for the bc635PCI-V2/PCIe is time code Decoding Mode, for the bc637PCI-V2/PCIe it is GPS Mode.

```
     Select Option: 9


0. Time Code Mode
1. Free Running Mode
2. 1PPS Mode
3. RTC Mode
4. GPS Mode
9. Cancel - back to main menu
   Select: 1


Verify the Mode selection:
```

Request Time Settings

```
     Select Option: 24


Time Settings:


Mode: Free Run
Time Format: Binary
Year: 2008
Local Offset: 0.0
 Propagation Delay: 0
Current Leap Seconds: 0
Scheduled Leap Event Time: 1069977600
Scheduled Leap Event Flag: Disable
GPS Time Format: UTC Format
IEEE Daylight Savings Flag: Enable


>>>>>>>>>>>>>>>>  Press Enter to continue  <<<<<<<<<<<<<<<<


     Select Option: 1
Binary Time: 10/10/2008  17:12:29.5482294   Status: 0
Binary Time: 10/10/2008  17:12:29.5630256   Status: 0
Binary Time: 10/10/2008  17:12:29.6196937   Status: 0
Binary Time: 10/10/2008  17:12:29.6318699   Status: 0
Binary Time: 10/10/2008  17:12:29.6508560   Status: 0
.
.
.
<enter> to stop reading the time.
```

Note that “Status: 0” indicates the board is in a locked state. For information on the status bits, refer to the Chapter 1.

Select Timecode Decoding Format

The PCI/PCIe TFP will decode IRIG A, IRIG B, IEEE 1344 or NASA 36. The default setting for the bc635PCI-V2 is IRIG B, amplitude modulated.

```
     Select Option: 10


Select Time Code Format:


1. IRIG A
2. IRIG B
3. IEEE 1344
4. NASA 36


    Select: 1


Select Time Code Modulation:


1. Modulated
2. DC Level Shift


    Select: 1


To verify the input time code selection:
```

Request Timecode Settings

```
     Select Option: 25


Time Code Settings:
```

```
Time Code: IRIG A
Code Modulation: AM
Time Code Out: IRIG B
Generator Time Offset: 0.0

>>>>>>>>>>>>>>>  Press Enter to continue  <<<<<<<<<<<<<<<<
```

Select Timecode Output Format

The PCI-V2/PCIe will output IRIG B or IEEE 1344, amplitude modulated or DC level shift.

```
    Select Option: 11

Select Time Code Output Format
  0. IRIG B
  1. IEEE 1344

Select: 1


Request Time Code Settings

    Select Option: 25

Time Code Settings:

Time Code: IRIG A
Code Modulation: AM
Time Code Out: IEEE 1344
Generator Time Offset: 0.0
```

```
>>>>>>>>>>>>>>>>  Press Enter to continue  <<<<<<<<<<<<<<<<
```

Select the Time Register Format

The card will output time in binary or grouped decimal formats.

```
    Select Option: 8

Select Time Format
  0. Decimal
  1. Binary

Select: 0


To verify the time format, check the time settings.

     Select Option: 24

Time Settings:

Mode: GPS
Time Format: Decimal
Year     : 2007
Local Offset: 0.0
 Propagation Delay: 0
Current Leap Seconds: 0
Scheduled Leap Event Time: 1069977600
Scheduled Leap Event Flag: Disable
GPS Time Format: UTC Format
IEEE Daylight Savings Flag: Disable
```

```
>>>>>>>>>>>>>>>>  Press Enter to continue  <<<<<<<<<<<<<<<<
```

Read Current Time

You may read the time in Binary or Decimal formats, selecting option 1 and <enter> to display the board's time. Note that the default time format is the binary format. It is recommended that you set (or verify) the time format prior to reading the time, using options 8 or 24. When in binary time format, the time output will indicate binary time.

```
     Select Option: 1
Binary Time: 01/01/2008  00:00:03.4308082   Status: 7
Binary Time: 01/01/2008  00:00:03.4525147   Status: 7
Binary Time: 01/01/2008  00:00:03.5417680   Status: 7
Binary Time: 01/01/2008  00:00:03.5615496   Status: 7
Binary Time: 01/01/2008  00:00:03.5810793   Status: 7
Binary Time: 01/01/2008  00:00:03.6010876   Status: 7
.
.
.
<enter> to stop reading the time.
```

Note that “Status: 7” indicates the board is in a flywheel state. For information on the status bits, refer to Chapter 1.

When in decimal time format, the time output will indicate decimal time.

```
     Select Option: 1
Decimal Time: 284 2008  17:11:26.5086369   Status: 7
Decimal Time: 284 2008  17:11:26.5684513   Status: 7
Decimal Time: 284 2008  17:11:26.6105457   Status: 7
Decimal Time: 284 2008  17:11:26.6484520   Status: 7
Decimal Time: 284 2008  17:11:26.6884591   Status: 7
Decimal Time: 284 2008  17:11:26.7297610   Status: 7
.
```

```
.
.

<enter> to exit
```

Set Current Time

You may choose to set the board time. This makes sense when the card does not have a time source and is counting time in a flywheel state, or when the card is in FreeRun or RTC modes. If the card is decoding a selected input source, the time written to the card will be overwritten. To set the board time, select option 3.

```
     Select Option: 3
Enter Time:
YYYY MM DD HH MM SS
2008 10 10 10 10 10
```

To verify the time:

```
     Select Option: 1
Binary Time: 10/10/2008  17:10:11.9932217   Status: 7
Binary Time: 10/10/2008  17:10:12.0358680   Status: 7
Binary Time: 10/10/2008  17:10:12.1087182   Status: 7
Binary Time: 10/10/2008  17:10:12.1288805   Status: 7
Binary Time: 10/10/2008  17:10:12.1456343   Status: 7
Binary Time: 10/10/2008  17:10:12.1671005   Status: 7
.
.
.

<enter> to exit
```

Set Current Year

```
     Select Option: 4
```

```
Enter Year (1990 - 2036): 2008
To verify the year:
     Select Option: 1
Binary Time: 10/11/2008  17:10:49.9202763   Status: 7
Binary Time: 10/11/2008  17:10:49.9372524   Status: 7
Binary Time: 10/11/2008  17:10:49.9871942   Status: 7
Binary Time: 10/11/2008  17:10:50.0070659   Status: 7
Binary Time: 10/11/2008  17:10:50.0270642   Status: 7
Binary Time: 10/11/2008  17:10:50.0483900   Status: 7
.
.
.
<enter> to exit
```

You may also use “Request Time Settings” to verify the year:

```
     Select Option: 24

Time Settings:

Mode: GPS
Time Format: Binary
Year: 2005
Local Offset: 0.0
Propagation Delay: 0
Current Leap Seconds: 0
Scheduled Leap Event Time: 1069977600
Scheduled Leap Event Flag: Disable
GPS Time Format: UTC Format
IEEE Daylight Savings Flag: Enable

>>>>>>>>>>>>>>>  Press Enter to continue  <<<<<<<<<<<<<<<<
```

Request Model Information

```
     Select Option: 28


Model Information


Model: BC637PCI
Serial: 272642420 (0x10403174)
Hardware Fab: 0
Assembly: 0


>>>>>>>>>>>>>>>>  Press Enter to continue  <<<<<<<<<<<<<<<<
```

DDS Frequency and New Time Codes

The current hardware supports DDS output and many new decoding and generating time codes. The DDS feature allows a much wider range of output frequency. These features are demonstrated in the new menu entries added to the bc63xPCIcfg program.

```
Select Option: 37


DDS Menu


1. Set DDS Frequency
2. Select Periodic or DDS Output
3. Enable/Disable Periodic/DDS Output
4. Set DDS Divider
5. Set DDS Divider Source
6. Set DDS Sync Mode
7. Set DDS Multiplier
8. Set DDS Period Value
9. Set DDS Tuning Word
0. Back to main menu
```

```
    Select:
```

You can set the DDS frequency.

```
    Select: 1

Enter DDS Frequency (in HZ): 12345.67

Done setting DDS frequency
>>>>>>>>>>>>>>>>  Press Enter to continue  <<<<<<<<<<<<<<<<
```

The DDS menu options 2 to 9 are for low level DDS controls, which gives the maximum flexibility. For information on the detailed DDS description, refer to Chapter 1.

