P2P Software Library and Utilities

User Manual

The Goebel Company

Revision: 1.20 **April 15, 2013**

Purpose

This manual describes the software for FCE bus P2P interface boards offered by The Goebel Company. This includes application programming interface library and management applications.

Notice

Information in this manual has been carefully reviewed and is believed to be accurate. The Goebel Company shall not be liable for errors contained herein. The Goebel Company reserves the right to make changes or additions to the software described herein.

Contact

For technical or other inquiries contact:

email: info@GoebelEtc.com

The Goebel Company
12486 Prowell
Leavenworth WA 98826
USA
Phone: 206-601-6010

In Europe contact:

email: info@muellersystemtechnik.de

Müller Systemtechnik GmbH
Klaus R. Muller
Mendelssohnstraße 34
D-81245 München
Tel: +49 (0)89 28659081
Fax: +49 (0)89 28659082

Table of Content

1 Introduction

P2P refers to the packet based Point-to-Point busses of the Flight Control Electronics of the Boeing 787. There are two speeds of P2P bus, a high speed 5Mbit/sec, and a low speed 400Kbit/sec. The busses are Manchester encoded with RS485 signaling. A preamble and postamble containing invalid Manchester encoding allows packets to be delineated without the possibility of data being interpreted as packet delineation.

The FCE equipment connected with these busses consists or Flight Control Modules (FCMs), Actuator Control Electronics (ACEs), REUs for remote actuation, and Direct Mode Rate Sensor (DMRS). High speed busses connect the FCMs and ACEs. Low speed busses connect the ACEs to REUs and DMRSs.

The API provides interfaces to transmit and receive frames complying with the P2P specification. A wrap protocol is used by FCM and ACE to confirm data path integrity, and LRU status. The P2P resource can provide the time critical wrap functions to simulate each type of LRU.

The P2P resource normally ensures error free transmissions. Error injection capabilities allow data to be transmitted with errors in data, CRC, packet length, or wrap violations. On reception error packets are captured with a detailed error status to aid in the detection of errors.

1.1 Board Versions

1.1.1 P2PSIM-X

The P2PSIM-X card is a 32 P2P channel full length PCI-X card containing IO PMC and Processor PMC. The processor PMC firmware provides advanced protocol processing and error injection capabilities to simultaneously drive all 32 channels in a mix of high speed and low speed modes.

1.1.2 P2P422SIM-X

The P2P422SIM-X card is a mix of 8 P2P channels plus 6 SDLC channels or 12 ASYNC channels on a full length PCI-X card. SDLC and ASYNC channels are configurable where each group or 4 twisted pair input/outputs can be configured as a single SDLC or dual ASYNC channels. Again the processor PMC firmware provides advanced protocol processing and error injection capabilities to simultaneously drive all P2P/SDLC/ASYNC channels.

1.1.3 P2PIO-MR

The P2PIO-MR card is a transformer coupled, conduction cooled and ruggedized PMC which drives 8 P2P channels, 4 in receive mode, and 4 in transmit mode.

1.1.4 Board IDs

P2P board versions can be identified by the “p2p id” command. This command returns a hardware id and fpga revision id. The following table outlines the differences.

hw_id	PN	Description	fpga_id	Description
1	P2PSIM-X	32 P2P channels	6	Hw1.2: Fix receiver reset on preamble noise
1	P2PSIM-X		9	Fixes receive of length error packets Allows pll speed control of +/- 20%
1	P2PIO-MR	8 P2P channel	1	FPGA equivalent to rev 6 of p2psim-x
2	P2P422SIM-X	8 P2P channel 6/12 SDLC/ASYNC channels	2	Fix voltage level of idle bus for SDLC protocol
3	P2PT20-X	20 transistor coupled P2P channels	1	Initial version of T20 FPGA

Table 1: P2P Hardware versions

1.2 P2P concepts

1.2.1 P2P packets

P2P packets consist of a preamble, followed by Manchester encoded data, CRC, and postamble. The packet data format consists of a label, followed by a length (number of 16 bit words in payload), followed by the payload consisting of 16 bit words. Data payloads are required to be an even number of 16 bit words.

1.2.2 Unidirectional and Bidirectional channels

A P2P channel can be a unidirectional connection from one LRU to another, or a bidirectional channel between LRUs. Bidirectional channels are used in low speed mode between ACEs and REUs. All other uses are for unidirectional channels. Unidirectional channels, have one transmitter, and up to 2 receivers in relatively close proximity. The close proximity allows the RS485 signal strength to sufficiently drive the receivers.

1.2.3 ADB - REU channels

Actuation Data Bus (ADB) channels is alternate terminology used to describe REU channels. REU channels provide a way to match the time critical response of REU to ACE transmissions. An REU must respond to ACE packets within 100 microseconds. In addition payload words of the ACE packet are wrapped back to the response, and the frame counter of the response packet is incremented. All this is done by the P2P resource autonomously without time critical guidance from the host.

1.2.4 IMB - ACE/FCE channels

Inter Module Bus (IMB) is alternate terminology used to describe the communication channels between ACE and FCE modules. ACE channels provide an encrypted wrap function to FCMs. Multiple packets are sent back-to-back within a frame. A flexible configuring capability allows the sequencing of packets reproducing the packet sequences of LRUs. The sequencing of packets is configured by using the p2p_sub_channel_config and p2p_rrsub_channel_config interfaces. Refer to these sections for information on sub-channels.

1.2.5 DMRS simulation

DMRS simulation provides a unidirectional channel from DMRS to the ACE. Packets are transmitted by the P2P resource autonomously, without time critical guidance from the host.

1.2.6 Honeywell restricted features

Honeywell has defined a feature set of sensitive functionality, that is restricted to certain customers. The functional details of these features are purposely not described in this manual, and require documentation from Honeywell International, or Boeing to define the functionality.

2 Hardware Architecture

2.1 PCIX dual PMC carrier

The P2P hardware (See Figure 1) consists of two PMC modules, hosted on a full length PCI carrier. The IO PMC has 32 channels routed out the front via a 68 pin SCSI II connector. The processor PMC is on the rear slot of the dual carrier, and does not require any cable connections. The carrier is universal type compatible with legacy 5V PCI as well as 3.3v PCI or PCIX. Note, for insertion on 32 bit slots, insure the base board does not have components which interfere with the rear tab of PCI-64 connections.

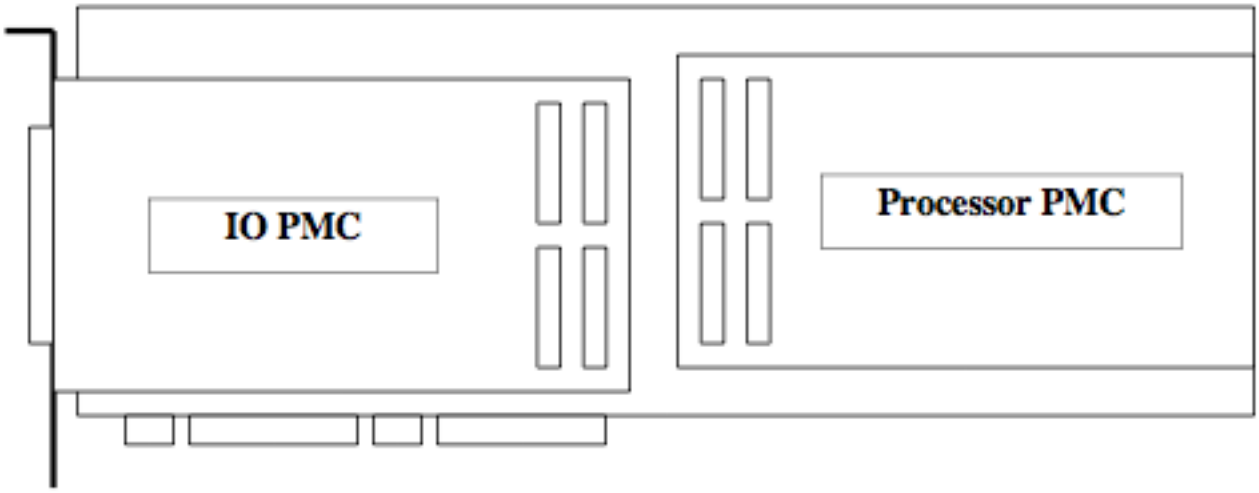


Figure 1 P2P on PCI dual PMC carrier

The PMC carrier is based on an Intel 31154 PCI-X bridge. The dual PMC-X carrier allows operation at PCI-X speeds of up to 133Mhz on a suitable host bus. In addition compatibility with legacy 5v 33Mhz/32 bit pci is supported while allowing the local PCI bus to remain at full speed.

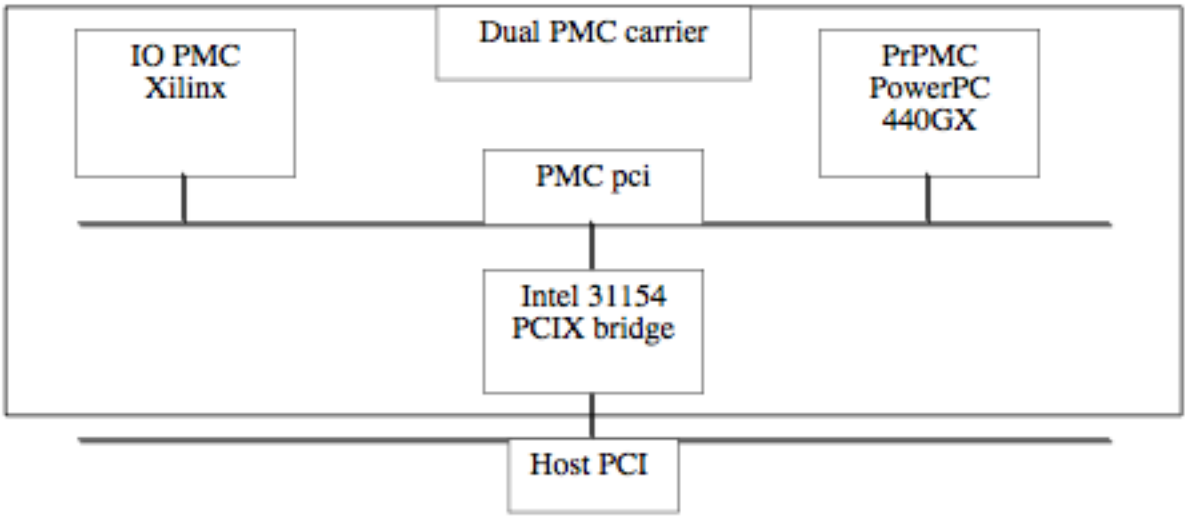


Figure 2 PCI bus diagram

The PMCs are connected by a local bus which is bridged to the host PCI via the Intel 31154 bridge. The processor and IO modules communicate on the local bus without any impact or interference from the host PCI bus. Time critical IO proceeds unencumbered by host PCI load. Compared to VME and cPCI the PCI carrier gives a higher performance at a lower unit cost.

2.2 PrPMC

PrPMC is the nomenclature for a processor on a PMC card. The PrPMC (See Figure 3) on the P2P test resource utilizes the low power IBM/AMCC PowerPC 440GX embedded Processor. With integrated PCI-X, DDR SDRAM, SRAM and Ethernet interfaces, the 440GX offers a high performance solution for general computing applications. A SO-DIMM DDR SDRAM memory slot and socketed boot flash provide flexible field upgradeability.

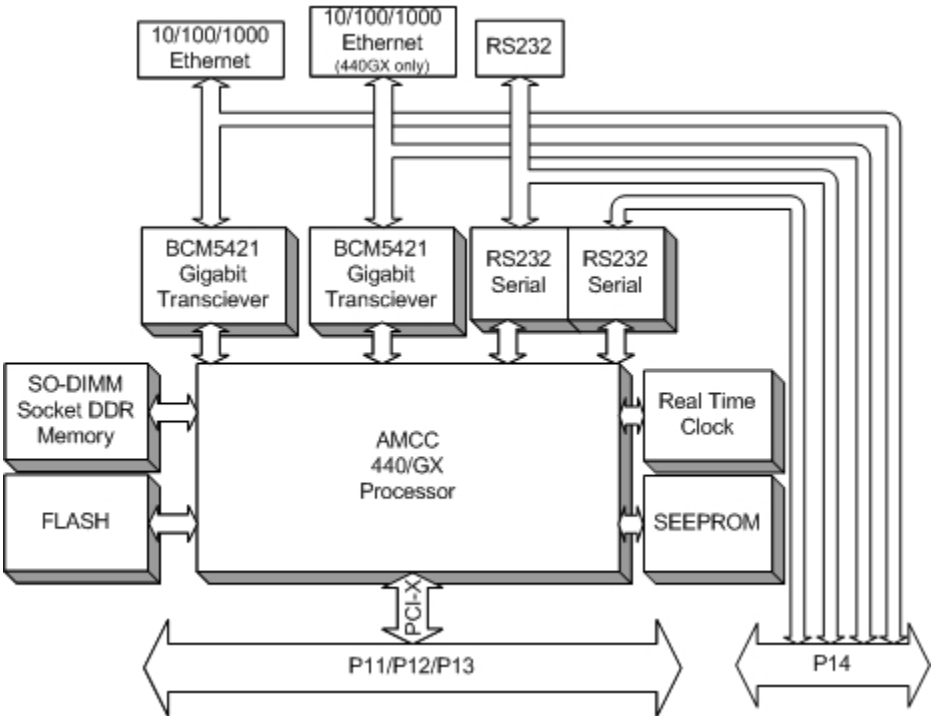


Figure 3 PrPMC

2.2.1 PCI interface

The PrPMC's PCI interface is capable of 133Mhz PCI-X speed. PCI-X is 3.3v and is adapted to 5V PCI systems via the bridged carrier. In this application, the PCI speed is limited by the speed of the IO PMC, which is 66 Mhz. Transfers between the processor and IO PMC are limited to 66Mhz/32 bit. Transfers to the host through the PCI-X bridge are done at 66Mhz/64 bit on the local bus. The speed on the host bus is not restricted by the local bus speed.

2.2.2 Flash boot rom

Flash memory of 16 MB provides ample storage for firmware, configurations and simulation models. Upgrades to firmware in flash can be downloaded over PCI with a firmware update application.

2.2.3 Memory

Socketed SO-DIMM DDR memory of 128 MB is utilized to provide low cost memory with expansion capability.

2.2.4 SRAM

The 440GX cpu provides on-board 256K SRAM. Care has been taken to optimally utilize the SRAM for firmware and packet data.

2.2.5 CPU

The PowerPC 440GX cpu is a dual issue 667 Mhz 32K instruction cache, 32K data cache embedded processor. At only 4.5 Watts of power dissipation, the 440GX provides an ideal processor to pair with the IO PMC on a variety of form factors.

2.3 IO PMC

The IO PMC is a Xilinx FPGA based PMC with RS485 drivers for 32 channels. The 32 channels are accessible via SCSI II connector on the front panel of the PCI carrier. When isolation is necessary it is provided by a breakout panel connected to the card's SCSI II connector.

All 32 channels are configurable as either high speed (5Mbit) or low speed (400Kbit) channels.

