
C-Interface Description

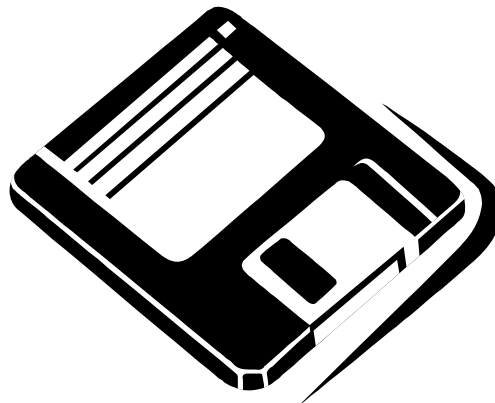
Windows 9x/ME – WindowsNT/2000/XP - Linux

ARINC 429 intelligent Interface Boards
A429USB | A429PCC | A429EPC | A429PC104
A429PCI | A429PCI-2G | A429VME | A429VXI

and A429IPM IP-Module

Version 3.64

CID-XPC-T-364, Rev F-1



TechSAT GmbH - Technische Systeme für Avionic und Test

Gruber Straße 46 c

D-85586 Poing

Germany

Phone (+49) (0)8121 703-0

Fax (+49) (0)8121 703-177

Internet: <http://www.techsat.com>

e-mail: ts-info@techsat.com

Product ARINC 429 Interface boards - A429xPC

Document C-Interface Description (DOS - Windows 9x/ME – WindowsNT/2000/XP - Linux)

Document# CID-XPC-T-364

Filed under

\\Bigmama\WORK\ToBeReleasedAndPublished\7_Modules\702096_A429USB\Records\Paperware\UserManual\CIFDESC_364.DOC

Revision F-1

Date 13-Jun-07

Copyright Notice

The Information in this document is the property of TechSAT and may not be copied in any form whatsoever without the prior express written permission of TechSAT. It has been checked and is thought to be entirely reliable. However, no responsibility is assumed in case of inaccuracies. Furthermore, TechSAT reserves the right to change any of the specifications described herein to improve reliability, function or design. TechSAT neither assumes any liability arising out of the information herein nor conveys any license under its patent rights or the rights of others.

Copyright © 1996-2007 TechSAT GmbH

Table of Contents

Revision history	6
Extended functions	10
Preface	11
ARINC 429 Data Transmission	12
ARINC 429 Data Transmission Commands	12
ARINC 429 Transmission Modes	13
Update Rate	14
Transmit Data Organization	15
Conditional Transmission, Trigger State Table	16
ARINC 429 Data Reception	17
ARINC 429 Receive Modes	17
Storage mode	17
Trigger Mode	18
Masked Mode	18
Manipulation Mode	18
Protocol Mode	19
Index Mode	19
Common Specifics & Differences between Interface boards	20
Time stamping	20
Backward compatibility 1	20
Backward compatibility 2: PCI-60: 16 Tx-/24 Rx-Channels	20
ARINC 429 Driver Power Down Mode [PCC]	21
Parity Generation [PCC][USB]	21
System clock	21
Wrap-around relays [USB][PCC][USB][EPC][PCI-2G]	21
Interface library basics	23
How to address the interface board	23
Headerfiles to be used	24
Definitions to be used to compile your application	24
Libraries/DLLs to be used to build your application	25
Basic ARINC 429 interface functions	26
_open_xpc() [all except HPVISA and VME]	26
_open_xpc() [HPVISA]	27
_open_xpc() [VME]	28
_close_xpc() [xPC]	29
_xpc_get_lasterror() [xPC] (currently only [PCC][USB])	29
_config_trans() [xPC]	30
_set_parity_mode() [PCC][USB] only	31
_set_tx_mode() [xPC]	32
_get_transmitter_set() [xPC]	33
_define_trans() [xPC]	34
_activate_trans() [xPC]	36
_delete_trans() [xPC]	37
_update_trans() [xPC]	38
_activate_tx_trigger_mode () [xPC]	39
_get_tx_definitions() [xPC]	40
_get_single_tx_definition() [xPC]	41
_get_tx_timeslice() [xPC]	42
_config_rec() [xPC]	43
_set_rec_buf_size() [xPC]	44
_get_rec_buf_count() [xPC]	46
_read_receiver_buffer() [xPC]	47
_activate_rec() [xPC]	48
_activate_rx_trigger_mode () [xPC] except [USB]	50
_get_status_word() [xPC]	51
_get_receiver_data_ready() [PCC][USB] (old interface)	53
_get_receiver_data_ready() [xPC] except [PCC][USB] (old interface)	54
_get_data_ready() [xPC]	55

_get_receiver_data_lost() [xPC]	56
_reset_data_lost() [xPC]	57
_get_receiver_data_exist() [xPC]	58
_activate_rx_indexed_mode() [xPC]	59
_read_indexed_data() [xPC]	60
_read_all_indexed_data() [xPC]	61
_get_receiver_mode() [xPC]	62
_define_trigger() [xPC] (xOUT not for [USB])	64
_get_trigger_state() [PCC][USB][HPVISA]	67
_get_trigger_state() [PC104][EPC][VME][PCI][PCI-2G]	68
_set_trigger_state() [PCC][USB]	69
_set_trigger_state() [EPC][PCI][PCI-2G][VME][PC104]	70
_set_trigger_state() [PC104][EPC][VME][PCI][PCI-2G]	71
_set_trigger_state() [HPVISA]	72
_reset_selftest() [PCC][USB] (old interface)	73
_reset_selftest() [EPC] (old interface)	74
_reset_and_selftest() [xPC]	75
_establish_loop() [PCC][USB][EPC][PCI-2G]	77
_reset_clock() [xPC]	78
_get_arinc_board_type() [xPC]	79
_get_firmware_version() [PCC][USB] (old interface)	80
_get_firmware_version() [EPC] (old interface)	80
_get_fw_version() [xPC]	81
_get_ticks_mpcboard() [PCC][USB] (old interface)	82
_get_ticks_mpcboard() [EPC] (old interface)	83
_get_ticks() [xPC]	84
_get_ext_ticks() [xPC]	85
_set_timestamp_synchro_mode() [xPC] except [USB]	86
_set_timestamp_res() [xPC]	88
_get_timestamp_res() [xPC]	89
_set_high_resolution() [xPC]	90
_set_out_state() [PCC][HPVISA] (old interface)	91
_set_out_state() [xPC] except [USB]	91
_get_in_state() [PCC][HPVISA] (old interface)	92
_get_in_state() [xPC] except [USB]	92
National Instruments LabView™ specific interface functions	94
_lv_get_transmitter_set() [xPC] except [HPVISA]	95
_lv_define_trans() [xPC] except [HPVISA]	96
_lv_activate_trans() [xPC] except [HPVISA]	97
_lv_delete_trans() [xPC] except [HPVISA]	98
_lv_update_trans() [xPC] except [HPVISA]	99
_lv_get_tx_definitions() [xPC] except [HPVISA]	100
_lv_get_single_tx_definition() [xPC] except [HPVISA]	101
_lv_read_receiver_buffer() [xPC] except [HPVISA]	102
_lv_activate_rec() [xPC] except [HPVISA]	103
_lv_define_trigger() [xPC] except [HPVISA]	104
A429IPM specific interface functions	105
_ipm_ipreadbyte() [xPC] except [PCC][USB][HPVISA]	105
_ipm_ipwritebyte() [xPC] except [PCC][USB][HPVISA]	106
_ipm_ipreadword() [xPC] except [PCC][USB][HPVISA]	107
_ipm_ipreadword() [HPVISA]	107
_ipm_ipwriteword() [xPC] except [PCC][USB][HPVISA]	108
_ipm_ipwriteword() [HPVISA]	108
_ipm_ipreadbytemem() [xPC] except [PCC][USB][HPVISA]	109
_ipm_ipwritebytemem() [xPC] except [PCC][USB][HPVISA]	110
_ipm_ipreadwordmem() [xPC] except [PCC][USB][HPVISA]	111
_ipm_ipreadwordmem() [HPVISA]	112
_ipm_ipwritewordmem() [xPC] except [PCC][USB][HPVISA]	113
_ipm_ipwritewordmem() [HPVISA]	114
_ipm_ipreadid() [xPC] except [PCC][USB][HPVISA]	115
_ipm_scan_ipm() [xPC] except [PCC][USB][HPVISA]	117

_ipm_scan_ipm() [HPVISA]	118
_ip429_reset() [xPC] except [PCC][USB]	118
A429EPC specific interface functions	119
_epc_write_routing() [EPC]	119
_epc_set_out_state() [EPC]	121
_epc_get_in_state() [EPC]	122
_epc_firmware_version() [EPC]	123
A429PCI-2G specific interface functions	124
_pci2g_set_out_state() [PCI-2G]	124
_pci2g_get_in_state() [PCI-2G]	125
_pci2g_firmware_version() [PCI-2G]	126
A429USB specific interface functions	127
_usb_get_discretes() [USB]	127
_usb_set_discretes() [USB]	128
Serial RS-xxx Interface functions [PCC][EPC]	129
_enable_async() [PCC][EPC]	129
_disable_async() [PCC][EPC]	130
_set_serial_rx_buf_size() [PCC][EPC]	130
_tx_serial_data() [PCC][EPC]	131
_rx_serial_data() [PCC][EPC]	131
_read_serial_buf() [PCC][EPC]	132
_get_serial_data_ready() [PCC][EPC]	133
IP carrier / IP module reset functions	134
_epc_reset() [EPC]	134
_pci_reset() [PCI]	134
_pci2g_reset() [PCI-2G]	135
_ipc01_reset() [VME]	135
_pc104_reset() [EPC]	135
_ipm_ipreset() [EPC][PCI-2G]	136
Appendix: xPCD.c – executable example for [EPC]	137
Main Menu	137
C – Configure	137
T – Trigger	138
M – Miscellaneous	138
R – Receive	139
T – Transmit	139
E – EPC features	140
D – Discrete Signals	140
example: loop back Tx-Data	141

Revision history

- 2.1 Initial document version
- 2.2 Driver and interface library totally reworked to make it more reliable
[TechSAT internal version (not published)]
- 2.3 'Typo' in `_establish_loop()` corrected
- 2.4 New functions introduced:
- `_get_fw_version()`
 - `_get_ticks()`
 - `_get_data_ready()`
 - `_reset_and_selftest()`
 - `_set_timestamp_synchro_mode()`
 - `_set_timestamp_res()`
 - `_epc_scan_ipm()`
 - `_epc_write_routing()`
- Typo corrected in interface description of `_get_receiver_data_ready()`[EPC] function.
- 2.5 New functions introduced:
- `_set_out_state()`
 - `_get_in_state()`
 - `_epc_set_out_state()`
 - `_epc_get_in_state()`
 - `_get_timestamp_res()`
- Preface section updated
- Attention:**
Shortly after close-out of Version 2.5, a bug in the C-interface library was detected. The function `_update_trans()` will return error code 2 (transmitter number out of range) if it is invoked with a transmitter number other than 0 or 1.
Work-around: Use a sequence of `_define_trans()` and `_activate_trans()` to update the value of a transmission definition.
This bug will be fixed in the next release.
- 2.6 Bug in C-interface library function `_update_trans()` fixed.
Description of function `_epc_firmware_version()` added to the document.
Section 'Interface library basics' enhanced.
Section 'Planned functions' added to the document.
- 2.7 Functions `_set_trigger_state()` and `_get_trigger_state()` corrected for A429EPC.
Description of these functions added to this manual.
- 2.8 Function `_epc_ipreset()` added.
Setup batch files corrected
- 2.9 Bug fixes in functions
`_read_receiver_buffer()`
- double the amount of bytes got transferred to the application which could lead to a application crash because other memory locations got overwritten.
`_activate_receiver()`
- the mask and match values got forwarded to the onboard software in the wrong order.

Setup batch files corrected

- 3.0 bug fix: setup.bat files mkdir sequence corrected (for NT)
new: support for field upgrade of A429IPM firmware added (does not involve Interface library).
- 3.01 bug fixes (only in PC104 version – FW V1.7)
- 3.02 new platform added – 32-bit DLL for IPC01 VME-carrier, _epc_ipreadmemword() – allocated space for returned data could crash the system.
- 3.03 new function: _epc_set_high_resolution() can set timestamp resolution 10 or 100 microseconds. IP-module FW V1.9/IP-carrier FW V1.3
- 3.04 new platform added: HPVISA VXI interface
- 3.1 upgraded this document ; added functions:
- _get_status_word)
 - _get_receiver_data_lost()
 - _reset_data_lost()
 - _reset_clock()
- Current FW-Versions: A429PCC V2.7 / A429IPC: V1.3 [01300208] / A429IPM V2.1
A429PCC PCMCIA-card 32-bit DLL cas95.dll/cas95.lib included.
- 3.20 Installation Procedure/delivery floppy and Installation Guidelines updated
- 3.21 [Feb/Mar 1999] - general clean-up – A429PCC 16-bit world included - upgraded this document – from V3.2 on A429MPC no more supported – A429PCI preparation - now consistently supported platforms for DOS/Windows 9x/Windows NT:
- A429PCC PCMCIA-card [not yet NT]
A429EPC enhanced ISA-board [DOS/WIN9x/NT]
A429PC104 PC/104 board [only DOS]
A429PCI PCI-board [DOS/WIN9x/NT] [preparation]
A429VME VME-board [not yet NT]
A429VXI VXI-board [only NT]
- added functions:
- _get_receiver_data_exist()
Are there any data items in the receive buffer (_get_receiver_data_ready()) returns only true if a buffer is filled).
 - _epc_get_card_ident()
Retrieve the EPC configuration record.
 - all serial RS-xxx support functions
 - _activate_tx_trigger_mode() / _activate_rx_trigger_mode
Functions to start/stop Rx-/Tx-activities triggered by the discrete input line
 - IP-carrier reset functions
 - several more, that have been implemented but not documented
- Current FW-Versions: A429PCC V2.8 / A429IPC: V1.4 [01400429] / A429IPM V2.2
- 3.22 no operational changes - Installation Procedure and delivery floppy for A429PCC updated – common 16-bit and 32-bit System Software floppy – triggered Tx/Rx mode also for A429PCC

- 3.30 **new:** A429PCC PCMCIA-card Windows NT support
- new:** Multi-A429PCC support for Windows 9x
- new:** A429PCI Linux support
- new:** A429xPC (A429PCC and all IP-carriers with A429IPM): special functions introduced that support protocol applications (e.g. ARINC 604 CMC/OMS/CFDS, ARINC 739 MCDU, ARINC 615 Dataloader,...) – the functions are described in a separate document – this document can be obtained on request.
- New function: `_xpc_get_lasterror()` – retrieves error code if `_open_xpc()` fails
- Various corrections in document
-
- 3.32 **new:** A429xPC Index-Mode: A new receive mode can be activated.
In this mode each received Label(/SDI) is the Index in a table where the latest received 32-Bit word + Timestamp is stored and can be retrieved by an application. This mode can run concurrently to the standard Storage-Mode, and in combination with any other mode (Trigger-Mode/Masked-Mode/...)
- This functionality requires A429IPM firmware version V2.5 or upper.
It will be implemented in A429PCC firmware 2.9.
-
- 3.33 A429PCC intermediate Firmware Versions V3.0 and V3.1 (in V3.1 bug in Protocol-Functions fixed – A429IPM V2.6)
- new:** Index-Mode for A429PCC implemented (new in Firmware V3.2 - not in earlier versions)
- Documentation: bug with `_set_trigger_state` fixed + various minor corrections
- Update of Documentation:
- `_define_trigger()`** : With new actions to set/reset discrete output lines
- `_set_trigger_state()`** : Set the trigger state of a discrete output line simultaneously with a trigger ID.
- new Appendix in this document: explanation/screen-dumps/example of **XPCD.c**
- new:** 32-Bit DLLs in PASCAL Calling-Convention (DELPHI) for A429EPC and A429PCC are now available
- Current FW-Versions: A429PCC V3.2 / A429IPC: V1.4 [01400429] / A429IPM V2.7
-
- 3.40 2. April 2001: Introduction of 6-slot IP-carrier PCI-60. (previous define “PCI” is now “PCI40” and “PCI60”).
- Updated KRFTech/Jungo Driver V4.33 for Windows NT and LINUX for PCI-40/60. The LINUX kernel sources are required for proper installation of the driver (independency from LINUX kernel versions are now supported). Installation procedure LINUX driver has changed.
- new** header-file: `a429ipm.h` (included for all platforms where the A429IPM is used)
- new** prototypes for all general purpose IP-module access functions: was `_epc_...` and is now `_ipm_...`
- prototype of function `_ipm_scan_ipm` (formerly `_epc_scan_ipm`) has changed
- new: `_ip429_reset()` resets the TechSAT A429IPM (software), `_ipm_ipreset()` performs a Hardware-Reset (only on A429EPC carrier – not supported by other carriers).
- Current FW-Versions: A429PCC V3.2 / A429IPC: V1.4 [01400429] / A429IPM V2.7

(unchanged).

- 3.41 7. November 2001: no functional changes, new Version V5.03 of Jungo Windriver for A429PCI (now also support for Windows 9x/ME and proper driver installation for Windows 2000).
A429PCC (PCMCIA): driver for Windows 2000
bug fix: masked receive mode did not work properly for A429PCC (PCMCIA)
- 3.42 22. April 2002: Bug fix in PCI40/60 drivers for Windows NT/2000 and Linux, IP scan function didn't work correctly for non-TechSAT modules.
New iflib function `_verify_iflib_version` - helps to detect use of outdated DLLs.
The header file `version.h` has been renamed to `xpc_vers.h` (to avoid name clashes with user files), must be installed in the same directory as the header `a429xpc.h`.
Extensions in preparation for A429IPM-2G, iflib and EPC carrier firmware changes (only affects internally-used functionality).
- 3.43 May 2002: Bug fix in function `_reset_and_selftest` (**[EPC]** platform), it now performs a reset on the IP. NOTE: this bug is still in the obsolete function `_reset_selftest`; this call should be replaced by `_reset_and_selftest`.
- All A429IPM platforms except EPC: IPscan changed so that the revision code of the A429IPM is no longer checked.
- August 2002: Bug fix in the new iflib function used to program A429IPM-2G; this bug only caused problems on W9X.
- 3.50 10. February 2003: **New supported boards:** A429PCI-2G2 (2 IP-slots) and A429PCI-2G4 (4 IP-slots), A429EPC compatible PCI-boards, existing A429PCI board (based on SBS Greenspring PCI-40/60) still supported. Identification for new board: **[PCI-2G]**
minor corrections and clean-up
current firmware versions: A429PCI-2G: V1.03 [01030083]
 A429EPC: V1.4 [01400429]
 A429IPM-2G: V2.9
 A429PCC V3.2
- 3.62 16. September 2000: modification for A429PCI-2G (various intermediate versions) – see Release Notes); new A429PCI-2G driver based on PLX V4.2 (no more Jungo driver); driver improved for faster access (Memory mapped instead of IO-mapped)
Boards require upgrade at TechSAT.
Labview functions activated again.
BUG fix in PCI2G firmware, caused sporadic command errors
current firmware versions: A429PCI-2G: V2.05 [02052174]
 A429EPC: V1.4 [01400429]
 A429IPM-2G: V3.0
 A429PCC V3.2
- 3.63 26. Oktober 2006: the new TechSAT ARINC 429 USB-device A429USB is now part of the family.
- 3.64 19. December 2006: the A429USB is now a non-HID device. With this the performance of the A429USB is similar to the one of A429PCC (PCMCIA)
May/June 2007: switch from A429PCI-2G V3.62 also to V3.64 – new PLX-Driver V4.4.0.0 necessary for A429PMC-Module

Extended functions

Special A429BASTplus related Functions dealing with

- **Data Manipulation**
- **Data Replay**
- **Transmit Functions**

are implemented in A429IPM, A429PCC and A429USB firmware and interface libraries and referenced in header-files but not officially documented – please refer to respective header-files and/or contact TechSAT.

These special functions are documented in ***ARINC 429 EPC/PCI and PCC C-Interface Description Internal Extensions V1.1*** [BSEXTRAS.DOC ;Feb. 22, 1999].

Protocolfunctions (e.g. *send_command_expect_response*) are documented in ***XPC Protocol Extensions*** [PROTOCOL.DOC; Feb. 25, 2000]. This document can be obtained on request.

Preface

This document contains a description of all interface functions for the TechSAT ARINC 429 Interface boards **AA429USB / A429PCC / A429EPC / A429PCI / A429PCI-2G / A429VME / A429VXI** used in the MS-Windows9x/ME™ and MS-WindowsNT/2000/XP™ environment. It also covers the LINUX (Red Hat) shared library for the **A429PCI/A429PCI-2G** board.

The **A429MPC** and the **A429EPC-RxTy** will not be further supported. Although all basic functions still apply and can be used by owners of the **A429MPC** or **A429EPC-RxTy**.

The **A429MPC** is no more referenced in this document.

The interface functions have been compiled under **Microsoft Visual C++ V5.0/V6.0 (for the MS-Windows 32-bit environment)** and are currently available as 32-bit Dynamic Link Library (DLL)/shared library for the Windows 9x and WindowsNT/2000/XP environment.

LINUX driver/shared object was built with **Redhat LINUX V8.0 gcc**.

Please refer to the Hardware User Manuals of the used Interface Board for a more detailed description of the implemented hardware features.

The design of the interface library allows you (with a few restrictions) to switch between the different Operating Systems and interface boards just by relinking your application with the appropriate library.