The new hardware has expanded greatly in terms of decoding and the generation of time codes. To access the supported new decoding time codes, use option 38 of the bc63xPCIcfg. For example, you can specify to decode IRIG E 1000 Hz no year as follows.

```
Select Option: 38

Select Time Code Format:

1. IRIG A000-A003, A130-A133 (no year)
2. IRIG A004-A007, A134-A137 (has year)
3. IRIG B000-B003, B120-B123 (no year)
4. IRIG B004-B007, B124-B127 (has year)
5. IRIG B TrueTime Legacy
6. IRIG B IEEE 1344
7. IRIG E001, E002, E121, E122 (no year)
8. IRIG E005, E006, E125, E126 (has year)
9. IRIG E001, E002, E111, E112 (no year)
10. IRIG E005, E006, E115, E116 (has year)
11. IRIG G001, G002, G141, G142 (no year)
12. IRIG G005, G006, G145, G146 (has year)
13. NASA 36
14. XR3
15. 2137
```

```
  0. Back to main menu


    Select: 7

Select Time Code Modulation:

1. Modulated
2. DC Level Shift

    Select: 1
```

To use the supported new generating time code, use option 39 of the bc63xPCIcfg. For example, you can specify to generate IRIG G as follows.

```
Select Option: 39

Select Time Code Output Format

 1. IRIG A000, A130 (BCD,CF,SBS)        2. IRIG A001, A131 (BCD,CF)
  3. IRIG A002, A132 (BCD)              4. IRIG A003, A133 (BCD,SBS)
  5. IRIG A004, A134 (BCD,YR,CF,SBS)    6. IRIG A005, A135 (BCD,YR,CF)
  7. IRIG A006, A136 (BCD,YR)           8. IRIG A007, A137 (BCD,YR,SBS)
 9. IRIG B000, B120 (BCD,CF,SBS)        10. IRIG B001, B121 (BCD,CF)
 11. IRIG B002, B122 (BCD)              12. IRIG B003, B123 (BCD,SBS)
 13. IRIG B004, B124 (BCD,YR,CF,SBS)    14. IRIG B005, B125 (BCD,YR,CF)
 15. IRIG B006, B126 (BCD,YR)           16. IRIG B007, B127 (BCD,YR,SBS)
 17. IRIG B TrueTime (BCD,CF,SBS)       18. IRIG B IEEE 1344
 19. IRIG E001, E121 (BCD,CF)           20. IRIG E002, E122 (BCD)
 21. IRIG E005, E125 (BCD,YR,CF)        22. IRIG E006, E126 (BCD,YR)
 23. IRIG E001, E111 (BCD,CF)           24. IRIG E002, E112 (BCD)
 25. IRIG E005, E115 (BCD,YR,CF)        26. IRIG E006, E116 (BCD,YR)
 27. IRIG G005, G145 (BCD,YR,CF)        28. NASA 36
 29. XR3                                30. 2137
 0. Back to main menu

     Select: 27
```

Note that option 10 and option 11 of the bc63xPCIcfg main menu, still works, but offers a small subset of what is available under option 38 and option 39. These options are still available for users who are familiar with the previous version of the program.

Compatibility with Old bc635PCI or bc637PCI Card

The bc63xPCIcfg.exe program works with the old bc635PCI or bc637PCI card. These cards are known as U and V1 hardware. When invoking a feature that is available only in V2 hardware, the program displays an error message.

```
Error: Getting data!!!!!!!!!!!!!!!!!!!!
```

Uninstall Instructions

NOTE: You must be logged in as root in order to uninstall.

1. Uninstall the driver service.

Do a /sbin/lsmod to check if the WinDriver module is in use by any application or by other modules. Make sure no programs are using WinDriver. If any application or module is using WinDriver, close all applications and do a /sbin/rmmod to remove any module using WinDriver.

Run the command "`/sbin/rmmod windrvr6`"

`rm -rf /dev/windrvr6` (Remove the old device node in the /dev directory.)

2. Delete the bc635pci installation directory.

Use the command rm -rf /usr/bin/bc635pci

4.3. Library Definitions

4.3.1. General

The interface library provides functions for each of the programming packets supported by the bc635PCI-V2, bc637PCI-V2, bc635PCIe, and bc637PCIe 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 Chapter 1.

For developers who have developed code based on the library using the V1 hardware, you will notice that a set of new API functions has been added. However, the existing API functions have been kept intact. This is to ensure that your program is source compatible with the new library. All you need to do is a recompile of your program to link with the new library.

4.3.2. Functions

bcStartPci	
Prototype	BC_PCI_HANDLE bcStartPci ();
Packet	N/A
Input Parameter	None
Returns	BC_PCI_HANDLE hBC_PCI On Success NULL On Failure
Description: This function opens the underlying hardware layer. The return handle is used with the rest of the functions. (See section 4.4 for a programming example.)	

bcStopPci	
Prototype	void bcStopPci (BC_PCI_HANDLE hBC_PCI);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function
Returns	None
Description: This function closes the underlying hardware layer and releases any used resources. (See section 4.4 for a programming example.)	

bcReadReg	
Prototype	BOOL bcReadReg (BC_PCI_HANDLE hBC_PCI, DWORD Offset, PDWORD Data);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function Offset = See defined Registers offsets in the "bcuser.h" header file. Data = pointer to unsigned long to return value requested.
Returns	TRUE On Success FALSE On Failure

bcReadReg
Description: Returns the contents of the requested register.

bcWriteReg	
Prototype	BOOL bcWriteReg (BC_PCI_HANDLE hBC_PCI, DWORD Offset, DWORD Data);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function Offset = See defined Registers offsets in the "bcuser.h" header file. Data = unsigned long value to be set.
Returns	TRUE On Success FALSE On Failure
Description: Sets the contents of the requested register.	

bcReadDPReg	
Prototype	BOOL bcReadDPReg (BC_PCI_HANDLE hBC_PCI, DWORD Offset, PBYTE Data);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function Offset = Byte offset in the Dual Port RAM area. This area is used to send a command to the timing engine and read a result from the timing engine. For information on command format and results returned, refer to the PCI/PCIe TFP Users Guide. Data = pointer to the return byte value requested.
Returns	TRUE On Success FALSE On Failure
Description: Returns the content of the requested byte in the Dual Port RAM area.	

bcWriteDPReg	
Prototype	BOOL bcWriteDPReg (BC_PCI_HANDLE hBC_PCI, DWORD Offset, BYTE Data);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function Offset = Byte offset in the Dual Port RAM area. This area is used to send a command to the timing engine and read a result from the timing engine. For information on command format and results returned, refer to the PCI/PCIe TFP Users Guide.

bcWriteDPReg	
	Data = byte value to be set.
Returns	TRUE On Success FALSE On Failure
Description: Sets the content of the requested byte in the Dual Port RAM area.	

bcReadBinTime	
Prototype	BOOL bcReadBinTime (BC_PCI_HANDLE hBC_PCI, PDWORD major, PDWORD min, PBYTE stat);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function major = unsigned long pointer to store major time (Unix format). min = unsigned long pointer to store microseconds. stat = unsigned char to store status bits.
Returns	TRUE On Success FALSE On Failure
Description: Latches and returns time captured from the time registers. (See section 4.4 for a programming example.)	

bcReadBinTimeEx	
Prototype	BOOL bcReadBinTimeEx (BC_PCI_HANDLE hBC_PCI, PDWORD major, PDWORD min, PWORD nano, PBYTE stat);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function major = unsigned long pointer to store major time (Unix format). min = unsigned long pointer to store microseconds. nano = unsigned short pointer to store 100 nano seconds count. stat = unsigned char to store status bits.
Returns	TRUE On Success FALSE On Failure
Description: Latches and returns time captured from the time registers. (See section 4.4 for a programming example.)	

bcReadDecTime	
Prototype	BOOL bcReadDecTime (BC_PCI_HANDLE hBC_PCI, struct tm *ptm, PDWORD ulpMin, PBYTE pstat);

bcReadDecTime	
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function ptm = pointer to tm stuct to store major time (calendar format). ulpMin = pointer to unsigned long to store microseconds. pstat = pointer to unsigned char to store status bits.
Returns	TRUE On Success FALSE On Failure
Description: Latches and returns time captured from the time registers. (See section 4.4 for a programming example.)	

bcReadDecTimeEx	
Prototype	BOOL bcReadDecTime (BC_PCI_HANDLE hBC_PCI, struct tm *ptm, PDWORD ulpMin, PWORD nano, PBYTE pstat);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function ptm = pointer to tm stuct to store major time (calendar format). ulpMin = pointer to unsigned long to store microseconds. nano = unsigned short pointer to store 100 nano seconds count. pstat = pointer to unsigned char to store status bits.
Returns	TRUE On Success FALSE On Failure
Description: Latches and returns time captured from the time registers. (See section 4.4 for a programming example.)	

bcSetBinTime	
Prototype	BOOL bcSetBinTime (BC_PCI_HANDLE hBC_PCI, DWORD newtime);
Packet	0x12
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function newtime = unsigned long time value to set (Unix format).
Returns	TRUE On Success FALSE On Failure
Description: Set the major time buffer.	

bcSetDecTime	
Prototype	BOOL bcSetDecTime (BC_PCI_HANDLE hBC_PCI, struct tm);
Packet	0x12
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function tm = tm struct containing new time values to set.
Returns	TRUE On Success FALSE On Failure
Description: Set the major time buffer.	

bcReqYear	
Prototype	BOOL bcReqYear (BC_PCI_HANDLE hBC_PCI, PDWORD year);
Packet	0x19
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function year = pointer to unsigned long value of new year (1990-2036).
Returns	TRUE On Success FALSE On Failure
Description: Request the current year value.	

bcSetYear	
Prototype	BOOL bcSetYear (BC_PCI_HANDLE hBC_PCI, DWORD year);
Packet	0x13
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function year = value of new year (1990-2036).
Returns	TRUE On Success FALSE On Failure
Description: Set the current year value.	

bcReadEventTime	
Prototype	BOOL bcReadEventTime (BC_PCI_HANDLE hBC_PCI, PDWORD maj, PDWORD min, PBYTE stat);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function maj = pointer to unsigned long to store major time (Unix format). min = pointer to unsigned long to store microseconds. stat = unsigned char to store status bits.
Returns	TRUE On Success FALSE On Failure

bcReadEventTime
Description: Latches and returns time captured caused by an external event.

bcReadEventTimeEx	
Prototype	BOOL bcReadEventTimeEx (BC_PCI_HANDLE hBC_PCI, PDWORD maj, PDWORD min, PWORD nano, PBYTE stat);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function maj = pointer to unsigned long to store major time (Unix format). min = pointer to unsigned long to store microseconds. nano = pointer to unsigned short to store 100 nano seconds count. stat = unsigned char to store status bits.
Returns	TRUE On Success FALSE On Failure
Description: Latches and returns time captured caused by an external event.	