2.3.1 PLL clock

A PLL clock drives the timing of all channels. This clock can be altered +/- 10% to allow variance of transmit speeds. When the PLL clock is changed, all transmit channels are affected.

2.3.2 PCI interface

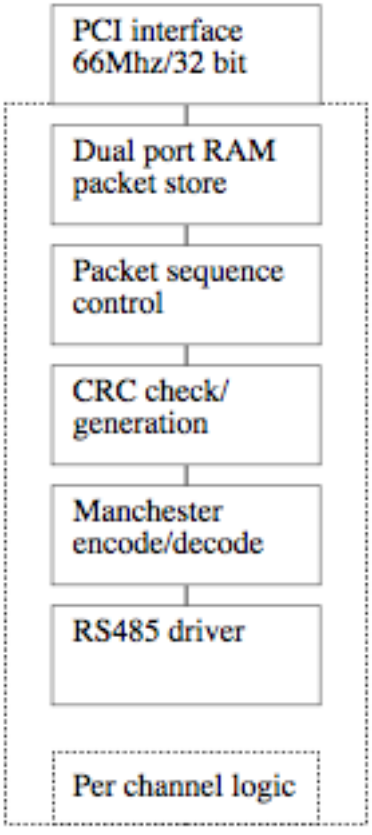
The PCI interface of the IO PMC is integrated in the Xilinx to optimize transfers to the Xilinx dual port RAMs. The interface runs at 66 Mhz 32 bit. P2P data in 16 bit words is always packed into 32 bit words to optimize access.

DMA of packet payload data to/from the IO module is done in burst mode under control of the IO PMC's DMA engine within the Xilinx.

2.3.3 Dual port RAM

Each channel has a dedicated dual port RAM of 2048 bytes. This amount of memory is sufficient to hold in excess of 3 milliseconds of transmit or receive data on a high speed channel. This exceeds the expected FCM transmit time of 1.4 milliseconds by over 100%.

For a bidirectional low speed channel (REU channel), the dual port RAM is divided into equal sized transmit and receive partitions.



2.3.4 Packet sequencer

The packet sequencer controls the back-to-back transmit or receive of packets for a single channel. Packets are stored consecutively in the dual port RAM to allow the maximum burst length of packets of at least 3 milliseconds. On transmit the sequencer controls the generation of truncated packets, packets with bad data or CRC, packets with truncated length, or missing postamble.

On reception the sequencer timestamps each packet with a microsecond clock, and saves error status on a per packet basis.

For error insertion purposes the packet sequencer is not limited to the 62 word payload size of the specification. It is possible to transmit and receive a single packet of up to 1018 payload words. In addition

truncated packets are possible, where transmission is terminated at any word following the packet preamble.

2.3.5 CRC generator/checker

On transmission the CRC generator can be deselected to allow the transmission of a user specified CRC, which may not match the transmitted data. On reception the CRC checker supplies the pass/fail status with the packet.

2.3.6 Manchester encoder/decoder

Manchester encoding as used in the P2P protocol defines a logic one as having the first half of the bit cell high and the second half low. The encoder runs at a nominal 5Mbit or 400Kbit rate depending on how the channel is configured. The encoder is driver by the PLL which can be altered to achieve rates deviating from the nominal.

The decoder uses an 16x over sampling technique to allow variance of the receive clock by approximately 12.5%.

2.3.7 RS485 drivers

The RS485 transceivers used are TI SN75HVD06. The IO board has software selectable termination. Alternately a breakout panel is available with jumperable termination, speed specific impedance matching, and circuit protection.

3 Software Architecture

The following diagram shows the main components of the P2P software.

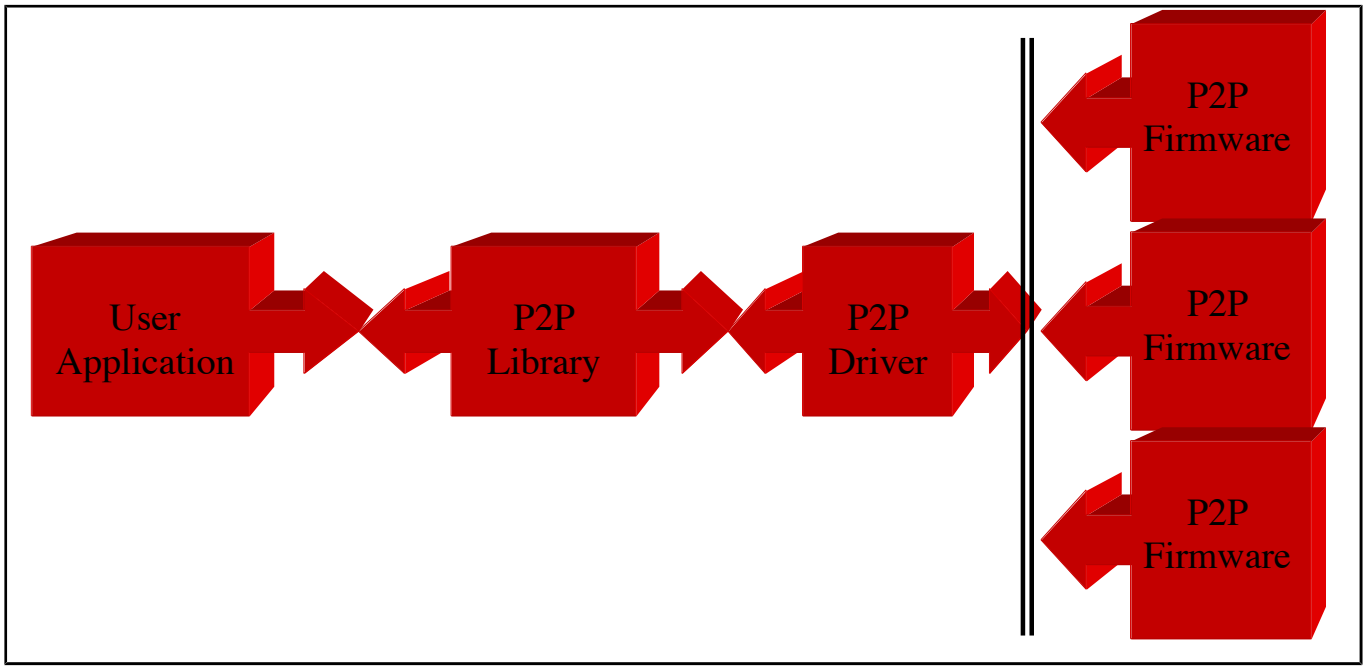


Figure 4 P2P Software structure

3.1 P2P Library

The P2P library consists of a set of functions through which application software accesses the P2P boards. The following functions are available:

- Opening, configuring, closing of P2P boards.
- Opening, configuring, closing of P2P channels.
- Send and receive of data frames.
- Reading statistic counters and board information.
- Debug functions.

3.2 P2P Driver

The driver is implemented as a kernel driver, or user driver depending on host system considerations. A kernel driver has the capability to control access to a board among multiple application processes. A user driver is limited for use by a single application process per board, but is more efficient as it does not have user space to kernel space context switching.

3.3 P2P Firmware

3.3.1 Channel capture

Packets received on channels are saved for reading by the API. Packets are time stamped with a microsecond counter, and checked for CRC integrity, Manchester encoding integrity, and preamble/postamble integrity.

3.3.2 REU simulation

The P2P resource is capable of simulating all REUs connected to one or more ACE LRUs. The ACE-REU channels are the only bidirectional channels of the FCE. A given ACE transmits packets on all it's REU transmit channels simultaneously. Within 100 microseconds all REU channels must transmit the reply packet back to the ACE. In addition the command word of the ACE to REU packet is transmitted back to the ACE in the reply packet.

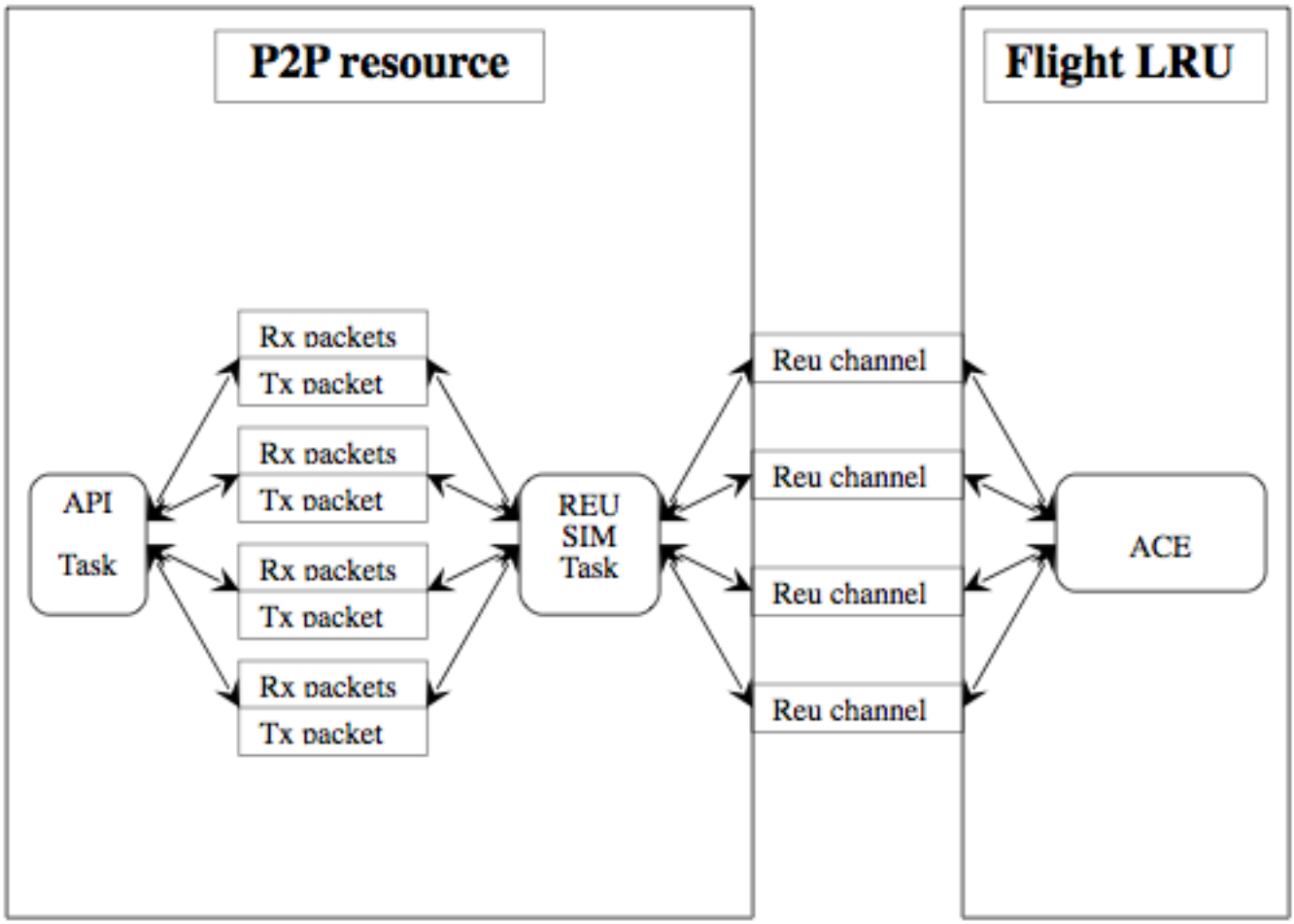


Figure 5 Firmware REU simulation

3.3.3 DMRS simulation

DMRS simulation is handled by a task on the P2P resource. This task is responsible for generating the packets transmitted to the ACE on the DMRS frame time. The most recent data supplied by the host is used to generate the transmit packet. If the host has not supplied new data the previously supplied data will be used to generate a new transmit packet.

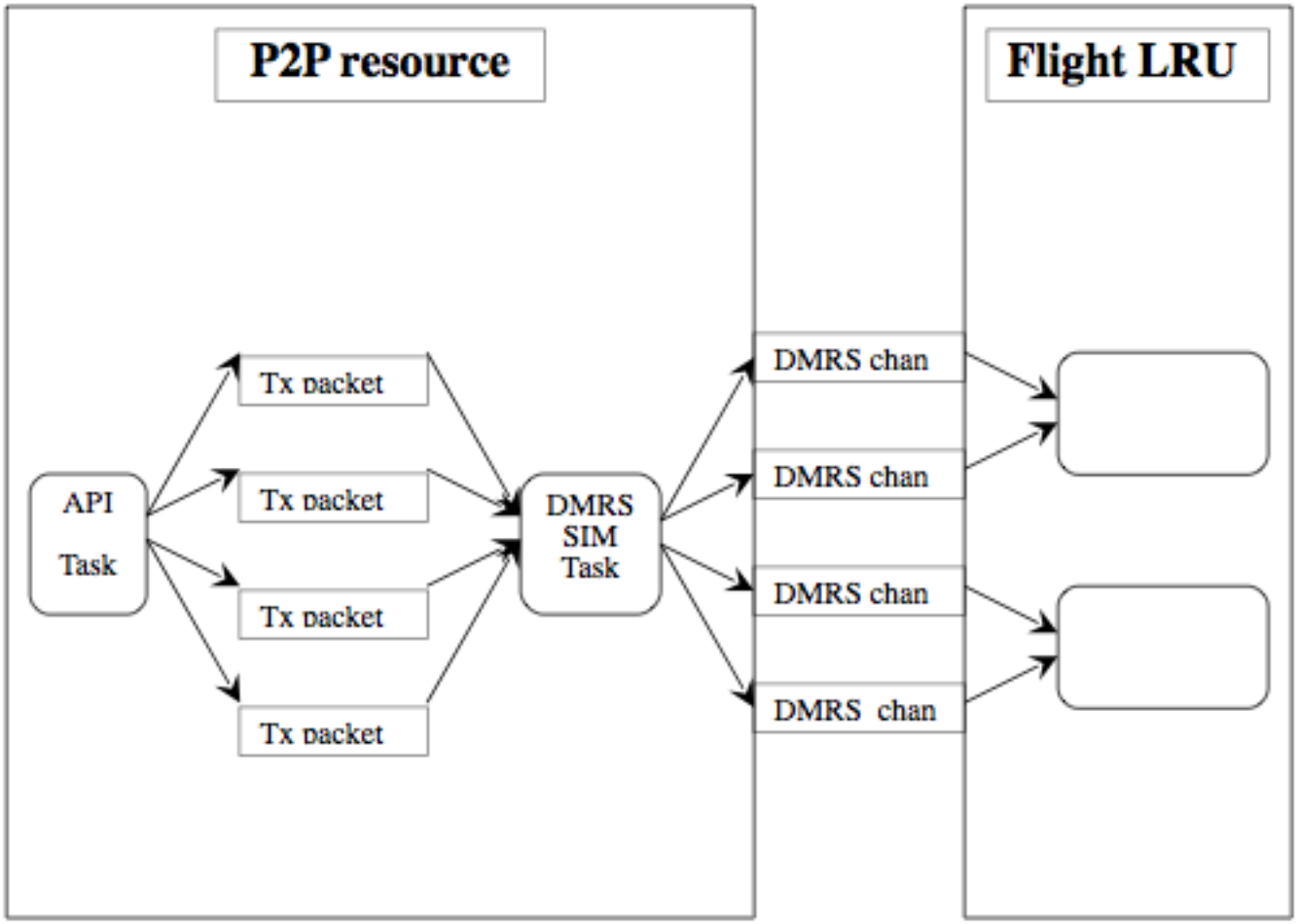


Figure 6 DMRS simulation

4 P2P Application Programming Interface

4.1 Board Control Functions

4.1.1 p2p_open

Synopsis

```
#include "p2p_lib.h"

p2p_hdl_t
p2p_open(int slot, unsigned options);
```

Description

This function establishes the connection to a P2P board identified by **slot** number. An application must call p2p_open with one of the **options** P2P_OPEN_BOARD, P2P_OPEN_BUFFERED, P2P_OPEN_READ_BUFFERED, P2P_OPEN_WRITE_BUFFERED before it can transmit and/or receive data. In addition any channels to be read or written must be opened and configured. P2P transmit and receive processing is stopped after a successful open. The buffered modes of open are used where efficiency of transfer is important. The non-buffered mode (P2P_OPEN_BOARD) is used when transmit latency is to be minimized, or when reading of individual channels is desired. Each board is limited to a single open for transmit and receive.