Due to the different hardware/platform capabilities, not all interface functions are available on all interface boards/platforms. The following notation has been used in this document to outline the availability of an interface function on a particular hardware type/platform:

[xPC]	available/applicable for all Interfaces
[USB]	available on A429USB (USB-device)
[PCC]	available on A429PCC (PCMCIA-card)
[EPC]	available on A429EPC (ISA-card)
[PC104]	available on A429PC104 (PC/104card, FLEX/104 carrier)
[PCI]	available on A429PCI (PCI/cPCI-card, PCI-40/-60/cPCI-200-carrier))
[PCI-2G]	available on A429PCI-2G (PCI-card 2 nd generation)
[VME]	available on A429VME (VME-card, IPC01 or MVME-162 carrier)
[HPVISA]	available on A429IPM using the HPVISA VXI interface

With the various libraries and DLLs a sample program is delivered as well, both in source and executable format. The source file shows you how to use the various interface functions; the executable can be used to verify the correct installation of the hard- and software. The filename of the source file is **xpcd.c**; the name of the executable file varies between the different platforms and board types (epcd32.exe, usb32d.exe, cas95ntd.exe, epcd16.exe, pc104d.exe, etc.). These files will be copied during the installation phase into the *samples* subdirectory of the directory you specified to copy the library files to (e.g. \epc\samples). The executable also might be useful to get familiar with the interface board prior to writing your own application. **xpcd.c** uses a simple menu structure to lead you to the different functions. (see Appendix to this document).

If you have any suggestions to improve this document and/or the software described herein, please send an e-mail message to either of the following addresses:

hjs@techsat.com
ts-info@techsat.com

Bug reports (if any), questions about the products and comments of any kind are welcome too !

ARINC 429 Data Transmission

ARINC 429 Data Transmission Commands

The basic information element is a digital word containing 32-bits, the so called 'ARINC 429 word' herein. The type of information contained in a word is identified by the first eight bits of the word. These eight bits are called 'label'. According to ARINC specification 429, most ARINC 429 words are transmitted repeatedly within a standardized transmit interval.

Before the ARINC 429 controller is able to send data words along the bus, it has to be informed about

- how data should be transmitted (**configuration**)
- what ARINC 429 data should be sent and how often (**transmission definition**)
- when and which data should be sent (**transmission activation & update**)

Configuration commands include the definition of the electrical and logical characteristics of a transmitter channel (high-speed or low-speed, optimized/burst transmit-mode).

Data definition commands provide the set of ARINC 429 words to transmit, their transmission intervals (update-rates) and initial contents.

Transmission control commands start or stop the transmission of one or more defined ARINC 429 words or change the contents of a particular ARINC 429 word.

Both configuration and data definition commands terminate any transmission in progress.

ARINC 429 Transmission Modes

Any particular ARINC 429 data word can be sent in different ways. The most common transmission mode is the unconditional mode. In this mode, data is sent over and over at a rate controlled by the transmit interval value (update rate). In conditional mode, data is sent if and only if an associated flag is set. Those flags are called 'trigger flags' and are kept in the trigger state table. This table has 256 entries (0 through 255 inclusively). A trigger state table entry is identified by its triggerID, which is the index of the trigger state table. More than one ARINC 429 word can refer to the same triggerID. In one-shot mode (update rate = 0), data is sent only once. Updating one-shot data prepares it for another 'shot'. One-shot mode and conditional mode may be combined.

Another aspect of transmission is the gap time between data items. Defining an update rate (see below) does not necessarily imply a timing relationship between data items having identical update rates. Those items may be sent with minimum gap time (burst mode) or maximum gap time (optimized mode) between each other.

Assuming data items (B,C,D) each having the same update rate, and a timeslice tick value determined by another data item (A), data may thus be sent in two different schemes:

Burst mode:

A B C D	A	A	A	A B C D	A	A	A	A B C D	A	A	A
------------------	---	---	---	------------------	---	---	---	------------------	---	---	---

Optimized mode:

A B	A C	A D	A	A B	A C	A D	A	A B	A C	A D	A
--------	--------	--------	---	--------	--------	--------	---	--------	--------	--------	---

Note, that in both cases the update rate is not affected by the relative gap time between the data items.

If an equal distribution is not achievable, a modified optimization scheme is applied:

Burst mode:

A B C D	A	A B C D	A	A B C D	A	A B C D	A	A B C D	A	A B C D	A
------------------	---	------------------	---	------------------	---	------------------	---	------------------	---	------------------	---

Optimized mode:

A B C	A D	A B C	A D	A B C	A D	A B C	A D	A B C	A D	A B C	A B
-------------	--------	-------------	--------	-------------	--------	-------------	--------	-------------	--------	-------------	--------

Update Rate

ARINC 429 data words usually are sent repeatedly at a certain rate, which is called update rate. The reciprocal is the transmit time interval. The controller maintains update rates with a resolution of 1msec, i.e. minimum transmit interval is 1msec. Transmission is controlled by interrupts occurring every <timeslice>msec, with <timeslice> being the greatest common divisor of all transmit time intervals of one transmitter.

The transmission of an ARINC 429 word takes a finite amount of time, therefore at most (<timeslice>/<word transmission time>) data words can be transmitted between two transmit request interrupts. The word transmission time is calculated using the formula

$$\text{word transmission time}[\mu\text{secs}] = \frac{36}{\text{transmission bitrate} [\text{kBit/sec}]}$$

Note : The calculation of the timeslice (interrupts) and the resulting amount of transmittable labels is done separately for one transmitter, and does not affect the second one of an A429IPM/A429PCC/A429USB at all.

A special situation arises, when one-shot data definitions (characterized by an update-rate of zero (0) milliseconds) exist. Because the time when a one-shot item should be sent is unpredictable, this kind of data is treated as having a transmit interval of infinite length. When transmitting, one-shot data is processed last, if and only if there is time left within the timeslice. Hence, one-shot data may not be sent at all, if all timeslices are fully occupied (maximum throughput with all data having the same update rate).

Transmit Data Organization

Before any data can be transmitted, it has to be defined. Data definition includes the data to send, the transmission interval and the transmitter to use. From the host's view any data definition will be referenced using a transmissionID. There are 256 transmissionIDs (0-255) per transmitter.

Data definitions are collected in the data definition list. The data definition list has 256 slots. Its organization is shown below.

Layout of the **data definition list**:

MSB	[16-Bit]	LSB
Assigned trigger-ID		
ARINC 429 low-word		
ARINC 429 high-word		
Transmit interval [msecs]		
Transmit interval [timeslice ticks]		
Timeslice ticks until transmission		
Flags for one-shot mode		
spare		

On entering a new transmit data definition in the data definition list the trigger state pointers (trigger IDs) of all transmit data definitions of the transmitter are assigned to trigger state table entry 0, which is always cleared, thus precluding all data definitions from transmission. When a data definition is selected for transmission, its trigger state pointer is replaced by the pointer to the trigger state supplied by the host. If a data definition is selected for unconditional transmission, its trigger state pointer points to trigger state 255, which is always set, thus enabling transmission.

Every time data definitions are entered, the timeslice value is evaluated. The timeslice value is the greatest common divisor of all transmit time intervals. It is expressed in system ticks (1msec) and is loaded in a timer, which is clocked by the system tick. The timer generates an internal interrupt every <timeslice> ticks.

This interrupt causes the data definitions being processed in the order given by the transmit interval index. A particular data definition is checked if it is ready for transmission. This is the case, whenever its ticks-until-transmission counter reaches zero. If a data definition is eligible for conditional transmission, it contains a pointer to a trigger table entry (triggerID). The state of the trigger table entry controls transmission of a data item. Whenever a data item is eligible for transmission, it is copied to a transmission list.

Its ticks-until-transmission counter is then updated by reloading the corresponding transmission interval value. Before copying to the transmission list, however, a data item is checked for a) whether it is controlled by a trigger event and b) if so, whether the trigger state inhibits this item. If neither is the case, the data is copied.

During transmission the data definition list is accessed in increasing order of transmit intervals. This order ensures, that ARINC 429 words with a high repetition rate are always handled first. This means, that no shift along the time axis occurs, i.e. transmit interval is exact, even if, at a particular point of time, more ARINC 429 words are sent than usual. This may be caused by the coincidence of labels with low repetition rates and those with high repetition rates.

Transmission itself is triggered by a timer interrupt, which occurs at a rate defined by the timeslice value. When this interrupt occurs, the transmitter FIFOs are loaded from the transmission lists, up to eight entries at a time. Once the loading process is triggered by the timeslice-interrupt, it is continued by interrupts issued by the transmitters, signalling readiness for a new set of FIFO data. If a transmission list is exhausted, transmitter interrupts are no longer accepted.

Conditional Transmission, Trigger State Table

The trigger state table is a list of flags, which enable or disable the transmission of data items referencing them. One or more data definitions can reference the same trigger table entry (=flag). A trigger flag is set or reset by some external event. There are three types of events that can modify trigger state table entries:

- A) the fulfillment of a receiver matching criterion
- B) explicit programming by the host.

There are 254 trigger table entries (1-254) that can be used for conditional transmission. All entries are initially zero, thus disabling transmission if being referenced by any data definition. A particular entry is identified by its index called triggerID. Two triggerIDs have a special meaning. They are not meant to be used for conditional transmission.

- triggerID 0 means 'transmit never'. This trigger table entry is always reset and cannot be modified.
- triggerID 255 means 'transmit always', effectively removing a trigger assignment from a data definition. This trigger table entry is always set and cannot be modified

ARINC 429 Data Reception

ARINC 429 Receive Modes

An ARINC 429 receiver may operate in 6 (six) fundamental receive modes and combinations thereof.

- ◆ Storage Mode
- ◆ Trigger Mode
- ◆ Masked Mode
- ◆ Manipulation Mode
- ◆ Protocol Mode
- ◆ Index Mode

Any combination is possible; the only restriction is that Protocol mode is only permitted in combination with Storage and/or Index mode. Masked mode alone is also not permitted. If Masked mode is used together with Index mode, the data removed by the Masked mode filter does not appear in the Indexed data.

Storage mode

Receive buffers are used to store incoming data. One data entry in a receive buffer consists of the 32-bit ARINC data word and a 32-bit timestamp.

In **storage mode**, all data received is stored in receive buffers. Each receiver has its own set of buffers, which size may be set by the host.

On the A429IPM/A429PCC/A429USB each receiver has 16 kByte of buffer-space for receive data, which results in a theoretical storage capacity of 2048 receive items. The buffer-space is divided into pages, which are called receiver buffers. The number of buffers is not fixed as for the A429MPC, but depends on the selected size of the receive buffers, which is determined with the interfacefunction `_set_rec_buf_size()`.

The maximum buffer size is 252 data items (32-bit data + 32-bit timestamp). This limitation is due to the memory window size allocated in the on-module RAM which is used for data exchange with the PC [limitation only for A429PCC, but maintained for A429IPM/A429USB for compatibility reasons].

It should be observed that internally the physical size of the receive buffers is rounded up to the nearest power of 2 of the selected receive buffer size. This is important to know for optimum usage of the internal buffer resources:

Example: User selects receiver buffer size of 100 items with function `_set_rec_buf_size()`. This results in allocation of 16 buffers each with a physical size of 128 entry items. Each buffer is only filled up until the configured receiver buffer size (in this case 100) is reached. This results in a storage capability of $16 \times 100 = 1600$ receive items.

If the user had selected a receiver buffer size of 128, the same number of buffers would have been allocated, and each of them would be completely filled up to 128 receive items. The on-card storage capability in this case is the theoretical limit of 2048 receive items. To determine the actual number of allocated receive buffers, the interface function `_get_rec_buf_count()` may be used. The number

of receive buffers determines the number of consecutive reads necessary to flush all receive buffers.

It is essential to understand, that a particular receiver buffer is only filled up to the configured receiver buffer size. If this happens, the next receiver buffer is used for storing inbound data. In addition, the data ready flag is raised, notifying the host, that at least the requested number of ARINC 429 words have been deposited in the buffer since the last time the host has read the received data. The host reads the data in chunks of at most the configured receiver buffer size. If all receive buffers get full, the receiver buffer holding the oldest data is reused, thus implementing a ring buffer scheme.

Note : It should be noted that reading of receive buffers is totally unsynchronized to filling them up. An application can issue the `_read_receiver_buffer()` function at any time; there is no need to observe the Receive Flags. Upon issuing the `_read_receiver_buffer()` function either the oldest filled-up buffer [if existing] or the currently filling buffer is read.

Flag	Meaning
data exist	this flag indicates that there is at least one data item available. [xPC]
data ready	this flag indicates that there is at least one of the receive buffers filled with data. [xPC]
data lost	this flag indicates that at least one of the receive buffers has been overwritten by the firmware before retrieved by the host application. This flag remains asserted until being reset by the host application. [xPC]

Trigger Mode

In **trigger mode** comparisons are received items (mask/match) are performed. An ARINC 429 word is investigated whether it should cause an internal event. Depending on the result, transmission of selected ARINC 429 words can be started or stopped. Entries of the so-called trigger-table are set/reset according to the trigger matching criterion.

Masked Mode

In **masked mode**, a received ARINC 429 word is processed (by any other Receive-Mode) only if it fulfills a matching criterion defined by the host. Otherwise it is ignored. The matching criterion consists of a 32bit mask pattern and a 32bit match pattern. The ARINC 429 word is ANDed with the mask to strip off the don't-care bits. The result is compared with the match pattern.

Manipulation Mode

The **manipulation mode** is a specific mode that allows for real-time manipulation and re-transmission of incoming data. It is used for the Data Manipulation functionality of A429BASTplus. The **manipulation mode** is not explained in this document – usage is quite complicated, and could confuse standard users.

Explanation and documentation can be obtained on request – please contact TechSAT.

Protocol Mode

A dedicated set of functions specifically support protocol applications like ARINC 739 MCDU, ARINC 604 CMC/OMS/CFDS, ARINC 615 Dataloader and any application with data handshake. For these functions receiver channels are configured to operate in the so called **protocol mode**. The protocol functions are described in a separate document which can be obtained on request – please contact TechSAT.

Index Mode

In the **index mode** each received Label(/SDI) is the index in a table where the latest received 32-Bit word + Timestamp is stored and can be retrieved by an application. This mode can run concurrently to the standard Storage-Mode, and in combination with any other mode (Trigger-Mode/Masked-Mode/...).

Common Specifics & Differences between Interface boards

This section outlines common specifics of all boards and functional differences between the different interface board types. For a more detailed description of the hardware capabilities, please refer to the appropriate Hardware User Manuals.

Time stamping

Interface board type	Resolution
[xPC]	the timestamp resolution can be configured to be either 1 msec or 100 μ sec. ([xPC] 10 μ sec with function <code>_set_high_resolution</code>). 1 msec overrun after 1193 hours 100 μ sec overrun after 119 hours [10 μ sec overrun after 11.9 hours]

ARINC 429 high-speed data word has a length of 360 μ sec, including gap-time. Hence it is possible that three ARINC 429 data words are received within one millisecond, all tagged with the same timestamp (1 msec resolution). Although the order of received data is maintained in the on-card receive buffers, any sorting algorithm used by an application program may distort the order. To avoid rearrangements of the order of received data which are in one millisecond, a second level of time-stamping resolution with 100 μ sec precision was introduced for [xPC].

Applying the function `_set_high_resolution` the 100 μ sec precision can be switched to 10 μ sec.

Backward compatibility 1

In addition to the standard implementation various functions are implemented as well as a so called **(old interface)**. This is for backward compatibility reasons which should enable users of our old A429MPC ISA-board and A429PCC PCMCIA-card to easily port applications to new platforms.

In order to do this it is necessary to define **COMPATIBILITY** in the old C-sources or make-file [see A429XPC.H].

In order to obtain compatibility for the future we recommend to avoid usage of **(old interface)** functions.

Backward compatibility 2: PCI-60: 16 Tx-/24 Rx-Channels

In all functions with Rx- or Tx-Numbers as parameters characters are used. The numbering was quite obvious until the PCI-60 with 12 Tx- and 24 Rx-channels has

been introduced. For backward compatibility reasons we did not change the parameter type - see below the assignment for Rx16 to Rx23 for the PCI-60:

Rx15 -	'f'
Rx16 -	'g'
Rx17 -	'h'
Rx18 -	'i'
Rx19 -	'j'
Rx20 -	'k'
Rx21 -	'l'
Rx22 -	'm'
Rx23 -	'n'

ARINC 429 Driver Power Down Mode [PCC]

The A429PCC allows to set the ARINC 429 line drivers in a power-down mode with minimum power consumption. This is done automatically from within the firmware and requires no user interaction.

Parity Generation [PCC][USB]

The ARINC 429 transceiver chips used on [PCC] and [USB] allow to disable the automatic parity generation. It's possible to turn parity generation off which results in bit 32 being transmitted as defined by the host, thus allowing to inject single parity errors in a transmit schedule. See function `_set_parity_mode()`.

System clock

[xPC] offer the possibility to reset the internal system clock either via interface function or hardware interrupt (discrete input line INPCC – except [USB]), thus allowing a synchronization of the timestamps across interface boards.

Wrap-around relays [USB][PCC][USB][EPC][PCI-2G]

[USB], [PCC], [EPC] and [PCI-2G] are equipped with on board wrap-around relays in order to establish Tx-Rx loop connections.

Loop connections can be used for selftest purposes. Whenever a loop connection is established, the corresponding receiver is disconnected from the outside world, thus allowing a selftest to be performed without the need for removal of any external signal sources.

Note : Looping back a transmitter to a not used receiver allows to determine the precise absolute instant when data has been transmitted. This mechanism can be used for comparison with receive times of data from a UUT, and to get an idea of its required time to react upon certain stimuli.

This function is very useful for verifying timing behavior in any handshaking-type communication between LRUs and [xPC] applications !

All loops are used during reset selftest in order to verify that transmitted data is received back. This is to verify proper function of the ARINC 429 line drivers. For all boards except [USB], [PCC], [EPC] and [PCI-2G] a selftest adapter has to be

used or the result of the external loops test in function `_reset_selftest()` has to be ignored.

Interface library basics

How to address the interface board

All interface functions, except for **_open_xpc()** have a variable of type **HANDLE** as first argument. Depending on the used operating system and/or interface board, this variable may have to following values:

MS-DOS/Windows[16-bit]- A429PC104

The value for **HANDLE** is the basic I/O port address jumpered on the PC/104 carrier board (0x300, 0x320 or 0x340).

MS-DOS/Windows[16-bit]- A429PCC

The value for **HANDLE** is the memory segment address ranging from 0xC000 to 0xFF00, in steps of 0x200. **This value must be the same as of the loadcard.ini in the appropriate directory.**

MS-DOS/Windows[32-bit]- A429PCC/A429PCI/A429VME

The value for **HANDLE** is the return value of the **_open_xpc()** call.

MS-DOS/Windows[16-bit and 32-bit] - A429EPC

The value for **HANDLE** is the basic I/O port address as jumpered on the board, ranging from 0x100 to 0x3f0, in steps of 0x10.

WindowsNT/2000/XP – A429USB/A429PCC/A429PCI/A429PCI-2G/A429EPC/A429VME/HPVISA

The value for **HANDLE** is the return value of the **_open_xpc()** call.

Even though available for all operating system platforms (for compatibility reasons), the **_open_xpc()** and **_close_xpc()** function calls are only relevant under **WindowsNT/2000/XP** for all boards and for 32-bit applications under **Windows 95/98/ME** for **[USB]**, **[PCC]**, **[PCI]**, **[PCI-2G]**, **[VME]** and **[HPVISA]**.

Headerfiles to be used

A number of C header files is delivered with the various forms of the interface library:

Header filename	Content
a429xpc.h	All definitions common to all of the available interface board types [xPC]
a429ipm.h	Definitions for all platforms where the A429IPM IP-module is used [PC104], [EPC], [PCI], [PCI-2G], [VME], [HPVISA]
a429usb.h	Definitions unique to the A429USB USB-device [USB]
a429epc.h	Definitions unique to the A429EPC ISA-bus Interface board [EPC]
a429pci2g	Definitions unique to the A429PCI-2G PCI interface board [PCI-2G]
a429pci.h	Definitions unique to the A429PCI PCI-bus interface board [PCI]
a429vme.h	Definitions unique to the A429VME VME Interface board [VME]
hpvisa.h	Definitions unique to the A429IPM using the HPVISA VXI interface [HPVISA]

The application programmer only has to include the headerfile appropriate to the board type used (e.g. a429epc.h), the common header a429xpc.h file will be included through the boardtype specific headerfile.

Definitions to be used to compile your application

Depending on the target operating system platform you compile your application for and the type of Hardware you are using, you have to use each one of the following definitions:

Operating System:

DOS	Application is to be build for the MS-DOS operating system
WIN16	16-Bit Application is to be build for the MS-WINDOWS 95/98 operating system
WIN9x	32-Bit Application is to be build for the MS-WINDOWS 95/98 operating system Note: in addition to WIN9x it is mandatory to define WIN32 !!!
WIN32	Application is to be build for the MS-WINDOWS 95/98 or WINDOWS-NT/2000/XP operating system
LINUX	Application is to be build for the Linux operating system (only [PCI])
XPASCAL	DLL/LIB is in PASCAL Calling-Convention

Hardware:

USB	Application is to be build for the A429USB USB-device
PCC	Application is to be build for the A429PCC PCMCIA-card
EPC	Application is to be build for the A429EPC ISA-board
PCI2G	Application is to be build for the A429PCI-2G board (2 or 4 IP-slots)
PCI40	Application is to be build for the A429PCI PCI-40 board (4 IP-slots)
PCI60	Application is to be build for the A429PCI PCI-60 board (6 IP-slots)
IPC01	Application is to be build for the A429VME VME-board
HPVISA	Application is to be build for the the A429IPM using the HPVISA VXI interface

Libraries/DLLs to be used to build your application

Depending on the target operating system platform you intend to build your application for, you've to use the appropriate .lib/.dll files in the link stream. The names of the libraries distributed reflect the interface board type and the target platform.