bcSetStrobeTime	
Prototype	BOOL bcSetStrobeTime (BC_PCI_HANDLE hBC_PCI, DWORD dMaj, DWORD dMin);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function dMaj = unsigned long value for strobe major time. dMin = unsigned long value for strobe minor time.
Returns	TRUE On Success FALSE On Failure
Description: Set the strobe time.	

bcReqTimeFormat	
Prototype	BOOL bcReqTimeFormat (BC_PCI_HANDLE hBC_PCI, PBYTE timeformat);
Packet	0x19
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function timeformat = pointer to unsigned char value for time format. The allowed values are defined in the 'bcuser.h' file: enum { FORMAT_DECIMAL = 0x00 }; enum { FORMAT_BINARY = 0x01 };
Returns	TRUE On Success

bcReqTimeFormat	
	FALSE On Failure
Description: Request current time format.	

bcSetTimeFormat	
Prototype	BOOL bcSetTimeFormat (BC_PCI_HANDLE hBC_PCI, BYTE tmfmt);
Packet	0x11
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function tmfmt = unsigned char value for time format. The allowed values are defined in the 'bcuser.h' file: enum { FORMAT_DECIMAL = 0x00 }; enum { FORMAT_BINARY = 0x01 };
Returns	TRUE On Success FALSE On Failure
Description: Set time format	

bcSetMode	
Prototype	void bcSetMode (BC_PCI_HANDLE hBC_PCI, BYTE mode);
Packet	0x10
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function mode = unsigned char value for new operating mode. The allowed values are defined in the 'bcuser.h' file: enum { MODE_IRIG = 0x00 }; enum { MODE_FREE = 0x01 }; enum { MODE_1PPS = 0x02 }; enum { MODE_RTC = 0x03 }; enum { MODE_GPS = 0x06 };
Returns	None
Description: Sets the operating mode of the board. (See section 4.4 for a programming example.)	

bcSetLocOff	
Prototype	BOOL bcSetLocOff (BC_PCI_HANDLE hBC_PCI, USHORT offset, BYTE half);
Packet	0x1D

bcSetLocOff	
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function Offset = hours from input time source. (-16 - +16). half = half hour increment 0: No Half Hour increment 1: Add Half Hour increment
Returns	TRUE On Success FALSE On Failure
Description: Programs the board to operate at an offset from UTC.	

bcSetGenOff	
Prototype	BOOL bcSetGenOff (BC_PCI_HANDLE hBC_PCI, USHORT offset, BYTE half);
Packet	0x1C
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function offset = hours from input time source. (-16 - +16). half = half hour increment 0: No Half Hour increment 1: Add Half Hour increment
Returns	TRUE On Success FALSE On Failure
Description: Programs the board time code generator to operate at an offset from UTC.	

bcSetPropDelay	
Prototype	BOOL bcSetPropDelay (BC_PCI_HANDLE hBC_PCI, long value);
Packet	0x17
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function value = long data for propagation delay.
Returns	TRUE On Success FALSE On Failure
Description: Sets the propagation delay offset.	

bcSetHbt	
Prototype	BOOL bcSetHbt (BC_PCI_HANDLE hBC_PCI, BYTE mode, USHORT n1, USHORT n2);
Packet	0x14
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function mode = unsigned char value for the heartbeat mode. n1 = unsigned short value for heartbeat counter 1. n2 = unsigned short value for heartbeat counter 2. The allowed values are defined in the 'bcuser.h' file: enum { PERIODIC_SYNC = 0x01 }; enum { PERIODIC_NOSYNC = 0x00 };
Returns	TRUE On Success FALSE On Failure
Description: Sets the heartbeat counters and mode.	

bcSetTcIn	
Prototype	BOOL bcSetTcIn (BC_PCI_HANDLE hBC_PCI, BYTE TcIn);
Packet	0x15
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function TcIn = unsigned char value for time code input. The allowed values are defined in the 'bcuser.h' file: enum { TCODE_IRIG_A = 0x41 }; enum { TCODE_IRIG_B = 0x42 }; enum { TCODE_IEEE = 0x49 }; enum { TCODE_NASA = 0x4E };
Returns	TRUE On Success FALSE On Failure
Description: Sets the input time code format.	

bcSetTcInMod	
Prototype	BOOL bcSetTcInMod (BC_PCI_HANDLE hBC_PCI, BYTE TcInMod);
Packet	0x16
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function TcInMod = unsigned char value for time code input modulation. The allowed values are defined in the 'bcuser.h' file: enum { TCODE_MOD_AM = 0x4D };

bcSetTcInMod	
	enum { TCODE_MOD_DC = 0x44 };
Returns	TRUE On Success FALSE On Failure
Description: Sets the input time code modulation.	

bcSetGenCode	
Prototype	BOOL bcSetGenCode (BC_PCI_HANDLE hBC_PCI, BYTE GenTc);
Packet	0x1B
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function GenTc = unsigned char value for the time code output. The allowed values are defined in the 'bcuser.h' file: enum { TCODE_IRIG_B = 0x42 }; enum { TCODE_IEEE = 0x49 };
Returns	TRUE On Success FALSE On Failure
Description: Sets the output time code format.	

cSetLeapEvent	
Prototype	BOOL bcSetLeapEvent (BC_PCI_HANDLE hBC_PCI, char flag, DWORD leapevt);
Packet	0x1E
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function flag = char value for the leap event flag. leapevt = unsigned long value for the leap event time.
Returns	TRUE On Success FALSE On Failure
Description: Sets the leap event time.	

bcSetClkSrc	
Prototype	BOOL bcSetClkSrc (BC_PCI_HANDLE hBC_PCI, BYTE clk);
Packet	0x20
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function clk = unsigned char value for the clock source. The allowed values are defined in the 'bcuser.h' file: enum { CLK_INT = 0x49 };

bcSetClkSrc	
	enum { CLK_EXT = 0x45 };
Returns	TRUE On Success FALSE On Failure
Description: Sets the clock source, Internal/External.	

bcSetDac	
Prototype	BOOL bcSetDac (BC_PCI_HANDLE hBC_PCI, USHORT dac);
Packet	0x24
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function dac = unsigned short value for the DAC.
Returns	TRUE On Success FALSE On Failure
Description: Sets the DAC value	

bcSetGain	
Prototype	BOOL bcSetGain (BC_PCI_HANDLE hBC_PCI, short gain);
Packet	0x25
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function gain = short value for the Gain.
Returns	TRUE On Success FALSE On Failure
Description: Sets the GAIN.	

bcSetJam	
Prototype	BOOL bcSetJam (BC_PCI_HANDLE hBC_PCI, BYTE jam);
Packet	0x21
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function jam = unsigned char value for enabling/disabling jam-sync. The allowed values are defined in the 'bcuser.h' file: enum { JAM_SYNC_ENA = 0x01 }; enum { JAM_SYNC_DIS = 0x00 };
Returns	TRUE On Success FALSE On Failure
Description: Sets the Jam-Sync.	

bcSetGpsTmFmt	
Prototype	BOOL bcSetGpsTmFmt (BC_PCI_HANDLE hBC_PCI, BYTE gpsfmt);
Packet	0x33
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function gpsfmt = unsigned char value for gps time format. The allowed values are defined in the 'bcuser.h' file: enum { GPS_TIME_FMT = 0x01 }; enum { UTC_TIME_FMT = 0x00 };
Returns	TRUE On Success FALSE On Failure
Description: Sets the GPS time format.	

bcSetGpsOperMode	
Prototype	BOOL bcSetGpsOperMode (BC_PCI_HANDLE hBC_PCI, BYTE gpsmode);
Packet	0x34
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function gpsmode = unsigned char value for gps mode. The allowed values are defined in the 'bcuser.h' file: enum { GPS_STATIC = 0x01 }; enum { GPS_NONE_STATIC = 0x00 };
Returns	TRUE On Success FALSE On Failure
Description: Sets the GPS operating mode.	

bcSetLocalOffsetFlag	
Prototype	BOOL bcSetLocalOffsetFlag (BC_PCI_HANDLE hBC_PCI, BYTE flagoff);
Packet	0x40
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function flagoff = unsigned char value for enabling/disabling local offset time. The allowed values are defined in the 'bcuser.h' file: enum { LOCAL_OFF_ENABLE = 0x01 }; enum { LOCAL_OFF_DISABLE = 0x00 };
Returns	TRUE On Success FALSE On Failure
Description: Sets the local offset flag.	

bcSetYearAutoIncFlag	
Prototype	BOOL bcSetYearAutoIncFlag (BC_PCI_HANDLE hBC_PCI, BYTE yrinc);
Packet	0x42
	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function yrinc = unsigned char value for enabling/disabling year auto-increment flag. The allowed values are defined in the 'bcuser.h' file: enum { YEAR_AUTO_ENA = 0x01 }; enum { YEAR_AUTO_DIS = 0x00 };
Returns	TRUE On Success FALSE On Failure
Description: Sets the year auto increment flag. Note: this function is no longer applicable. The year is always auto incremented in the V2 hardware. Please refer to the Chapter 1 for details.	

bcAdjustClock	
Prototype	BOOL bcAdjustClock (BC_PCI_HANDLE hBC_PCI, long cval);
Packet	0x29
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function cval = long value for adjusting the clock
Returns	TRUE On Success FALSE On Failure
Description: Advance/Retard clock value.	