The handle returned by p2p_open is subsequently used to start/stop or configure the board or to open channels for configuring, reading or writing.

Other values of **options** exist on p2p_open but are not typically used by user applications. These other options are used by various p2p utility programs. Opening a board with the P2P_OPEN_CONSOLE option is always possible, regardless of an existing connection for transmit/receive. This option is used by the **p2p** utility to retrieve status information from a board. The P2P_OPEN_FLASH option is used by the **p2pload** utility to load firmware.

It is possible to reset the card upon open. Note no other applications should have the board open when a reset is done.

Parameters

slot numerical identifier for a P2P board. The interpretation of the parameter is platform dependent:

IRIX	the instance number of the P2P board starting with 1.
LINUX	the instance number of the P2P board starting with 1.
Windows	the instance number of the P2P board starting with 1.

options:

P2P_OPEN_BOARD	for standard P2P transmit/receive capability.
P2P_OPEN_READ_BUFFERED	for buffered read capability.
P2P_OPEN_WRITE_BUFFERED	for buffered write capability.

P2P_OPEN_BUFFERED for buffered read/write capability.
P2P_OPEN_ASYNC for asynchronous read capability.
P2P_OPEN_SCATTERGATHER for scatter/gather read/write capability.

The following options are normally used only by P2P utility applications.

P2P_OPEN_CONSOLE for monitoring status or console messages.
P2P_OPEN_FLASH for loading firmware.
P2P_OPEN_TAP for analyzer use.
P2P_OPEN_REPLAY for analyzer use.
P2P_OPEN_RESET Reset the board before opening.

Note: only a single open to each board is allowed in an application.

Returns

Handle for the board addressed by **slot** or P2P_CALL_FAILED in case of an error.

Errors

EINVAL:
slot or **options** is invalid
EBUSY:
The device with the given **slot** number is already open
ENODEV:
No operational device was found with the given **slot** number

4.1.2 p2p_close

Synopsis

```
#include "p2p_lib.h"

int
p2p_close(p2p_hdl_t hdl);
```

Description

This closes a connection to a board. For boards opened with P2P_OPEN_BOARD, the P2P scheduler stops transmit and receive processing. All channels opened on the board are closed.

Parameters

hdl value returned by p2p_open.

Returns

0 on success, P2P_CALL_FAILED on error.

Errors

EINVAL:
hdl does not indicate a board that was opened.

4.1.3 p2p_reset

Synopsis

#include <p2p_lib.h>

int
p2p_reset(int **lbn**);

Description

This function reset the board or channel identified by **lbn**.

Note, no other applications should have an active connection to the board when a reset is done.

Parameters

lbn logical board number 1-n.

Returns

0 on success, P2P_CALL_FAILED on failure

Errors

EINVAL:
lbn is invalid
ENODEV:
No operational device was found with the given **lbn** number

4.1.4 p2p_start

Synopsis

#include <p2p_lib.h>

int
p2p_start(p2p_hdl_t **hdl**, int **options**);

Description

This function starts the resource identified by **hdl**. This can be a board (**hdl** from p2p_open), channel (**hdl** from p2p_channel_open) or label (**hdl** from p2p_label_open).

Starting the board starts the transmit, receive and simulation processing. All channels should be opened and configured prior to p2p_start. The board can be written to before the p2p_start call. In this case the transmit processing will immediately start transmitting queued data. No receive data has been captured prior to the start call.

When the p2p_start function is called with a channel's handle, transmit, receive or simulation processing is started for that channel only. Note that if a channel has been stopped, and started, there can be data present in transmit and receive buffers. If this is not desired, include P2P_FLUSH in the options flags when starting. The board must also be started for the channel to start.

When the p2p_start function is called with a label's transmit, receive or simulation processing is started for that label only. The channel and board must also be started for the label to start.

Boards and channels have statistics counters associated with them. To reset these counters, include P2P_COUNTER_RESET in the options flags.

Parameters

hdl value returned by p2p_open or p2p_channel_open.

Options is a set of flags which can include:

P2P_RX_FLUSH	flush receive data in the board, or channel.
P2P_TX_FLUSH	flush transmit data in the board, or channel.
P2P_FLUSH	flush receive and transmit data in the board, or channel.
P2P_COUNTER_RESET	zero the statistics counters for the board or channel.

Returns

0 on success, P2P_CALL_FAILED on failure

Errors

EINVAL:

hdl is not a valid board or channel handle.

hdl is for a board (**hdl** is from p2p_open) and the board is already started.

4.1.5 p2p_stop

Synopsis

```
#include <p2p_lib.h>

int
p2p_stop(p2p_hdl_t hdl);
```

Description

This function applies to all object types, board, channel, label or data generation protocol identified by **hdl**. A board has **hdl** from p2p_open, a channel has **hdl** from p2p_channel_open.

If **hdl** is for a board the board stops transmit, receive and simulation processing. If **hdl** is for a channel, then transmit, receive or simulation processing is stopped only for that channel. Note that any currently scheduled packets will be transmitted after the stop call. Receive packets are dropped after a stop.

Parameters

hdl value returned by p2p_open or p2p_channel_open.

Returns

0 on success, P2P_CALL_FAILED on failure

Errors

EINVAL:

hdl is not a valid board or channel handle.
hdl is for a board (**hdl** is from p2p_open) and the board is already stoped.

4.1.6 p2p_time_config

Synopsis

#include <p2p_lib.h>

int
p2p_time_config (p2p_hdl_t **hdl**, unsigned long **secs**, unsigned long **usecs**);

Description

p2p_time_config sets a boards time clock to Greenwich time specified in seconds and microseconds. The board is identified by **hdl**, returned from p2p_open. Greenwich time consists of the number of seconds and microseconds since January 1 1970.

If absolute time is not desired, setting **secs** and **usecs** to zero, will zero the clock, such that times returned are relative to the p2p_time_config call.

If system time is desired, set **secs** and **usecs** to ~0. This will indicate that the driver should set the clock to system time and continue to keep the board clock in sync with system time.

Parameters

hdl	value returned by p2p_open.
secs	Greenwich time is secs plus usecs since January 1 1970.
usecs	Greenwich time is secs plus usecs since January 1 1970.

Returns

0 on success, P2P_CALL_FAILED on failure

Errors

EINVAL:
hdl is not a valid board handle.

4.1.7 p2p_xml_config

Synopsis

#include <p2p_lib.h>

int
p2p_xml_config (p2p_hdl_t **hdl**, char ***string**, int **length**);

Description

p2p_xml_config provides a generalized and extensible API for implementing controls on various object types. The xml controls are specific to the object handle supplied. That is some controls will only apply to board handles, while others may apply to channel, label, or other handles.

Parameters

hdl handle created by any of a variety of API calls.
string A character string in xml element format.
length the length of **string**.

Returns

0 on success, P2P_CALL_FAILED on failure
some calls may return a handle which is valid in other object handle APIs, such as p2p_stop/p2p_start.

Errors

EINVAL:
hdl is not a valid handle.

4.1.8 p2p_xml_file_config

Synopsis

#include <p2p_lib.h>

int
p2p_xml_file_config (p2p_hdl_t **hdl**, char ***filename**);

Description

p2p_xml_file_config provides a generalized and extensible API for implementing controls on various object types. The xml controls are specific to the object handle supplied. That is some controls will only apply to board handles, while others may apply to channel, label, or other handles. This API allows the configuration parameters to be passed via a file.

Parameters

hdl handle created by any of a variety of API calls.
filename A character string containing the name of the file containing config parameters.

Returns

0 on success, P2P_CALL_FAILED on failure
some calls may return a handle which is valid in other object handle APIs, such as p2p_stop/p2p_start.

Errors

EINVAL:
hdl is not a valid handle.

4.2 *General Functions*

4.2.1 p2p_fw_version

Synopsis

```
#include <p2p_lib.h>

const char *
p2p_fw_version(p2p_hdl_t hdl);
```

Description

This function returns the firmware version string. The current version described by this manual is: "The Goebel Company, P2P GXP1000 firmware Rev 0.1.0 <date> <time>".
Upgrades to firmware will increment the last number.

Parameters

hdl value returned by p2p_open.

Returns

A pointer to a constant character buffer containing the version string, P2P_CALL_FAILED on failure.

Errors

EINVAL
hdl is not a valid handle returned from p2p_open

4.2.2 p2p_lib_version

Synopsis

```
#include <p2p_lib.h>

const char *
p2p_lib_version();
```

Description

This function retrieves the P2P interface library version. The current version described by this manual is: "P2P library rev 1.0 Copyright The Goebel Company 2005"

Parameters

None

Returns

A pointer to a constant character buffer containing the version string.

Errors

None

4.3 P2P channel functions

4.3.1 p2p_channel_open

Synopsis

#include <p2p_lib.h>

p2p_hdl_t
p2p_channel_open(p2p_hdl_t **hdl**, int **channel**);

Description

This function opens a P2P channel on the board indicated by **hdl**. The channel number is indicated by **channel**. Once the channel is opened, the type of channel must be configured by a call to one of the functions p2p_XXX_channel_config. This configures the channel as type XXX.

Parameters

hdl	value returned by p2p_open.
channel	tchannel number from 1 to 32.

Returns

A channel handle, which is used to identify the channel in all other channel related functions, P2P_CALL_FAILED on failure.

Errors

EINVAL
hdl is not a valid handle returned from p2p_open

EBUSY
The **channel** is already opened.

4.3.2 p2p_reu_channel_config

Synopsis

#include <p2p_lib.h>

int
p2p_reu_channel_config(p2p_hdl_t **hdl,** int **delay,** int **mode**);

Description

This function configures a channel as an REU simulation channel. This channel is the only bidirectional variant of FCE channel. An REU simulation channel waits for an ACE command packet to arrive. When the ACE packet is received, a reply packet is generated and transmitted back on the same channel within 100 microseconds.

The response packet generated is dependent on REU type. The REU type is specified with the mode parameter, ie. P2P_MODE_PRIMARY_REU, P2P_MODE_HYDRAULIC_SPOILER_REU, P2P_MODE_EMCU_REU, or P2P_MODE_HIGH_LIFT_EMCU_REU. If an REU type is not

specified P2P_MODE_PRIMARY_REU is the default. If an REU response is desired other than one of the standard types listed, the REU response can be configured with p2p_reu_wrap_config. This command can configure the wrap of ACE command packet data to the REU response in a customized way. To see the details of the response protocol consult the Honeywell P2P specifications.

The response packet is only generated if at least one packet is written to the channel. When new packets are written, these will be used in place of the previous packet. A packet is reused as a response packet until a new one is written to the channel.

The packet transmitted in response to the ACE packet is that which is written to the channel, with the exception of the command wrap and frame count fields. These fields are nominally supplied by the P2P resource. The correct values normally provided by the P2P resource can be overridden by selecting the appropriate error injection flags to p2p_channel_write_hdr.

The **delay** parameter should only be set non-zero for error injection cases. This will hold off the reply packet to the ACE for the number of microseconds specified by the **delay** value.

The **mode** parameter can be used to configure the channel in an initially stopped state. In this case **p2p_start** must be used to start the channel.

Parameters

hdl value returned by p2p_channel_open.

delay delay in microseconds for reply packet sent to ACE.

mode Use P2P_MODE_STOPPED to configure the channel in stopped state.
Use one of:
P2P_MODE_PRIMARY_REU
P2P_MODE_HYDRAULIC_SPOILER_REU
P2P_MODE_EMCU_REU
P2P_MODE_HIGH_LIFT_EMCU_REU
P2P_MODE_TYPE2_REU
P2P_MODE_TYPE3_REU
If REU type is not specified:
P2P_MODE_PRIMARY_REU is the default.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Error

EINVAL
The channel is not stopped or **hdl** is not an opened channel.

4.3.3 p2p_reu_config

Synopsis

#include <p2p_lib.h>

int
p2p_reu_config(p2p_hdl_t **hdl,** int **delay,** int **packets,** int **mode**);

Description

This function adds the packet parameter to p2p_reu_channel_config. Use this function with the **packets** parameter to increase the queue size to hold additional packets to avoid drops when reads may not be services regularly.

4.3.4 p2p_reu_wrap_config

This call allows the customization of the individual REU response components. The individual components are specified by up to six integer values, which are generated with supplied macros. Each component specifies one field of the response packet and how this field is generated. The basic macro used to specify the response fields is REU_WRAP, defined below.

REU_WRAP(source, destination, mask)

Where source is one of

N	A number 0-N indicating the payload index of the source word.
REU_INVERT(n)	The payload index of a source word to be inverted.
REU_ACTIVITY	The source word is taken from an increasing frame counter for the channel.
REU_RANDOM	The source word is a random number.

Where destination is one of

N	A number 0-N indicating the payload index of the destination word.
REU_OR_IN(n)	Indicates the value is Ored into the destination word.

Where mask is anded with the source word and the result transferred to the destination word.

Example:

p2p_reu_wrap_config(channel_handle, wrap1, wrap2, wrap3, wrap4, wrap5, wrap6);

```
int     wrap1=REU_WRAP(0,0,0xfff0);
int     wrap2=REU_WRAP(REU_ACTIVITY,REU_OR_IN(0),0xf);
int     wrap3=REU_WRAP(1,1,0xffff);
int     wrap4=REU_WRAP(REU_INVERT(3),7,0xffff);
int     wrap5=REU_WRAP(REU_INVERT(3),8,0xfff0);
int     wrap6=REU_WRAP(REU_RANDOM,REU_OR_IN(8),0xf);
```

REU_WRAP(0,0,0xfff0) transfers upper 12 bits of ACE command payload word 0 to REU response payload word 0.

REU_WRAP(REU_ACTIVITY,REU_OR_IN(0),0xf) Ores in lower 4 bits of frame counter to REU response word 0.

REU_WRAP(1,1,0xffff) transfers ACE command payload word 1 to REU response payload word 1.

REU_WRAP(REU_INVERT(3),7,0xffff) transfers ACE command payload word 3 inverted to REU response payload word 7.

REU_WRAP(REU_INVERT(3),8,0xfff0) transfers upper 12 bits of ACE command payload word 3 inverted to REU response payload word 8.