The basic template for the library name has the following format: **xxxnn.lib**, where

xxx is one of **epc**, **usb**, **pcc**, **pci** or **pci2g** denoting the type of interface board

nn is one of **16**, **9x** or **32**, denoting the target operating system platform
16 for all 16-bit applications (DOS, WINDOWS 3.x/95/98) and
9x for all 32-bit applications (WINDOWS 95/98)
32 for all 32-bit applications (WINDOWS-NT/2000/XP)

All **xxxnnp.lib/.dll** are in PASCAL Calling-Convention.

Note: The A429PCC 32-bit library/DLL for Windows95/98 and Windows NT/2000/XP is called

cas95.lib/cas95.dll

and is located on the 32-bit A429xPC C-Interface Software floppy.

Exceptions: ipc01.lib/dll – for A429VME
hpvisa.lib/dll – for A429IPM using the HPVISA VXI library

Basic ARINC 429 interface functions

_open_xpc() [all except HPVISA and VME]

Synopsis HANDLE _open_xpc(lbn)

Description Opens an ARINC-429 **[xPC]** device and returns a handle to it. This function has to be used before any of the other functions can be used **[WindowsNT/2000/XP with A429EPC and Windows 95/98 for A429PCC, A429USB only]**. On other operating system platforms this function will return the value of *lbn*.

Argument	Explanation
short lbn	Logical board number in the range 0 to 9. The logical board number reflects the devicename shown in the registry [NT/2000/XP only] (e.g. to access the device with device name epc2 you have to use the logical board number 2). Note: For the A429PCC/A429USB with WIN9x and WIN32 always use lbn = 0 !!! _open_xpc() scans the system and retrieves the handle for the first detected A429PCC/A429USB If more than one A429PCC are installed use the following assignments: lbn = 1 : PCMCIA-slot 0 / USB 0 lbn = 2 : PCMCIA-slot 1 / USB 1 lbn = 3 : PCMCIA-slot 2 / USB 2 lbn = 4 : PCMCIA-slot 3 / USB 3

Return Value	Explanation
INVALID_HANDLE_VALUE	The function returns a handle to the opened device or if the device couldn't be accessed [WindowsNT/2000/XP with A429EPC and Windows 95/98 for A429PCC, A429USB only] Use function <i>_xpc_get_lasterror()</i> to retrieve reason for INVALID_HANDLE_VALUE For portability reasons, on other operating system platforms this function will return the value of its argument <i>lbn</i> .

See also *_close_xpc()*, *xpc_get_lasterror()*

`_open_xpc()` [HPVISA]

Synopsis `HANDLE _open_xpc(handle, vxibus, vxiaddr)`

Description Opens an ARINC-429 IPM device and returns a handle to it. This function has to be used before any of the other functions can be used.

Argument	Explanation
<code>HANDLE handle</code>	Either the handle of a previous call to <i>open_xpc()</i> or 0 (zero) for the first call.
<code>short vxibus</code>	VXIbus number
<code>short vxiaddr</code>	VXIbus address of device
Return Value	Explanation
	The function returns a handle to the opened device or
<code>INVALID_HANDLE_VALUE</code>	if the device couldn't be accessed.

See also `_close_xpc()`

Note For the first call to *open_xpc()* you have to specify 0 (zero) as handle to force the interface library function to perform a *viOpendefaultRM()* call.
Please use the returned handle as argument for all successive calls to *open_xpc()*.

`_open_xpc()` [VME]

Synopsis `HANDLE _open_xpc(base16, base32, ipmemsize, ivec, ilevel)`

Description Opens an ARINC-429 VME device/board and returns a handle to it. This function has to be used before any of the other functions can be used.

Argument	Explanation
unsigned long base16	board 16-bit base address of IPC01 carrier board (see HW Manual) Range: 1000H-F000H
unsigned long base32	A32 base address
long ipmemsize	A429IPM memory size – 8Mbyte required for A429IPM
int ivec	VME Interrupt vector
int ilevel	VME Interrupt level
Return Value	Explanation
	The function returns a handle to the opened device or
XPC_SUCCESS	if the device couldn't be accessed. (should be INVALID_HANDLE_VALUE – wrong implementation)

See also `_close_xpc()`

Note: please refer to IPC01 HW User Manual

`_close_xpc()` [xPC]

Synopsis `int _close_xpc(hmpc)`

Description Closes the connection to a A429xPC device, previously opened by a call to `_open_xpc()`.

Argument	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call

Return Value	Explanation
XPC_SUCCESS	on success
XPC_FAILED	on failure

See also `open_xpc()`

`_xpc_get_lasterror()` [xPC] (currently only [PCC][USB])

Synopsis `int _xpc_get_lasterror()`

Description Retrieves an error-code that identifies the reason why `_open_xpc()` failed.

Argument	Explanation
[none]	

Return Value	Explanation
for A429PCC / A429USB :	
XPC_ERROR_HANDLE	logical board number not supported
XPC_INIPC_INITBOARD	error from VARINCD.DLL (Softing) ([PCC] only)
XPC_CANPC_RESET	error on PCMCIA, USB reset
XPC_PCC_INIT	error on PCMCIA, USB initialisation

See also `open_xpc()`

_config_trans() [xPC]

Synopsis `int _config_trans(hmpc, tx_number, speed, bitrate, parity)`

Description Sets the basic operating characteristics for a transmitter channel.

Argument	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
char tx_number	The number of the transmitter number to configure, range: '0'..'1' [PCC][USB][HPVISA] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
unsigned short speed	Selects the basic speed the transmitter shall operate in 0 = high speed 1 = low speed
unsigned short bitrate	obsolete – only for backward compatibility to A429MPC
unsigned short parity	Selects the parity the transmitter shall generate: 0 = odd parity 1 = even parity
Return Value	Explanation
XPC_SUCCESS	Transmitter successfully configured
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_ERROR_SPEEDSEL	Invalid speed selected
XPC_ERROR_BITRATE	Wrong bitrate selected
XPC_ERROR_PARITY	Wrong parity selected
XPC_FAILED	Configuration failed for an unknown reason.

Note On selection of the transmit speed the transmit waveform slope (5V/usec for HS and 1 V/usec) is automatically adjusted.

Any transmission activities on the affected channel will be terminated immediately. All transmit data definitions for the respective transmitter are lost.

This command is optional.

Unless overridden by this command, default transmitter configuration is

- low speed (12.5 kBit/sec)
- odd parity
- slope 1 V/*sec

`_set_parity_mode()` [PCC][USB] only

Synopsis `int _set_parity_mode(hmpc, tx_number, unsigned int parity)`

Description With this command it is possible to switch-off/on the automatic parity generation capability of the A429PCC. It can be used for error injection purposes.

Argument	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
char tx_number	The number of the transmitter number to configure, range: '0'..'1' [PCC][USB]
unsigned int parity	Defines the parity-mode: 0: enable parity generation 1: disable parity generation
Return Value	Explanation
XPC_SUCCESS	Transmitter successfully configured
XPC_FAILED	Configuration failed for an unknown reason.

Note The ARINC 429 transceiver chips allow to disable the automatic parity generation as it is default for the selected type of chip. It is possible to switch off the parity generation, which results in Bit 32 of transmit data being transmitted as defined by the host. This allows to inject single parity errors in a transmit schedule, whereas a channel parity configuration would be valid for all transmit data.

_set_tx_mode() [xPC]

Synopsis `int _set_tx_mode(hmpc, tx_number, mode)`

Description Sets the transmit mode of a certain transmitter to optimized or burst mode

Arguments	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
char tx_number	Transmitter number, range '0'..'1' [PCC][USB][HPVISA] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
unsigned short mode	Mode selector: 0 = burst mode: minimum gap time between transmissions with the same update rate 1 = optimized mode: equidistant gap time over all transmissions with the same update rate.
Return Value	Explanation
XPC_SUCCESS	Mode set successfully
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_FAILED	Mode set failed for an unknown reason.

Note This command is used to set a transmitter either in time-optimized- or in burst transmission mode.

It can be executed at any time, and has no impact on any transmit data definitions and transmit activities (except their relative timing).

After changing the transmission mode, some update intervals may be distorted for a short time. Transmission resumes a stable timing after the longest specified update interval, maintaining all defined update-rates.

In time-optimized transmit mode the transmitter tries to expand the gap time of ARINC words with the same update rate until all time slices are equally loaded, but still maintaining the update rates.

Note : The optimized transmit-mode will only work satisfactory when the resulting time-slice of the transmitter is less (any factor greater than 1) than the update-rates of the labels whose transmission-timing (the gap-time in between them) should be expanded (see also chapter 'Transmission Modes' in Hardware User Manual of respective Board).

`_get_transmitter_set()` [xPC]

Synopsis `int _get_transmitter_set(hmpc, tx_number, &buffer)`

Description Returns the basic operating settings of a certain transmitter channel.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
char tx_number	The transmitter number, range '0'..'1' [PCC][USB][HPVISA] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
tx_set_t *buffer	Pointer to structure of the format: { unsigned short bitrate_par_slope_set; unsigned short amplitude_set; } tx_set_t; The bit designation of bitrate_par_slope_set is as follows: bit 15..9: don't care bit 8: parity generation 0 = disabled 1 = enabled bit 7: transmit mode 0 = normal burst mode 1 = time optimized mode bit 6..5: slope 00 = 5 V/us [xPC] = high speed 10 = 1.0 V/us [xPC] = low speed bit 4: speed 0 = high 1 = low bit 3: parity 0 = odd 1 = even bit 2..0: bitrate in kBit/sec for low speed / high speed 001 = 12.50 / 100.0 [xPC] amplitude_set contains the amplitude with a resolution of 0.1 mV LSB
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_FAILED	Execution of command failed

See also *get transmitter mode* in the appropriate Hardware User Manual.

`_define_trans()` [xPC]

Synopsis `int _define_trans(hmpc, tx_number, count, &buffer, &result, ×lice)`

Description Defines transmissions by assigning transmission IDs and ARINC 429 32-bit data with certain transmit intervals to a transmitter

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
char tx_number	Transmitter number, range '0'..'1' [PCC][USB][HPVISA] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
unsigned short count	Number of definitions in <i>buffer</i> (1..256)
tx_definition_t *buffer	Pointer to an array of type tx_definition_t buffers with the following layout: { unsigned short transmission_ID; unsigned short data_low; unsigned short data_high; unsigned short transmit_interval; } tx_definition_t; transmission_ID = range of 0..255 data_low = the two least significant bytes of the ARINC 429 word data_high = the two most significant bytes of the ARINC 429 word transmit_interval = in milliseconds, a value of zero denotes a 'one-shot'
unsigned short *result	Result information regarding the definition: 0 = definition accepted and inserted 1 = too many definitions for timeslice 2 = resulting definition count is zero
unsigned short *timeslice	The calculated timeslice in milliseconds
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware - see <i>result</i> for further information
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_ERROR_DEFCOUNT	Count out of range
XPC_FAILED	Execution of command failed

See also *define transmit data* in the appropriate Hardware User Manual.

Note

Transmit data definition tells the controller, what data should be sent and how often (repetition rate). This set of information is assigned a unique number between 0 and 255, referred to as transmissionID. Each transmitter has its own set of 256 transmissionIDs. If more than one definition is given with the same transmissionID, the last one encountered takes precedence, thus overwriting any previous definition.

Entering a data definition immediately stops transmission on the affected channel. Transmission is started by selecting particular data definitions. Once transmission is started, data is sent every <transmit interval> milliseconds. If the transmit interval is zero, the data definition is a one-shot definition, i.e. data is sent only once. After sending an one-shot item, its transmission is inhibited, independent from its associated trigger state, if any.

Every time a data definition is submitted to the controller, the timeslice value is evaluated and can be accessed.

The timeslice value is the greatest common divisor of all transmit intervals. For maximum throughput, all update rates should be integer multiples of each other. Because data transmission is interrupt driven inside the controller, the timeslice value determines the rate, at which these interrupts have to occur to maintain all required update rates. The timeslice value also limits the number of data words, that can be sent between two consecutive timeslice interrupts. This number is calculated as follows:

$$\text{word transmission time[usecs]} = \frac{(\text{word length[bits]} + \text{gap time[bits]} \times 1000)}{\text{transmission bitrate [kBit/sec]}}$$

$$\text{number of ARINC429 words within timeslice} = \frac{\text{timeslice[usecs]}}{\text{word transmission time[usecs]}}$$

When entering data definitions, the A429IPM on-module CPU checks, whether above conditions are met. It does so by summing the word transmission times of all data definitions ignoring one-shot data. For this calculation the true word transmission times of the transmit data, taking into respect the speed, are used. This sum should be less than or equal the timeslice value. Special care should be exercised when mixing one-shot data and continuous data. The controller tries to sent one-shot data whenever it has time to do so. This is always the case, if repetitive data with different update rates are defined, because from time to time there exists a timeslice, that is not completely occupied. If all repetitive data have an equal update rate, however, all timeslices are equally loaded. When operating at maximum throughput, no one-shot data is sent at all. To guarantee transmission of one-shot data, it is sufficient, to define at least one update rate, that is different from the others.

On this command any transmission activities on the affected channel will be immediately terminated. To any already defined transmission ID of the transmitter the trigger ID zero (0) is assigned, that inhibits transmission.

Important :

If the number of transmission definitions does not fit the calculated timeslice (i.e. too much definitions for the timeslice; impossible to guarantee the observance of the defined update-rates) it is absolutely necessary to delete transmission definitions (see command 'delete transmit data') until the number fits again.

Activation of transmit data definitions in a state where too much transmit data have been defined will not guarantee correct transmission of all definitions. Definitions with the biggest transmit time interval will either not be transmitted, or with an incorrect update.

_activate_trans() [xPC]

Synopsis `int _activate_trans(hmpc, tx_number, count, &buffer)`

Description Activates and assigns trigger IDs to transmission IDs of a certain transmitter

Arguments	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
char tx_number	The transmitter number, range '0'..'1' [PCC][USB][HPVISA] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
unsigned short count	Activation count (1..256)
tx_activation_t *buffer	Pointer to array of tx_activation_t structures with the following layout: { unsigned char transmission_ID; unsigned char trigger_ID; } tx_activation_t;
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware – successful transmission activation
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_ERROR_ACTCOUNT	Count out of range
XPC_FAILED	Execution of command failed

Note Any data defined will not be transmitted until explicitly activated for transmission. A previously defined data entry (identified by its transmissionID) is activated by assigning a triggerID between 0 and 255 to it. The default triggerID zero (0) (assigned automatically when defined) deselects (inhibits) data transmission. A triggerID of 255 selects it for unconditional transmission. Any other triggerID causes conditional transmission, controlled by the corresponding trigger state table entry.

At any time it is possible to assign any trigger ID to a certain transmission ID. Any number of transmission ID's can be assigned to one trigger ID.

Attention:

The attempt to activate a transmissionID, for which no data definition exists, is ignored.

Re-selection of a one-shot data definition (update rate = 0) causes this data to be sent once again.

See also *activate transmit data* in the appropriate Hardware User Manual

_delete_trans() [xPC]

Synopsis `int _delete_trans(hmpc, tx_number, count, &buffer, &result, timeslice)`

Description Deletes transmission definitions of a certain transmitter

Arguments	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
char tx_number	Transmitter number, range '0'..'1' [PCC][USB][HPVISA] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
unsigned short count	Number of deletions (1..256)
tx_delete_t *buffer	Pointer to array of structures with the following layout: { unsigned short transmission_ID; } tx_deletion_t; All the given transmission IDs will be removed from the transmission definition list.
unsigned short *result	Result information regarding the deletion: 0 = deletion accepted 1 = too many definitions for timeslice 2 = resulting definition count is zero
unsigned short *timeslice	The calculated timeslice in milliseconds
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware - see <i>result</i> for further information
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_ERROR_DELCOUNT	Count out of range
XPC_FAILED	Execution of command failed

Note This command causes specified transmit data definitions to be removed from the internal tables. It immediately stops transmission on the affected channel, setting all trigger ID's to zero (0).

Upon execution of this command remaining transmit data definitions are reorganized; the timeslice is recalculated. The same mechanisms as with the command 'define transmit data' are performed; the same considerations must be taken into account..

Attention:

Any transmission activities on the affected channel will be immediately terminated.

See also *delete transmit data* in the appropriate Hardware User Manual.

_update_trans() [xPC]

Synopsis `int _update_trans(hmpc, tx_number, count, &buffer)`

Description Updates the data values of already defined ARINC-429 transmissions

Arguments	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
char tx_number	The transmitter number:, range '0'..'1' [PCC][USB][HPVISA] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
unsigned short count	Number of definitions in <i>buffer</i>
tx_update_t *buffer	Pointer to array of structures holding the new ARINC 429 values: { unsigned short transmission_ID; unsigned short data_low; unsigned short data_high; unsigned short dont_care; } tx_update_t; transmission_ID = ID of the transmission whose data value should be updated data_low = the two least significant bytes of the ARINC 429 word data_high = the two most significant bytes of the ARINC 429 word dont_care = reserved for future use.
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware – successful Transmission value(s) update
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_ERROR_ACTCOUNT	Count out of range
XPC_FAILED	Execution of command failed

Note The ARINC 429 word of a transmit data definition can be modified at any time, regardless of its transmission state. The immediate next transmission is performed with the new ARINC 429 word. If multiple entries for the same transmissionID are given, the last one encountered takes precedence. Updating one-shot data does not automatically re-select it for transmission. This is accomplished exclusively using the command *activate transmit data*.

Updating data does not affect trigger state, i.e. the new data is not transmitted until the associated trigger state is set.

Data for which no data definition was specified before is ignored.

See also *update transmit data* in the appropriate Hardware User Manual.

`_activate_tx_trigger_mode () [xPC]`

Synopsis `int _activate_tx_trigger_mode(hmpc, tx_number, act_code)`

Description Activates the discrete I/O triggered start/stop mode for a selected transmitter.

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
char tx_number	The transmitter number:, range '0'..'1' [PCC][USB][HPVISA] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
unsigned short act_code	activation code: (not [USB]) 0: de-activate triggered TX-mode 1: activate triggered TX-mode
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_FAILED	Execution of command failed

Note This function activates or de-activates the 'start/stop TX on trigger' mode. When in this mode, calls that normally start the transmitter only prepare the transmitter for starting; the actual start is performed when the IPMx_IN/INPCC signal next changes.

If the IPMx_IN/INPCC signal subsequently changes back to its original state, the transmitter will be stopped.

This function can be used to precisely synchronize transmit activities with external events.

See also `_activate_rx_trigger_mode` in this document

`_get_tx_definitions()` [xPC]

Synopsis `int _get_tx_definitions(hmpc, tx_number, &buffer, &result, &ndef)`

Description Retrieves the transmit definitions for a certain transmitter

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
char tx_number	The transmitter number, range '0'..'1' [PCC][USB][HPVISA] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
read_tx_definition_t *buffer	Pointer to an array of structures with the following layout: { unsigned char transmission_ID; unsigned char trigger_ID; unsigned short data_low; unsigned short data_high; unsigned short transmit_interval_millis; unsigned short transmit_interval_timesl; unsigned short ticks_til_transmission; unsigned short transmission_flags; unsigned short space_word; } read_tx_definition_t;
unsigned short *result	0 = transmission definitions retrieved 1 = no transmission definitions available
unsigned short *ndef	number of definitions retrieved from board
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware - see <i>result</i> for further information
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_FAILED	Execution of command failed

Note Using this command, all transmit data definitions can be read back. The complete transmit data definition list is read (see chapter Transmit Data Organisation in Hardware User Manual).

Due to the fact, that transmit data definitions are read asynchronously with respect to the transmission process, the ticks-until-transmission value is an approximate value only.

Attention : The user is responsible to provide enough memory for the maximum number of transmit data definitions in his application.

See also *get all transmit data definitions* in the appropriate Hardware User Manual.

_get_single_tx_definition() [xPC]

Synopsis `int _get_single_tx_definition(hmpc, tx_number, trans_ID, &buffer, &result)`

Description Retrieves a single transmit data definition of a certain receiver

Arguments	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
char tx_number	The transmitter number, range '0'..'1' [PCC][USB][HPVISA] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
unsigned short trans_ID	The transmission ID (0..255) for which the definition should be retrieved
read_tx_definition_t *buffer	Pointer to buffer with the following layout: { unsigned char transmission_ID; unsigned char trigger_ID; unsigned short data_low; unsigned short data_high; unsigned short transmit_interval_millis; unsigned short transmit_interval_timesl; unsigned short ticks_til_transmission; unsigned short transmission_flags; unsigned short spare_word; } read_tx_definition_t;
unsigned short *result	0 = transmit data definition copied to buffer 1 = no definition available for this transmission ID
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware - see <i>result</i> for further information
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_ERROR_TX_ID	Invalid transmission ID specified
XPC_FAILED	Execution of command failed

Note: Using this command, a single transmit data definitions, identified by its transmission ID can be read back. One entry of the transmit data definition list is read (see chapter Transmit Data Organisation in Hardware User Manual).

See also *get single transmit data definition* in the appropriate Hardware User Manual.