bcCommand	
Prototype	void bcCommand (BC_PCI_HANDLE hBC_PCI, BYTE cmd);
Packet	0x1A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function cmd = unsigned char value for software reset. The allowed value is defined in the 'bcuser.h' file: enum { CMD_WARMSTART = 0x01 };
Returns	None
Description: Software reset.	

bcForceJam	
Prototype	BOOL bcForceJam (BC_PCI_HANDLE hBC_PCI);
Packet	0x22

bcForceJam	
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function
Returns	TRUE On Success FALSE On Failure
Description: Forces a Jam-Sync.	

bcSyncRtc	
Prototype	BOOL bcSyncRtc (BC_PCI_HANDLE hBC_PCI);
Packet	0x27
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function
Returns	TRUE On Success FALSE On Failure
Description: Sync RTC clock with current time.	

bcDisRtcBatt	
Prototype	BOOL bcDisRtcBatt (BC_PCI_HANDLE hBC_PCI);
Packet	0x28
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function
Returns	TRUE On Success FALSE On Failure
Description: Disable battery.	

bcReqSerialNum	
Prototype	BOOL bcReqSerialNum (BC_PCI_HANDLE hBC_PCI, PDWORD serial);
Packet	0xFE
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function serial = pointer to unsigned long value for serial number of the board.
Returns	TRUE On Success FALSE On Failure
Description: Request serial number of the board.	

bcReqHardwarFab	
Prototype	BOOL bcReqHardwarFab (BC_PCI_HANDLE hBC_PCI, PWORD fab);
Packet	0xF5
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function fab = pointer to unsigned short value for the hardware fab.
Returns	TRUE On Success

bcReqHardwarFab	
	FALSE On Failure
Description: Request hardware fab of the board.	

bcReqAssembly	
Prototype	BOOL bcReqAssembly (BC_PCI_HANDLE hBC_PCI, PWORD num);
Packet	0xF4
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function num = pointer to unsigned short value for the assembly number.
Returns	TRUE On Success FALSE On Failure
Description: Request assembly number of the board.	

bcReqOscData	
Prototype	BOOL bcReqOscData (BC_PCI_HANDLE hBC_PCI, OscData *pdata);
Packet	0x19
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function pdata = pointer to OscData structure. The structure is defined in the "bcuser.h" header file.
Returns	TRUE On Success FALSE On Failure
Description: Request Oscillator data of the board.	

bcReqTimeCodeData	
Prototype	BOOL bcReqTimeCodeData (BC_PCI_HANDLE hBC_PCI, TimeCodeData *pdata);
Packet	0x19
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function pdata = pointer to TimeCodeData structure. The structure is defined in the "bcuser.h" header file.
Returns	TRUE On Success FALSE On Failure
Description: Request time code data of the board.	

bcReqTimeData	
Prototype	BOOL bcReqTimeData (BC_PCI_HANDLE hBC_PCI, TimeData *pdata);

bcReqTimeData	
Packet	0x19
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function pdata = pointer to TimeData structure. The structure is defined in the "bcuser.h" header file.
Returns	TRUE On Success FALSE On Failure
Description: Request time data of the board.	

bcReqOtherData	
Prototype	BOOL bcReqOtherData (BC_PCI_HANDLE hBC_PCI, OtherData *pdata);
Packet	0x19
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function pdata = pointer to OtherData structure. The structure is defined in the "bcuser.h" header file.
Returns	TRUE On Success FALSE On Failure
Description: Request other data of the board.	

bcReqOtherDataEx	
Prototype	BOOL bcReqOtherDataEx (BC_PCI_HANDLE hBC_PCI, OtherDataEx *pdata);
Packet	0x19
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function pdata = pointer to OtherDataEx structure. The structure is defined in the "bcuser.h" header file.
Returns	TRUE On Success FALSE On Failure
Description: Request other data of the board.	

bcReqVerData	
Prototype	BOOL bcReqVerData (BC_PCI_HANDLE hBC_PCI, VerData *pdata);
Packet	0x19
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function pdata = pointer to VerData structure.

bcReqVerData	
	The structure is defined in the "bcuser.h" header file.
Returns	TRUE On Success FALSE On Failure
Description: Request version data of the board.	

bcReqModel	
Prototype	BOOL bcReqModel (BC_PCI_HANDLE hBC_PCI, ModelData *pdata);
Packet	0x19
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function pdata = pointer to ModelData structure. The structure is defined in the "bcuser.h" header file.
Returns	TRUE On Success FALSE On Failure
Description: Request model data of the board.	

bcGPSReq	
Prototype	BOOL bcGPSReq (BC_PCI_HANDLE hBC_PCI, GpsPkt *pktout);
Packet	0x31
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function pktout = structure commands information detailing the packet to retrieve and the buffer
Returns	TRUE On Success FALSE On Failure
Description: Retrieve a data packet from the GPS receiver. Refer to chapter for more details regarding this command. (See packet 0x31 definition)	

bcGPSSnd	
Prototype	BOOL bcGPSSnd (BC_PCI_HANDLE hBC_PCI, GpsPkt *pktin);
Packet	0x30

bcGPSSnd	
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from "bcStartPci" pktin = structure commands information detailing the packet to send and the buffer.
Returns	TRUE On Success FALSE On Failure
Description: Send a data packet to the GPS receiver. Refer to Chapter 1 for more details regarding this command. (See packet 0x30 definition)	

bcGPSMan	
Prototype	BOOL bcGPSMan (BC_PCI_HANDLE hBC_PCI, GpsPkt *pktin, GpsPkt *pktout);
Packet	0x32
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function pktin = structure commands information detailing the packet to send and the buffer. pktout = structure commands information detailing the packet to retrieve and the buffer
Returns	TRUE On Success FALSE On Failure
Description: Manually send and retrieve data packets from the GPS receiver. Refer to Chapter 1 for more details regarding this command. (See Packet 0x32 definition)	

bcStartInt	
bcStartInt	
Prototype	BOOL bcStartInt (BC_PCI_HANDLE hBC_PC, BC_PCI_INT_HANDLER pCallback);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function PCallback function to receive interrupts
Returns	TRUE On Success FALSE On Failure
Description: Start the interrupt thread. This thread will execute bcShowInt() function every time an interrupt is detected. (See section 4.4 for a programming example.)	

bcStartIntEx	
Prototype	BOOL bcStartIntEx (BC_PCI_HANDLE hBC_PC, BC_PCI_INT_HANDLER pCallback, DWORD intMask);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function pCallback = function to receive interrupts intMask = the interrupt mask value
Returns	TRUE On Success FALSE On Failure
Description: Start the interrupt thread. This thread will execute bcShowInt() function every time an interrupt is detected. The interrupt mask defines the source of interrupts. (See section 4.4 for a programming example.)	

bcStopInt	
Prototype	void bcStopInt (BC_PCI_HANDLE hBC_PCI);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function
Returns	None
Description: Stops the interrupt thread. (See section 4.4 for a programming example.)	

bcSetInt	
Prototype	BOOL bcSetInt (BC_PCI_HANDLE hBC_PCI, BYTE IntVal);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function IntVal = unsigned char value for selecting the interrupt source. The allowed values are defined in the 'bcuser.h' file: enum { INTERRUPT_EVENT = 0x01 }; enum { INTERRUPT_PERIODIC = 0x02 }; enum { INTERRUPT_STROBE = 0x04 }; enum { INTERRUPT_1PPS = 0x08 }; enum { INTERRUPT_GPS = 0x10 }; enum { INTERRUPT_EVENT2 = 0x20 }; enum { INTERRUPT_EVENT3 = 0x40 };

bcSetInt	
Returns	TRUE On Success FALSE On Failure
Description: Enable one-interrupt sources. (See section 4.4 for a programming example.)	

bcReqInt	
Prototype	BOOL bcReqInt (BC_PCI_HANDLE hBC_PCI, PBYTE Ints);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function Ints = pointer on unsigned char value for current used interrupt. The allowed values are defined in the 'bcuser.h' file: enum { INTERRUPT_EVENT = 0x01 }; enum { INTERRUPT_PERIODIC = 0x02 }; enum { INTERRUPT_STROBE = 0x04 }; enum { INTERRUPT_1PPS = 0x08 }; enum { INTERRUPT_GPS = 0x10 }; enum { INTERRUPT_EVENT2 = 0x20 }; enum { INTERRUPT_EVENT3 = 0x40 };
Returns	TRUE On Success FALSE On Failure
Description: Query the current enabled interrupt.	

bcShowInt	
Prototype	void bcShowInt (BC_PCI_HANDLE hBC_PCI);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function
Returns	None
Description: This function is used as an interrupt service routine. The user can add any code in this function to perform tasks once an interrupt is detected.	

bcReqRevisionID	
Prototype	BOOL bcReqRevisionID (BC_PCI_HANDLE hBC_PCI, PWORD id);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function

bcReqRevisionID	
	id = pointer to unsigned short value for the hardware revision id.
Returns	TRUE On Success FALSE On Failure
Description: Request hardware revision ID. The hardware revision id is the Revision ID field in the PCI configuration register space (offset 08h). The current hardware revision ID is in the range of [0x20, 0x2F]. The original hardware (V1 hardware) revision ID is less than 0x20. You can use the revision ID to run code specific to each hardware version of the PCI / PCIe card.	