REU_WRAP(REU_RANDOM,REU_OR_IN(8),0xf) Ores in lower 4 bits of a random number to REU response payload word 8.

4.3.5 p2p_ace_reu_channel_config

Synopsis

#include <p2p_lib.h>

int
p2p_reu_channel_config(p2p_hdl_t **hdl,** int **frame_time,** int **mode**);

Description

This function configures a channel as an ACE to REU simulation channel. This channel is the only bidirectional variant of FCE channel. An ACE to REU simulation channel supplies the ACE command packet to the REU. When the REU receives the ACE packet, a reply is generated and transmitted back on the same channel within 100 microseconds. The command word received in the ACE packet is wrapped back to provide an integrity check. The ACE to REU packets are generated on a periodic **frame_time**. If no new packet is supplied by the API, the previous packet is retransmitted with updated frame counter.

The packet transmitted in response to the ACE packet is that which is written to the channel, with the exception of the command wrap and frame count fields. These fields are nominally supplied by the P2P resource. The correct values normally provided by the P2P resource can be overridden by selecting the appropriate error injection flags to p2p_channel_write_hdr.

The **mode** parameter can be used to configure the channel in an initially stopped state. In this case **p2p_start** must be used to start the channel.

Parameters

hdl	value returned by p2p_channel_open.
Frame_time	frame time in microseconds for ACE packets to be sent.
mode	Use P2P_MODE_STOPPED to configure the channel in stopped state.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Error

EINVAL
The channel is not stopped or **hdl** is not an opened channel.

4.3.6 p2p_dmrs_channel_config

Synopsis

#include <p2p_lib.h>

int
p2p_dmrs_channel_config(p2p_hdl_t **hdl,** int **frame_time,** int **mode**);

Description

This function configures a channel as a DMRS simulation channel. This channel transmits to the ACE every **frame_time** microseconds. The packet activity counter in the fourth payload word low 5 bits is automatically configured when using this call.

The **frame_time** parameter can be set to zero to get the default DMRS frame time. Otherwise set to the number of microseconds to select.

The **mode** parameter can be used to configure the channel in an initially stopped state. In this case **p2p_start** must be used to start the channel.

Parameters

hdl	value returned by p2p_channel_open.
frame_time	frame time in microseconds for packet sent to ACE.
mode	Use P2P_MODE_STOPPED to configure the channel in stopped state.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Error

EINVAL
The channel is not stopped or **hdl** is not an opened channel.

4.3.7 p2p_ace_dmrs_channel_config

Synopsis

#include <p2p_lib.h>

int
p2p_dmrs_channel_config(p2p_hdl_t **hdl,** int **mode**);

Description

This function configures a channel as the ACE receiver of a DMRS channel. The DMRS transmits a packet on a periodic basis.

The **mode** parameter can be used to configure the channel in an initially stopped state. In this case **p2p_start** must be used to start the channel.

Parameters

hdl value returned by p2p_channel_open.
mode Use P2P_MODE_STOPPED to configure the channel in stopped state.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Error

EINVAL
The channel is not stopped or **hdl** is not an opened channel.

4.3.8 p2p_rx_channel_config

Synopsis

#include <p2p_lib.h>

int
p2p_rx_channel_config(p2p_hdl_t **hdl,** int **speed**, unsigned **max_packets**, int **mode**);

Description

This function configures a channel as a receive channel. The **speed** parameter selects a high or low speed channel. Buffer size on the board is selected by the **max_packets** parameter.

Parameters

hdl value returned by p2p_channel_open.

speed speed for the channel (P2P_SPEED_400K or P2P_SPEED_5M).
This is the actual speed in Hz.

max_packets number of packets that will be buffered for the channel.

mode Use P2P_MODE_STOPPED to configure the channel in stopped state.
Use P2P_MODE_SAMPLING to read only the most recent packet.
Use P2P_MODE_QUEUING to read all packets received.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Error

EINVAL
The channel is not stopped or **hdl** is not an opened channel.

4.3.9 p2p_tx_channel_config

Synopsis

#include <p2p_lib.h>

int
p2p_tx_channel_config(p2p_hdl_t **hdl,** int **speed**, unsigned **max_packets**, unsigned **frame_time**, int **mode**);

Description

This function configures a channel as a transmit channel. The **speed** parameter selects a high or low speed channel. Buffer size on the board is selected by the **max_packets** parameter. If the channel is to transmit periodically with the most recent data, the **frame_time** is used to define the periodic rate.

Parameters

hdl	value returned by p2p_channel_open.
speed	speed for the channel (P2P_SPEED_400K or P2P_SPEED_5M).
max_packets	is the number of packets that will be buffered for the channel.
Frame_time	the frame time in microseconds if a periodic transmit is desired.
mode	Use P2P_MODE_STOPPED to configure the channel in stopped state. Use P2P_MODE_SAMPLING to transmit the most recently written packet. Use P2P_MODE_QUEUING to transmit all packets written. Use P2P_MODE_SQUEUING to transmit all packets written, and retransmit the last packet written until a new packet is available. Use P2P_MODE_RQUEUING when it is desired to loop on transmitting the entire queue. When the end of queue is reached, it is reset to the beginning.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Error

EINVAL
The channel is not stopped or **hdl** is not an opened channel.

4.3.10 p2p_tx_frame_time

Synopsis

#include <p2p_lib.h>

int
p2p_tx_frame_time(p2p_hdl_t **hdl,** unsigned **frame_time**);

Description

This function allows the channel frame time to be dynamically changed. The **frame_time** parameter is the microsecond rate used to define the channel transmit frame period. All labels on the channel are some multiple of the channel rate. That is a label may be transmitted every channel **frame_time**, or some multiple of the channel **fame_time**.

Parameters

frame_time	the frame time in microseconds of the channel transmit.

Error

EINVAL
The **hdl** is not an opened transmit channel.

4.3.11 p2p_label_open

Synopsis

```
#include <p2p_lib.h>

p2p_hdl_t
p2p_label_open(p2p_hdl_t hdl, int ordinal);
```

Description

This function associates a handle with a label on the channel indicated by **hdl**. For transmit channels, the label ordinal is used to determine the transmit order. An ordinal value of P2P_ORDINAL_NEXT will allocate the next available ordinal. Once the label is opened, it must be configured by a call to p2p_tx_label_config or p2p_rx_label_config. This initializes the queue and queuing mode associated with the label.

Parameters

hdl	value returned by p2p_channel_open.
ordinal	either **P2P_ORDINAL_NEXT** or a number from 1 to 31 indicating the order in the transmit sequence.

Returns

A label handle, which is used to identify the label in all other label related functions, P2P_CALL_FAILED on failure.

Errors

EINVAL
hdl is not a valid handle returned from p2p_open
The channel is already opened.

4.3.12 p2p_tx_label_config

Synopsis

```
#include  <p2p_lib.h>

int
p2p_tx_label_config(p2p_hdl_t  hdl, unsigned max_packets, unsigned max_size, int mode);

<deprecated>
int
p2p_sub_channel_config(p2p_hdl_t  hdl, unsigned sub_channel, unsigned max_packets, unsigned
max_size, int mode);
```

Description

This function is used to configure a label within a channel. Creating labels within a transmit channel allow multiple labels to be sent per frame. Each label to be transmitted must be configured with this call. Each label has it's own queue and position in the transmit sequence. The position in the transmit sequence is defined by **ordinal** when the label is opened with **p2p_label_open**.

Tx labels are opened and configured under a channel. The transmit channel is identified by **hdl**. The label's ordinal is a number between 1 and 31. Ordinal 0 has already been configured when the transmit channel is configured.

Each label is transmitted in ordinal order according to the following rules.

1. The label must be written with a packet to transmit.
   1a. If the label is in sampling mode, the most recent packet is transmitted.
   1b. If the label is in queuing mode, the next packet in that label's queue is transmitted.
2. If the label has a scheduling rule configured, an evaluation is made as to determine if the label should be scheduled in this frame. The appropriate rule from 1a or b is used to determine if and which packet is transmitted.

Once the last of the labels is evaluated, the channel is done transmitting until the next frame, at which time the labels are reevaluated, starting with label ordinal zero.

The maximum number of packets which can be queued in for a label is selected by the **max_packets** parameter. If this parameter is left zero, a default value will be selected.

The maximum size of packets which can be written to the label is selected by the **max_size** parameter. If this parameter is left zero, a default value will be selected that allows maximal size packets to be transmitted. If larger than maximal size packets are desired, or space optimization is important use this parameter to select the maximum sized packet to transmit.

Use the **mode** parameter to select queuing or sampling mode for the label.

Parameters

hdl	value returned by p2p_channel_open and donfigured with p2p_tx_channel_config.
max_packets	number of packets that will be buffered for the channel.
max_size	maximum size of a packet to be written to the channel.
mode	Use P2P_MODE_SAMPLING to transmit the most recently written packet. Use P2P_MODE_QUEUING to transmit all packets written. Use P2P_MODE_SQUEUING to transmit all packets written, and retransmit the last packet written until a new packet is available.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Error

EINVAL

The channel is not stopped or **hdl** is not an opened channel.

4.3.13 p2p_tx_label_schedule

Synopsis

#include <p2p_lib.h>

int
p2p_tx_label_schedule(p2p_hdl_t **hdl**, unsigned **frame_divisor**, unsigned **frame_offset**);

Description

This function is used to schedule a label which is not transmitted every frame. Using this function it is possible to control the scheduling of different transmit labels within a scheduling slot. To schedule a label every other frame use a **frame_divisor** of 2, every third frame a **frame_divisor** of 3, etc. When N labels are to
occupy the same scheduling slot the **frame_divisor** for each frame is set to N, and each label is given a unique **frame_offset** from 0 to N-1.

This function replaces the round-robin concept of the previous version of API. This function allows the flexibility to have scheduling slots that transmit a single label from a group of labels at less than the frame rate. For example to transmit one of two 25 hz labels in a given scheduling slot of a 100 hz frame, configure each label as follows:

```
label1_hdl    = p2p_label_open(channel_hdl, 1);
status   = p2p_tx_label_config(channel_hdl, max_packets, max_size, mode);
istatus  = p2p_tx_label_schedule(label1_hdl, 4, 0);

label2_hdl    = p2p_label_open(channel_hdl, 2);
status   = p2p_tx_label_config(channel_hdl, max_packets, max_size, mode);
istatus  = p2p_tx_label_schedule(label2_hdl, 4, 1);
```

This example would transmit label1 in frame 1, label2 in frame 2, then no label would be transmitted in frames 3 and 4, then the sequence would repeat starting at label1.

Parameters

hdl	value returned by p2p_label_open.
frame_divisor	A divisor of N schedules the label every N frames.
frame_offset	offset from the start of the scheduling group.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Error

EINVAL
The channel is not stopped or **hdl** is not an opened channel.

4.3.14 p2p_rx_label_config

Synopsis

#include <p2p_lib.h>

int
p2p_rx_label_config(p2p_hdl_t **hdl,** unsigned **label**, unsigned **max_packets**, unsigned **max_size**, int **mode**);

Description

This function is used in two cases. First, if only selected labels are desired, then this call is used to filter the desired labels. When this function is not used a receive channel will receive all labels in a single receive buffer which is read via the channel's handle. When this function is used, label specific buffers are configured and are available to be read with the handle returned from this call. Labels received, which are not configured are routed to the buffer associated with the channel handle. If this handle is not read, the messages will be dropped. It is suggested in this case the channel be configured in sampling mode, such that the messages dropped do not increment error counters.

Second, if the gather method of reading data is used, then this function configures receive labels which can be selected in the gather operation. Gather data is found by the label's handle passed to p2p_channel_gather_config. Configure the labels in sampling mode, since the labels will never be read, only the gather payload words are read.

The maximum number of packets which can be queued is selected by the **max_packets** parameter. If this parameter is left zero, a default value will be selected. Sampling mode implies no queuing of packets, in which case **max_packets** is not meaningful.

The maximum size of packets which can be received is selected by the **max_size** parameter. If this parameter is left zero, a default value will be selected that allows maximal size packets to be transmitted. If larger than maximal size packets are desired, or space optimization is important use this parameter to select the maximum sized packet to transmit.

Use the **mode** parameter to select queuing or sampling mode for the label.

Parameters

hdl	value returned by p2p_channel_open and configured for receive.
max_packets	number of packets that will be buffered for the channel.
max_size	maximum size in bytes of a packet to be received by the channel.
mode	Use P2P_MODE_SAMPLING to receive the most recent packet. Use P2P_MODE_QUEUING to receive all packets.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Error

EINVAL
The channel is not stopped or **hdl** is not an opened channel.

4.3.15 p2p_scatter_config

Synopsis

#include <p2p_lib.h>

p2p_hdl_t
p2p_scatter_config (p2p_hdl_t **hdl**, unsigned **word**, unsigned **index**, unsigned **ic**);

Description

This call associates a specific **word** in a transmit label, with a given **index** in the scatter buffer. Whenever the label is transmitted, the current value of the scatter buffer word is used to update the payload word within the packet.

The use of a scatter buffer allows the efficient update of selected variables in a number of transmit channel packet payloads. The scatter buffer is a buffer of 16 bit words which is written to the board. The board then performs a scatter operation to place each 16 bit variable in it's appropriate packet and payload position. The payload position is noted by **word** and the scatter buffer index is noted by **index**. The initial value of the variable can be specified by **ic**. The definition of payload positions to fill is done by successive calls to **p2p_scatter_config**. The configuration of the scatter buffer is done once prior to starting a board. Once configured, a single write is all that is required during a frame to update all packet variables.

Parameters

hdl	The handle of the label containing the transmit packet.
word	Word index into payload.
index	The index into the scatter buffer which is an array of short.
ic	Initial value of the payload word.

Returns

A handle on success, P2P_CALL_FAILED on failure

Errors

EINVAL:
hdl is not a valid channel handle.
ENOMEM:
Not enough resources to add scatter config.

4.3.16 p2p_gather_config

Synopsis

#include <p2p_lib.h>

p2p_hdl_t
p2p_gather_config (p2p_hdl_t **hdl**, unsigned **word**, unsigned **index**, unsigned **ic**);

Description

This call associates a specific **word** in a receive label, with a given **index** in the gather buffer. Whenever the label is received, the payload word within the packet is used to update the value of the gather buffer word.

The use of a gather buffer allows the efficient reading of the most current value of selected variables gathered from a number of receive label packet payloads. As each packet is received the payload words are transferred to the gather buffer according to the definition of gather payload words defined by successive calls to **p2p_gather_config**. The gather buffer can be read at any time by a call to p2p_gather. The configuration of the gather buffer is done once prior to start of board operation.

Parameters

hdl	The handle of the label containing the receive packet.
word	Word index into payload.
index	The index into the scatter buffer which is an array of short.
ic	Initial value of the payload word.

Returns

A handle on success, P2P_CALL_FAILED on failure

Errors

EINVAL:
hdl is not a valid board handle.
Board is not stopped.

4.3.17 p2p_start

See the description under Board Control Functions

4.3.18 p2p_stop

See the description under Board Control Functions

4.3.19 p2p_channel_close

Synopsis

```
#include <p2p_lib.h>

int
p2p_channel_close(p2p_hdl_t hdl);
```

Description

This function closes the P2P channel identified by **hdl**. All resources allocated for the channel will be freed and all data queued for the channel is discarded.