_get_tx_timeslice() [xPC]

Synopsis `int _get_tx_timeslice(hmpc, tx_number, ×lice)`

Description Returns the current timeslice value of a certain transmitter.

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
char tx_number	The transmitter number, range '0'..'1' [PCC][USB] [HPVISA] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
unsigned short *timeslice	The returned timeslice value in milliseconds. A value of 0 (zero) denotes that there hasn't been a timeslice value calculated for this transmitter. This means that no cyclic transmit definitions have been defined for this particular transmitter.

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_FAILED	Command rejected

See also *get transmitter timeslice* in the appropriate Hardware User Manual

Note If only one-shot data are defined for a transmitter the returned timeslice value is the required transmit time for all defined one-shot data.

`_config_rec()` [xPC]

Synopsis `int _config_rec(hmpc, rx_pair, speed)`

Description Configures the speed for the given receiver pair.

Argument	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
char rx_pair	The number of the receiver pair to be configured, range: '0'..'1' [PCC][USB][HPVISA] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME] '0' == receiver 0 and 1, '1' == receiver 2 and 3, etc.
unsigned short speed	The speed, the receiver pair shall operate in: 0 = high speed 1 = low speed

Return Value	Explanation
XPC_SUCCESS	Receiver configured successfully
XPC_ERROR_RX_NUMBER	Invalid receiver pair number specified
XPC_ERROR_SPEEDCODE	Invalid speed selected
XPC_FAILED	Configuration failed for another, unknown reason.

Note This command selects the speed mode of a receiver pair and initializes it. Receivers 0 and 1 form one pair, receivers 2 and 3 the second pair of receivers, and so on. Due to hardware-reasons (ARINC 429 transceiver chip) it is not possible to select the speed of single receivers.

Data reception will be disabled upon execution of the command, until re-enabled by appropriate instructions. The receivers are no more active. Any data in the receiver buffers is lost. The size of the receiver buffers is reset to zero (0).

This command is optional. Unless overridden by this command, default receiver speed mode is low speed (see also command 'configure transmitter').

`_set_rec_buf_size()` [xPC]

Synopsis `int _set_rec_buf_size(hmpc, rx_number, size, &result, &status)`

Description Sets the buffer size for a certain receiver

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
char rx_number	The receiver number: '0'..'3' [PCC][USB][HPVISA] '0'..'7'['b'] [PC104] '0'..'9','a'..'f'['n'] [EPC][PCI][PCI-2G][VME]
unsigned short size	Denotes the size of the receiver buffer and thus the number of ARINC-429 words after which the <i>data ready</i> flag is set (range: 0..252[xPC]). A size of 0 (zero) forces the receiver to operate in non-storage mode.
unsigned short *result	0 = size accepted 1 = given size out of range - previous definition remains unchanged
unsigned short *status	Shows the mode of operation for the receiver: bit 0 set: receiver operates in trigger mode bit 1 set: receiver operates in storage mode bit 2 set: receiver operates in masked mode
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware - see <i>result</i> and <i>status</i> for further information
XPC_ERROR_RX_NUMBER	Invalid receiver number specified
XPC_FAILED	Execution of command failed

Note This command initializes the receive buffers and determines the amount of receive items for one receiver, after which 'data ready' (see command *get data ready*) is set. As a consequence of the size of the receive buffers also the number of allocated receive buffers is determined (see section ARINC 429 Data Reception)

The maximum size of each receive buffer is 252 receive items, corresponding to 253 32-bit ARINC 429 words plus 32-bit timestamp. The maximum number of receive buffers per receiver is 8, each with the maximum size of 252 receive items.

Each receiver has its set of buffers, whose size may be set by the host. These buffers hold the ARINC 429 data word and the time of reception (in milliseconds resp. tenths of milliseconds) since the last card reset, or on-card clock reset.

On performing this command any data already stored in the receiver buffer is purged. To prevent data loss, this command should only be given, if the receiver is stopped.

Performing the command 'configure receiver' resets the receiver's buffers size and number of buffers to zero (0).

Reading out the contents of the receiver buffers is independant from the size, but the maximum number of items in the buffers, that can be read at one time can never exceed the receiver buffers size. On performing the function `_read_receiver_buffer` always the buffer with the oldest data is read.

Attention : The user must provide at least as much memory for reading out the receiver buffers (see command 'read receiver buffer') as he had set the respective receiver buffer size.

Changing the receiver buffers size to zero (0) causes the receiver to work in non-storage mode, e.g. no data that is received will be stored in the receiver buffers, unless they can set/reset trigger ID's (see command 'define trigger events').

See also *set receiver buffer size* in the appropriate Hardware User Manual and section *On-card receive buffers* in this document.

`_get_rec_buf_count()` [xPC]

Synopsis `int _get_rec_buf_count(hmpc, rx_number, &nbuf, &bsize)`

Description Retrieve number of receiver buffers and its size.

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
char rx_number	The receiver number: '0'..'3' [PCC][USB][HPVISA] '0'..'7'['b'] [PC104] '0'..'9','a'..'f'['n'] [EPC][PCI][PCI-2G][VME]
unsigned short *nbuf	Number of receiver buffers [xPC]
unsigned short *bsize	Actual buffersize in number of data items [xPC]

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_RX_NUMBER	Invalid receiver number specified
XPC_FAILED	Execution of command failed

See also *set receiver buffer size* in the appropriate Hardware User Manual and the `_set_rec_buf_size()` in this manual. See also section *On-Card Receiver Buffers*.

Note This command retrieves the number of receive buffers which have been allocated due to the receiver buffer size determined by command *set receiver buffer size*.

Additionally to the number of buffers the size of the buffers is returned. The number of buffers is important to know when all received data up to a specific point in time should be retrieved.

The buffer size determines the setting of the receiver data ready flag. When at least one buffer has been filled up to the receiver buffer size the data ready flag is set.

`_read_receiver_buffer()` [xPC]

Synopsis `int _read_receiver_buffer(hmpc, rx_number, &buffer, &result, &nentries)`

Description Retrieves received data from the board.

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
char rx_number	The receiver number, range '0'..'3' [PCC][USB][HPVISA] '0'..'7'['b'] [PC104] '0'..'9','a'..'f'['n'] [EPC][PCI][PCI-2G][VME]
rx_data_t *buffer	Pointer to array of structures with the following layout: { unsigned long receive_time; unsigned short arinc_low; unsigned short arinc_high; } rx_data_t; Note: The dimension of buffer should be big enough to hold the number of items defined in the <code>_set_receiver_buffer()</code> call.
unsigned short *result	0 = data available and stored in buffer 1 = no data available
unsigned short *nentries	number of elements stored in buffer
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware - see <i>result</i> for further information
XPC_ERROR_RX_NUMBER	Invalid receiver number specified
XPC_FAILED	Execution of command failed

See also *read receiver buffer* in the appropriate Hardware User Manual.

Attention If the size of **buffer** is too small, the results will be unpredictable.

Note When this command is given, the buffer with the oldest data of the requested receiver is copied into the data exchange buffer ([PCC] Dual Port RAM) and from there copied into the user-defined PC-memory. The internal buffer (the respective region only) is flushed. The data ready flag, if set, is cleared if no other filled buffer is available for this receiver. It remains set if any other receive buffer for this receiver is full.

The `_read_receiver_buffer()` function can be called at any time, independent from the state of the data ready flag.

The maximum amount of data read at a time is the receiver buffer size, determined with the command 'set receiver buffer size'. The user is responsible to provide sufficient memory for reading the receive data, to avoid any overflow.

If no data is available the number of entries is zero (0), and the data exist flag is cleared.

_activate_rec() [xPC]

Synopsis `int _activate_rec(hmpc, rx_number, act_code, &buffer, &status)`

Description Sets the receive mode for a certain receiver

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call
char rx_number	The receiver number, range '0'..'3' [PCC][USB][HPVISA] '0'..'7' [PC104] '0'..'9','a'..'f' [EPC][PCI][PCI-2G][VME]
unsigned short act_code	Activation code: 0 = stop receiving 1 = start unmasked receiving 2 = start masked receiving 3 = continue in previously selected mode
rx_mode_t *buffer	Pointer to structure with the following layout (used only in conjunction with act_code = 2 masked receiving) : { unsigned short mask_low; unsigned short mask_high; unsigned short match_low; unsigned short match_high; } rx_mode_r;
unsigned short *status	Shows the mode of operation for the receiver: bit 0 set: receiver operates in trigger mode bit 1 set: receiver operates in storage mode bit 2 set: receiver operates in masked mode
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware - see <i>result</i> for further information
XPC_ERROR_RX_NUMBER	Invalid transmitter number specified
XPC_ERROR_ACTIONCODE	Invalid action code specified
XPC_FAILED	Execution of command failed

See also *activate receiver* in the appropriate Hardware User Manual.

Note This command starts, stops or continues data reception on a particular receiver and optionally defines a mask for received ARINC 429 data. This command can be given at any time.

'Start' and 'continue' are equivalent, with the exception, that 'start' controls mask definition and -usage. A mask is used for conditional reception. If a mask is specified, two 32-bit patterns are prepared. The first pattern, referred to as 'mask', is used to select the bits, that should compare equal with the second pattern, referred to as 'match'. In other words: if ((ARINC 429data AND mask) XOR match) does not evaluate to zero, ARINC 429 data is ignored.

Attention: unmarked bits (0) in 'mask' have to be set to '0' in 'match' as well.

'Continue' resumes reception without changing mode (masked or unmasked). If 'continue' is given after reset, the command is ignored.

Although a receiver is stopped still the last received data are stored in its receiver buffers, and can be read. Performing the command 'read receiver buffer' as often as much receive buffers have been allocated (see command *get receiver buffer count*) for the particular receiver, will deliver the last received items.

`_activate_rx_trigger_mode ()` [xPC] except [USB]

Synopsis `int _activate_rx_trigger_mode(hmpc, rx_number, act_code)`

Description Activates the discrete I/O triggered start/stop mode for a selected receiver

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
char rx_number	The receiver number:, range '0'..'3' [HPVISA][PCC] '0'..'7'['b'] [PC104] '0'..'9','a'..'f'['n'] [EPC][PCI][PCI-2G][VME]
unsigned short act_code	activation code: 0: de-activate triggered RX-mode 1: activate triggered RX-mode

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware XPC_ERROR_RX_NUMBER Invalid receiver number specified
XPC_FAILED	Execution of command failed

Note This function activates or de-activates the 'start/stop RX on trigger' mode. When in this mode, calls that normally start the receiver only prepare the receiver for starting; the actual start is performed when the IPMx_IN/INPCC signal next changes.

If the IPMx_IN/INPCC signal subsequently changes back to its original state, the receiver will be stopped.

This function can be used to precisely synchronize receive activities with external events.

See also `_activate_tx_trigger_mode` in this document

`_get_status_word()` [xPC]

Synopsis `int _get_status_word(hmpc, ip_num, &status)`

Description Returns the 32-bit status word of a particular IP-Module or PCMCIA-card

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
short ip_num	The IPM slot number 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME] ignored for [PCC][USB][HPVISA]
unsigned long status	The 32-bit Status word:

Bit	Description
0	0: controller is ready to accept commands 1: controller is busy
1	1: interrupt is pending
2	1: Rx0 Receiver buffer full (data ready flag set)
3	1: Rx1 Receiver buffer full (data ready flag set)
4	1: Rx2 Receiver buffer full (data ready flag set)
5	1: Rx3 Receiver buffer full (data ready flag set)
6/7	10: IP-Module A429IPM 01: PCMCIA-Card A429PCC 11: USB-device A429USB
8	Reserved for future use
9	1: A429PCC power save mode
10	1: Tx0 off
11	1: Tx1 off
12	State of external TTL-out IPMx_OUT/OUTPCC (not [USB])
13	State of external TTL-out 1 (optional [EPC]) (not [USB])
14	State of external TTL-in 1 (optional [EPC]) (not [USB])
15	State of external TTL-in 0 IPMx_IN/INPCC (not [USB])
16	1: Rx0 Receiver buffer not empty (data exist flag set)
17	1: Rx1 Receiver buffer not empty (data exist flag set)
18	1: Rx2 Receiver buffer not empty (data exist flag set)
19	1: Rx3 Receiver buffer not empty (data exist flag set)
20	1: Rx0 Receiver buffer overrun (data lost flag set)
21	1: Rx1 Receiver buffer overrun (data lost flag set)

<continued>

22	1: Rx2 Receiver buffer overrun (data lost flag set)
23	1: Rx3 Receiver buffer overrun (data lost flag set)
24	1: Rx0 high-resolution timestamp configured
25	1: Rx1 high-resolution timestamp configured
26	1: Rx2 high-resolution timestamp configured
27	1: Rx3 high-resolution timestamp configured
28	1: Tx0 data request (Data Replay)
29	1: Tx1 data request (Data Replay)
30	1: Tx0 error
31	1: Tx1 error

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed
XPC_ERROR_IPNUM	Given IP-Number is out of range

See also see further explanation of the various Status-bits in the appropriate Hardware Reference Manuals

Note A **Data Ready flag** set to one indicates that at least one receiver buffer of the respective receiver is filled up and can be read. If a number of data equal or more to the receivers buffers size has been received, the respective receiver Data Ready flag is set. Although not set, received data can be read at any time.

A **Data Exists flag** set to one indicates that at least one receiver buffer of the respective receiver is not empty, and can be read. The flags indicate if any data are available in the receive buffers of the receivers. This is independant from the size of the receive buffers.

A **Data Lost flag** set to one indicates that at least one receiver buffer of the respective receiver has been flushed for not having been read by the host. The performance of the A429IPM allows to guarantee that no data received by the transceiver chips are lost. However it is possible that the host does not want, or is not able to, retrieve data from the modules receiver buffers. Any time one of the internal receive-buffers of a receiver is internally flushed, and overridden with new receive data, this flag is set.

This flag remains set until explicitly resetted by the host (see function `_reset_data_lost()`)

_get_receiver_data_ready() [PCC][USB] (old interface)

Synopsis `int _get_receiver_data_ready(hmpc, &rx0, &rx1, &rx2, &rx3)`

Description Returns the data ready flag for all receivers.

Arguments	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
unsigned short *rx0	data ready flag receiver 0
unsigned short *rx1	data ready flag receiver 1
unsigned short *rx2	data ready flag receiver 2
unsigned short *rx3	data ready flag receiver 3
A data ready flag value of 0 (zero) signals that the no buffer is filled yet whereas a value of 1 (one) signals that at least one buffer has been filled.	
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

See also *get receiver data ready* in the appropriate Hardware User Manual.

Note A **Data Ready flag** set to one indicates that at least one receiver buffer of the respective receiver is filled up and can be read. If a number of data equal or more to the receivers buffers size has been received, the respective receiver Data Ready flag is set. Although not set, received data can be read at any time.

_get_receiver_data_ready() [xPC] except [PCC][USB] (old interface)

Synopsis `int _get_receiver_data_ready(hmpc, rxnum)`

Description Returns the data ready flag for a particular receiver.

Arguments	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
char rxnum	The receiver number, range '0'..'3' [HPVISA] '0'..'7' [PC104] '0'..'9','a'..'f'['n'] [EPC][PCI][PCI-2G][VME]
Return Value	Explanation
0	no Receiver buffer is filled yet
1	at least one Receiver buffer is filled with data
XPC_ERROR_RX_NUMBER	rxnum out of range
XPC_FAILED	Execution of command failed

See also *get receiver data ready* in the appropriate Hardware User Manual.

_get_data_ready() [xPC]

Synopsis `int _get_data_ready(hmpc, rxnum)`

Description Returns the data ready flag for a particular receiver.

Arguments	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
char rxnum	The receiver number, range '0'..'3' [PCC][USB][HPVISA] '0'..'7' [PC104] '0'..'9','a'..'f'['n'] [EPC][PCI][PCI-2G][VME]
Return Value	Explanation
0	Receiver buffer isn't filled yet
1	Receiver buffer is filled with data
XPC_ERROR_RX_NUMBER	Receiver number out of range
XPC_FAILED	Execution of command failed

See also *get receiver data ready* in the appropriate Hardware User Manual.

_get_receiver_data_lost() [xPC]

Synopsis `int _get_receiver_data_lost(hmpc, rx_number)`

Description Retrieves the datalost-flag of a particular Receiver. The data-lost-flag indicates if at least one receive-buffer has not been retrieved before it was flushed and overwritten with new data.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
char rx_number	The number of the receiver, range: '0'..'3' [PCC][USB][HPVISA] '0'..'7' [PC104] '0'..'9','a'..'f'['n'] [EPC][PCI][PCI-2G][VME]
Return Value	Explanation
XPC_FAILED	Execution of command failed
XPC_SUCCESS	Execution of command succeeded/datalost-flag is not set
XPC_ERROR_RX_NUMBER	Invalid receiver number specified
1	datalost-flag is set

See also `_reset_data_lost()`

Note A **Data Lost flag** set to one indicates that at least one receiver buffer of the respective receiver has been flushed for not having been read by the host. The performance of the A429IPM allows to guarantee that no data received by the transceiver chips are lost. However it is possible that the host does not want, or is not able to, retrieve data from the modules receiver buffers. Any time one of the internal receive-buffers of a receiver is internally flushed, and overridden with new receive data, this flag is set.

This flag remains set until explicitly reset by the host (see function `_reset_data_lost()`)

`_reset_data_lost()` [xPC]

Synopsis `int _reset_data_lost(hmpc, rx_number)`

Description Resets the datalost-flag of a particular Receiver. The data-lost-flag indicates if at least one receive-buffer has not been retrieved before it was flushed and overwritten with new data.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
char rx_number	The number of the receiver, range: '0'..'3' [PCC][USB][HPVISA] '0'..'7' [PC104] '0'..'9','a'..'f'['\n'] [EPC][PCI][PCI-2G][VME]
Return Value	Explanation
XPC_FAILED	Execution of command failed
XPC_SUCCESS	Execution of command succeeded
XPC_ERROR_RX_NUMBER	Invalid receiver number specified

See also `_get_receiver_data_lost()`

_get_receiver_data_exist() [xPC]

Synopsis `int _get_receiver_data_exist(hmpc, rx_number)`

Description Retrieves the data-exist-flag of a particular Receiver. The data-exist-flag indicates if at least one receive-buffer has data that can be retrieved.

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
char rx_number	The number of the receiver, range: '0'..'3' [PCC][USB][HPVISA] '0'..'7' [PC104] '0'..'9','a'..'f'['n'] [EPC][PCI][PCI-2G][VME]
Return Value	Explanation
XPC_FAILED	Execution of command failed
XPC_SUCCESS	Execution of command succeeded/dataexist-flag is not set
XPC_ERROR_RX_NUMBER	Invalid receiver number specified
1	dataexist-flag is set

Note A **Data Exists flag** set to one indicates that at least one receiver buffer of the respective receiver is not empty, and can be read. The flags indicate if any data are available in the receive buffers of the receivers. This is independant from the size of the receive buffers.

`_activate_rx_indexed_mode()` [xPC]

Synopsis `int _activate_rx_indexed_mode(hmpc, rx_number, act_code, &status)`

Description This function activates the Indexed Receive Mode. Each Label(+SDI) has a slot in an Index-Table from which an application can always retrieve the most actual 32-bit dataword plus receive timestamp.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
char rx_number	The number of the receiver, range: '0'..'3' [PCC][USB][HPVISA] '0'..'7' [PC104] '0'..'9','a'..'f'['n'] [EPC][PCI][PCI-2G][VME]
unsigned short act_code	the activation code: 0 : deactivate Index-Mode 1 : activate Index-Mode in 10-Bit mode (Label+SDI) 2: activate Index-Mode in 8-Bit mode (only Label)
unsigned short *status	return status: 0 : function (see act_code) executed 1 : invalid combination of Index-Mode
Return Value	Explanation
XPC_FAILED	Execution of command failed
XPC_SUCCESS	Execution of command succeeded
XPC_ERROR_RX_NUMBER	Invalid receiver number specified
XPC_ERROR_ACTCODE	Invalid activation code

See also `_read_indexed_data()` and `_read_all_indexed_data()` in this document

Note Indexed mode is allowed in combination with any other permitted mode combination; the only restriction is that Protocol mode is only permitted in combination with Storage and/or Indexed mode. Masked mode alone is also not permitted. If Masked mode is used together with Indexed mode, the data removed by the Masked mode filter does not appear in the Indexed data.

`_read_indexed_data()` [xPC]

Synopsis `int _read_indexed_data(hmpc, rx_number, label, sdi, &arinc_data,×tamp)`

Description This function returns one data entry (ARINC 429 data + timestamp) for the Label/SDI combination. If the activation code was 2 in function **`_activate_rx_indexed_mode()`** the SDI is ignored..

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
char rx_number	The number of the receiver, range: '0'..'3' [PCC][USB][HPVISA] '0'..'7' [PC104] '0'..'9','a'..'f'['n'] [EPC][PCI][PCI-2G][VME]
unsigned short label	the ARINC 429 Label-number
unsigned short sdi	the SDI-value: 0-3
unsigned long *arinc_data	the ARINC 429 32-Bit word
unsigned long *timestamp	the 32-Bit timestamp of the ARINC 429 word

Return Value	Explanation
XPC_FAILED	Execution of command failed
XPC_SUCCESS	Execution of command succeeded
XPC_ERROR_RX_NUMBER	Invalid receiver number specified

See also **`_activate_rx_indexed_mode()`** and **`_read_all_indexed_data()`** in this document

Note 1: The timestamps returned by the **`_read_indexed_data()`** function are in the currently selected receiver timestamp accuracy.