bcReqTimeCodeDataEx	
Prototype	BOOL bcReqTimeCodeDataEx (BC_PCI_HANDLE hBC_PCI, TimeCodeDataEx *pdata);
Packet	0x19
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function pdata = pointer to TimeCodeDataEx structure. The structure is defined in the "bcuser.h" header file. The inputFormat and outputFormat are the time code type. Both are defined as enums 'TCODE_<aaa>'. The inputSubType and outputSubType are the IRIG time code sub type. They are defined as enums 'TCODE_IRIG_SUBTYPE_<a>'.
Returns	TRUE On Success FALSE On Failure
Description: This function requests time code data of the board. It extends original bcReqTimeCodeData(). This function returns time code sub type in its extended structure.	

bcSetPeriodicDDSSelect	
Prototype	BOOL bcSetPeriodicDDSSelect (BC_PCI_HANDLE hBC_PCI, BYTE bSel);
Packet	0x43
	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function bSel = unsigned char value to select periodic output or DDS output. The allowed values are defined in the 'bcuser.h' file: enum { SELECT_PERIODIC_OUT = 0x0 }; enum { SELECT_DDS_OUT = 0x1 };
Returns	TRUE On Success FALSE On Failure
Description: This function selects the periodic output or DDS output. Note that this selects the output choice. To enable the output, you have to call bcSetPeriodicDDSEnable.	

bcSetPeriodicDDSEnable	
Prototype	BOOL bcSetPeriodicDDSEnable (BC_PCI_HANDLE hBC_PCI, BYTE bEnable);
Packet	0x44
	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function bEnable = unsigned char value for enabling periodic or enabling DDS output. The allowed values are 1 to enable and 0 to disable.
Returns	TRUE On Success FALSE On Failure
Description: Depending on the selected periodic or DDS output choice (bcSetPeriodicDDSSelect), this function enables or disables that choice.	

bcSetDDSDivider	
Prototype	BOOL bcSetDDSDivider (BC_PCI_HANDLE hBC_PCI, BYTE bDiv);
Packet	0x45
	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function bDiv = unsigned char value to select divider value. The allowed values are defined in the 'bcuser.h' file: enum { DDS_DIVIDE _BY_1E0 = 0x0 }; enum { DDS_DIVIDE _BY_1E1 = 0x1 }; enum { DDS_DIVIDE _BY_1E2 = 0x2 }; enum { DDS_DIVIDE _BY_1E3 = 0x3 }; enum { DDS_DIVIDE _BY_1E4 = 0x4 }; enum { DDS_DIVIDE _BY_1E5 = 0x5 };

bcSetDDSDivider	
	enum { DDS_DIVIDE _BY_1E6 = 0x6 }; enum { DDS_DIVIDE _BY_1E7 = 0x7 }; enum { DDS_DIVIDE _BY_PREG = 0xF }; For information on the detailed DDS description, refer to Chapter 1.
Returns	TRUE On Success FALSE On Failure
Description: Sets the DDS divider value for the DDS frequency output.	

bcSetDDSDividerSource	
Prototype	BOOL bcSetDDSDividerSource (BC_PCI_HANDLE hBC_PCI, BYTE bSrc);
Packet	0x46
	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function bSrc = unsigned char value to select divider source. The allowed values are defined in the 'bcuser.h' file: enum { DDS_DIVIDER_SRC_DDS = 0x0 }; enum { DDS_DIVIDER_SRC_MULT = 0x1 }; enum { DDS_DIVIDER_SRC_100MHZ = 0x2 }; For information on the detailed DDS description, refer to Chapter 1..
Returns	TRUE On Success FALSE On Failure
Description: Sets the DDS divider source for the DDS frequency output.	

bcSetDDSSyncMode	
Prototype	BOOL bcSetDDSSyncMode (BC_PCI_HANDLE hBC_PCI, BYTE bMode);
Packet	0x47
	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function bMode = unsigned char value to select synchronization mode. The allowed values are defined in the 'bcuser.h' file: enum { DDS_SYNC_MODE_FRAC = 0x0 }; enum { DDS_SYNC_MODE_CONT = 0x1 };

bcSetDDSSyncMode	
	For information on the detailed DDS description, refer to Chapter 1.
Returns	TRUE On Success FALSE On Failure
Description: This sets the DDS synchronization mode for the DDS frequency output.	

bcSetDDSMultiplier	
Prototype	BOOL bcSetDDSMultiplier (BC_PCI_HANDLE hBC_PCI, BYTE bMult);
Packet	0x48
	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function bMult = unsigned char value to select continuous synchronization mode multiplier. The allowed values are defined in the 'bcuser.h' file: enum { DDS_MULTIPLY_BY_1 = 0x1 }; enum { DDS_MULTIPLY_BY_2 = 0x2 }; enum { DDS_MULTIPLY_BY_3 = 0x3 }; enum { DDS_MULTIPLY_BY_4 = 0x4 }; enum { DDS_MULTIPLY_BY_6 = 0x6 }; enum { DDS_MULTIPLY_BY_8 = 0x8 }; enum { DDS_MULTIPLY_BY_10 = 0xA }; enum { DDS_MULTIPLY_BY_16 = 0x10 }; For information on the detailed DDS description, refer to Chapter 1.
Returns	TRUE On Success FALSE On Failure
Description: This sets the continuous synchronization mode multiplier for the DDS frequency output.	

bcSetDDSPeriodValue	
Prototype	BOOL bcSetDDSDividerSyncMode (BC_PCI_HANDLE hBC_PCI, DWORD period);
Packet	0x49
	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function period = unsigned long value in the range of [0, 0xFFFFFF] for period value.

bcSetDDSPeriodValue	
	For information on the detailed DDS description, refer to Chapter 1.
Returns	TRUE On Success FALSE On Failure
Description: This sets the DDS period value for the DDS frequency output.	

bcSetDDSTuningWord	
Prototype	BOOL bcSetDDSTuningWord (BC_PCI_HANDLE hBC_PCI, DWORD tuneWord);
Packet	0x4A
	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function tuneWord = unsigned long value for the turning word. For information on the detailed DDS description, refer to Chapter 1.
Returns	TRUE On Success FALSE On Failure
Description: This sets the DDS tuning word for the DDS frequency output.	

bcSetDDSFrequency	
Prototype	BOOL bcSetDDSFrequency (BC_PCI_HANDLE hBC_PCI, DOUBLD freq);
Packet	N/A
	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function freq = double value to specify the DDS frequency. The DDS frequency can have fractional values. For information on the detailed DDS description, refer to Chapter 1.
Returns	TRUE On Success FALSE On Failure
Description: This sets the frequency for the DDS frequency output. Note that this function automatically selects DDS output (bcSetPeriodicDDSSelect) and sets the synchronization mode to DDS_ SYNC_MODE_FRAC (bcSetDDSDividerSyncMode).	

bcSetTcInEx	
Prototype	BOOL bcSetTcInEx (BC_PCI_HANDLE hBC_PCI, BYTE TcIn, BYTE SubType);
Packet	0x15
	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function TcIn = unsigned char value for time code input. SubType = unsigned char value for time code subtype. The allowed values are defined in the 'bcuser.h' file: TCODE_IRIG_A, TCODE_IRIG_SUBTYPE_NONE ('A' - IRIG A no year) TCODE_IRIG_A, TCODE_IRIG_SUBTYPE_Y ('AY' - IRIG A with year) TCODE_IRIG_B, TCODE_IRIG_SUBTYPE_NONE ('B' - IRIG B no year) TCODE_IRIG_B, TCODE_IRIG_SUBTYPE_Y ('BY' - IRIG B with year) TCODE_IRIG_B, TCODE_IRIG_SUBTYPE_T ('BT' - IRIG B Legacy TrueTime) TCODE_IEEE, TCODE_IRIG_SUBTYPE_NONE ('I' - IRIG B IEEE 1344) TCODE_IRIG_E, TCODE_IRIG_SUBTYPE_NONE ('E' - IRIG E 1000Hz no year) TCODE_IRIG_E, TCODE_IRIG_SUBTYPE_Y ('EY' - IRIG

bcSetTcInEx	
	E 1000Hz with year) TCODE_IRIG_e, TCODE_IRIG_SUBTYPE_NONE ('e' - IRIG E 100Hz no year) TCODE_IRIG_e, TCODE_IRIG_SUBTYPE_Y ('eY' - IRIG E 100Hz with year) TCODE_IRIG_G, TCODE_IRIG_SUBTYPE_NONE ('G' - IRIG G no year) TCODE_IRIG_G, TCODE_IRIG_SUBTYPE_Y ('GY' - IRIG G with year) TCODE_NASA, TCODE_IRIG_SUBTYPE_NONE ('N' - NASA 36) TCODE_XR3, TCODE_IRIG_SUBTYPE_NONE ('X' - XR3) TCODE_2137, TCODE_IRIG_SUBTYPE_NONE ('2' - 2137)
Returns	TRUE On Success FALSE On Failure
Description: This sets the input time code format and subtype. It extends the function bcSetTcIn to support new time code formats and subtypes.	

bcReadEvet2TimeEx	
Prototype	BOOL bcReadEvent2TimeEx (BC_PCI_HANDLE hBC_PCI, PDWORD major, PDWORD min, PWORD nano, PBYTE stat);
Packet	N/A
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function major = unsigned long pointer to store major time (Unix format). min = unsigned long pointer to store microseconds. nano = unsigned short pointer to store 100 nano seconds count. stat = unsigned char to store status bits.
Returns	TRUE On Success FALSE On Failure
Description: Latches and returns time captured from the event2. (See section 4.4 for a programming example.)	