Parameters

hdl is the value returned by p2p_channel_open.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno will contain the error code.

Error

EINVAL

The **hdl** is not for an opened channel.

4.4 Data Generation Functions

Data generation functions can be applied to any transmit label. Transmit labels are specified by a label handle as returned from p2p_label_open or p2p_channel_open.

Data generation functions are applied in the order that the config calls are made. This is particularly important for functions which rely on other data in the packet to complete their computation. For example, a CRC spanning the entire packet would be configured last so that all data within the packet is generated prior to the CRC.

Note that data generation functions return a handle. This handle is for use in calling the p2p_channel_pause_config function.

4.4.1 p2p_channel_activity_config

Synopsis

#include <p2p_lib.h>

p2p_hdl_t
p2p_channel_activity_config(p2p_hdl_t **hdl,** unsigned **word**, unsigned **count,** unsigned **mask**);

Description

This function causes an activity count to be generated prior to transmitting a packet. The activity count starts with the value **count** and increments until it reaches mask and then rolls over to zero. The payload word indexed by **word** is where the 16 bit frame count is inserted.

Parameters

hdl	either a channel or label handle.
word	the index of the payload word to receive the crc.
count	the initial value of the frame count.
mask	the bits which are inserted into word.

Returns

On success a handle is returned, on error P2P_CALL_FAILED is returned and errno contains the error code. The handle returned is for use with p2p_channel_pause_config.

Error

EINVAL
The channel is not stopped or **hdl** is not an opened channel.
ENOMEM
There are not sufficient resources for this data generation function.

4.4.2 p2p_channel_copy_config

Synopsis

#include <p2p_lib.h>
p2p_hdl_t
p2p_channel_copy_config(p2p_hdl_t **dst_hdl,** p2p_hdl_t **src_hdl,** unsigned **dst**, unsigned **src**, unsigned **words**, unsigned **mask**);

Description

This function copies data from another label into the transmit packet. The label to be copied from is specified by **src_hdl**. This is the handle returned by p2p_channel_open or p2p_label_open. The data is copied from the most recent packet of the source label. Data is copied from source packet payload word **src** to destination packet payload word **dst** for **words** payload words. If only a portion of a word is to be copied, **mask** can be used to specify the bits to copy.

Parameters

dst_hdl a channel or label handle to copy to.
src_hdl a channel or label handle to copy from.
dst starting payload word (0-n) to receive data.
src starting payload word (0-n) of source data.
words number of payload word to copy.
mask bits to copy if less than a whole word.

Returns

On success a handle is returned, on error P2P_CALL_FAILED is returned and errno contains the error code. The handle returned is for use with p2p_channel_pause_config.

Error

EINVAL
dst_hdl is not an opened channel or label.
dst_hdl is not an opened channel or label.
ENOMEM
There are not sufficient resources for this data generation function.

4.4.3 p2p_channel_crc16_config

Synopsis

```
#include  <p2p_lib.h>

p2p_hdl_t
p2p_channel_crc16_config(p2p_hdl_t  hdl, unsigned word, unsigned first);
```

Description

This function causes a 16 bit crc to be generated prior to transmitting a packet. The crc computation starts at the payload word indexed by **first** and includes payload words up to the payload word indexed by **word** which is where the 16 bit crc is inserted.

Parameters

hdl either a channel or label handle.
word index of the payload word to receive the crc.
first index of the payload word where the computation starts.

Returns

On success a handle is returned, on error P2P_CALL_FAILED is returned and errno contains the error code. The handle returned is for use with p2p_channel_pause_config.

Error

EINVAL
The channel is not stopped or **hdl** is not an opened channel.
ENOMEM
There are not sufficient resources for this data generation function.

4.4.4 p2p_channel_crc32_config

Synopsis

#include <p2p_lib.h>

p2p_hdl_t
p2p_channel_crc32_config(p2p_hdl_t **hdl,** unsigned **word**, unsigned **first**);

Description

This function causes a 32 bit crc to be generated prior to transmitting a packet. The crc computation starts at the payload word indexed by **first** and includes payload words up to the payload word indexed by **word.** The 32 bit crc is inserted at payload words indexed by **word** and **word+1**.

Parameters

hdl either a channel or label handle.
word the index of the payload word to receive the crc.
first the index of the payload word where the computation starts.

Returns

On success a handle is returned, on error P2P_CALL_FAILED is returned and errno contains the error code. The handle returned is for use with p2p_channel_pause_config.

Error

EINVAL
The channel is not stopped or **hdl** is not an opened channel.
ENOMEM
There are not sufficient resources for this data generation function.

4.4.5 p2p_channel_diffcount_config

Synopsis

#include <p2p_lib.h>
p2p_hdl_t
p2p_channel_diffcount_config(p2p_hdl_t **hdl,** unsigned **src**, unsigned **words**, unsigned **word**, unsigned **mask**);

Description

This function checks a range of payload words for a difference since the last transmit. When a difference is detected a counter in payload word **word** is incremented. Data is checked from packet payload word **src** for **words** payload words. If the counter field is a subset of the word the bits of the counter field are specified by **mask**.

Parameters

hdl a channel or label handle to copy to perform diffcount on.
src starting payload word (0-n) of data to check for difference.
words number of payload word to check for difference (currently limited to 2).
word payload word number (0-n) of difference counter.
mask bits to copy if less than a whole word.

Returns

On success a handle is returned, on error P2P_CALL_FAILED is returned and errno contains the error code. The handle returned is for use with p2p_channel_pause_config.

Error

EINVAL

hdl is not a valid channel or label.

words is less than 1 or grreater than 2.

ENOMEM

There are not sufficient resources for this data generation function.

4.4.6 p2p_channel_framecount_config

Synopsis

#include <p2p_lib.h>

p2p_hdl_t
p2p_channel_framecount_config(p2p_hdl_t **hdl,** unsigned **word**, unsigned **count**);

Description

This function causes a 16 bit frame count to be generated prior to transmitting a packet. The frame count starts with the value **count** and increments until it reaches 0xffff an then rolls over to zero. The payload word indexed by **word** is where the 16 bit frame count is inserted.

Parameters

hdl either a channel or label handle.
word the index of the payload word to receive the crc.
count the initial value of the frame count.

Returns

On success a handle is returned, on error P2P_CALL_FAILED is returned and errno contains the error code. The handle returned is for use with p2p_channel_pause_config.

Error

EINVAL

The channel is not stopped or **hdl** is not an opened channel.

ENOMEM

There are not sufficient resources for this data generation function.

4.4.7 p2p_channel_frameinc_config

Synopsis

#include <p2p_lib.h>
p2p_hdl_t
p2p_channel_framecount_config(p2p_hdl_t **hdl,** unsigned **word**, unsigned **count,** unsigned **mask,** int **increment**);

Description

This function causes a 16 bit frame count to be generated prior to transmitting a packet. The frame count starts with the value **count** and increments by **increment** until it reaches 0xffff an then rolls over to zero. The payload word indexed by **word** is where the 16 bit frame count is inserted. The

frame count can be limited to a portion of the word by specifying a **mask** value other than 0xffff. A **mask** value of zero implies the entire word is used for the frame count.

Parameters

hdl either a channel or label handle.
word the index of the payload word to receive the crc.
count the initial value of the frame count.
mask 1 bits are set for the bits in the word to contain the frame count.
Increment the value to increment the frame count by.

Returns

On success a handle is returned, on error P2P_CALL_FAILED is returned and errno contains the error code. The handle returned is for use with p2p_channel_pause_config.

Error

EINVAL
The channel is not stopped or **hdl** is not an opened channel.
ENOMEM
There are not sufficient resources for this data generation function.

4.4.8 p2p_channel_framelimit_config

Synopsis

```
#include <p2p_lib.h>

p2p_hdl_t
p2p_channel_framelimit_config(p2p_hdl_t hdl, unsigned limit);
```

Description

This function causes a channel to stop transmitting after limit **frames**.

Parameters

hdl either a channel or label handle.
limit number of frames to transmit.

Returns

On success a handle is returned, on error P2P_CALL_FAILED is returned and errno contains the error code. The handle returned is for use with p2p_channel_pause_config.

Error

EINVAL
The channel is not stopped or **hdl** is not an opened channel.
ENOMEM
There are not sufficient resources for this data generation function.

4.4.9 p2p_channel_heartbeat_config

Synopsis

```
#include <p2p_lib.h>

p2p_hdl_t
```

p2p_channel_heartbeat_config(p2p_hdl_t **hdl,** unsigned **word**, unsigned **fc_word**, unsigned **label_word**, unsigned **lane**);

Description

This function causes a 16 bit heartbeat to be generated prior to transmitting a packet. The heartbeat computation must know where the frame count and label words reside in the packet. The payload index of these words is specified by **fc_word** and **label_word**. The lane parameter is zero for MON, and one for COM. The payload word indexed by **word** is where the heartbeat is inserted.

Parameters

hdl	either a channel or label handle.
word	index of the payload word to receive the heartbeat.
fc_word	payload index of the frame count.
label_word	payload index of the application label.
lane	zero for MON, and one for COM.

Returns

On success a handle is returned, on error P2P_CALL_FAILED is returned and errno contains the error code. The handle returned is for use with p2p_channel_pause_config.

Error

EINVAL
The channel is not stopped or **hdl** is not an opened channel.
ENOMEM
There are not sufficient resources for this data generation function.

4.4.10 p2p_channel_pattern_config

Synopsis

#include <p2p_lib.h>

p2p_hdl_t
p2p_channel_pattern_config(p2p_hdl_t **hdl,** unsigned **pattern**);

Description

This function fills the entire payload with the specified pattern prior to transmitting a packet. If other data generation functions are desired they should always follow this one, as the data generation happens in the order of the config calls and this call causes the entire payload to be generated.

Parameters

hdl either a channel or label handle.
pattern is one of the following.

P2P_PATTERN_ZERO	pattern is zeros for all payload words.
P2P_PATTERN_ONES	pattern is ones for all payload words.
P2P_PATTERN_2525	pattern is 0x2525 for all payload words.
P2P_PATTERN_INC2	pattern is incrementing payload words.
P2P_PATTERN_INC4	pattern is incrementing long payload words.

P2P_PATTERN_5252 pattern is 0x5252 for all payload words.

Returns

On success a handle is returned, on error P2P_CALL_FAILED is returned and errno contains the error code. The handle returned is for use with p2p_channel_pause_config.

Error

EINVAL
The channel is not stopped or **hdl** is not an opened channel.
ENOMEM
There are not sufficient resources for this data generation function.

4.4.11 p2p_channel_sawtooth_config

Synopsis

#include <p2p_lib.h>

p2p_hdl_t
p2p_channel_sawtooth_config(p2p_hdl_t **hdl**, unsigned **word**, unsigned **period,** unsigned **delay,** int **bias,** unsigned **amplitude**);

Description

This function generates a payload word following a sawtooth pattern. The data pattern repeats over period microseconds. This can be a period of up to 1,000 seconds (1,000,000,000 microseconds). The starting phase offset is set with delay. Delay is also in microseconds and must be less than period. The data values have a peak-to-peak range specified by amplitude. Maximum amplitude is 0xfff8. An amplitude of 0xfff8 and bias of zero would result in values ranging from –0x7ffc to +0x7ffc. A non-zero bias will reduce the maximum amplitude allowed by 2*bias. The nominal starting value of the sawtooth wave is zero with a rising slope.

Parameters

hdl	either a channel or label handle.
word	index of the payload word to receive the data.
period	time in microseconds that the pattern repeats.
delay	the stating phase offset time in microseconds.
bias	value added to the nominal.
amplitude	the maximal range of values.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Error

EINVAL
The channel **hdl** is not an opened channel.
Bias +/- amplitude/2 is > 0x7ffc or < -0x7ffc.
ENOMEM
There are not sufficient resources for this data generation function.

4.4.12 p2p_channel_sinewave_config

Synopsis

#include <p2p_lib.h>

p2p_hdl_t
p2p_channel_sinewave_config(p2p_hdl_t **hdl**, unsigned **word**, unsigned **period,** unsigned **delay,** int **bias,** unsigned **amplitude**);

Description

This function generates a payload word following a sinewave pattern. The data pattern repeats over **period** microseconds. This can be a period of up to 1,000 seconds (1,000,000,000 microseconds). The starting phase offset is set with **delay**. Delay is also in microseconds and must be less than period. The data values have a peak-to-peak range specified by **amplitude**. Maximum amplitude is 0xfff8. An amplitude of 0xfff8 and bias of zero would result in values ranging from –0x7ffc to +0x7ffc. A non-zero **bias** will reduce the maximum amplitude allowed by 2*bias. The nominal starting value of the sine wave is zero with a rising slope.

Parameters

hdl	either a channel or label handle.
word	index of the payload word to receive the data.
period	time in microseconds that the pattern repeats.
delay	the stating phase offset time in microseconds.
bias	value added to the nominal.
amplitude	the maximal range of values.

Returns

On success a handle is returned, on error P2P_CALL_FAILED is returned and errno contains the error code. The handle returned is for use with p2p_channel_pause_config.

Error

EINVAL
The channel **hdl** is not an opened channel.
Bias +/- amplitude/2 is > 0x7ffc or < -0x7ffc.
ENOMEM
There are not sufficient resources for this data generation function.

4.4.13 p2p_channel_squarewave_config

Synopsis

#include <p2p_lib.h>

p2p_hdl_t
p2p_channel_squarewave_config(p2p_hdl_t **hdl**, unsigned **word**, unsigned **period,** unsigned **delay,** int **bias,** unsigned **amplitude**);

Description

This function generates a payload word following a squarewave pattern. The data pattern repeats over **period** microseconds. This can be a period of up to 1,000 seconds (1,000,000,000 microseconds). The starting phase offset is set with **delay**. Delay is also in microseconds and must be less than period. The data values have a peak-to-peak range specified by **amplitude**. Maximum amplitude is 0xfff8. An amplitude of 0xfff8 and bias of zero would result in values ranging from –0x7ffc to +0x7ffc. A non-zero **bias** will reduce the maximum amplitude allowed by 2*bias. The nominal starting value of the square wave is +amplitude/2.

Parameters

hdl	either a channel or label handle.
word	index of the payload word to receive the data.
period	time in microseconds that the pattern repeats.
delay	the stating phase offset time in microseconds.
bias	the value added to the nominal.
amplitude	the maximal range of values.

Returns

On success a handle is returned, on error P2P_CALL_FAILED is returned and errno contains the error code. The handle returned is for use with p2p_channel_pause_config.

Error

EINVAL
The channel **hdl** is not an opened channel.
Bias +/- amplitude/2 is > 0x7ffc or < -0x7ffc.
ENOMEM
There are not sufficient resources for this data generation function.

4.4.14 p2p_channel_validation_config

Synopsis

```
#include  <p2p_lib.h>

p2p_hdl_t
p2p_channel_validation_config(p2p_hdl_t  hdl, unsigned word, unsigned label);
```

Description

This function generates a validation block. The six word validation block starts at payload word indexed by **word**. The validation block has the **label** included in this call.