Note 2: The label is in binary/hex representation: e.g. Label 206oct (Bits 8-1 in word: 011|000|01 bin) has to passed as 0x61hex.

Note 3: If the receiver does not operate in Index Mode the function **`_read_indexed_data()`** return invalid arinc_data/timestamp values.

`_read_all_indexed_data()` [xPC]

Synopsis `int _read_all_indexed_data(hmpc, rx_number, &arinc_data,×tamp)`

Description This function returns one all data entries (ARINC 429 data + timestamp) of the Index-Mode Table.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
char rx_number	The number of the receiver, range: '0'..'3' [PCC][USB][HPVISA] '0'..'7' [PC104] '0'..'9','a'..'f'['n'] [EPC][PCI][PCI-2G][VME]
unsigned long *arinc_data	array for ARINC 429 32-Bit words
unsigned long *timestamp	array for 32-Bit timestamps of the ARINC 429 words

Return Value	Explanation
XPC_FAILED	Execution of command failed
XPC_SUCCESS	Execution of command succeeded
XPC_ERROR_RX_NUMBER	Invalid receiver number specified

See also `_activate_rx_indexed_mode()` and `_read_indexed_data()` in this document

Note 1: The timestamps returned by the `_read_all_indexed_data()` function are in the currently selected receiver timestamp accuracy.

Note 2: **The `arinc_data[]` and `timestamp[]` arrays size has to be 256 for `act_code = 2` !**
 The `arinc_data[]` and `timestamp[]` arrays size has to be 1024 for `act_code = 1` !

Note 3: If the receiver does not operate in Index Mode the function `_read_all_indexed_data()` return invalid `arinc_data/timestamp` values.

_get_receiver_mode() [xPC]

Synopsis `int _get_receiver_mode(hmpc, rx_number, &mode)`

Description Retrieve the receiving mode for a receiver

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
char rx_number	Receiver number, range '0'..'3' [PCC][USB][HPVISA] '0'..'7' [PC104] '0'..'9','a'..'f'['n'] [EPC][PCI][PCI-2G][VME]
unsigned short *mode	The mode, the the receiver operates in: bit 0 set: receiver operates in trigger mode bit 1 set: receiver operates in storage mode bit 2 set: receiver operates in masked mode bit 3 set: receiver operates in data manipulation mode bit 4 set: receiver operates in protocol mode bit 5 set: receiver operates in indexed mode

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_RX_NUMBER	Invalid receiver number specified
XPC_FAILED	Execution of command failed

See also *get receiver mode* in the appropriate Hardware User Manual.

Note This command reports the current operating modes for a particular receiver. There are three possibilities, and any combination of them:

trigger mode

A trigger (mask & match value and action code; see function `_define_trigger`) is defined for the particular receiver channel.

storage mode

Receiver buffers with a size different from 0 (zero) are defined for the particular receiver.

masked mode

A filter (mask & match value, see function `_activate_rec`) is defined for the particular receiver channel.

manipulation mode

receiver operates in Data Manipulation Mode (see A429BASTplus: Data Manipulation)

protocol mode

receiver operates in Protocol Mode (see Protocol Mode functions)

indexed mode

receiver operates in Indexed Mode (see functions `_activate_rx_indexed_mode`, `_read_indexed_data`, `_read_all_indexed_data`)

Note:

Indexed mode is allowed in combination with any other permitted mode combination; the only restriction is that Protocol mode is only permitted in combination with Storage and/or Indexed mode.

Masked mode alone is also not permitted. If Masked mode is used together with Indexed mode, the data removed by the Masked mode filter does not appear in the Indexed data.

`_define_trigger()` [xPC] (xOUT not for [USB])

Synopsis `int _define_trigger(hmpc, rx_number, count, &buffer, &status)`

Description Defines up to eight trigger events for a certain receiver.

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
char rx_number	The number of the receiver to define the trigger events for, range: '0'..'3' [PCC][USB][HPVISA] '0'..'7' [PC104] '0'..'9','a'..'f'['n'] [EPC][PCI][PCI-2G][VME]
unsigned short count	Denotes the number of trigger events to define. A value of zero terminates trigger mode.
trig_definition_t *buffer	Pointer to an array of structures of the following layout: { unsigned char trigger_ID; unsigned char act_code; unsigned short trigger_mask_low; unsigned short trigger_mask_high; unsigned short trigger_match_low; unsigned short trigger_match_high; } trig_definition_t; The range for trigger_ID is 1..254 (do not use 0 and 255 !!). The value of act_code denotes the action to be taken: 0 = Stop on match 1 = Start on match 2 = Stop on mismatch 3 = Start on mismatch 4 = Stop on match and set xOUT to LOW(0) 5 = Start on match and set xOUT to HIGH(1) 6 = Stop on mismatch and set xOUT to LOW(0) 7 = Start on mismatch and set xOUT to HIGH(1) [8-11 not used] note: codes 12-15 provision for alternate discrete OUT
unsigned short *status	Shows upon completion the mode the receiver is operating in: bit 0 set = receiver operates in trigger mode bit 1 set = receiver operates in storage mode bit 2 set = receiver operates in masked mode

Return Value	Explanation
XPC_SUCCESS	Trigger definition accepted
XPC_ERROR_RX_NUMBER	Invalid receiver number specified
XPC_ERROR_TRIG_COUNT	Trigger count out of range
XPC_FAILED	Command execution failed

See also *define trigger* in the appropriate Hardware User Manual

Note This command (re)defines up to eight trigger events for each receiver. This command can be given at any time. A trigger event definition supercedes all previous definitions, if any.

If trigger events are defined, a matching process is performed, whenever an ARINC 429 word passes the receiver mask. Depending on the action indicated, the corresponding trigger state table entry (referred by its trigger ID) is set/cleared if the match is successful or not.

Additionally it is possible to concurrently set the state of A429IPM/A429PCC discrete output IPMx_OUT resp. OUTPCC dependent on the match/mismatch criteria.

For each event definition, the action code, a selector byte as well as two 32bit patterns have to be prepared. The 'mask' is used to select the bits, that should compare equal with the second pattern, referred to as 'match'. In other words: if ((ARINC 429data AND mask) XOR match) does/doesnot evaluate to zero, no further actions are performed, otherwise the trigger state table entry is modified as required. This process is repeated until all trigger event definitions are exhausted.

Four (4) different trigger ID actions can be performed :

start-on-match :	The corresponding trigger ID is set if the matching criterion is fulfilled. Referring transmit definitions will be transmitted.
stop-on-match :	The corresponding trigger ID is cleared if the matching criterion is fulfilled. Referring transmit definitions will not be transmitted.
start-on-mismatch :	the corresponding trigger ID is set if the matching criterion is not fulfilled. Referring transmit definitions will be transmitted.
stop-on-mismatch :	the corresponding trigger ID is cleared if the matching criterion is not fulfilled. Referring transmit definitions will not be transmitted.

Concurrently four different actions concerning the state of discrete output IPMx_OUT/OUTPCC can be performed :

LOW-on-match : set the state of IPMx_OUT/OUTPCC to TTL voltage level LOW if the matching criterion is fulfilled.

HIGH-on-match : set the state of IPMx_OUT/OUTPCC to TTL voltage level HIGH if the matching criterion is fulfilled.

LOW-on-mismatch : set the state of IPMx_OUT/OUTPCC to TTL voltage level LOW if the matching criterion is not fulfilled.

HIGH-on-mismatch : set the state of IPMx_OUT/OUTPCC to TTL voltage level HIGH if the matching criterion is not fulfilled.

The mismatch criteria are useful for monitoring a data stream for any change.

`_get_trigger_state()` [PCC][USB][HPVISA]

Synopsis `int _get_trigger_state(hmpc, trig_ID, &flag)`

Description Returns the current state of a certain trigger ID.

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
unsigned short trig_ID	The trigger ID whose state shall be returned (Range: 0..255)
unsigned short *flag	The current trigger state: 0 = all transmission IDs referring to this trigger ID will NOT be transmitted 1 = all transmission IDs referring to this trigger ID will be transmitted.
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_TRIG_ID	Invalid trigger ID specified
XPC_FAILED	Command execution failed

See also *get trigger state* in the appropriate Hardware User Manual

Note This command is used to check the state of a particular trigger state table entry (trigger ID).
The state for triggerID 0 is always 'cleared', for triggerID 255 it is always 'set'.

`_get_trigger_state()` [PC104][EPC][VME][PCI][PCI-2G]

Synopsis `int _get_trigger_state(hmpc, ipm, trig_ID, &flag)`

Description Returns the current state of a certain trigger ID.

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
short ipm	The number of the IP-Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME]
unsigned short trig_ID	The trigger ID whose state shall be returned (Range: 0..255)
unsigned short *flag	The current trigger state: 0 = all transmission IDs referring to this trigger ID will NOT be transmitted 1 = all transmission IDs referring to this trigger ID will be transmitted.
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_TRIG_ID	Invalid trigger ID specified
XPC_ERROR_IPNUM	Invalid IP-Module number specified
XPC_FAILED	Command execution failed

See also *get trigger state* in the appropriate Hardware User Manual

`_set_trigger_state()` [PCC][USB]

Synopsis `int _set_trigger_state(hmpc, trig_ID, trig_flag, out_flag)`

Description Set the state of a certain trigger ID together with the state of the OUTPCC discrete signal

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
unsigned short trig_ID	The trigger ID whose state should be changed (Range 1..254).
unsigned short trig_flag	New trigger state: 0 = all transmission ID's referring to this trigger ID will NOT be transmitted. 1 = all transmission ID's referring to this trigger ID will be transmitted.
unsigned short out_flag	New OUTPCC state (ignored for [USB]) 0 = OUTPCC is LOW 1 = OUTPCC is HIGH
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_TRIG_ID	Invalid trigger ID specified
XPC_ERROR_TRIG_FLAG	Invalid trigger state specified.
XPC_ERROR_OUTPCC_STATE	Invalid trigger OUTPCC state/flag specified.

See also *set trigger state* in the appropriate Hardware User Manual

Note A trigger state table is used to control conditional transmission of ARINC 429 data items. Any ARINC 429 transmit data definition refers to a trigger index (trigger ID), the state of the corresponding trigger table entry controls transmission. A trigger state table entry can be explicitly set or reset using this command, affecting transmission of all data items referencing it. A trigger state table entry can also be modified by fulfillment of some receiver matching criteria (see command *define_trigger_event*).

Note, that both transmitters of an A429PCC/A429USB share the same trigger state table, thus being affected simultaneously.

Trigger states 0 and 255 may not be modified. An attempt to do so is ignored.

This functions allows to simultaneously set/reset the state of the discrete output OUTPCC together with the state of a trigger ID. It is very useful to trigger external devices upon start of ARINC 429 data transmission.

_set_trigger_state() [EPC][PCI][PCI-2G][VME][PC104]

Synopsis `int _set_trigger_state(hmpc, ipm, trig_ID, trig_flag)`

Description Set the state of a certain trigger

Arguments	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
short ipm	The IP-Module number: 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME]
unsigned short trig_ID	The trigger ID whose state should be changed (Range 1..254).
unsigned short trig_flag	New trigger state: 0 = all transmission ID's referring to this trigger ID will NOT be transmitted. 1 = all transmission ID's referring to this trigger ID will be transmitted.
unsigned short out_flag	New OUTPCC state: 0 = OUTPCC is LOW 1 = OUTPCC is HIGH
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_TRIG_ID	Invalid trigger ID specified
XPC_ERROR_TRIG_FLAG	Invalid trigger state specified.

See also *set trigger state* in the appropriate Hardware User Manual

Note A trigger state table is used to control conditional transmission of ARINC 429 data items. Any ARINC 429 transmit data definition refers to a trigger index (trigger ID), the state of the corresponding trigger table entry controls transmission. A trigger state table entry can be explicitly set or reset using this command, affecting transmission of all data items referencing it. A trigger state table entry can also be modified by fulfillment of some receiver matching criteria (see command *define_trigger_event*).

Note, that both transmitters of an A429IPM IP-Module share the same trigger state table, thus being affected simultaneously.

Trigger states 0 and 255 may not be modified. An attempt to do so is ignored.

`_set_trigger_state()`[PC104][EPC][VME][PCI][PCI-2G]

Synopsis `int _set_trigger_state(hmpc, ipm, trig_ID, trig_flag)`

Description Set the state of a certain trigger ID.

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
short ipm	The IP-Module number: 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME]
unsigned short trig_ID	The trigger ID whose state should be changed (Range 1..254).
unsigned short trig_flag	New trigger state: 0 = all transmission ID's referring to this trigger ID will NOT be transmitted. 1 = all transmission ID's referring to this trigger ID will be transmitted.
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	Invalid IP-Module number specified
XPC_ERROR_TRIG_ID	Invalid trigger ID specified
XPC_ERROR_TRIG_FLAG	Invalid trigger state specified.

See also *set trigger state* in the appropriate Hardware User Manual

Note A trigger state table is used to control conditional transmission of ARINC 429data items. Any ARINC 429 transmit data definition refers to a trigger index (trigger ID), the state of the corresponding trigger table entry controls transmission. A trigger state table entry can be explicitly set or reset using this command, affecting transmission of all data items referencing it. A trigger state table entry can also be modified by fulfillment of some receiver matching criteria (see command *define_trigger_event*).

Note, that both transmitters of an A429IPM or A429PCC, A429USB share the same trigger state table, thus being affected simultaneously.

Trigger states 0 and 255 may not be modified. An attempt to do so is ignored.

A extension of command set trigger state is realized with the interface function `_set_trigger_state_dis` [not yet implemented in V3.21]. This functions allows to simultaneously set the state of the discrete output IPM_OUT/OUTPCC together with the state of a trigger ID. It is very useful to trigger external devices upon start of ARINC 429 data transmission.

`_set_trigger_state()`[HPVISA]

Synopsis `int _set_trigger_state(hmpc, trig_ID, trig_flag)`

Description Set the state of a certain trigger ID.

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
unsigned short trig_ID	The trigger ID whose state should be changed (Range 1..254).
unsigned short trig_flag	New trigger state: 0 = all transmission ID's referring to this trigger ID will NOT be transmitted. 1 = all transmission ID's referring to this trigger ID will be transmitted.
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_TRIG_ID	Invalid trigger ID specified
XPC_ERROR_TRIG_FLAG	Invalid trigger state specified.

See also *set trigger state* in the appropriate Hardware User Manual

Note A trigger state table is used to control conditional transmission of ARINC 429data items. Any ARINC 429 transmit data definition refers to a trigger index (trigger ID), the state of the corresponding trigger table entry controls transmission. A trigger state table entry can be explicitly set or reset using this command, affecting transmission of all data items referencing it. A trigger state table entry can also be modified by fulfillment of some receiver matching criteria (see command *define_trigger_event*).

Note, that both transmitters of an A429IPM or A429PCC, A429USB share the same trigger state table, thus being affected simultaneously.

Trigger states 0 and 255 may not be modified. An attempt to do so is ignored.

_reset_selftest() [PCC][USB] (old interface)

Synopsis `int _reset_selftest(hmpc, &result, &err_bits)`

Description Performs a board reset with selftest and returns the resulting information.

Arguments	Explanation
-----------	-------------

HANDLE hmpc	The handle returned by the _open_xpc() call.
-------------	---

unsigned short *result	The selftest result; a value of zero (0) denotes no errors, otherwise the bits 0 to 11 (if set) denote the detected errors:
------------------------	---

Bit	Description
0	- don't care -
1	DPRAM failure
2	SRAM/exchange buffer failure
3	Transceiver A (Tx0,Rx0,Rx1) bad
4	Transceiver B (Tx1, Rx2,Rx3) bad
5	error in external loop test
6	adapter box not connected (only [PCC])
7	- don't care -
8	error in loop Tx0 - Rx0
9	error in loop Tx0 – Rx2
10	error in loop Tx1 – Rx1
11	error in loop Tx1 - Rx3
12-15	not used

unsigned short *error_bits	not used [tbd] – only for compatibilty reason with former A429MPC Interfaceboard
----------------------------	--

See also *perform reset/selftest* in the appropriate Hardware User Manual

_reset_selftest() [EPC] (old interface)

Synopsis `int _reset_selftest(hmpc, ipnum, &result)`

Description Performs a board reset and selftest and returns the resulting information.

Arguments	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
short ipnum	The number of the IP Module, range 0..3
unsigned short *result	The selftest result; a value of zero (0) denotes no errors, otherwise the bits 0 to 11 (if set) denote the detected errors:

Bit	Description
0	- don't care -
1	DPRAM failure
2	SRAM/exchange buffer failure
3	Transceiver A (Tx0,Rx0,Rx1) bad
4	Transceiver B (Tx1, Rx2,Rx3) bad
5	Error in external loop test
6	- don't care -
7	- don't care -
8	Error in loop Tx0 - Rx0
9	Error in loop Tx0 – Rx2
10	Error in loop Tx1 – Rx1
11	Error in loop Tx1 - Rx3
12-15	not used

See also *perform reset/selftest* in the appropriate Hardware User Manual
 _reset_and_selftest(), **_epc_reset()** and **_epc_IPreset()** in this Manual

`_reset_and_selftest()` [xPC]

Synopsis `int _reset_and_selftest(hmpc, ipnum, &result, &err_bits)`

Description Performs a board reset with selftest and returns the resulting information.

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
short ipnum	The IPM slot number : 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME] ignored for [PCC][USB][HPVISA]
unsigned short *result	The selftest result; a value of zero (0) denotes no errors, otherwise the bits 0 to 11 (if set) denote the detected errors:

Bit	Description
0	don't care
1	DPRAM failure [xPC]
2	SRAM/exchange buffer [xPC]
3	Transceiver A (Tx0,Rx0,Rx1) bad [xPC]
4	Transceiver B (Tx1, Rx2,Rx3) bad [xPC]
5	error in external loop test [xPC]
6	- don't care - [all except PCC] adapter box not connected [PCC]
7	- don't care -
8	error in loop Tx0 - Rx0 [xPC]
9	error in loop Tx0 – Rx2 [xPC]
10	error in loop Tx1 – Rx1 [xPC]
11	error in loop Tx1 - Rx3 [xPC]
12-15	not used

unsigned short *error_bits	not used [tbd] – only for compatibilty reason with former A429MPC Interfaceboard
----------------------------	--

See also *perform reset/selftest* in the appropriate Hardware User Manual

Note The A429IPM/A429PCC performs various selftest routines to ensure the integrity of its hardware and software components. These routines are performed with this command, initiating a reset and selftest of an A429IPM on a carrier board or the A429PCC PCMCIA-card with adapterbox. These tests check the major A429IPM/A429PCC building blocks and report their result.

If a malfunction in the A429IPM/A429PCC memory is detected (either SRAM or the Dual Port RAM), the card remains in busy state. It then no longer responds to any command.

The ARINC 429 part is tested as follows:

First, an internal loop test is performed, i.e. data is sent to each transmitter and echoed on-chip. If data received matches the stimulus pattern, the ARINC 429 controller chips as well as the clock generator circuitry are assumed to work properly. Next, loop connections are established to connect each transmitter output to two receiver inputs – only **[PCC][USB],[EPC]** and **[PCI-2G]**. This tests the ARINC 429 driver circuitry and the receiver inputs.

On all boards [PC104],[PCI],[VME] and [HPVISA] the external loops will fail unless the loops have been externally wired through a selftest adapter.

A malfunction in the ARINC 429 part of the controller does not inhibit the controller. In this case, proper operation is improbable, however.

Note that, while selftest is in progress, certain data bursts appear on the ARINC 429 outputs. These bursts send data all with label 000 (octal). If any device connected to the **[xPC]** card cannot cope with this situation, it should be disconnected. Data present on the ARINC 429 bus during selftest does not influence the internal selftest.

After a reset/selftest of the interface board neither any transmit data definition nor any other activation/configuration or setting is valid any more.

`_establish_loop()` [PCC][USB][EPC][PCI-2G]

Synopsis `int _establish_loop(hmpc, loop_code)`

Description Establish Transmitter - Receiver loops.

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
unsigned short loop_code	Denotes the internal loops: For [PCC][USB] bit 0 set = loop Tx0 to Rx0 bit 1 set = loop Tx1 to Rx1 bit 2 set = loop Tx0 to Rx2 bit 3 set = loop Tx1 to Rx3 all other bits are not evaluated. For [EPC][PCI-2G] bits 0..3 refer to IP module 0 bits 4..7 refer to IP module 1 bits 8..11 refer to IP module 2 bits 12..15 refer to IP module 3 The four bits designated to an IP module denote the routing between the transmitters and receivers of that module as depicted above for the [PCC][USB].
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_LOOPCODE	Invalid loop_code specified
XPC_FAILURE	Command rejected

See also *establish loop connections* in appropriate Hardware User Manual

Note The receivers affected by this call will be disconnected from the 'outside world' while the transmitters remain connected. Loop connections can be used for selftest purposes. Whenever a loop connection is established, the corresponding receiver is disconnected from the outside world, thus allowing a selftest to be performed without the need for removal of any external signal sources.

Looping back a transmitter to a not used receiver allows to determine the precise absolute instant when data has been transmitted. This mechanism can be used for comparison with receive times of data from a UUT, and to get an idea of its required time to react upon certain stimuli.

This function is very useful for verifying timing behavior in any handshaking-type communication between LRUs and [PCC][USB][EPC][PCI-2G] applications !