bcReadEvent3TimeEx	
Prototype	BOOL bcReadEvent3TimeEx (BC_PCI_HANDLE hBC_PCI, PDWORD major, PDWORD min, PWORD nano, PBYTE stat);
Packet	N/A

bcReadEvent3TimeEx	
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function major = unsigned long pointer to store major time (Unix format). min = unsigned long pointer to store microseconds. nano = unsigned short pointer to store 100 nano seconds count. stat = unsigned char to store status bits.
Returns	TRUE On Success FALSE On Failure
Description: Latches and returns time captured from the event3. (See section 4.4 for a programming example.)	

bcReqEventsData	
Prototype	BOOL bcReqEventsData (BC_PCI_HANDLE hBC_ PCI, EventsData *pdata);
Packet	0x19
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function pdata = pointer to EventsData structure. The structure is defined in the "bcuser.h" header file.
Returns	TRUE On Success FALSE On Failure
Description: Returns event, event2 and event3 data. The information for each event includes enabled, sense and capture lock.	

bcSetEventsData	
Prototype	BOOL bcSetEventsData (BC_PCI_HANDLE hBC_ PCI, EventsData *pdata);
Packet	0x19
Input Parameter	BC_PCI_HANDLE hBC_PCI : Handle returned from 'bcStartPci' function pdata = pointer to EventsData structure. The structure is defined in the "bcuser.h" header file.
Returns	TRUE On Success FALSE On Failure
Description: Sets event, event2 and event3 data. The information for each event includes enabled, sense and capture lock.	

4.4. Programming Examples

4.4.1. General

The example code fragments in this chapter are written in the C programming language. These examples are extracted from the bc63xPCIcfg application included in this kit.

4.4.2. Starting and Stopping the Device

The following example starts and stops the device:

```
BC_PCI_HANDLE hBC_PCI;

// Start the device
hBC_PCI = bcStartPci();
if (!hBC_PCI)
{
printf ("Error Opening Device Driver\n");
return -1;
}
…
…
…
// Stop the device
bcStopPci(hBC_PCI);
```

4.4.3. Reading Time On Demand

The following example reads the time from the TFP register:

```
DWORD maj, min;
WORD nano;
BYTE stat;
```

```
struct tm *majtime;
```

Reading in Binary Time Format

```
if ( bcReadBinTimeEx (hBC_PCI, &maj, &min, &nano, &stat) == TRUE )
{
majtime = gmtime( &maj );
printf( "\nBinary Time: %02d/%02d/%d  %02d:%02d:%02d.%06lu%d
Status: %d",
      majtime->tm_mon+1, majtime->tm_mday, majtime->tm_year+1900,
      majtime->tm_hour, majtime->tm_min, majtime->tm_sec, min, nano,
stat);

}
```

Reading in Decimal Time Format

```
if ( bcReadDecTimeEx (hBC_PCI, majtime, &min, &nano, &stat) == TRUE )
{
printf( "\nDecimal Time: %d %d  %02d:%02d:%02d.%06lu%d Status: %d",
majtime->tm_yday, majtime->tm_year+1900,
majtime->tm_hour, majtime->tm_min, majtime->tm_sec, min, nano, stat);
}
```

4.4.4. Setting theTFP Mode

The following example sets the TFP mode to GPS:

```
bcSetMode (hBC_PCI, MODE_GPS);
```

4.4.5. Setting Interrupts

The following example sets a 1PPS Interrupt:

```
// Define an Interrupt handler function
void bcIntHandlerRoutine(BC_PCI_HANDLE hBC_PCI, DWORD dwSource)
{
printf("Got Interrupt Number: %d\n", dwSource);
}

// Start the interrupt routine
bcStartIntEx(hBC_PCI, bcIntHandlerRoutine, INTERRUPT_1PPS);

// Set the interrupt type
bcSetInt(hBC_PCI, INTERRUPT_1PPS);

// To stop interrupt generation
bcStopInt(hBC_PCI);
```

5. Solaris SDK

5.1. Introduction

5.1.1. General

The Solaris bc63xPCI-V2 / PCIe Driver Kit provides a device driver and a sample program useful in the development of applications which access features of the bc635PCI-V2, bc637PCI-V2, bc635PCIe, and bc637PCIe Time and Frequency Processor. This kit is designed to provide an interface between the PCI/PCIe TFP and applications developed for Solaris 8, 9, and 10 on SPARC and x86_64 Platforms. The driver and sample program source code are provided to give a better understanding of the kit features and benefits.

5.1.2. Features

The main features of the Driver Kit include:

- The device driver and example program with source
- This section of the User's Guide providing library definitions

5.1.3. Overview

The Driver Development Kit provides an interface to the bc635PCI-V2, bc637PCI-V2, bc635PCIe, and bc637PCIe Time and Frequency Processor in the 64 bit environments of Solaris 8, 9, and 10. The example program provides sample code shows 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 program was developed using discrete functions for each operation, which allows the developer to copy any useful code and use it in their own applications.

This release supports both the SPARC and x64_64 platforms. You need to use the platform specific tar file for installation.

5.2. Installation

5.2.1. 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 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 PCI / PCIe slot, plug the bc635PCI-V2, bc637PCI-V2, bc635PCIe, and bc637PCIe into it, and then install the device driver. Be sure to consult the user documentation that came with your particular workstation for any specific card installation instructions. When installing the PCI / PCIe card, use good ESD precautions.

5.2.2. Software Installation

The software is created as a Solaris package. To install the software package follow these steps.

1. Untar the platform specific package tar file.

```
/export/home/user> su

# tar xf BCPCI-V<aaa>-sparc64-build<bbb>.tar
```

Note 'user' is the name of your user. <aaa> is the release version, and <bbb> is the three digit build number. For x86_64 platform, the file is BCPCI-V<aaa>-x86_64-build<bbb>.tar.

2. Use the 'pkgadd' command to install the package. Answer 'y' to the question when prompted.

```
# pkgadd -d . BC635PCI

Processing package instance <BC635PCI> from </export/home/user/bc635pci>

BC635/637PCIe-V2 Solaris driver(sparc) 1.0,REV=8.0V0
Symmetricom, Inc.
Using </> as the package base directory.
## Processing package information.
## Processing system information.
   1 package pathname is already properly installed.
## Verifying disk space requirements.
## Checking for conflicts with packages already installed.
```

```
## Checking for setuid/setgid programs.

This package contains scripts which will be executed with super-user
permission during the process of installing this package.

Do you want to continue with the installation of <BC635PCI> [y,n,?] y

Installing BC635/637PCI-V2/PCIe Solaris driver as <BC635PCI>

## Installing part 1 of 1.
/kernel/drv/sparcv9/stfp
/opt/BC635PCI/include/stfpio.h
/opt/BC635PCI/src/Makefile
/opt/BC635PCI/src/bc635pci.c
/opt/BC635PCI/src/bc63xPCIcfg
/opt/BC635PCI/src/stfp.c
[ verifying class <none> ]
## Executing postinstall script.
Driver installed. Please reboot now.

Installation of <BC635PCI> was successful.
```

3. Reboot your computer.

This driver should work correctly in its current binary form without the need for recompilation. Alternatively, you can also use the provided 'install.sh' script to install the software. This script is in the directory where you find the package tar file. You must run 'install.sh' as root. The script combines the 'untar' and 'pkgadd' together. Again, remember to reboot your machine after finishing the script.

The install script for SPARC platform is shown below:

```
#!/bin/sh

# Install easy script for BC635PCI package.
```

```
# Copyright (C) 1997, EIS Computers, Symmetricom Inc.
# Copyright (C) 2007-2009, Symmetricom Inc.

#
# Run this simple script from the same directory as the

# BCPCI-V800-sparc64-build<aaa>.tar file you have received.
# Note <aaa> is the three digit build number.
#



TARFILE=`ls BCPCI-V800-sparc64-build*.tar`
if [ $? -ne 0 ]; then
    echo "The tar file for the BCPCI software does not exist!"
    exit 1
fi
tar xf $TARFILE
PATH=$PATH:/usr/bin:/usr/sbin
export PATH
PKG=BC635PCI
pkgadd -d . $PKG
```

Note: On the SPARC platform, the driver stfp is installed to /kernel/drv/sparcv9 directory. On the x86_64 platform, the driver stfp is installed to /kernel/drv/amd64 directory.