Parameters

hdl	either a channel or label handle.
word	the index of the starting payload word to receive the data.
label	the label word of the validation block.

Returns

On success a handle is returned, on error P2P_CALL_FAILED is returned and errno contains the error code. The handle returned is for use with p2p_channel_pause_config.

Error

EINVAL
The channel **hdl** is not an opened channel.
ENOMEM
There are not sufficient resources for this data generation function.

4.4.15 p2p_channel_validation_inc_config

Synopsis

#include <p2p_lib.h>
p2p_hdl_t
p2p_channel_validation_inc_config(p2p_hdl_t **hdl**, unsigned **word**, unsigned **label,** int **increment**);

Description

This function generates a validation block. The six word validation block starts at payload word indexed by **word**. The validation block has the **label** included in this call. The **increment** of the frame_count word can be varied to provide other than the default increment of one.

Parameters

hdl either a channel or label handle.
word the index of the starting payload word to receive the data.
label the label word of the validation block.
increment the amount to increment the frame count on each frame.

Returns

On success a handle is returned, on error P2P_CALL_FAILED is returned and errno contains the error code. The handle returned is for use with p2p_channel_pause_config.

Error

EINVAL
The channel **hdl** is not an opened channel.
ENOMEM
There are not sufficient resources for this data generation function.

4.5 Channel passthru

The P2Psim card has capabilities to transmit data received on one channel onto another channel. In the process it is possible to change data or insert errors in the resulting transmit stream. Data changes may include corruption of label and length, modification of payload, or corruption of CRC.

4.5.1 p2p_channel_passthru_config

Synopsis

#include <p2p_lib.h>
p2p_hdl_t
p2p_channel_passthru_config(p2p_hdl_t **hdl**, unsigned **channel**, unsigned **label**);

Description

This function causes all data received on the channel or label to be passed to another **channel** for retransmit. Presently the firmware requires knowing the starting **label** of the frame to group a frame's labels into one transmission. Note that when overriding label data, or injecting errors for a label a label handle should be created on the receive channel, and overrides or error injections, applied to that handle.

Parameters

hdl either a channel or label handle.
channel the channel for retransmission.
label the first label of the frame.

Returns

On success a handle is returned, on error P2P_CALL_FAILED is returned and errno contains the error code. The handle returned is for use with p2p_channel_pause_config.

Error

EINVAL
The channel **hdl** is not an opened channel or label.
ENOMEM
There are not sufficient board resources for this function.

4.6 Data Transfer Concepts

4.6.1 Buffered and unbuffered modes

Data transfers are done in one of two modes. First, is buffered mode where multiple packets are transferred between the application and P2P board in one board command. Buffered mode is the most efficient in delivering data to or from the application.

For buffered mode reads only newly received data is read from sampling channels or labels. This contrasts with reading the sampling channel or label directly, where the most recent data is read, even when no new data has been received since the last read.

When using buffered mode one reads from the board, rather than from a channel. The following example shows a buffered mode read.

```
board_hdl      = p2p_open(1, P2P_OPEN_BUFFERED);
channel_hdl    = p2p_open_channel(board_hdl, 1);

/* Configure channel here */
/* Note in buffered mode we read from board but write to channel */

length  = p2p_read(board_hdl, packet_hdr, sizeof(packet_hdr) + sizeof(packet));

status  = p2p_channel_write(channel_hdl, packet, length);
```

Second, is per packet mode where individual packets are transferred to or from specific channels. This mode might be used when reads of individual channels or labels are preferred. Buffered mode may be selected on reads, writes or both. The API commands for write are identical. The read commands are different depending on the mode.

```
board_hdl      = p2p_open(1, P2P_OPEN_BOARD);
channel_hdl    = p2p_open_channel(board_hdl, 1);

/* should configure channel here */
/* Note in unbuffered mode we read from channel and write to channel */

length  = p2p_channel_read(channel_hdl, packet, sizeof(packet));

status  = p2p_channel_write(channel_hdl, packet, length);
```

Buffered mode should be used unless the application needs to be structured to read on a per channel or per label basis. To use buffered mode refer to the p2p_read and p2p_read_next functions. To use per packet mode refer to the p2p_channel_read function.

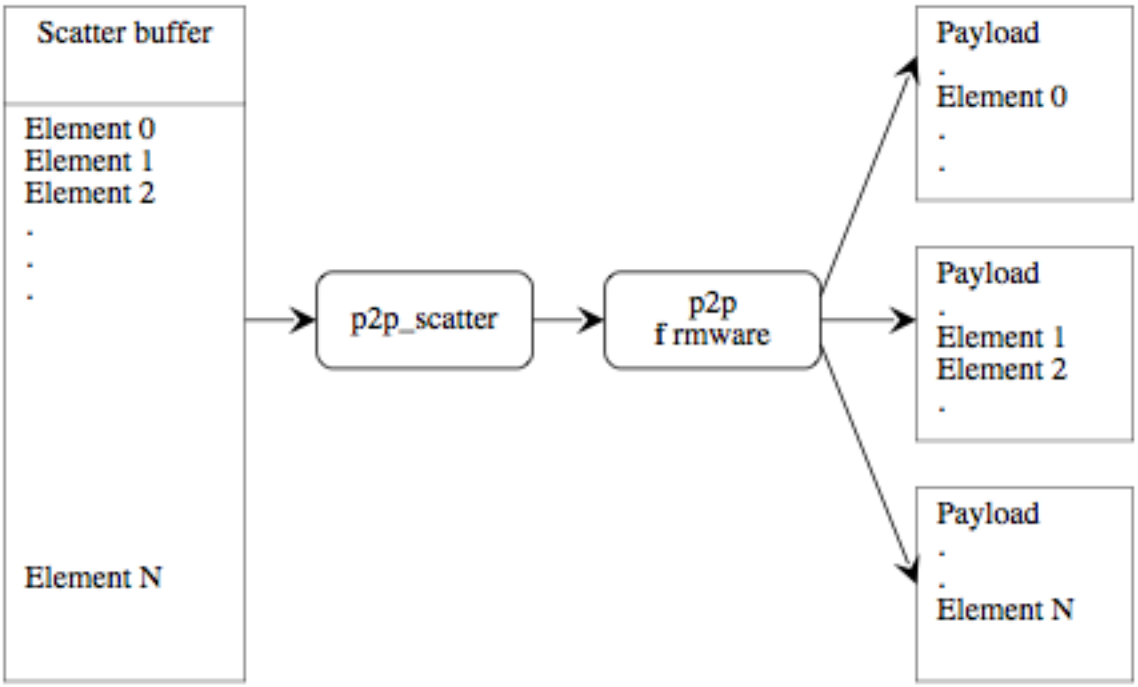
4.6.2 Asynchronous read

It is possible to have the P2P board transfer p2p packets directly to the user application memory as they arrive. This capability is referred to as asynchronous reads. This mode is selected on open by specifying P2P_OPEN_ASYNC_READ. When in asynchronous read mode the board handle is read via p2p_read_next and p2p_read to retrieve the next packet. This is the same as reads in buffered mode.

4.6.3 Scatter gather modes

Data transfers can be optimized when only a subset of data is required on read, or a subset of packet variables need to be updated on write. The method of supplying or retrieving these subsets is called scatter/gather for write/read respectively.

In the case of a scatter operation below, the application first defines how the scatter buffer is to be mapped into packet payloads.



Once this mapping is made at initialization time via calls to p2p_channel_scatter_config, writes of the scatter buffer are done via calls to p2p_scatter. The firmware routes the elements to the outbound packets at the time the packets are built. Gather operations follow an inverse data flow.

Scatter/gather IO does not preclude the use of normal reads and writes if certain channels need normal read/write processing. A mixture of scatter/gather and regular IO is possible even within the same channels and labels.

4.6.4 RX Packet headers

All P2P packets read by the APIs which follow, include a packet header pre-pended by the P2P board. The following is the format of this header:

```
typedef struct p2p_rx_hdr {
```

```
  unsigned short channel;      /* channel number */

  unsigned short length;       /* length in bytes of data returned, not including p2p_rx_hdr_t */
  unsigned       error;        /* error flags */

  int            secs;         /* seconds since January 1 1970 GMT */
  int            usec;         /* microseconds for above */
} p2p_rx_hdr_t;
```

length is the total number of bytes in the packet. Normally this length does not include the CRC. When the CRC is requested to be returned (channel is configured with P2P_MODE_RX_CRC), then the CRC is returned following the payload data, and the packet header length includes the CRC.

where error is a mask of the following:

ERR_APP_CRC	application crc error detected
ERR_CRC	media crc error detected
ERR_SEQ	frame sequence counter validation fails
ERR_WRAP	data wrap check fail
ERR_LENGTH	packet size does not match length indicated in word 1 of packet

4.6.5 TX Packet headers

For transmit packets, a transmit header can be supplied to the p2p_channel_write_hdr call. This is done for error injection or data replay purposes only. In this case the following header is supplied:

```
struct p2p_tx_hdr {
  unsigned short channel;      /* channel number */

  unsigned short length;       /* length of packet, not including p2p_hdr_t */
  unsigned       error;        /* errors to generate from set below */

  unsigned int   secs;         /* time for replay device only */
  unsigned int   usec;         /* time for replay device only */
} p2p_tx_hdr_t;
```

length is the total number of bytes in the packet. Normally this length does not include the CRC. When the CRC is supplied (ERR_CRC is selected in the error mask), then the CRC is included following the payload data, and the packet header length includes the CRC.

where error is a mask of the following:

ERR_CRC	media crc is taken from packet data
ERR_LENGTH	packet length is taken from size of write, post-amble is disabled.

4.6.6 Packets

The packet structure is defined as follows:

```
struct p2p_packet {
  unsigned short label;          /* label field */
  unsigned short length;         /* number of payload words, not including label, length, CRC */
  unsigned short data[62+1];     /* space for payload data and CRC */
} p2p_packet_t;
```

Note: the length supplied in the packet structure is the number of 16 bit payload words. This should be an even number between 2 and 62. Invalid values, ie odd or zero, or > 62 are possible on transmit for error injection purposes.

Packet structures including headers are defined as follows:

```
struct p2p_rx_packet {
  p2p_rx_hdr_t  hdr;             /* Header describing packet */
  p2p_packet_t  pkt;             /* packet data */
} p2p_rx_packet_t;

struct p2p_tx_packet {
  p2p_tx_hdr_t  hdr;             /* Header describing packet */
  p2p_packet_t  pkt;             /* packet data */
} p2p_tx_packet_t;
```

4.6.7 Endian issues on X86

The P2P bus transports 16 bit words in big endian order. That is each 16 bit word is MSB first. The CPU which drives the P2P bus is also big endian. When data is transferred from an X86 architecture to the P2P board, data is swapped on a 32 bit word basis. The following diagram shows the ordering of data on the wire and as received from each host type.

On wire
0011 2233 4455 6677 ….

Data received from big endian host
0011 2233 4455 6677 ….

Data ordering on little endian host
1100 3322 5544 7766 ….

Data received from little endian host
2233 0011 6677 4455 ….

The ordering of data on little endian hosts presents a problem in that the natural swapping of 32 bit words in PCI transfers to the P2P board does not fit with the swapping of 16 bit words which is required for the P2P bus. For this reason the API on X86 architectures performs a swapping function in transferring data to the P2P board. This swapping function must be done with the knowledge of the data structures being transferred. Hardware swapping of 32 bit words is sufficient for 32 bit quantities, however an additional

step of swapping 16 bit words is required when dealing with 16 bit values. This swapping step is done within the API when data structures are known. This is the case for all API calls where data is explicitly typed.

One prerequisite for this swapping is that all data buffers be full multiples of 32 bit words. Transfers of odd multiples of 16 bit words will be rounded up in size, such that full 32 bit words are utilized.

4.7 Channel data transfer functions

4.7.1 p2p_read

Synopsis

#include <p2p_lib.h>

int
p2p_read(p2p_hdl_t **hdl**, p2p_rx_packet_t *__packet__, int **size**);

Description

This function reads data from any channel on the board identified by **hdl.** Note the board must have been opened in buffered read mode. This is in contrast to p2p_channel_read, which only reads from a specific channel. Data is returned as a packet header, followed by packet data including CRC. To minimize data movement use p2p_read_next to look ahead to the channel of the next packet. This allows the packet to be read directly into the buffer for the correct channel.

If **size** is insufficient to hold the packet header plus packet data, the data will be truncated, and only **size** bytes is returned. The rest of the packet data is discarded.

Parameters

hdl value returned by p2p_open or p2p_channel_open.
pac
ket pointer to a buffer to receive data.
size total length of data returned.

Returns

The function returns the length of the data returned to the buffer, including header data. Zero is returned if no data is available.

If an error occurred, P2P_CALL_FAILED is returned and errno will contain the error code.

Errors

EINVAL
hdl is not a valid handle returned from p2p_open

4.7.2 p2p_read_ptr

Synopsis

#include <p2p_lib.h>

p2p_rx_packet_t *
p2p_read_ptr(p2p_hdl_t **hdl**);

Description

This function returns a pointer to the next packet of data in the read buffer. Note BUFFERED mode must be selected when using this function. By using this function the copy of packet data from the read buffer to a user buffer can be avoided. Note, the packet data must be consumed by the caller before subsequent calls to p2p_read_pkt. It is allowed to change data in the packet buffer, however, care should be taken to not exceed the packet size indicated in hdr.length.

Example

```
p2p_rx_packet_t *pbuf;

pbuf      = p2p_read_pkt(boardHdl);
if (pbuf == 0 || pbuf == P2P_CALL_FAILED)
  return errno; /* error */
length    = pbuf->hdr.length;
```

Parameters

hdl board handle returned by p2p_open with P2P_OPEN_READ_BUFFERED or P2P_OPEN_READ_ASYNC selected.

Returns

The function returns a pointer to the next read packet, or zero if the end of read data is reached.

If an error occurred, P2P_CALL_FAILED is returned and errno will contain the error code.

Errors

EINVAL
hdl is not a valid handle returned from p2p_open

4.7.3 p2p_channel_read

Synopsis

```
#include <p2p_lib.h>

int
p2p_channel_read(p2p_hdl_t hdl, p2p_rx_packet_t *packet, int size);
```

Description

This function reads a packet from the channel identified by **hdl**. This function is used to read from a specific channel. This differs from a buffered mode, where the board handle is read, and packets from any channel may be returned.

A packet header precedes the packet data returned.

Parameters

hdl value returned by p2p_channel_open or p2p_label_open.

packet pointer to a buffer to receive packets.

size total length of data returned including packet header.

Returns

The function returns the length of the data returned to the buffer, including the header. Zero is returned if no data is available.

If an error occurred, P2P_CALL_FAILED is returned and errno will contain the error code.

Errors

EINVAL
hdl is not a valid handle returned from p2p_open

EFBIG
The size parameter is too small for the data.

ENODEV
hdl is not a valid receive channel

4.7.4 p2p_channel_write

Synopsis

#include <p2p_lib.h>

int
p2p_channel_write(p2p_hdl_t **hdl**, p2p_packet_t ***packet**, int **size**);

Description

This function writes a packet to the channel identified by **hdl**. The **packet** buffer consists of the packet data excluding CRC. The P2P board adds CRC.