`_reset_clock()` [xPC]

Synopsis `int _reset_clock(hmpc, ipnum)`

Description This function resets the system clock [millisecond timer] of the particular IP-module resp. PCMCIA-card, USB-Device

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
short ipnum	The IPM slot number. 0..1 [PC104] 0..3[5] [EPC][PCI][VME] ignored for [PCC][USB][HPVISA]

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	IPM number out of range [EPC][PCI-2G]
XPC_ERROR_SLOTEEMPTY	IPM slot is empty [EPC][PCI-2G]
XPC_FAILED	Command rejected by firmware.

Note The internal clock automatically starts from 0. Timestamp-values of data already stored in the receive-buffer become obviously obsolete.

See also `_get_ticks()/_get_ticks_mpc_board()`

_get_arinc_board_type() [xPC]

Synopsis `int _get_arinc_board_type(hmpc, ipnum, &board_type)`

Description Returns the type number of the installed board.

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call
short ipnum	The IPM slot number. 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME] ignored for [PCC][USB][HPVISA]
unsigned short *board_type	Denotes the board type returned by the firmware: 0 = tbd 1 = A429PCC 2 = A429IPM on any carrier 3 = A429USB (was formerly A429MPC - old TechSAT/GPA ISA-board)
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Command rejected

See also *get board type* in the appropriate Hardware User Manual

`_get_firmware_version()` [PCC][USB] (old interface)

Synopsis `int _get_firmware_version(hmpc, &version_no)`

Description Returns the version number of the installed firmware

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
unsigned short *version_no	The version number returned by the firmware bit 0..3 = minor version number bit 4..7 = major version number latest revision: <div style="text-align: right;">[PCC] V3.2 [June 2000] [USB] V1.1 [October 2006]</div>
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Command execution failed

See also *get firmware version* in the appropriate Hardware User Manual

`_get_firmware_version()` [EPC] (old interface)

Synopsis `int _get_firmware_version(hmpc, ipnum, &version_no)`

Description Returns the version number of the installed firmware on the A429IPM IP-Module

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
short ipnum	The number of the IP Module, range 0..3
unsigned short *version_no	The version number returned by the firmware bit 0..3 = minor version number bit 4..7 = major version number latest revision: V2.7 [June 2000]
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Command execution failed

See also *get firmware version* in the appropriate Hardware User Manual and ***_epc_firmware_version()*** in this document

`_get_fw_version()` [xPC]

Synopsis `int _get_fw_version(hmpc, ipnum, &version_no)`

Description Returns the version number of the installed firmware

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
short ipnum	The number of the IP Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME] ignored for [PCC][USB][HPVISA]
unsigned short *version_no	The version number returned by the firmware bit 0..3 = minor version number bit 4..7 = major version number A429IPM latest revision: V2.9 [Feb 2003] A429PCC latest revision: V3.2 [Feb 2000] A429USB latest revision: V1.1 [Oct 2006]

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Command execution failed

See also *get firmware version* in the appropriate Hardware User Manual and ***_epc_firmware_version()***, ***_pci2g_firmware_version*** in this document

`_get_ticks_mpcboard()` [PCC][USB] (old interface)

Synopsis `int _get_ticks_mpcboard(hmpc, &elapsed_ms)`

Description Returns the A429PCC timer value with the number of elapsed milliseconds since the last PCMCIA-card reset.

Argument	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
unsigned long *elapsed_ms	Pointer to variable to hold the result, the elapsed milliseconds since the last board reset.
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Command rejected by firmware.

See also *get board timer value* in the appropriate Hardware User Manual.

Note This command returns the A429PCCs/A429USBs timer-value containing the number of elapsed milliseconds since the last A429PCC/A429USB reset.
When operating with several boards this function is very useful to retrieve the relative receive-timing of receivers from different boards.

`_get_ticks_mpcboard()` [EPC] (old interface)

Synopsis `int _get_ticks_mpcboard(hmpc, ipnum, &elapsed_ms)`

Description Returns the A429IPM IP-Module timer value with the number of elapsed milliseconds since the last board reset.

Argument	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
short ipnum	The number of the IP Module 0..3 [EPC]
unsigned long *elapsed_ms	Pointer to variable to hold the result, the elapsed milliseconds since the last board reset.

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Command rejected by firmware.

See also *get board timer value* in the appropriate Hardware User Manual.

Note This command returns the A429IPMs timer-value containing the number of elapsed milliseconds since the last A429IPM module reset.

When operating with several boards this function is very useful to retrieve the relative receive-timing of receivers from different boards.

_get_ticks() [xPC]

Synopsis `int _get_ticks(hmpc, ipnum, &elapsed_ms)`

Description Returns the A429xPC timer value with the number of elapsed milliseconds since the last board reset.

Argument	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
short ipnum	The number of the IP Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME] ignored for [PCC][USB][HPVISA]
unsigned long *elapsed_ms	Pointer to variable to hold the result, the elapsed milliseconds since the last board reset.

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	IPM number out of range, all except [PCC][USB]
XPC_ERROR_SLOTEEMPTY	IPM slot is empty, all except [PCC][USB]
XPC_FAILED	Command rejected by firmware.

See also *get board timer value* in the appropriate Hardware User Manual.

Note This command returns the A429IPM/A429PCC/A429USB timer-value containing the number of elapsed milliseconds since the last A429IPM/A429PCC/A429USB reset.

When operating with several boards/cards this function is very useful to retrieve the relative receive-timing of receivers from different boards.

`_get_ext_ticks()` [xPC]

Synopsis `int _get_ext_ticks(hmpc, ipnum, &elapsed_ms, &elapsed_100ms)`

Description Returns the A429xPC timer value with the number of elapsed milliseconds and the number of the high-resolution timer since the last board reset. Units of the high-resolution timer is determined by function `_set_high_resolution()`. Possible values are 100 microseconds or 10 microseconds.

Argument	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
short ipnum	The number of the IP Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME] ignored for [PCC][USB][HPVISA]
unsigned long *elapsed_ms	Pointer to variable to hold the elapsed milliseconds since the last board reset.
unsigned long *elapsed_100ms	Pointer to variable to hold the number of elapsed 100 (or 10) microseconds since the last board reset

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	IPM number out of range, all except [PCC][USB]
XPC_ERROR_SLOTEEMPTY	IPM slot is empty, all except [PCC][USB]
XPC_FAILED	Command rejected by firmware.

See also *get board timer value* in the appropriate Hardware User Manual.

Note This command returns the A429IPM/A429PCC/A429USB timer-value containing the number of elapsed milliseconds/100 (or 10) microseconds since the last A429IPM/A429PCC module reset.

When operating with several boards this function is very useful to retrieve the relative receive-timing of receivers from different boards.

`_set_timestamp_synchro_mode()` [xPC] except [USB]

Synopsis `int _set_timestamp_synchro_mode(hmpc, ipnum, mode, &status)`

Description Sets the timestamp synchronization mode for a certain A429PCC board resp. A429IPM.

Argument	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
short ipnum	The number of the IP-Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME] ignored for [PCC][HPVISA]
unsigned short mode	0 - independent mode (default after power on) 1 - slave mode 2 - master mode
unsigned short *status	Function status, zero (0) unless there's a resource conflict.

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	IPM number out of range, all except [PCC]
XPC_ERROR_SLOTEMPTY	IPM slot is empty, all except [PCC]
XPC_ERROR_SYNCHRO	Unknown mode specified
XPC_FAILED	Command rejected by firmware.

Note Each A429IPM/A429PCC has an internal timer that generates timestamps for received ARINC 429 and RS-xxx data. The resolution of the timestamps can either be configured to be 1msec or 100 microseconds.

The on-card oscillator has a ppm tolerance. In applications where several A429IPM/A429PCC are used and it is necessary that the modules are synchronized in respect to the technically obvious drift of their timers and consequently of the timestamps of the received data, there is a feature implemented that allows inter-module synchronization.

The cards are synchronized using the a discrete input serving as as msec-timer reset and sec-counter increment Interrupt-line. Any external 1 Hz/TTL compatible external signal can be used for synchronization. If no external signal is available an A429IPM/A429PCC can be configured to be synchronization-master, generating a 1 Hz signal on a discrete output line.

The involved A429PCC/A429IPM can be configured to be either SLAVE or MASTER (or INDEPENDANT) using the interface-function `_set_timestamp_synchro_mode()`.

Attention A429IPMs being configured as SLAVE are using the rising edge of a discrete input-line to reset their msec-timer, and to increment their Software sec-timer. Any spike on this line will let these boards jump ahead by 1 second. It is absolutely mandatory to avoid any externally generated spikes or signal distortions on the discrete input line. The wires have to be extremely well shielded and 100% protected from external impacts.

Once synchronized IP modules can differ (from the beginning) by exactly 1 milliseconds, which is the maximum resolution of the synchronization precision.

Any application for which this unavoidable timer-value difference is essential has to compensate this offset by means of software: Retrieve all timer values after synchronization - evaluate if difference - store flag determining if to add or subtract millisecond for the respective boards.

See also [_epc_write_routing](#) in this document: this **[EPC]** function allows to program so-called routing chips on the carrier board which establish the proper connection of the synchronization signals

`_set_timestamp_res()` [xPC]

Synopsis `int _set_timestamp_res(hmpc, ipnum, rx0, rx1, rx2, rx3)`

Description Sets the timestamp resolution mode for a certain A429PCC board resp. A429IPM module.

Argument	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
short ipnum	The number of the IP-Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME] ignored for [PCC][USB][HPVISA]
unsigned char rx0	The mode for receiver 0
unsigned char rx1	The mode for receiver 1
unsigned char rx2	The mode for receiver 2
unsigned char rx3	The mode for receiver 3
	Mode takes the following values: 0 - timestamp resolution 1msec 1 - timestamp resolution 0.1msec

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	IPM number out of range, all except [PCC][USB]
XPC_ERROR_SLOTEMPTY	IPM slot is empty, all except [PCC][USB]
XPC_ERROR_TSRES0	Mode for rx0 out of range
XPC_ERROR_TSRES1	Mode for rx1 out of range
XPC_ERROR_TSRES2	Mode for rx2 out of range
XPC_ERROR_TSRES3	Mode for rx3 out of range
XPC_FAILED	Command rejected by firmware.

Note This command sets the resolution of the 32-bit timestamp of received data per receiver to either 100 microsecs or 1 msec

Switching timestamp resolution while receiving data will invalidate timestamps of the up to then collected data items.

Default value after reset is a resolution of 1 msec for all receivers.

See also `_get_timestamp_res()` and `_set_high_resolution()`

`_get_timestamp_res()` [xPC]

Synopsis `int _get_timestamp_res(hmpc, ipnum, rx0, rx1, rx2, rx3)`

Description Gets the timestamp resolution mode for a certain A429PCC board resp. A429IPM module.

Argument	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
short ipnum	The number of the IP-Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME] ignored for [PCC][USB][HPVISA]
unsigned char *rx0	The mode of receiver 0
unsigned char *rx1	The mode of receiver 1
unsigned char *rx2	The mode of receiver 2
unsigned char *rx3	The mode of receiver 3
	Mode takes the following values: 0 - timestamp resolution 1ms 1 - timestamp resolution 0.1ms

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	IPM number out of range, all except [PCC][USB]
XPC_ERROR_SLOTEMPTY	IPM slot is empty, all except [PCC][USB]
XPC_FAILED	Command rejected by firmware.

Note This command retrieves the resolution of the 32-bit timestamp of either 100 microsecs or 1 msec of receivers

See also `_set_timestamp_res()` and `_set_high_resolution()`

`_set_high_resolution()` [xPC]

Synopsis `int _set_high_resolution(hmpc, slot, res)`

Description With this function it is possible to set the resolution of the high-resolution timestamp for time-tagging of receive-data to either 100 microseconds (default) or 10 microseconds.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
short slot	The number of the IP-Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME] ignored for [PCC][USB][HPVISA]
short res	The value for the resolution 10 = 10 microseconds 100 = 100 microseconds

Note: Only receivers which are configured for high-resolution timestamp are affected [use function `_set_timestamp_res()`]

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	IPM number out of range, all but [PCC][USB]
XPC_ERROR_SLOTEEMPTY	IPM slot is empty, all but [PCC][USB]
XPC_FAILED	Execution of command failed

See also `_set_timestamp_res()` and `_get_timestamp_res()`

_set_out_state() [PCC][HPVISA] (old interface)

Synopsis `int _set_out_state(hmpc, state)`

Description Sets the state of the discrete output line of a certain A429PCC board resp. A429IPM on VXI-carrier

Argument	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
short state	Linestate [0..1] 0 = low, 1 = high
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Command rejected by firmware.

See also `_get_in_state()` and `_epc_get_in_state()` **[EPC]** and `_pci2g_get_in_state()` and `_pci2g_set_out_state()` **[PCI-2G]**

Note (see below)

_set_out_state() [xPC] except [USB]

Synopsis `int _set_out_state(hmpc, ipnum, state)`

Description Sets the state of the discrete output line of a certain A429PCC board resp. A429IPM on any carrier

Argument	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
short ipnum	The number of the IP-Module 0..1 [PC104] 0..3 [EPC][PCI][PCI-2G][VME] ignored for [PCC][HPVISA]
short state	Linestate [0..1] 0 = low, 1 = high
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	IPM number out of range, all but [PCC]
XPC_ERROR_SLOTEMPTY	IPM slot is empty, all but [PCC]
XPC_FAILED	Command rejected by firmware.

See also `_get_in_state()` and `_epc_get_in_state()` [EPC] and `_pci2g_get_in_state()` and `_pci2g_set_out_state()` [PCI-2G]

Note This command is used to manually set the state (TTL voltage level LOW or HIGH) of the discrete output line IPMx_OUT/OUTPCC of one of the installed A429IPM modules or A429PCC.

`_get_in_state()` [PCC][HPVISA] (old interface)

Synopsis `int _get_in_state(hmpc)`

Description Gets the state of the discrete input line of a certain A429PCC board resp. A429IPM on VXI-carrier

Argument	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
Return Value	Explanation
0	State of line is low
1	State of line is high
XPC_FAILED	Command rejected by firmware.

Note (see below)

`_get_in_state()` [xPC] except [USB]

Synopsis `int _get_in_state(hmpc, ipnum)`

Description Gets the state of the discrete input line of a certain A429PCC board resp. A429IPM on any carrier

Argument	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
short ipnum	The number of the IP-Module 0..1 [PC104] 0..3 [EPC][PCI][PCI-2G][VME] ignored for [PCC][HPVISA]
Return Value	Explanation
0	State of line is low

1	State of line is high
XPC_ERROR_IPNUM	IPM number out of range, all but [PCC]
XPC_ERROR_SLOTEEMPTY	IPM slot is empty, all but [PCC]
XPC_FAILED	Command rejected by firmware.

Note This command is used to retrieve the state (TTL voltage level LOW or HIGH) of the discrete input line IPM_x_IN of one of the installed A429IPM or the INPCC of an A429PCC board.

National Instruments LabView™ specific interface functions

During the development of the LabVIEW™ Virtual Instrument driver library it was discovered that LabVIEW has difficulties with pointers to arrays of structures, as used in several interface functions. For this reason all those functions were duplicated as far as functionality is concerned but with a different set of arguments (pointers to arrays). The function names have been prefixed with the letters **_lv**.

_lv_get_transmitter_set() [xPC] except [HPVISA]

Synopsis `int _lv_get_transmitter_set(hmpc, tx_number, &bitrate, &litude)`

Description Returns the basic operating settings of a certain transmitter channel.

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
char tx_number	Transmitter number, range '0'..'1' [PCC][USB] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
unsigned short *bit_rate	The bit designation of bitrate_par_slope_set is as follows: The bit designation of bitrate_par_slope_set is as follows: bit 15..9: don't care bit 8: parity generation 0 = disabled 1 = enabled bit 7: transmit mode 0 = normal burst mode 1 = time optimized mode bit 6..5: slope 00 = 5 V/us [xPC] = high speed 10 = 1.0 V/us [xPC] = low speed bit 4: speed 0 = high 1 = low bit 3: parity 0 = odd 1 = even bit 2..0: bitrate in kBit/sec for low speed / high speed 001 = 12.50 / 100.0 [xPC]
unsigned short *amplitude	amplitude_set contains the amplitude with a resolution of 0.1 mV LSB

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_FAILED	Execution of command failed

See also *get_transmitter_mode* in the appropriate Hardware User Manual.
 _get_transmitter_set in this document.

_lv_define_trans() [xPC] except [HPVISA]

Synopsis `int _lv_define_trans(hmpc, tx_number, count, &trans_ID[0], &datalow[0], &datahigh[0], &ti[0], &result, ×lice)`

Description Defines transmissions by assigning transmission IDs with certain transmit intervals to a transmitter

Arguments	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
char tx_number	Transmitter number, range '0'..'1' [PCC][USB] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
unsigned short count	Number of definitions in arrays (1..256)
unsigned short *trans_ID	Pointer to array holding the transmission ID's. (Range 0..255)
unsigned short *datalow	Pointer to array holding the two least significant bytes of the ARINC 429 word
unsigned short *datahigh	Pointer to array holding the two most significant bytes of the ARINC 429 word
unsigned short *ti	Pointer to array holding the transmit interval in milliseconds
unsigned short *result	Result information regarding the definition: 0 = definition accepted and inserted 1 = too many definitions for timeslice 2 = resulting definition count is zero
unsigned short *timeslice	The calculated timeslice in milliseconds
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware - see <i>result</i> for further information
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_ERROR_DEFCOUNT	Count out of range
XPC_FAILED	Execution of command failed

Note Invocation of this function stops all transmission activities. No more than 256 transmissions can be defined for a single transmitter.

See also *define transmit data* in the appropriate Hardware User Manual and *_define_trans()* in this document.

`_lv_activate_trans()` [xPC] except [HPVISA]

Synopsis `int _lv_activate_trans(hmpc, tx_number, count, &trans_ID[0], &trig_ID[0], &buffer)`

Description Activates and assigns trigger IDs to transmission IDs of a certain transmitter

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
char tx_number	Transmitter number, range '0'..'1' [PCC][USB] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
unsigned short count	Activation count (1..256)
unsigned char *trans_ID	Pointer to array holding the transmission ID's.
unsigned char *trig_ID	Pointer to array holding the trigger ID's.
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_ERROR_ACTCOUNT	Count out of range
XPC_FAILED	Execution of command failed

Note Each trig_ID element defines the new state of the transmission for the corresponding trans_ID element:
255 = unconditional transmit
1..254 = conditional transmit
0 = don't transmit

See also *activate transmit data* in the appropriate Hardware User Manual
_activate_trans() in this document.

_lv_delete_trans() [xPC] except [HPVISA]

Synopsis `int _lv_delete_trans(hmpc, tx_number, count, &trans_ID[0], &result, timeslice)`

Description Deletes transmission definitions of a certain transmitter

Arguments	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
char tx_number	Transmitter number, range '0'..'1' [PCC][USB] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
unsigned short count	Number of deletions (1..256)
unsigned short *trans_ID	Pointer to array holding the transmission ID's to be deleted. All the given transmission ID's will be removed from the transmission definition list.
unsigned short *result	Result information regarding the deletion: 0 = deletion accepted 1 = too many definitions for timeslice 2 = resulting definition count is zero
unsigned short *timeslice	The calculated timeslice in milliseconds
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware - see <i>result</i> for further information
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_ERROR_DELCOUNT	Count out of range
XPC_FAILED	Execution of command failed

Note Stops all transmit activities for the given transmitter.

See also *delete transmit data* in the appropriate Hardware User Manual
 _delete_trans() in this document.

_lv_update_trans() [xPC] except [HPVISA]

Synopsis `int _lv_update_trans(hmpc, tx_number, count, &trans_ID[0],
 &datalow[0], &datahigh[0])`

Description Updates the data values of already defined ARINC-429 transmissions

Arguments	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
char tx_number	Transmitter number, range '0'..'1' [PCC][USB] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
unsigned short count	Number of definitions in buffers
unsigned short *trans_ID	Pointer to array holding the transmission ID's to be updated.
unsigned short *datalow	Pointer to array holding the two least significant bytes of the ARINC 429 word
unsigned short *datahigh	Pointer to array holding the two most significant bytes of the ARINC 429 word

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_ERROR_DEFCOUNT	Count out of range
XPC_FAILED	Execution of command failed

Note The state of the transmission isn't changed.

See also *update transmit data* in the appropriate Hardware User Manual
 _update_trans() in this document.

`_lv_get_tx_definitions()` [xPC] except [HPVISA]

Synopsis `int _lv_get_tx_definitions(hmpc, tx_number, &trans_ID[0],
&trig_ID[0], &datalow[0], &datahigh[0], &ti_ms[0], &ti_ts[0],
&ticks[0], &flags[0], &result, &ndef)`

Description Retrieves the transmit definitions for a certain transmitter

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
char tx_number	Transmitter number, range '0'..'1' [PCC][USB] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
unsigned char *trans_ID	Pointer to array to hold the transmission ID's.
unsigned char *trig_ID	Pointer to array to hold the trigger ID's.
unsigned short *datalow	Pointer to array to hold the two least significant bytes of the ARINC 429 word
unsigned short *datahigh	Pointer to array to hold the two most significant bytes of the ARINC 429 word.
unsigned short *ti_ms	Pointer to array to hold the transmit interval value.
unsigned short *ti_ts	Pointer to array to hold the timeslice value.
unsigned short *ticks	Pointer to array to hold the ticks-til-transmission value.
unsigned short *flags	Pointer to array to hold the flag value.
unsigned short *result	0 = transmission definitions retrieved 1 = no transmission definitions available
unsigned short *ndef	number of definitions retrieved from board
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware - see <i>result</i> for further information
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_FAILED	Execution of command failed

Note If the size of the **arrays** is too small the results will be unpredictable.