After your computer has rebooted, you can inspect the installed package using the Solaris 'pkginfo' command (output of other install packages are removed for clarity).

```
/export/home/user> pkginfo | grep BC635PCI
Symmetricom BC635PCI  BC635/637PCIe-V2 Solaris driver
```

In case, you want to remove the software package, use the Solaris 'pkgrm' command. Answer 'y' to the two questions when prompted. The output for removing BC635PCI package on the SPARC platform is shown below.

```
# pkgrm BC635PCI

The following package is currently installed:
   BC635PCI  BC635/637PCI-V2/PCIe Solaris driver
             (sparc) 1.0,REV=8.0V0

Do you want to remove this package? [y,n,?,q] y

## Removing installed package instance <BC635PCI>

This package contains scripts which will be executed with super-user
permission during the process of removing this package.

Do you want to continue with the removal of this package [y,n,?,q] y
## Verifying package <BC635PCI> dependencies in global zone
## Processing package information.
## Removing pathnames in class <none>
/opt/BC635PCI/src/stfp.c
/opt/BC635PCI/src/bc63xPCIcfg
/opt/BC635PCI/src/bc635pci.c
/opt/BC635PCI/src/Makefile
/opt/BC635PCI/src
/opt/BC635PCI/include/stfpio.h
/opt/BC635PCI/include
/opt/BC635PCI
/opt <shared pathname not removed>
/kernel/drv/sparcv9/stfp
## Executing postremove script.
```

```
BC635/637 PCI driver has been removed from the kernel.
## Updating system information.

Removal of <BC635PCI> was successful.
```

5.2.3. Test Installation

The software is installed in /opt/BC635PCI. Use the sample program to test the installation.

```
# cd /opt/BC635PCI
# cd src
# ./bc63xPCIcfg

--------------------------------------------------------------------------
                            Symmetricom - TT & M
                         bc635/637PCIe-V2 Configurator
                  Version 3.0
--------------------------------------------------------------------------

1. Read Current Time (Press enter to return to menu)
2. Read Event Time   (Press enter to return to menu)
3. Set Current Time
4. Set Current Year
5. Set Strobe Time
6. Program Control Register
7. Program Leap Event Seconds
8. Select Time Format
9. Select Operational Mode
10. Select Decoding TimeCode Format
11. Select TimeCode Output Format
12. Select Clock Source
13. Select Output Frequency
14. Program Heartbeat counters
15. Set Local Time Offset           16. Set Generator Time Offset
17. Set Propagation Delay           18. Set Local Time Offset Flag
19. Set Year Auto Increment Flag    20. Sync RTC to External Time Data
21. Set GPS Time Format             22. Set GPS Mode Flag
23. Software Reset                  24. Request Time Settings
```

```
25. Request Clock Settings          26. Request Offset Settings
27. Request UTC Information         28. Request Model Information
29. Request Firmware Version        30. Advanced Menu
31. Interrupts Menu                 32. GPS Packets Menu - bc637PCI
                                    Only
33. PCI Revision ID                 34. Read from Dual Port RAM
35. Write to Dual Port RAM          36. Write ACK - Send DP Command
37. DDS Menu - V2 Hardware Only
38. Select TimeCode Input Format - V2 Hardware
39. Select TimeCode Output Format - V2 Hardware
40. Read from Register              41. Write to Register
42. Select events settings
43. Read Event2 Time  (Press enter to return to menu)
44. Read Event3 Time  (Press enter to return to menu)
0.  Exit the Pro-
gram.....................................................

Select:
```

The newer TFP hardware is referred to as the V2 hardware. The older hardware is referred to as the V1 hardware. You can use option 33 (PCI Revision ID) to find out the hardware version. The V2 hardware has Revision ID in the range [0x20, 0x2F]. The revision ID is stored in the PCI configure space field 'Revision ID'.

The option 37 works for the V2 hardware only. The options 38 and 39 mainly work for the V2 hardware. Only a small set of options 38 and 39 work for the V1 hardware.

The V2 hardware supports time stamping external event2 and event3 if they are enabled. Refer to Chapter 1 for details. The interrupt mask values for event2 and event3 are INT_EVENT2 and INT_EVENT3 that are defined in 'stfpio.h'.

5.2.4. Driver Compilation

The driver module should work as is in its current binary format without the need for recompilation, so this section is for completeness only.

A Makefile is provided with the source files. If you have 'make' or 'gnu-make' installed, you can simply type 'make' or 'gmake' to compile. The Makefile for the SPARC platform is shown below.

```
# Makefile to build the stfp driver and the bc63xPCIcfg program
```

```
#

DRVFLAGS = -I. -m64 -xarch=sparc -xcode=abs32 -xregs=no%appl -xO3
APPFLAGS = -I. -m32 -xO3

DRIVER   = stfp
PCIDEMO  = bc63xPCIcfg

OBJS     = stfp.o bc635pci.o

.PHONY: all
all: $(DRIVER) $(PCIDEMO)

.PHONY: clean
clean:
@rm -f $(OBJS)
@rm -f $(DRIVER)
@rm -f $(PCIDEMO)

$(DRIVER): stfp.o
ld -r -o $@ $+

$(PCIDEMO): bc635pci.o
$(CC) -o $@ $+

stfp.o: stfp.c
$(CC) -D_KERNEL -c $(DRVFLAGS) -o $@ $<

bc635pci.o: bc635pci.c
$(CC) -c $(APPFLAGS) -o $@ $<
```

```
Copy stfp into the / kernel/drv/sparcv9 directory. The driver is ready
to be installed using following add_ drv( 1M) function.


# 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.


# modunload -i module-id


The Makefile for the x86_64 platform is also provided. Make sure copy
stfp into the /kernel/drv/amd64 directory on the x86 platform.
```

5.3. Driver Function Definitions

5.3.1. General

The 'STFP' device driver provides functions for each of the programming packets supported by the bc635PCI-V2, bc637PCI-V2, bc635PCIe, and bc637PCIe 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 Chapter 1.

5.3.2. Functions

The 'STFP' device driver supports the bc635PCI-V2, bc637PCI-V2, bc635PCIe, and bc637PCIe Time and Frequency Processor (TFP) modules. 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. A variety of 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, or a DDS frequency clock in the range from less than 1Hz to greater than 100MHz.

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 accessed 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 Chapter 1.

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 of Chapter 1. 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 Chapter 1 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 Chapter 1) 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 and bc637PCIe. 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 Chapter 1 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

REQREVID, *int

Reads the Revision ID field in the PCI configure register space

RDDPOFFSET, *struct dp_rdwr_t

Reads the Dual Port RAM byte data at the specified offset

WRDPOFFSET, *struct dp_rdwr_t

Sets the Dual Port RAM byte data at the specified offset

SENDCOMMAND

This ioctl control code exposes the internal function that sends the command in the Dual Port RAM input area to the timing engine for execution. In fact, many ioctl codes work by taking the input data from the application to set up a command in the Dual Port RAM input area and then send the command to the timing engine for execution. The ioctl interface encapsulates this detail from the application code. However, with the above WRDPOFFSET ioctl control code, now application code can set up any Dual Port RAM command by writing the command and its data at the specified offsets. The application can invoke SENDCOMMAND to send the command to the timing engine for execution. This provides the maximum flexibility for application code to exercise features of the hardware when suitable ioctl control codes may not be available.

DDSCOMMAND, *struct dds_command_t

Executes DDS commands. The DDS commands are Dual Port RAM commands that are sent to the timing engine and be executed. The application code uses the 'cmd_id'field of the 'struct dds_

command_t' to specify which DDS command to invoke. The source file 'bc635pci.c' for the demo program defines a set of DDS functions, such as, bcSetPeriodicDDSSelect(),bcSetPeriodicDDSEnable(), etc. See the source code for details.

TCFORMATEX, *struct tc_command_t

Selects the time code input format and its sub type or time code output format and sub type. The hardware supports a variety of input and output time codes. This ioctl control code supersedes both SELTCFORMAT and SETTCOUTFMT ioctl codes. See stfpio.h for the definitions of supported time codes and sub types. The sample code showing how to set input time code and its sub type is in the function bcSetTcInEx() inside the file 'bc635pci.c'. Similarly, the function bcSetGenCodeEx() inside the file 'bc635pci.c' shows how to set output time code and its sub type.

RDREGOFFSET, *struct reg_rdwr_t

Reads the register value from the specified address

WRREGOFFSET, *struct reg_rdwr_t

Sets the register value to the specified address

RDEVENT2TV, *struct stfp_timeval

Reads time from the EVENT2 registers assuming the time format is binary.

RDEVENT3TV, *struct stfp_timeval

Reads time from the EVENT3 registers assuming the time format is binary.

5.4. Example Program

5.4.1. General

The bc63xPCIcfg.exe is an example program that provides sample code. The program exercises the device driver functions. It is also 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. This allows the developer to clip any useful code and use it in their own applications.

5.4.2. 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%d 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.hnsec, stm.status);`
	
	`/* Binary Time Format */`
	`ioctl (fd, RDTIMETVREQ, &tvTime);`
	`printf ("Binary Time: %lu.%06ld%d status: %x ", tvTime.tv.tv_sec, tvTime.tv.tv_usec, tvTime.hnsec, 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_read_event2_time
Description	Reads the event2 time in binary time format
Example	ioctl (fd, RDEVENT2TV, &tvTime);
Registers	pci_read_event3_time
Description	Reads the event3 time in binary time format
Example	`ioctl (fd, RDEVENT3TV, &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);`

Registers	pci_revision ID
Description	Reads Revision ID field of the PCI configure space
Example	`ioctl (fd, REQREVID, &rev_id);`

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

Command 0x11	pci_time_format
Description	Sets the time format, binary or decimal
Example	ioctl (fd, SELTIMEFORMAT, TIME_BINARY);
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 0x13	pci_set_year

Description	Sets the TFP year
Example	`ioctl (fd, SETYEAR, year);`

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

Command 0x15	pci_time_code and its sub type
Description	Sets the time code input format and its sub type

Example	`struct tc_command_t tccmd;`
	`tccmd.cmd_id = COM_SETTCFORMAT;`
	`tccmd.type = TC_IRIGB;`
	`tccmd.subtype = TC_SUBTYPE_Y;`
	`ioctl (fd, TCFORMATEX, &tccmd);`

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

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);`

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

Command 0x19	pci_req_time_settings
Description	Request time settings

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 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 0x1A	pci_sw_reset
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 0x1B	pci_tc_out_format and its sub type
Description	Sets the time code output format and its sub type

Example	`struct tc_command_t tccmd;`
	`tccmd.cmd_id = COM_SETTCOUTFMT;`
	`tccmd.type = TC_IRIGE;`
	`tccmd.subtype = TC_SUBTYPE_1;`
	`ioctl (fd, TCFORMATEX, &tccmd);`