If the channel is a sampling channel, the data is copied to a channel buffer. It will be transmitted when the sampling channel scheduler on the P2P board determines that the channel is due for transmission.

If the channel is a queuing channel, the data is appended to the channels queue, and will be transmitted with all other messages queued when the TX scheduler on the P2P board determines that the channel is due for transmission. This will happen immediately if not prevented by scheduling constraints.

Note that if the board is opened with write buffering enabled, P2P_OPEN_WRITE_BUFFERED, then the write places the data in a buffer for subsequent transfer to the board. When in write buffered mode, issue p2p_write_flush to transfer the data to the board for scheduling.

In order to support channel error injection use p2p_channel_write_hdr where, a block of p2p_tx_hdr_t is passed to specify a set of errors to force.

Parameters

hdl value returned by p2p_channel_open or p2p_label_open.
packet pointer to a buffer holding the packet data.
size size of packet excluding CRC.

Returns

On success the amount of data transferred is returned, on error P2P_CALL_FAILED is returned and p2p_errno() will return the error code.

Errors

EINVAL
hdl is not a valid handle returned from p2p_open
The **size** parameter is incompatible with the channel configuration.

ENOMEM
No buffers are available for queuing data

ENODEV
hdl is not a valid transmit channel

4.7.5 p2p_channel_write_hdr

Synopsis

#include <p2p_lib.h>

int
p2p_channel_write_hdr(p2p_hdl_t **hdl**, p2p_tx_hdr_t ***hdr**, p2p_packet_t ***packet**);

Description

This function writes a packet to the channel identified by **hdl**. The header indicates errors which may be generated for this packet. For example, a CRC error can be generated, if the header error field contains ERR_CRC. Other errors are possible, see section 4.6.5 (TX Packet header) for details.

Parameters

hdl value returned by p2p_channel_open or p2p_label_open.
hdr pointer to a buffer holding a packet header.
packet pointer to a buffer holding the packet data.

Returns

On success the amount of data transferred is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Errors

EINVAL
hdl is not a valid handle returned from p2p_open
The **size** parameter is out of bounds.

ENOMEM
No buffers are available for queueing data

ENODEV
hdl is not a valid transmit channel

4.7.6 p2p_write_flush

Synopsis

```
#include <p2p_lib.h>

int
p2p_write_flush(p2p_hdl_t hdl);
```

Description

This function flushes the write buffer to the board identified by **hdl**. The board must be in buffered mode, ie. p2p_open with P2P_OPEN_WRITE_BUFFERED, in order for this function to be effective. This function does nothing if not in buffered write mode.

Parameters

hdl value returned by p2p_open.

Returns

On success returns 0, on error P2P_CALL_FAILED is returned and errno will return the error code.

Errors

EINVAL
hdl is not a valid handle returned from p2p_open

4.8 Device data transfer functions

4.8.1 p2p_read

Synopsis

```
#include <p2p_lib.h>

int
p2p_read(p2p_hdl_t hdl, void  *data, int size);
```

Description

When this function is called with a device handle identified by **hdl** it reads an opaque structure specific to the device. When called with a channel handle, see the description under Channel data transfer functions.

If **size** is insufficient to hold the data, only **size** bytes is returned.

Parameters

hdl	value returned by p2p_open.
data	pointer to a buffer to receive data.
size	total length of data returned.

Returns

The function returns the length of the data returned to the buffer. Zero is returned if no data is available.

If an error occurred, P2P_CALL_FAILED is returned and errno will contain the error code.

Errors

EINVAL
hdl is not a valid handle returned from p2p_open

4.8.2 p2p_write

Synopsis

```
#include <p2p_lib.h>

int
p2p_write(p2p_hdl_t hdl, void *data, int size);
```

Description

This function writes an opaque data buffer to the device identified by **hdl**. The **data** buffer consists of data appropriate for the device specified. This function is mainly used by p2p utilities to write to the special devices, ie flash, replay.

Parameters

hdl	value returned by p2p_channel_open or p2p_label_open.
data	a buffer holding the data to be written.
size	size of packet excluding CRC.

Returns

On success the amount of data transferred is returned, on error P2P_CALL_FAILED is returned and p2p_errno() will return the error code.

Errors

EINVAL
hdl is not a valid handle returned from p2p_open
The **size** parameter is incompatible with the channel configuration.

ENOMEM
No space is available for writing the data

ENODEV
hdl is not a valid device.

4.8.3 p2p_gather

Synopsis

```
#include <p2p_lib.h>

int
p2p_gather(p2p_hdl_t hdl, short *gather, int size);
```

Description

This function reads a gather buffer from the board identified by **hdl**. This function is used to read from a data items from different labels and channels. Note **hdl** is from a p2p_open with P2P_OPEN_SCATTERGATHER.

Parameters

hdl	value returned by p2p_open.
gather	pointer to a buffer to receive data.
size	size of gather buffer in bytes.

Returns

The function returns the length in bytes of the data returned to the buffer.

If an error occurred, P2P_CALL_FAILED is returned and errno will contain the error code.

Errors

EINVAL
hdl is not a valid handle returned from p2p_open

4.8.4 p2p_scatter

Synopsis

#include <p2p_lib.h>

int
p2p_scatter(p2p_hdl_t **hdl**, short ***scatter**, int **size**);

Description

This function writes a scatter buffer to the board identified by **hdl**. The **scatter** buffer consists of the data elements previously configured by p2p_channel_scatter_config.

Parameters

hdl	value returned by p2p _open.
scatter	pointer to a buffer holding the data elements.
size	size of scatter buffer in bytes.

Returns

On success the amount of data transferred is returned, on error P2P_CALL_FAILED is returned and p2p_errno() will return the error code.

Errors

EINVAL
hdl is not a valid handle returned from p2p_open

4.9 Error creation functions

4.9.1 p2p_reu_delay_config

Synopsis

#include <p2p_lib.h>

int
p2p_reu_delay_config(p2p_hdl_t **hdl,** int **delay**);

Description

This function changes the response delay of an REU to an ACE command packet. This will hold off the reply packet to the ACE for the number of microseconds specified by the **delay** value. This function is valid while the channel is started.

Parameters

hdl	value returned by p2p_channel_open.
delay	delay in microseconds for reply packet sent to ACE.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Error

EINVAL **hdl** is not a configured REU channel.

4.9.2 p2p_channel_crc_error_config

Synopsis

#include <p2p_lib.h>

p2p_hdl_t
p2p_channel_crc_error_config(p2p_hdl_t **hdl,** unsigned **wait_frames**, unsigned **error_frames,** unsigned **value**);

Description

This function inserts a user specified crc in place of the hardware generated crc for a given number of frames. The error insertion can wait for **wait_frames** until it is applied. Once applied it can be repeated for **error_frames**. The 16 bit crc is inserted following the payload words as defined by the length field.

Parameters

hdl	value returned by p2p_channel/label_open and configured as a tx channel/label.
wait_frames	number of frames to wait before inserting the crc.

error_frames number of frames to apply the error crc.
value value of the crc to apply.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Error

EINVAL
The channel is not stopped or **hdl** is not an opened channel.
ENOMEM
There are not sufficient resources for this data generation function.

4.9.3 p2p_channel_override_config

Synopsis

#include <p2p_lib.h>
p2p_hdl_t
p2p_channel_override_config(p2p_hdl_t **hdl,** unsigned **wait_frames**, unsigned **override_frames,** unsigned **word**, unsigned **mask**, unsigned **value**);

Description

The override function will override a word of packet data. The override is done after wait_frames have elapsed, and will continue for override_frames. It is possible to overide a portion of the word by specifying a mask value. The mask has bits set for each bit of value to apply to the packet word.

Parameters

hdl value returned by p2p_channel_open or p2p_label_open.
wait_frames number of frames to process normally before pausing.
override_frames number of frames to override the target data.
word index of the payload word to override. Note word of -2 overrides label, -1 overrides length.
mask the bits which are inserted into word.
value the value to insert into word. payload[word] = (payload[word] & ~mask) | (value & mask)

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Error

EINVAL
The channel **hdl** is not an opened channel.
ENOMEM
There are not sufficient resources for this data generation function.

Override XML API

Override via p2p_xml_config allows extensions to the API while allowing backward compatability. The following parameters extend the above described API. The **channel** and **label** parameters avoid having to know or open the channel and label handles. **priority** allows specifying overrides to happen before or after crc or other data generation functions.

This call is only valid using the board handle.

channel number of channel to override, instead of using hdl.
label number of label to override, instead of using hdl.
priority indicates order override is performed relative to other data generation functions.
repeat number of times to repeat the wait_frames, override_frames sequence.

XML example

```
char *oride1 = "<override channel='1'
                label='0x4d30'
                word='3'
                mask='0xffff'
                value='0x1234'
                wait_frames='200'
                override_frames='3'
                priority='50'
                repeat='10'
                />"
p2p_xml_config(board_handle, oride1, strlen(oride1) );
```

4.9.4 p2p_channel_pause_config

Synopsis

#include <p2p_lib.h>

p2p_hdl_t
p2p_channel_pause_config(p2p_hdl_t **hdl,** p2p_hdl_t **dhdl**, unsigned **wait_frames**, unsigned **pause_frames**);

Description

This function causes a data generation function to pause for a given number of frames. This can be used to halt the generation of new values of any data elements configured by functions in section 4.4.

Parameters

hdl value returned by p2p_channel_open or p2p_label_open.
dhdl value returned by a data generation function.
wait_frames number of frames to process normally before pausing.
pause_frames number of frames to pause the target data generation function.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Error

EINVAL
The channel **hdl** is not an opened channel.
The data generation handle **dhdl** is not valid.
ENOMEM
There are not sufficient resources for this data generation function.

4.9.5 p2p_channel_pause_restore

Synopsis

#include <p2p_lib.h>

p2p_hdl_t
p2p_channel_pause_restore(p2p_hdl_t **hdl,** p2p_hdl_t p**hdl**);

Description

This function cancels a pause function initiated by **p2p_channel_pause_config**.

Parameters

hdl	value returned by p2p_channel_open or p2p_label_open.
phdl	value returned by **p2p_channel_pause_config**.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Error

EINVAL
The channel **hdl** is not an opened channel.
The pause handle p**hdl** is not valid.

4.9.6 p2p_pll_config

Synopsis

#include <p2p_lib.h>

int
p2p_pll_config(p2p_hdl_t **hdl,** int **pll_adjust**);

Description

This function causes pll clock to deviate from nominal speed by the percentage given in pll_adjust. This deviation can be in the range of –10 to +10 percent. The pll clock is what drives the transmit data. A value of **pll_adjust** would cause high speed channels to transmit at approximately 4.5 Mhz instead of the nominal value of 5 Mhz.

Parameters

hdl	value returned by p2p_open.
pll_adjust	percentage to vary the pll clock.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno contains the error code.

Error

EINVAL

The board **hdl** is not valid, or the board is not in a stopped state.

4.10 Status Functions

4.10.1 p2p_get_counter

Synopsis

#include <p2p_lib.h>

int
p2p_get_counter(p2p_hdl_t **hdl**, void ***counter**, int **size**);

Description

This function retrieves statistic counters for the P2P board or channel identified by **hdl**.

Parameters

hdl	value returned by p2p_open or p2p_channel_open.
counter	a pointer to a structure of type p2p_counter_t. See p2p.h for an explanation.
size	size of buffer to receive counters.

Returns

On success 0 is returned, on error 1 is returned and errno will contain the error code.

Errors

None

4.10.2 p2p_reset_counter

Synopsis

#include <p2p_lib.h>

int
p2p_reset_counter(p2p_hdl_t **hdl**);

Description

This function resets the statistic counter for the P2P board or channel identified by **hdl**.

Parameters

hdl	value returned by p2p_open or p2p_channel_open.

Returns

On success 0 is returned, on error 1 is returned and errno will contain the error code.

Errors

None

4.11 Debugging Functions

The following functions have been useful for debugging firmware and may be useful for debugging user programs in exceptional situations.

4.11.1 p2p_bit

Synopsis

```
#include "p2p_lib.h"

int
p2p_bit(p2p_hdl_t hdl, unsigned channel, char *buffer, int len);
```

Description

This function directs the firmware to perform a loopback test on the specified channel. If channel one is specified, then a loopback is performed between channels one and two. All odd and even pairs of channels can be tested with this interface.

Note a loopback connector must be present to execute this test, unless software loopback mode is selected, see section Error: Reference source not found.

Parameters

hdl value returned by p2p_open.

Returns

On success 0 is returned, on error a message is returned to **buffer** indicating the error.

Errors

None

4.11.2 p2p_debug

Synopsis

```
#include <p2p_lib.h>

int
p2p_debug(p2p_hdl_t hdl, unsigned long flags);
```

Description

Debug messages can be enabled in the driver software and board firmware. The driver outputs debug messages to the system log file, SYSLOG, for Unix/Linux systems. The firmware outputs debug messages to the console or to the serial channel on the front of the board. Debug output written to the console can be read from the console device (p2p_open(lbn, P2P_OPEN_CONSOLE)). To enable firmware messages to the board's serial channel, select the P2P_CONSOLE_DEBUG flag.

This function sets the debug options specified by **flags**. Each flag turns on or off a certain type of output. There is one flag which has proven useful to user programs. That is the flag p2p_ERROR_DEBUG. It gives information about the test that failed, resulting in an error returned by firmware. Errors detected at the library level do not result in firmware calls and do not generate error messages to the firmware status device.

To see the information routed to the console device do the following:
IRIX "cat /hw/p2p1/console"

Note: some debug information is only available with a special firmware version where all debug is enabled.

Parameters

hdl value returned by p2p_open.

flags can be one or more of the following:

P2P_CONFIG_DEBUG	Shows configuration related information mainly during board boot.
P2P_OPEN_DEBUG	Shows information when opening boards or channels.
p2p_READ_DEBUG	Shows information about frames received.
p2p_WRITE_DEBUG	Shows information about frames transmitted.
P2P_COMMAND_DEBUG	Shows information about commands issued to a board.
p2p_INTR_DEBUG	Shows information during interrupts (requires debug firmware).
p2p_VERBOSE_DEBUG	May show more detailed information for the other options.
p2p_ERROR_DEBUG	Shows information about user call errors.
P2P_FATAL_DEBUG	Shows information about serious firmware conditions.
P2P_CONSOLE_DEBUG	Shows information on the serial channel as well as console device.
p2p_TIMING_DEBUG	Enables timing calculations returned by p2p_get_counter.

By default the following flags are selected: P2P_CONFIG_DEBUG, p2p_ERROR_DEBUG, P2P_FATAL_DEBUG.

Returns

On success 0 is returned, on error P2P_CALL_FAILED is returned and errno will contain the error code.

Errors

None

4.11.3 p2p_show

Synopsis

```
#include "p2p_lib.h"

int
p2p_show(p2p_hdl_t hdl);
```

Description

This function instructs the P2P board to show information about the board or channel associated with hdl. The information is available on the console device. **hdl** may be a board or channel

handle. The information dumped is available by reading the console device. On IRIX this can be done via the command “cat /hw/P2P1/console” for the slot 1 device.

Parameters

hdl value returned by p2p_open.

Returns

On success 0 is returned, on error 1 is returned and errno will contain the error code.