See also *get all transmit data definitions* in the appropriate Hardware User Manual.
_get_tx_definitions() in this document.

_lv_get_single_tx_definition() [xPC] except [HPVISA]

Synopsis `int _lv_get_single_tx_definition(hmpc, tx_number, trans_ID, &trig_ID, &datalow, &datahigh, &ti_ms, &ti_ts, &ticks, &flags, &result)`

Description Retrieves a single transmit data definition of a certain receiver

Arguments	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
char tx_number	Transmitter number, range '0'..'1' [PCC][USB] '0'..'3' [PC104] '0'..'7'['b'] [EPC][PCI][PCI-2G][VME]
unsigned short trans_ID	The transmission ID (0..255) for which the definition should be retrieved
unsigned char *trig_ID	Pointer to variable to hold trigger ID.
unsigned short *datalow	Pointer to variable to hold the least significant two bytes of the ARINC 429 word.
unsigned short *datahigh	Pointer to variable to hold the most significant two bytes of the ARINC 429 word.
unsigned short *ti_ms	Pointer to variable to hold the transmit interval.
unsigned short *ti_ts	Pointer to variable to hold the timeslice value.
unsigned short *ticks	Pointer to variable to hold the ticks-til-transmission value.
unsigned short *flags	Pointer to variable to hold the transmit flag value.
unsigned short *result	0 = transmit data definition copied to buffer 1 = no definition available for this transmission ID
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware - see <i>result</i> for further information
XPC_ERROR_TX_NUMBER	Invalid transmitter number specified
XPC_ERROR_TX_ID	Invalid transmission ID specified
XPC_FAILED	Execution of command failed

See also *get single transmit data definition* in the appropriate Hardware User Manual
_get_single_tx_definition() in this document.

`_lv_read_receiver_buffer()` [xPC] except [HPVISA]

Synopsis `int _lv_read_receiver_buffer(hmpc, rx_number, &rtime[0], &alow[0], &ahigh[0], &result, &nentries)`

Description Retrieves received data from the board.

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
char rx_number	The receiver number: '0'..'3' [PCC][USB] '0'..'7' [PC104] '0'..'9','a'..'f'['n'] [EPC][PCI][PCI-2G][VME]
unsigned long *rtime	Pointer to array to hold the timestamp values.
unsigned short *alow	Pointer to array to hold the two least significant bytes of the ARINC 429 word.
unsigned short *ahigh	Pointer to array to hold the two most significant bytes of the ARINC 429 word.
	Note: The number of elements for the above arrays should be the same as defined with the <code>set_rec_buf_size()</code> call.
unsigned short *result	0 = data available and stored in the arrays 1 = no data available
unsigned short *nentries	number of elements stored in arrays
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware - see <i>result</i> for further information
XPC_ERROR_RX_NUMBER	Invalid receiver number specified
XPC_FAILED	Execution of command failed

Note If the size of **arrays** is too small, the results will be unpredictable.

See also *read receiver buffer* in the appropriate Hardware User Manual
 `_read_receiver_buffer()` in this document.

_lv_activate_rec() [xPC] except [HPVISA]

Synopsis `int _lv_activate_rec(hmpc, rx_number, act_code, lmask, hmask, lmatch, hmatch, &status)`

Description Sets the receive mode for a certain receiver

Arguments	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call
char rx_number	The receiver number: '0'..'3' [PCC][USB] '0'..'7' [PC104] '0'..'9','a'..'f'['n'] [EPC][PCI][PCI-2G][VME]
unsigned short act_code	Activation code: 0 = stop receiving 1 = start unmasked receiving 2 = start masked receiving 3 = continue in previously selected mode
unsigned short lmask	least significant two bytes of mask word
unsigned short hmask	most significant two bytes of mask word
unsigned short lmatch	least significant two bytes of match word
unsigned short hmatch	most significant two bytes of match word
	Note: The mask and match words are only evaluated if act_code == 2 is specified.
unsigned short *status	Shows the mode of operation for the receiver: bit 0 set: receiver operates in trigger mode bit 1 set: receiver operates in storage mode bit 2 set: receiver operates in masked mode
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_RX_NUMBER	Invalid transmitter number specified
XPC_ERROR_ACTIONCODE	Invalid action code specified
XPC_FAILED	Execution of command failed

See also *activate receiver* in the appropriate Hardware User Manual
 _activate_rec() in this document.

`_lv_define_trigger()` [xPC] except [HPVISA]

Synopsis `int _lv_define_trigger(hmpc, rx_number, count, &trig_ID[0],
&acode[0], &lmask[0], &hmask[0], &lmatch[0], &hmatch[0],
&status)`

Description Defines up to eight trigger events for a certain receiver.

Arguments	Explanation
HANDLE hmpc	The handle returned by the <code>_open_xpc()</code> call.
char rx_number	The receiver number: '0'..'3' [PCC][USB] '0'..'7' [PC104] '0'..'9','a'..'f'['n'] [EPC][PCI][PCI-2G][VME]
unsigned short count	Denotes the number of trigger events to define. A value of zero terminates trigger mode.
unsigned char *trig_ID	Pointer to array with the trigger ID's The range for trig_ID is 0..255.
unsigned char *acode	Pointer to array with the action codes The value of acode denotes the action to be taken: 0 = Stop on match 1 = Start on match 2 = Stop on mismatch 3 = Start on mismatch
unsigned short *lmask	Pointer to array with the two least significant bytes of the mask words.
unsigned short *hmask	Pointer to array with the two most significant bytes of the mask words.
unsigned short *lmatch	Pointer to array with the two least significant bytes of the match words.
unsigned short *hmatch	Pointer to array with the two most significant bytes of the match words.
unsigned short *status	Shows upon completion the mode the receiver is operating in: bit 0 set = receiver operates in trigger mode bit 1 set = receiver operates in storage mode bit 2 set = receiver operates in masked mode
Return Value	Explanation
XPC_SUCCESS	Trigger definition accepted
XPC_ERROR_RX_NUMBER	Invalid receiver number specified
XPC_ERROR_TRIG_COUNT	Trigger count out of range
XPC_FAILED	Command execution failed

See also *define trigger* in the appropriate Hardware User Manual and
_define_trigger() in this document

A429IPM specific interface functions

This section outlines all interface library functions, that are common to all IP-carriers equipped with the A429IPM IP-Module or any other.

These functions are general purpose functions that can be used for any type of IndustryPack-module from any vendor installed on the appropriate IP-carrier board !

Information in terms of respective resources (IO-space/Memory-space/ID-space) and how to access them through the IP-carrier (offset,mapping,...) must be retrieved from documentation of the IP-module and the IP-carrier

`_ipm_ipreadbyte()` [xPC] except [PCC][USB][HPVISA]

Synopsis `int _ipm_ipreadbyte(hmpc, slot, reg, &data)`

Description Read a byte from an IP byte register - for register numbers please refer to the documentation of the IP module concerned.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
short slot	The number of the IP-Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME]
short reg	The IP register number to read a byte from
unsigned char *data	Pointer to a variable where to store the read data.

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	Invalid slot number specified
XPC_ERROR_SLOTEEMPTY	There's no IP module installed in the chosen slot number.
XPC_FAILED	Execution of command failed

See also `_ipm_ipreadword()`, `_ipm_ipwritebyte()`, `_ipm_ipwriteword()` in this manual

`_ipm_ipwritebyte()` [xPC] except [PCC][USB][HPVISA]

Synopsis `int _ipm_ipwritebyte(hmpc, slot, reg, data)`

Description Write a byte to an IP byte register - for register numbers please refer to the documentation of the IP module concerned.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
short slot	The number of the IP-Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME]
short reg	The IP register number to read a byte from
unsigned char data	Data that should be written to the register.
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	Invalid slot number specified
XPC_ERROR_SLOTEMPTY	There's no IP module installed in the chosen slot number.
XPC_FAILED	Execution of command failed

See also `_ipm_ipreadword()`, `_ipm_ipreadbyte()`, `_ipm_ipwriteword()` in this manual

`_ipm_ipreadword()` [xPC] except [PCC][USB][HPVISA]

Synopsis `int _ipm_ipreadword(hmpc, slot, reg, &data)`

Description Read a word from an IP word register - for register numbers please refer to the documentation of the IP module concerned.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
short slot	The number of the IP-Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME]
short reg	The IP register number to read a byte from
unsigned short *data	Pointer to variable to hold the read data
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	Invalid slot number specified
XPC_ERROR_SLOTEEMPTY	There's no IP module installed in the chosen slot number.
XPC_FAILED	Execution of command failed

See also `_ipm_ipwritebyte()`, `_ipm_ipreadbyte()`, `_ipm_ipwriteword()` in this manual

`_ipm_ipreadword()` [HPVISA]

Synopsis `int _ipm_ipreadword(hmpc, reg, &data)`

Description Read a word from an IP word register - for register numbers please refer to the documentation of the IP module concerned.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
short reg	The IP register number to read a byte from
unsigned short *data	Pointer to variable to hold the read data
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

See also `_ipm_ipwritebyte()`, `_ipm_ipreadbyte()`, `_ipm_ipwriteword()` in this manual

`_ipm_ipwriteword()` [xPC] except [PCC][USB][HPVISA]

Synopsis `int _ipm_ipwriteword(hmpc, slot, reg, data)`

Description Write a word to an IP word register - for register numbers please refer to the documentation of the IP module concerned.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
short slot	The number of the IP-Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME]
short reg	The IP register number to read a byte from
unsigned short data	Data that should be written to the register
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	Invalid slot number specified
XPC_ERROR_SLOTEEMPTY	There's no IP module installed in the chosen slot number.
XPC_FAILED	Execution of command failed

See also `_ipm_ipwritebyte()`, `_ipm_ipreadbyte()`, `_ipm_ipreadword()` in this manual

`_ipm_ipwriteword()` [HPVISA]

Synopsis `int _ipm_ipwriteword(hmpc, reg, data)`

Description Write a word to an IP word register - for register numbers please refer to the documentation of the IP module concerned.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
short reg	The IP register number to read a byte from
unsigned short data	Data that should be written to the register
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

See also `_ipm_ipwritebyte()`, `_ipm_ipreadbyte()`, `_ipm_ipreadword()` in this manual

`_ipm_ipreadbytemem()` [xPC] except PCC][USB][HPVISA]

Synopsis `int _ipm_ipreadbytemem(hmpc, slot, off, len, &data)`

Description Read bytes from the IP memory - for memory offsets please refer to the documentation of the IP module concerned.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
short slot	The number of the IP-Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME]
unsigned long off	The IP memory offset
short len	The number of bytes to read 1..4096
unsigned char *data	Pointer to variable to hold the read data
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	Invalid slot number specified
XPC_ERROR_SLOTEEMPTY	There's no IP module installed in the chosen slot number.
XPC_ERROR_BYTECOUNT	The <code>len</code> parameter is out of range
XPC_FAILED	Execution of command failed

See also `_ipm_ipwritebytemem()`, `_ipm_ipreadwordmem()`, `_ipm_ipwritewordmem()` in this manual

`_ipm_ipwritebytemem()` [xPC] except [PCC][USB][HPVISA]

Synopsis `int _ipm_ipwritebytemem(hmpc, slot, off, len, data)`

Description Write bytes to the IP memory - for memory offsets please refer to the documentation of the IP module concerned.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
short slot	The number of the IP-Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME]
unsigned long off	The IP memory offset
short len	The number of bytes to write 1..4096
unsigned char *data	Data to write to the board
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	Invalid slot number specified
XPC_ERROR_SLOTEEMPTY	There's no IP module installed in the chosen slot number.
XPC_ERROR_BYTECOUNT	The <i>len</i> parameter is out of range
XPC_FAILED	Execution of command failed

See also `_ipm_ipreadbytemem()`, `_ipm_ipreadwordmem()`, `_ipm_ipwritewordmem()` in this manual

`_ipm_ipreadwordmem()` [xPC] except [PCC][USB][HPVISA]

Synopsis `int _ipm_ipreadwordmem(hmpc, slot, off, len, &data)`

Description Read words from the IP memory - for memory offsets please refer to the documentation of the IP module concerned.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
short slot	The number of the IP-Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME]
unsigned long off	The IP memory offset
short len	The number of words to read 1..2048
unsigned char *data	Pointer to variable to hold the read data

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	Invalid slot number specified
XPC_ERROR_SLOTEEMPTY	There's no IP module installed in the chosen slot number.
XPC_ERROR_WORDCOUNT	The <i>len</i> parameter is out of range
XPC_FAILED	Execution of command failed

See also `_ipm_ipwritebytemem()`, `_ipm_ipreadbytemem()`, `_ipm_ipwritewordmem()` in this manual

`_ipm_ipreadwordmem()` [HPVISA]

Synopsis `int _ipm_ipreadwordmem(hmpc, off, len, &data)`

Description Read words from the IP memory - for memory offsets please refer to the documentation of the IP module concerned.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
unsigned long off	The IP memory offset
short len	The number of words to read 1..2048
unsigned char *data	Pointer to variable to hold the read data
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_WORDCOUNT	The <i>len</i> parameter is out of range
XPC_FAILED	Execution of command failed

See also **`_ipm_ipwritebytemem()`, `_ipm_ipreadbytemem()`, `_ipm_ipwritewordmem()`** in this manual

`_ipm_ipwritewordmem()` [xPC] except [PCC][USB][HPVISA]

Synopsis `int _ipm_ipwritewordmem(hmpc, slot, off, len, data)`

Description Write words to the IP memory - for memory offsets please refer to the documentation of the IP module concerned.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
short slot	The number of the IP-Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME]
unsigned long off	The IP memory offset
short len	The number of words to write 1..2048
unsigned char *data	Data to write to the board
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	Invalid slot number specified
XPC_ERROR_SLOTEEMPTY	There's no IP module installed in the chosen slot number.
XPC_ERROR_WORDCOUNT	The <i>len</i> parameter is out of range
XPC_FAILED	Execution of command failed

See also `_ipm_ipreadbytemem()`, `_ipm_ipwritebytemem()`, `_ipm_ipreadwordmem()` in this manual

`_ipm_ipwritewordmem()` [HPVISA]

Synopsis `int _ipm_ipwritewordmem(hmpc, off, len, data)`

Description Write words to the IP memory - for memory offsets please refer to the documentation of the IP module concerned.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
unsigned long off	The IP memory offset
short len	The number of words to write 1..2048
unsigned char *data	Data to write to the board
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_WORDCOUNT	The <i>len</i> parameter is out of range
XPC_FAILED	Execution of command failed

See also `_ipm_ipreadbytemem()`, `_ipm_ipwritebytemem()`, `_ipm_ipreadwordmem()` in this manual

_ipm_ipreadid() [xPC] except [PCC][USB][HPVISA]

Synopsis `int _ipm_ipreadid(hmpc, slot, len, &data)`

Description Read the IP ID from the IPs EPROM.

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
short slot	The number of the IP-Module 0..1 [PC104] 0..3[5] [EPC][PCI][PCI-2G][VME]
short len	Number of Bytes to read
unsigned char *data	Pointer to a character array where to store the ID string. The user has to dimension the array big enough to hold the ID string.

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_ERROR_IPNUM	Invalid slot number specified
XPC_ERROR_SLOTEEMPTY	There's no IP module installed in the chosen slot number.
XPC_FAILED	Execution of command failed

See also

_ipm_ipreadid() [HPVISA]

Synopsis `int _ipm_ipreadid(hmpc, len, &data)`

Description Read the IP ID from the Ips EPROM.

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
short len	Number of Bytes to read
unsigned char *data	Pointer to a character array where to store the ID string. The user has to dimension the array big enough to hold the ID string.
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

See also

_ipm_scan_ipm() [xPC] except [PCC][USB][HPVISA]

Synopsis `int _ipm_scan_ipm(hmpc, &mod_inf)`

Description Perform an IP module scan on the carrier board.

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
char *mod_inf[x]	Pointer to character array to hold type information of IP module [x], x=0-5
	The following type information will be returned in mod_inf: 0 = No module installed in that slot 1 = Unprogrammed A429IPM (TechSAT) 2 = Programmed A429IPM (TechSAT) 3 = Third party IP module
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

See also

_ipm_scan_ipm() [HPVISA]

Synopsis `int _ipm_scan_ipm(hmpc, &mod_inf)`

Description Perform an IP module scan on the VXI carrier board.

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
char *mod_inf	Pointer to character to hold type information of the module
	The following type information will be returned in mod_inf: 0 = No module installed in that slot 1 = Unprogrammed A429IPM (TechSAT) 2 = Programmed A429IPM (TechSAT) 3 = Third party IP module
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

See also

_ip429_reset() [xPC] except [PCC][USB]

Synopsis `int _ip429_reset(hmpc, ipnum)`

Description Reset the TechSAT ARINC 429 IP-Module (via SW).

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
short ipnum	The number of the IP-Module 0..3[5] [xPC]
Return Value	Explanation
XPC_FAILED	Execution of command failed
XPC_SUCCESS	Execution of command succeeded
XPC_ERROR_SLOTEMPTY	No IP-Module installed in the given slot number.
XPC_ERROR_IPNUM	Given IP-Number is out of range.

See also

A429EPC specific interface functions

This section outlines all interface library functions, that are specific to the A429EPC-board.

`_epc_write_routing()` [EPC]

Synopsis `int _epc_write_routing(hmpc, chipno, rtype)`

Description Write the programming information to a certain ispGDS14 routing chip.
With this function you can alter the timestamping behaviour of the associated IP-Module.

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
short chipno	The number of the routing chip [0..3] The chip number denotes the affected IP-Module (chipno 0 == IPM 0).
short rtype	The type of routing to program: 0 = erase programming 1 = use associated IPM in timestamping slave mode, master on same board 2 = use associated IPM in timestamping slave mode, master external (lab signal or a different A429EPC board). 3 = use associated IPM in timestamping master mode

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed
XPC_ERROR_CHIPNO	Chip number out of range

See also `_set_timestamp_synchro_mode()`

Note The user is responsible to wire the signals externally (connect the 1 HZ signal to all used IPMx_SYNCH_IN lines) when an external source is used. If IPM0 is the MASTER the connections from IPM0_SYNCH_IN/OUT to the IPMx_SYNCH_IN lines have to be routed internally on the A429EPC carrier. The routing is implemented with in-system programmable switches ispGDS14.

In order to configure correct routing the interface function `_epc_write_routing` **[EPC]** has to be used.

The configuration modes:

erase erase previously configured programming/routing.

Attention: Erasing the ispGDS14 also disconnects the A429IPM discrete signals IPMx_IN and IPMx_OUT from the IO-connector. In any other mode these signals are routed to the IO-connector.

internal slave route signals, so that respective module is internally synchronized from module IPM0

external slave route signals, so that respective module is externally synchronized

master route signals, so that respective module is generating 1 Hz signal as synchronization master

Attention:

The ispGDS14 chips can be re-programmed up to 10000 times. Be careful not to use this function in applications that perform this function continuously.

A once programmed routing should only be modified if required. You can use the stand-alone executable epcd16.exe (or epcd32.exe)

in order to manually program the switches.

`_epc_set_out_state()` [EPC]

Synopsis `int _epc_set_out_state(hmpc, state)`

Description Sets the state of the discrete output line IPC_OUT of the A429EPC board/carrier

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
short state	The state to set line to [0..1] 0 = low, 1 = high

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

See also `_epc_get_in_state()`

_epc_get_in_state() [EPC]

Synopsis `int _epc_get_in_state(hmpc)`

Description Gets the state of the discrete input line IPC_IN of the A429EPC carrier/board

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
Return Value	Explanation
0	State of line is LOW
1	State of line is HIGH
XPC_FAILED	Execution of command failed

See also `_epc_set_out_state()`

_epc_firmware_version() [EPC]

Synopsis `int _epc_firmware_version(hmpc, vstring)`

Description Retrieve the firmware version string from the A429EPC board

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
unsigned char *vstring	Pointer to a string to hold the version string retrieved from the board. The dimension of <i>vstring</i> has to be big enough to hold eight characters. <i>vstring</i> will not be terminated with a null character. The returned string has the following format: <i>jjnndddy</i> where jj major version number nn minor version number ddd day of year the firmware was built y last digit of year the firmware was built thus, a version string like 01050217 would read: 01.05 is the firmware version and the firmware was built on the 21 st day of 1997. Current version [June 2000]: 01400429
Return Value	Explanation
XPC_FAILED	Execution of command failed
XPC_SUCCESS	Execution of command succeeded

Note **The A429EPC carrier firmware can be customized on request !!!**

A429PCI-2G specific interface functions

This section outlines all interface library functions, that are specific to the A429PCI-2G board.