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

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 0x43	DDS or periodic output
Description	Select DDS output or periodic output

Example	`struct dds_command_t ddscmd;`
	`ddscmd.cmd_id = COM_SET_PRD_DDS_SEL;`
	`ddscmd.data.cmd_byte[0] = SELECT_DDS_OUT;`
	`ddscmd.cmd_len = 1;`
	`ioctl (fd, DDSCOMMAND, &ddscmd);`

Command 0x44	DDS or periodic output enable
Description	Enables or disables DDS output or periodic output

Example	`struct dds_command_t ddscmd;`
	`ddscmd.cmd_id = COM_SET_PRD_DDS_ENA;`
	`ddscmd.data.cmd_byte[0] = 1;`
	`ddscmd.cmd_len = 1;`
	`ioctl (fd, DDSCOMMAND, &ddscmd);`

Command 0x45	DDS divider
Description	Sets DDS divider value

Example	`struct dds_command_t ddscmd;`
	`ddscmd.cmd_id = COM_SET_DDS_DIVIDER;`
	`ddscmd.data.cmd_byte[0] = DDS_DIVIDE_BY_1E2;`
	`ddscmd.cmd_len = 1;`
	`ioctl (fd, DDSCOMMAND, &ddscmd);`

Command 0x46	DDS divider source
Description	Sets DDS divider source

Example	`struct dds_command_t ddscmd;`
	`ddscmd.cmd_id = COM_SET_DDS_DIV_SRC;`
	`ddscmd.data.cmd_byte[0] = DDS_DIVIDER_SRC_DDS;`
	`ddscmd.cmd_len = 1;`
	`ioctl (fd, DDSCOMMAND, &ddscmd);`

Command 0x47	DDS sync mode
Description	Sets DDS sync mode

Example	`struct dds_command_t ddscmd;`
	`ddscmd.cmd_id = COM_SET_DDS_DIV_SYNC;`
	`ddscmd.data.cmd_byte[0] = DDS_SYNC_MODE_FRAC;`
	`ddscmd.cmd_len = 1;`
	`ioctl (fd, DDSCOMMAND, &ddscmd);`

Command 0x48	DDS multiplier
Description	Sets DDS multiplier value

Example	`struct dds_command_t ddscmd;`
	`ddscmd.cmd_id = COM_SET_DDS_MULTI;`
	`ddscmd.data.cmd_byte[0] = DDS_MULTIPLY_BY_3;`
	`ddscmd.cmd_len = 1;`
	`ioctl (fd, DDSCOMMAND, &ddscmd);`

Command 0x49	DDS period value
Description	Sets DDS period value

Example	`struct dds_command_t ddscmd;`
	`ddscmd.cmd_id = COM_SET_DDS_PERIOD;`
	`ddscmd.data.cmd_number = period;  // [0, 0xFFFFFF]`
	`ddscmd.cmd_len = 4;`
	`ioctl (fd, DDSCOMMAND, &ddscmd);`

Command 0x4A	DDS tuning word
Description	Sets DDS tuning word

Example	`struct dds_command_t ddscmd;`
	`ddscmd.cmd_id = COM_SET_DDS_TUNEWORD;`
	`ddscmd.data.cmd_number = tuneWord;`
	`ddscmd.cmd_len = 4;`
	`ioctl (fd, DDSCOMMAND, &ddscmd);`

Command 0x4F	pci_req_fw_ver
Description	Request firmware version
Example	/* Get Firmware Version */ get.arg = GETDATA_DTFW; 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);

5.4.3. Example 1: GPS Packet 46 - Health Packet 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);
```

Note: Symmetricom bus cards support different GPS receivers. Please consult the Users Guide for your bus card to access the GPS receiver commands that apply to your card.

5.4.4. Example 2: 1PPS Interrupt Sample

```
main ()
```

```
{
    /* Open device */
    open ("/dev/stfp0", O_RDWR);

    /* Initialize to free Running Mode */
    ioctl (fd, SELTIMINGMODE, MODE_FREERUN);
    /* Initialize to binary time format */
    ioctl (fd, SELTIMEFORMAT, TIME_BINARY);

    /* Setup interrupt signal handler */
    signal (SIGUSR1, sigHandler);
    printf ("Hit RETURN to quit. . .\n");
    /* enable 1PPS interrupt signals */
    ioctl (fd, SETINTSIGNAL, INT_1PPS);

    /* wait for RETURN key */
    scanf ("%c", &junk);

    /* disable interrupt signals */
    ioctl (fd, SETINTSIGNAL, 0);

    /* Close device */
    close(fd);
}

void sigHandler (int sig, int code, struct sigcontext *scp, char *addr)
{
    int intServiced;
    /* Read Signal */
    ioctl (fd, RDINTSIGNAL, &intServiced);
```

```
    printf ("Got interrupt signal: %d  Source: 0x%02X\n", sig, int-
Serviced);

}
```

The example program has demonstration for interrupt handling. You can access the interrupt menu through choice 31 of the main bc63xPCIcfg menu. Refer to 'pci_set_ints()' and 'intr_handler()' in 'bc635pci.c' for source code.

Glossary

The following is a glossary of key terms used in the discussion of timing operations: An expanded glossary of terms is available on-line at:

http://www.symmetricom.com/resource/glossary/

BCD: Binary Coded Decimal. Also called packed decimal, this is the representation of each digit of a decimal number by four-bit binary numbers. For example, the number 42 would be encoded as 0100 0010 .

Coordinated Universal Time (UTC): See UTC.

COTS: Commercial Off-The-Shelf products or services that are generally available and not built to customized specifications.

DCLS: Direct Current Level Shift, or digital IRIG.

Discipline: The word discipline, as used in this manual, means to adjust the frequency of the 10 MHz oscillator to track the incoming reference signal.

DPRAM: Dual Port RAM.

Epoch: A reference time or event. Epoch often refers to a one pulse per second event.

Event: An event is defined here as a transition of a digital signal (rising or falling), which can be used to time stamp the event.

Flywheel: To maintain time or frequency accuracy when the reference source has been lost or removed.

GPS: Global Positioning System. Originally designated NAVSTAR (Navigation System with Timing And Ranging), GPS was developed by the US Department of Defense to provide all-weather round-the-clock navigation capabilities for military ground, sea, and air forces.

HW: Hardware.

IRIG: Serial time format standard maintained by the Inter Range Instrumentation Group.

ISA: Industry Standard Architecture; desktop PC adapter board specification.

Jamsync: Is the process of abruptly synchronizing with a time reference, as opposed to gradually adjusting to match up with the time reference.

Major Time: Units of time larger than or equal to seconds.

MHz: A MegaHertz is one million (1,000,000) cycles per second.

Minor Time: Sub-second time to whatever resolution is supported.

MTBF: Mean Time Between Failure, a measure of reliability. The longer the time span between failures, the more reliable the device.

MTTR: Mean Time To Repair.

NASA 36: National Aeronautics & Space Administration 1-second BCD 36-bit Time Code.

NIST: National Institute of Standards and Technology, the National Measurement Institute in the United States.

OCXO: Oven-Controlled Crystal Oscillator

OS: Operating System.

Packet: A group of bytes conforming to a defined structure. Packets are usually used in bit serial or byte serial data transmissions to allow framing of the transmitted data. The bc637PCI-V2 uses data packets to communicate with the optional GPS receiver.

PCI: Peripheral Component Interconnect, a local bus that supports high-speed connection with peripherals. It plugs into a PCI slot on the motherboard.

PCIe: PCI Express, (Peripheral Component Interconnect Express), officially abbreviated as PCIe, is a computer expansion card standard designed to replace the older PCI, PCI-X, and AGP standards. Introduced by Intel in 2004, PCIe is the latest standard for expansion cards that is available on mainstream personal computers. PCI Express is used in consumer, server, and industrial applications, both as a motherboard-level interconnect (to link motherboard-mounted peripherals) and as an expansion card interface for add-in boards. A key difference between PCIe and earlier PC buses is a topology based on point-to-point serial links, rather than a shared parallel bus architecture.

PCISIG: PCI Special Interest Group.

PCM: Pulse Code Modulation.

Periodic: A programmable frequency that is obtained by dividing the TFP reference frequency. Periodics are sometimes referred to as “heartbeats.” PICMG: PCI Industrial Computer Manufacturers Group.

PLL: Phase-Locked Loop.

PPM: parts per million.

PPS: pulse per second.

RAM: Random Access Memory.

Resolution: Resolution of a time code refers to the smallest increment of time, whether it is days, hours, seconds, or other increments.

Strobe: The strobe is a programmable “alarm.” It compares the reference time with a user-programmed time, and outputs a signal when the two values are the same. The signal is indicated by a transition from low to high voltage. The duration of the signal is equal to 1 uSec. The Strobe function is also referred to as Time Compare.

SW: Software.

TCXO: Temperature Compensated Crystal Oscillator

TFP: Time and Frequency Processor is the name given to the bc63x family of products.

USNO: U.S. Naval Observatory, in Washington, D.C., where the atomic clock that serves as the official source of time for the United States is maintained.

UTC: The international time standard is called Universal Coordinated Time or, more commonly, UTC, for “Universal Time, Coordinated”. This ITU standard has been in effect since 1972. UTC is

maintained by the Bureau International de l'Heure (BIH), which forms the basis of a coordinated dissemination of standard frequencies and time signals.

VCXO: Voltage-Controlled Crystal Oscillator.

Index