Errors

None

5 P2P Firmware load utility

5.1 p2pload

When firmware updates are released they are loaded into flash memory via the **p2pload** utility. The default location for firmware is in the **/usr/local/p2p directory**. This directory contains the currently installed firmware whose name is in the format p2p-x.y.z-a.elf. This indicates revision x.y.z-a of firmware. It may also contain previous versions of firmware. A link named "firmware" points to the most recently installed firmware.

When the **p2pload** program is run without specifying any parameters, the current firmware from /usr/local/p2p is loaded in all P2P boards which are present. When loading firmware specify the -r option to reset the board, and execute the newly loaded firmware. Without the reset option, a reboot is necessary to execute the newly loaded firmware.

The p2pload program check to see if the requested firmware is already loaded. If so, it does not reload the same firmware version unless the -F option is specified.

When software is installed in other than /usr/local, the firmware name must be specified on the p2pload command. The example below show how to load firmware when the p2p software is installed in directory name /p2p.

Example

Firmware loaded in default location:

p2pload

Firmware loaded in /p2p:

p2pload -r –f /p2p/p2p-0.1.12.elf.

Synopsis

p2pload [-r] [-F] [-f file_name] [1-4]

Files

/usr/local/p2p/firmware	link to current firmware file.
/usr/local/p2p/p2p-x.y.z.elf	revision x.y.z firmware file.

6 P2P Tests

6.1 p2p

This is an internal test program that demonstrates various features of the P2P board. It is provided as a basic test program to validate board functionality. In addition source code is provided in the hope that it may prove useful as an example for programming.

This program will prompt for parameters if none are given. Alternatively the default parameters will be used when the command is given with the test type option (ex "p2p firmware").

To get a list of options enter "p2p" with no options. The following output will result.

```
> p2p
enter test type:
              count   Print p2p internal counters
         countreset   Reset p2p internal counters
            console   Print p2p debug console
           firmware   Show firmware revision
              debug   Set debug logging options
               dmrs   DMRS loopback test
                fce   FCE loopback test
                reu   REU loopback test
                txh   Transmit high speed test
                 tap   tap test
                  tx   Data transmit test
```

This shows the options of the program. Enter an option, and you will be prompted for additional parameters. Default values are selected by entering <cr>.

6.1.1 p2p count

One common use of this program by user's would be to check on activity of the p2p bus. To obtain a list of activity counts enter the following command:

```
      > p2p count
lbn 1
           3200 packets queued for transmit
           3200 packets transmited
           3200 packets received
           3200 packets delivered to user
```

This particular example shows the counts after running the "p2p fce" test, another valuable option of this program. There are many more counts possible other than those listed in this example. In particular a number of error counters are present as well as some obscure statistics.

6.1.2 p2p countreset

To clear the counters shown in the “p2p count” command, enter the command “p2p countreset”.

6.1.3 p2p console

Debug messages are printed to a console buffer on the p2p board. To see these messages use the command “p2p console”.

Sample output

```
device 1 id 2210EE
The Goebel Company, P2P xp1000 firmware Rev 0.0.6, Feb 24 2006 16:21:07
enable IMU XCR:2200000 BR1:401C008 BR2:18085
sizeof p2p F0
p2p_board: done
rdc_console: done
xp1000_flash: done
rdc_scatter: done
tap:6EFFA00 end:7EFF9FC
p2p_tap: done
p2p_replay: done
p2p_channel: done
p2p_channel: done
p2p_channel: done
p2p_channel: done
rdcInit: done
Init done
p2p started
imu1:isr2 value FFFFFFFF
pciHostMemBase: FFFFFFFF:1CC0000 - 100000
POM0PCIAL: 1000000
POM0PCIAH: 0
device 1 id 2210EE
io_board_init:90320000 id 10006
```

6.1.4 p2p firmware

To show the firmware revision and date on the P2P board, enter “p2p firmware”. A string such as the following will be shown:

Sample output

```
5 maclinux:/home/goebel % p2p firmware
p2p1 firmware:
The Goebel Company, P2P xp1000 firmware Rev 0.0.6, Feb 27 2006 10:47:50
```

6.1.5 p2p debug

This option sets debug flags for printing information to the board console. This option can cause delays which may prevent normal board functionality.

6.1.6 p2p dmrs

This test performs low speed transmits between pairs of channels connected via loopback connector.
Sample output

```
6 maclinux:/home/goebel % p2p dmrs

P2P dmrs test on board 1
This test uses standard loopback connector, ch1<->ch2, ch3<->ch4
etc
DMRS channels transmit from ch1->ch2, ch3->ch4 etc
Test 32 DMRS channels
p2p1 firmware:
The Goebel Company, P2P xp1000 firmware Rev 0.0.6, Feb 27 2006
10:47:50

Initialize 32 channels
Test runs for 10 seconds
               16 packets queued for transmit
            82176 packets transmited
            82176 packets received
            82176 packets delivered to api
```

6.1.7 p2p fce

This test performs high speed transmits between pairs of channels connected via loopback connector.

Sample output

```
7 maclinux:/home/goebel % p2p fce

P2P fce test on board 1
This test uses standard loopback connector, ch1<->ch2, ch3<->ch4
etc
FCE channels transmit from ch1->ch2, ch3->ch4 etc
Test 32 FCE channels
p2p1 firmware:
The Goebel Company, P2P xp1000 firmware Rev 0.0.6, Feb 27 2006
10:47:50

Initialize 32 channels
Test runs for 10 seconds
Test complete
channel ( 1: 2)  (transmitted:received) = (1094:1094)  passed
channel ( 3: 4)  (transmitted:received) = (1094:1094)  passed
channel ( 5: 6)  (transmitted:received) = (1094:1094)  passed
channel ( 7: 8)  (transmitted:received) = (1094:1094)  passed
channel ( 9:10)  (transmitted:received) = (1094:1094)  passed
channel (11:12)  (transmitted:received) = (1094:1094)  passed
channel (13:14)  (transmitted:received) = (1094:1094)  passed
channel (15:16)  (transmitted:received) = (1094:1094)  passed
channel (17:18)  (transmitted:received) = (1094:1094)  passed
channel (19:20)  (transmitted:received) = (1094:1094)  passed
```

```
channel (21:22)  (transmitted:received) = (1094:1094)  passed
channel (23:24)  (transmitted:received) = (1094:1094)  passed
channel (25:26)  (transmitted:received) = (1094:1094)  passed
channel (27:28)  (transmitted:received) = (1094:1094)  passed
channel (29:30)  (transmitted:received) = (1094:1094)  passed
channel (31:32)  (transmitted:received) = (1094:1094)  passed
```

6.1.8 p2p reu

This test performs an ACE to REU transmit/receive test on pairs of channels connected via loopback connector.

Sample output

```
8 maclinux:/home/goebel % p2p reu

P2P reu test on board 1
This test uses standard loopback connector, ch1<->ch2, ch3<->ch4
etc
p2p1 firmware:
The Goebel Company, P2P xp1000 firmware Rev 0.0.6, Feb 27 2006
10:47:50

Initialize 32 channels
Test runs for 10 seconds
                32 packets queued for transmit
            174368 packets transmited
            174368 packets received
            174368 packets delivered to api
```

6.1.9 p2p txh

This option transmits multiple labels per frame exercising the scheduling capabilities of the board.

Sample output

```
9 maclinux:/home/goebel % p2p txh

P2P high speed transmit test on 32 channels
p2p1 firmware:
The Goebel Company, P2P xp1000 firmware Rev 0.0.6, Feb 27 2006
10:47:50

Test runs for 10 seconds
            352000 packets transmited
            352000 packets received
                10 receive errors
                10 packets with wrong postamble
        0x34c80012 spare2
            196046 spare4
               221 spare8
```

6.1.10 p2p tap

This option utilizes the analyzer interface to tap into the transmit and receive data streams.

6.1.11 p2p tx

This option performs a transmit test on one or more channels.

Sample output

```
10 maclinux:/home/goebel % p2p tx

P2P trasnmit test on channel 1
p2p1 firmware:
The Goebel Company, P2P xp1000 firmware Rev 0.0.6, Feb 27 2006
10:47:50

Initialize channel 1 low speed
Test runs for 10 seconds
                1 packets queued for transmit
             4999 packets transmited
```

7 Installation

7.1 IRIX

To install start up “swmgr” as root. Select the distribution directory, on CD or download directory. Select the rdc (remote driver call) and p2p packages via the check boxes, and select start. Reboot after installation is complete as instructed.

P2P base software is installed in the /usr/local directory. The subdirectories are as follows:

bin The executables, p2p and p2pload
p2p P2P firmware and subdirectories

The development sub-package contains include files and libraries for program development. These files reside in /use/include, and /usr/lib32.

7.2 Linux

Linux releases are in the form of RPM files. Standard Linux RPM install procedures are used. An install under Linux takes the form of the following command.

rpm -i p2p-0.1.16-a.x86_64.rpm

7.3 Windows

The P2P software for Windows is currently released as a zip file. To install unzip the file in the \usr\local directory, or a directory of your choosing.

7.3.1 Driver install

A driver install will be required for both PMCs on the P2P card. Once installed one will be noted as P2P and the other P2Pio. Follow the install procedure below twice to install both drivers. Both drivers are located in the same location (\usr\local\drivers).

≅ A driver install dialog box should appear upon booting with the P2P card installed.

≅ From this dialog box select the bottom option to install from specific places.

≅ Select the bottom option (advanced) to select software location.

≅ Select the software location \usr\local\drivers if you have installed in the default location, or select <your_install_directory>\drivers.

≅ Start the install.

7.4 Installation verification

Verification of installation is accomplished by running a set of tests with the p2p program. Use the following tests preferable with a loopback interface connected to verify software functionality. Compare with the sample results shown in the P2P utility section. Note counts will not match between runs for time limited tests.

p2p firmware
p2p dmrs
p2p fce
p2p reu
p2p txh
p2p tx

7.5 Client server configuration

Access to p2p cards across ethernet networks is possible by configuring cards as network access. To set up this configuration, copy the file /etc/sysconfig/goebel/rdc.conf shown below to p2p.conf, and add the remote p2p line as documented in this file Once this configuration file is present, and the romote host has the version p2p-0.1.20-a or newer version installed, remote access will be enabled.

```
#
#       rdc client server configuration
#
#       on the client side create a file for your device with lines of
the form:
#
#       <alias> <lbn> <target_ip> <port> <target_lbn> <target_name>
<device_name>#
#
#       where:
#       alias   name for future use
#       lbn     logical board number, this should be one greater than
local lbns
#       target_ip ip address of the remote host
#       port    port remote host is configured to listen on
(default=5000)
#       target_lbn lbn on the remote host
#       target_name name on the remote host (not presently used)
#       device_name name on the remote host (not presently used)
#
#       to enable remote p2p device 1 as local p2p device 2 create
p2p.conf:
#       p2p2 2 198.168.2.152 5000 1 /dev/p2p/1 p2p1.
```

7.5.1 RDC server for p2p

The rdc service handles the remote p2p requests. To check the rdc service being active use

chkconfig --list rdc

It should output:

rdc 0:off 1:off 2:off 3:on 4:on 5:on 6:off

7.5.2 RDC log file

The log for remote p2p access is in /var/log/rdcd.log. This log contains messages for the opening and closing by client p2p users.

8 Change log

Changes are organized by firmware versions. Changes are documented for versions after 0.0.7h, the first widely deployed complete version. Missing firmware version numbers (0.1.1-3, 0.0.8) are for versions which were not deployed.

8.1 P2P-0.0.20-a

1. Add p2p_xml_file_config.
2. Add ACE and FCM emulation capability.

8.2 P2P-0.0.19-a

3. Add p2p_passthru_config.

8.3 P2P-0.0.18-a

1. REU wrap changes. See Boeing documentation for specification of proprietary information defining these protocols.

8.4 P2P-0.0.17-a

1. REU wrap changes. See Boeing documentation for specification of proprietary information defining these protocols.

8.5 P2P-0.0.16

1. Add REU wrap protocols TYPE2 and TYPE3. See Boeing documentation for specification of proprietary information defining these protocols.
2. Add console control and status commands for board firmware debug.

8.6 P2P-0.1.15

1. Have p2pload corectly recognize firmware reload required for older firmware versions.
2. Fix board reset hang when interrupt switch enabled on carrier.
3. Correct buffering problem for REUs introduced in 0.1.14.
4. Correct loop termination check which results in hang on async 422 channels.

8.7 P2P-0.1.14

1. Add p2p_tx_frame_time() to dynamically change frame rate.
2. Allow packets of 2 bytes.

8.8 P2P-0.1.13

1. Check firmware version loaded compared to firmware file to prevent loading firmware already loaded. Firmware can be forced to load with "-F" parameter.
2. Correct behavior for writing to a full queue. Old data should be dropped and new data queued.

8.9 P2P-0.1.12 – IRIX release

1. Rearange async read buffer pointers to facilitate proper flush of IO cache lines for SGI. This prevents possible loss of buffer pointers, with possible stall or halt of incomming data.

8.10 P2P-0.1.11 – IRIX release

1. Fix problem of overwriting incomming data when packets forced to cache line boundaries for SGI hardware io cache problem. Note problem only occurs when overrun of async read buffer.

8.11 P2P-0.1.10

1. Correct problem with transmitter idle pattern left on when channel stopped.
2. Support payload length of 2.
3. Changes to Primary REU wrap protocol.

8.12 P2P-0.1.9

1. Add board reset capability.
2. Add support of encrypted wrap.

8.13 P2P-0.1.8

1. Add interface to vary validation block increment count, p2p_channel_validation_inc_config.
2. Add interface to rescind pause funtions, p2p_channel_pause_restore.
3. Change pll transmit speed control limits to +/- 20%. Note this functionality requires FPGA rev 9.
4. Add command interface to show FPGA revisiion (p2p id).

8.14 P2P-0.1.7

1. Correct DMRS activity count error introduced in 0.1.3.

8.15 P2P-0.1.6

1. Add synchronous queuing mode for transmit (Orion requirement).

8.16 P2P-0.1.5

1. Improve host clock time sync accuracy.

8.17 P2P-0.1.4

1. Add support for P2P422SIM-X FPGA rev 2.2.
2. Add support for 203.4 Kbaud 422 channels.

8.18 P2P-0.1.0

1. Add support for P2P422SIM-X board with RS422 SDLC and ASYNC channels for PN:P2P422SIM-X.

2. Added p2p_channel_pause_restore.
3. Added firmware support for PLL clock modificaion of +/- 10%.
4. Add capability to inhibit REU reply on crc error of ACE command packet.
5. Support for replay of multiple channels.

8.19 P2P-0.0.10

1. Added p2p_channel_diffcount_config. This is in support of encrypted wrap.
2. Added p2p_channel_copy_config This is in support of encrypted wrap.
3. Correct wrap protocol for primary REUs.
4. Smallest packet size supported reduced to 2 payload words in support of CEV/Orion.
5. Turn off the transmitter idle pattern when a channel is turned off.
6. Correct REU rx queue buffering for SQUEUING mode to queue request instead of behaving like SAMPLING mode.

8.20 P2P-0.0.9

1. Avoid REU reply when receive command has hardware error.