_pci2g_set_out_state() [PCI-2G]

Synopsis `int _epc_set_out_state(hmpc, state)`

Description Sets the state of the discrete output lines (open/close of the solid state relays) of the A429PCI-2G board/carrier

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
short state	The state of the discrete Output lines, resp. open/close contact of solid state relays: [0..1] 0 = open, 1 = close: Bit 0: Discrete Out 0 (J2, Pins 19/20) Bit 1: Discrete Out 1 (J2, Pins 21/22) Bit 2: Discrete Out 2 (J2, Pins 23/24) Bit 3: Discrete Out 2 (J2, Pins 25/26)

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

See also `_pci2g_get_in_state()`

`_pci2g_get_in_state()` [PCI-2G]

Synopsis `int _epc_get_in_state(hmpc, &ddata)`

Description Retrieves the state of the discrete input lines of the A429PCI-2G board/carrier

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
unsigned short *ddate	Parameter holding the state of the various discrete input lines: Type Open/GND: 0 = open / 1 = GND Type 28V/Open: 0 = open / 1 = 28V Bit 0 : 28V/Open IN0 Bit 1 : 28V/Open IN1 Bit 2 : 28V/Open IN2 Bit 3 : 28V/Open IN3 Bit 4 : 28V/Open IN4 Bit 5 : 28V/Open IN5 Bit 6 : 28V/Open IN6 Bit 7 : 28V/Open IN7 Bit 8: Open/GND IN8 Bit 9: Open/GND IN9 Bit 10: Open/GND IN10 Bit 11: Open/GND IN11 Bit 12: Open/GND IN12 Bit 13: Open/GND IN13 Bit 14: Open/GND IN14 Bit 15: Open/GND IN15

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

See also `_pci2g_set_out_state()`

_pci2g_firmware_version() [PCI-2G]

Synopsis `int _epc_firmware_version(hmpc, vstring)`

Description Retrieve the firmware version string from the A429PCI-2G board

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
unsigned char *vstring	Pointer to a string to hold the version string retrieved from the board. The dimension of <i>vstring</i> has to be big enough to hold eight characters. <i>vstring</i> will not be terminated with a null character. The returned string has the following format: jjnndddy where jj major version number nn minor version number ddd day of year the firmware was built y last digit of year the firmware was built thus, a version string like 01050217 would read: 01.05 is the firmware version and the firmware was built on the 21 st day of 1997. Current version [Feb 2003]: 01030083
Return Value	Explanation
XPC_FAILED	Execution of command failed
XPC_SUCCESS	Execution of command succeeded

Note **The A429PCI-2G carrier firmware can be customized on request !!!**

A429USB specific interface functions

This section outlines all interface library functions, that are specific to the A429USB USB-Device.

`_usb_get_discretes()` [USB]

Synopsis `int _usb_get_discretes(hmpc, discr)`

Description Gets the state of the discrete input lines of a certain A429USB device.

Argument	Explanation
HANDLE hmpc unsigned short *discr	The handle returned by the <code>_open_xpc()</code> call. Each bit represent a discrete input line: Bit 0: Open/Gnd Input #1 [0= open; 1= GND] Bit 1: Open/Gnd Input #2 [0= open; 1= GND] Bit 2: Open/Gnd Input #3 [0= open; 1= GND] Bit 3: Open/Gnd Input #4 [0= open; 1= GND] Bit 4: Open/Gnd Input #5 [0= open; 1= GND] Bit 5: Open/Gnd Input #6 [0= open; 1= GND] Bit 6: n/a Bit 7: n/a Bit 8: 28V/Open Input #1 [1= open; 0= 28V] Bit 9: 28V/Open Input #2 [1= open; 0= 28V] Bit 10: 28V/Open Input #3 [1= open; 0= 28V] Bit 11: 28V/Open Input #4 [1= open; 0= 28V] Bit 12: 28V/Open Input #5 [1= open; 0= 28V] Bit 13: 28V/Open Input #6 [1= open; 0= 28V] Bit 14: n/a Bit 15: n/a

Return Value	Explanation
XPC_FAILED	Command rejected by firmware.

Note

See also `_usb_set_discretes()`

_usb_set_discretes() [USB]

Synopsis `int _usb_set_discretes(hmpc, mask, data)`

Description Sets the state of the discrete input lines of a certain A429USB device.

Argument	Explanation
HANDLE hmpc	The handle returned by the _open_xpc() call.
unsigned short mask	Mask for the respective discrete output(s) that should be set. Bit 0: output FD4 (Pin 25) Bit 1: output FD3 (Pin 23) Bit 3: output FD2 (Pin 22) Bit 4: output FD1 (Pin 21)
unsigned short data	Value for the respective discrete output:. Bit 0: 0 = output FD4 (Pin 25) to Pin 24: open 1 = output FD4 (Pin 25) to Pin 24: closed Bit 1: 0 = output FD3 (Pin 23) to Pin 24: open 1 = output FD3 (Pin 23) to Pin 24: closed Bit 2: 0 = output FD2 (Pin 22) to Pin 9: open 1 = output FD2 (Pin 22) to Pin 9: closed Bit 3: 0 = output FD1 (Pin 21) to Pin 9: open 1 = output FD1 (Pin 21) to Pin 9: closed
Return Value	Explanation
XPC_FAILED	Command rejected by firmware.

Note

See also `_usb_get_discretes()`

Serial RS-xxx Interface functions [PCC][EPC]

This section outlines all interface library functions dealing with the serial RS-xxx Interface functions of [PCC] and [EPC] .

The RX-xxx feature is an option to the A429EPC and A429PCC.

`_enable_async()` [PCC][EPC]

Synopsis `int _enable_async(hmpc, serial_cfg_t *config_data)`

Description configures/enables the serial interface (RS-232/422) on the A429PCC resp. A429EPC carrier-board

Arguments	Explanation
HANDLE hmpc	The handle returned by <code>_open_xpc()</code>
serial_cfg_t *config data	Pointer to structure with the following layout:

```
typedef struct
{
    unsigned char data_frame;           /*1: 8-bit , 2: 7-bit, 3: 9-bit */
    unsigned char stop_bits;           /*0: one stop bit, 1: two stop bits*/
    unsigned char enable_receiver;     /*0: Rx disabled, 1: Rx enabled*/
    unsigned char parity_check_enable; /*0: not enabled, 1: enabled*/
    unsigned char framing_check_enable; /*0: not enabled, 1: enabled*/
    unsigned char overrun_check_enable; /*0: not enabled, 1: enabled*/
    unsigned char parity;              /*0: even parity, 1: odd parity*/
    unsigned char loopback;            /*0: disabled, 1: enabled*/
    unsigned short baudrate_reload_val
    /*baudrate_reload_value = CPU-frequency (20 MHz) / (32 * baudrate) - 1*/
}
serial_cfg_t;
```

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

Note

_disable_async() [PCC][EPC]

Synopsis `int _disable_async(hmpc)`

Description disables the serial interface (RS-232/422) on the A429PCC resp. A429EPC carrier-board

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

Note

_set_serial_rx_buf_size() [PCC][EPC]

Synopsis `int _set_serial_rx_buf_size(hmpc, size)`

Description defines the size of the serial RS-xxx receive-buffer on the A429PCC resp. A429EPC carrier-board

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
unsigned short size	size of buffer – numer of <code>serial_data_t</code> array elements (see function <code>_read_serial_buf()</code>); allowed values: 0..128
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

Note

_tx_serial_data() [PCC][EPC]

Synopsis `int _tx_serial_data(hmpc, data)`

Description Transmits one data unit (7,8 or 9 bits) on the serial interface (RS-232/422)

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
unsigned short data	The data to
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

Note

_rx_serial_data() [PCC][EPC]

Synopsis `int _rx_serial_data(hmpc, &byte_available, &serial_error, &data)`

Description Receives one data unit (7,8 or 9 bits) on the serial interface (RS-232/422)

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
unsigned short *byte_available	Indicates if a serial item could be retrieved: 0 : item was vailable 1 : no item was available
unsigned short *serial_error	Indicates if an error occurred: 0 : no error 1 : receive error occurred unsigned short
unsigned short *data	the data item
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

Note

_read_serial_buf() [PCC][EPC]

Synopsis `int _read_serial_buf(hmpc, serial_data_t *buffer, &result, &num_entry)`

Description Retrieves contents of a receiver buffer on the serial interface (RS-232/422)

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
serial_data_t *buffer	Pointer to array of structures structure with the following layout:
<pre>typedef struct { unsigned short serial_data; /*the receive item */ unsigned short error_status; /*error status*/ unsigned long receive_time; /*receive time in msecs */ }</pre>	
serial_data_t;	error_status:
	bit 0 = Parity error
	bit 1 = Framing error
	bit 2 = On-Chip Buffer Overrun error
	bit 3 = Receiver Buffer Overrun error
unsigned short *result	Indicates if a serial data could be retrieved: 0 : no data available 1 : data available 2 : data with errors available
unsigned short *num_entry	number of valid items in the receive buffer
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

Note not yet implemented for A429EPC-board !

_get_serial_data_ready() [PCC][EPC]

Synopsis `int _get_serial_data_ready (hmpc,&rx_rdy)`

Description Returns the number of data items currently available in the receiver data buffer on the serial interface (RS-232/422)

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
unsigned short *rx_rdy	number of available items in receive buffer
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

Note

IP carrier / IP module reset functions

This section outlines all interface library functions, that are necessary to reset IndustryPack carrier boards.

_epc_reset() [EPC]

Synopsis `int _epc_reset(hmpc)`

Description Perform a firmware reset on the carrier board and all installed IP Modules. Additionally the loopback relays will be reseted too.

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

Note It is mandatory to perform this command prior to any other interface function.

_pci_reset() [PCI]

Synopsis `int _pci_reset(hmpc)`

Description Perform a firmware reset on the carrier board and all installed IP Modules.

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

Note It is mandatory to perform this command prior to any other interface function.

_pci2g_reset() [PCI-2G]

Synopsis `int _pci2g_reset(hmpc)`

Description Perform a firmware reset on the carrier board and all installed IP Modules.

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

Note It is mandatory to perform this command prior to any other interface function.

_ipc01_reset() [VME]

Synopsis `int _ipc01_reset(hmpc)`

Description Perform a reset on the VME IPC01 carrier board and all installed IP Modules.

Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

Note It is mandatory to perform this command prior to any other interface function.

_pc104_reset() [EPC]

Synopsis `int _pc104_reset(hmpc)`

Description Perform a reset on the PC/104 carrier board and all installed IP Modules.

Arguments	Explanation
-----------	-------------

HANDLE hmpc	The handle returned by _open_xpc()
-------------	---

Return Value	Explanation
XPC_SUCCESS	Command accepted by firmware
XPC_FAILED	Execution of command failed

Note It is mandatory to perform this command prior to any other interface function.

_ipm_ipreset() [EPC][PCI-2G]

Synopsis `int _ipm_ipreset(hmpc, ipnum)`

Description Reset a particular A429IPM IP-Module. This function applies only to IP-modules installed on the A429EPC or A429PCI-2G boards. The IP reset signal is toggled - the implementation is carrier specific. Most of the IP-modules are reset on power-up.

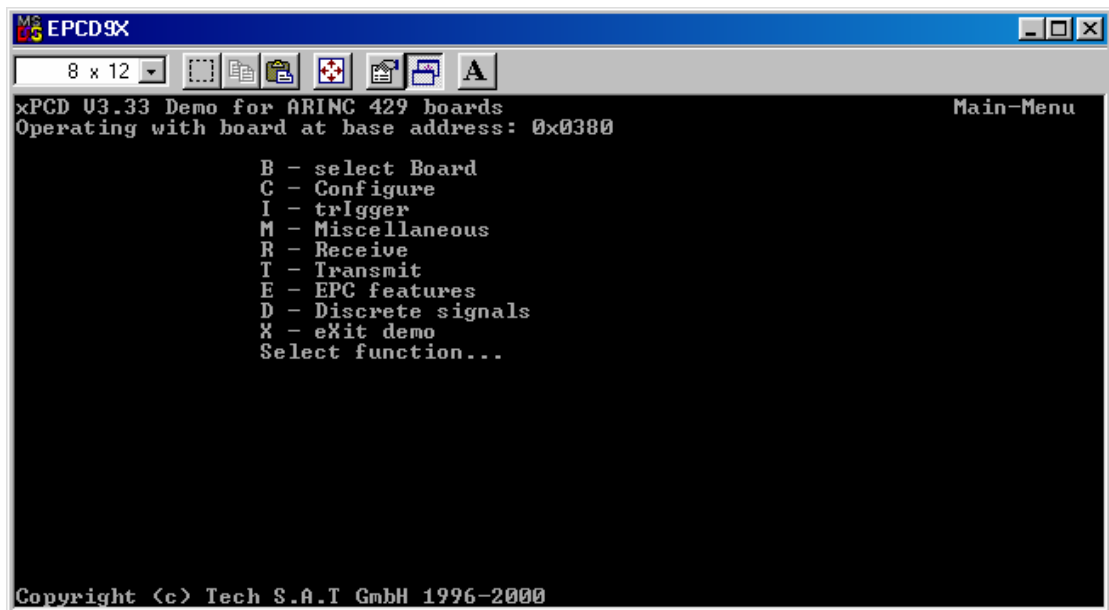
Arguments	Explanation
HANDLE hmpc	The handle returned by _open_xpc()
short ipnum	The number of the IP-Module 0..3 [EPC][PCI-2G]

Return Value	Explanation
XPC_FAILED	Execution of command failed
XPC_SUCCESS	Execution of command succeeded
XPC_ERROR_SLOTEMPTY	No IP-Module installed in the given slot number.
XPC_ERROR_IPNUM	Given IP-Number is out of range.

See also

Appendix: xPCD.c – executable example for [EPC]

Main Menu



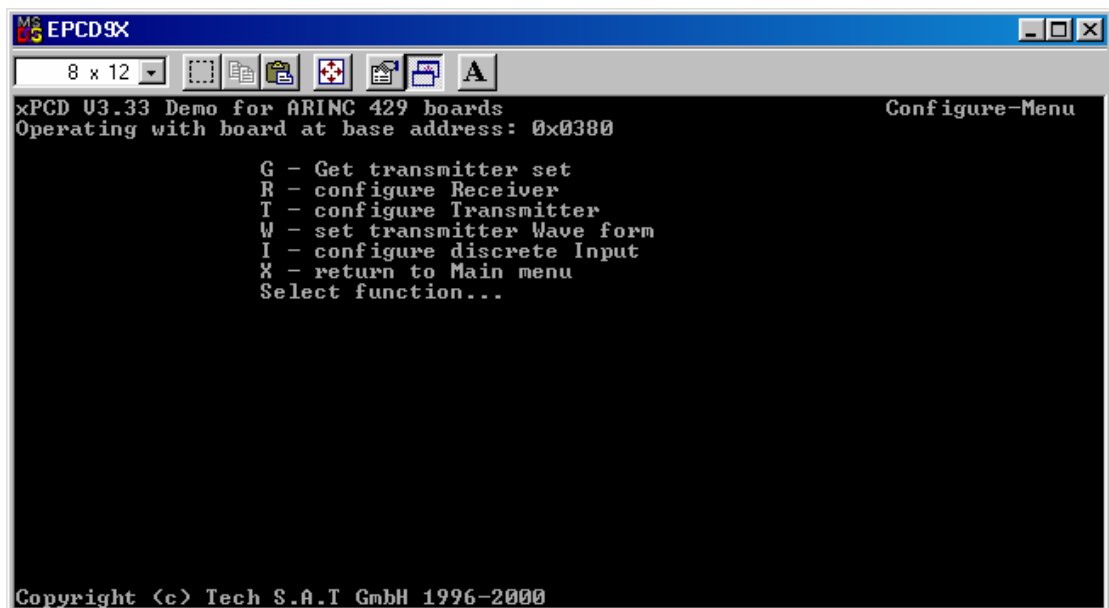
The screenshot shows a Windows-style window titled "EPCD9X". The menu bar contains a font size dropdown set to "8 x 12" and several icons. The main text area has a black background with white text. At the top left, it says "xPCD U3.33 Demo for ARINC 429 boards" and "Operating with board at base address: 0x0380". At the top right, it says "Main-Menu". In the center, there is a list of functions: B - select Board, C - Configure, I - trigger, M - Miscellaneous, R - Receive, T - Transmit, E - EPC features, D - Discrete signals, X - eXit demo, followed by "Select function...". At the bottom, it says "Copyright (c) Tech S.A.T GmbH 1996-2000".

```
xPCD U3.33 Demo for ARINC 429 boards
Operating with board at base address: 0x0380

B - select Board
C - Configure
I - trigger
M - Miscellaneous
R - Receive
T - Transmit
E - EPC features
D - Discrete signals
X - eXit demo
Select function...

Copyright (c) Tech S.A.T GmbH 1996-2000
```

C – Configure



The screenshot shows a Windows-style window titled "EPCD9X". The menu bar contains a font size dropdown set to "8 x 12" and several icons. The main text area has a black background with white text. At the top left, it says "xPCD U3.33 Demo for ARINC 429 boards" and "Operating with board at base address: 0x0380". At the top right, it says "Configure-Menu". In the center, there is a list of functions: G - Get transmitter set, R - configure Receiver, T - configure Transmitter, W - set transmitter Wave form, I - configure discrete Input, X - return to Main menu, followed by "Select function...". At the bottom, it says "Copyright (c) Tech S.A.T GmbH 1996-2000".

```
xPCD U3.33 Demo for ARINC 429 boards
Operating with board at base address: 0x0380

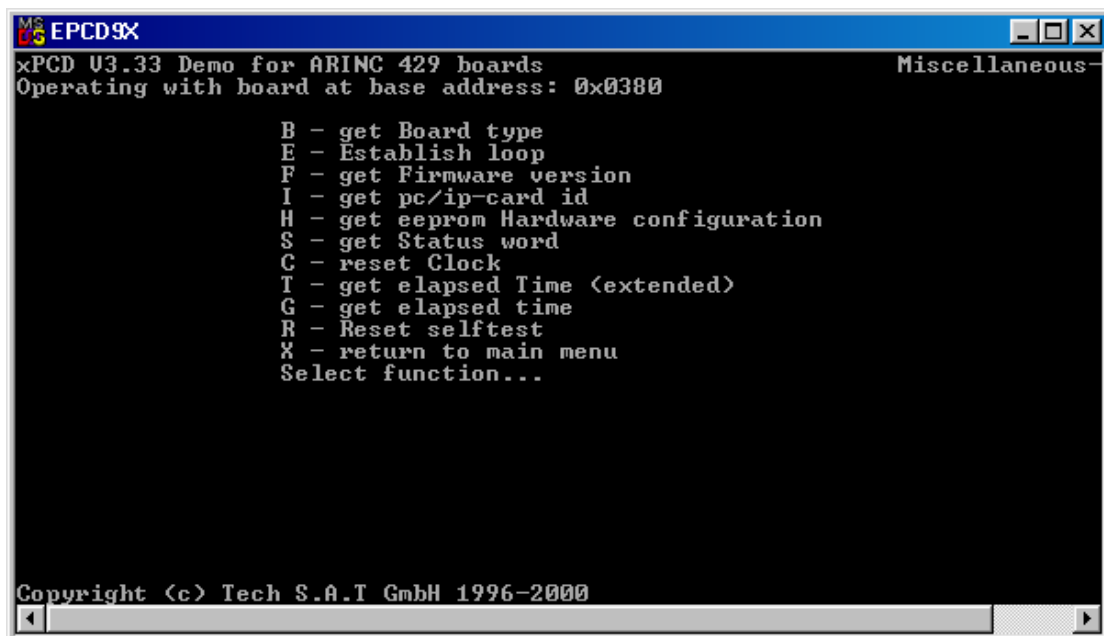
G - Get transmitter set
R - configure Receiver
T - configure Transmitter
W - set transmitter Wave form
I - configure discrete Input
X - return to Main menu
Select function...

Copyright (c) Tech S.A.T GmbH 1996-2000
```

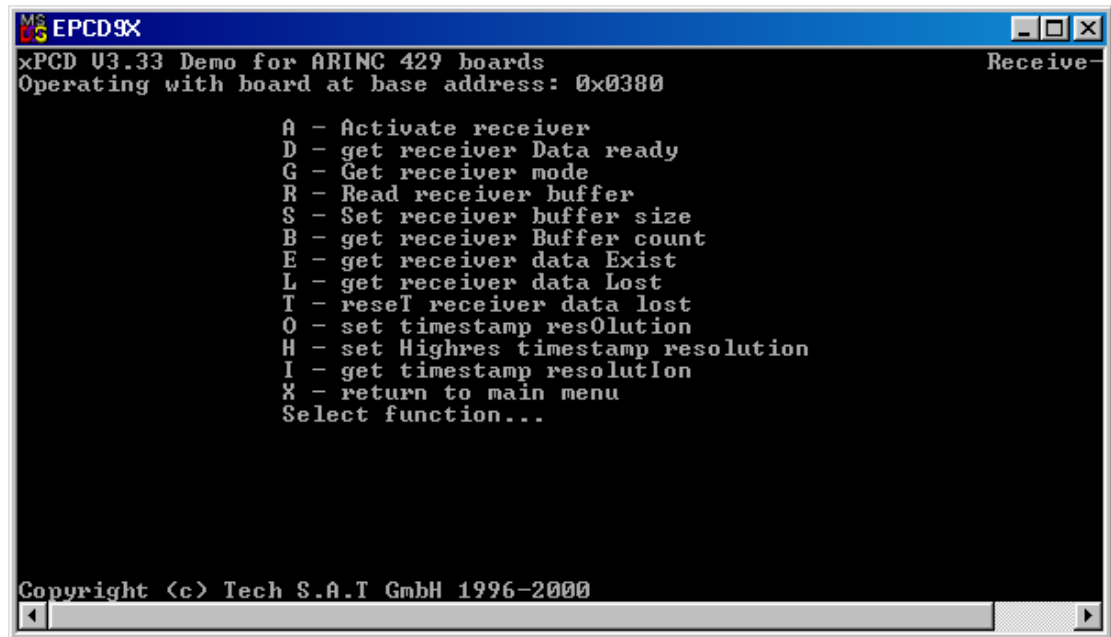
T – Trigger



M – Miscellaneous



R – Receive



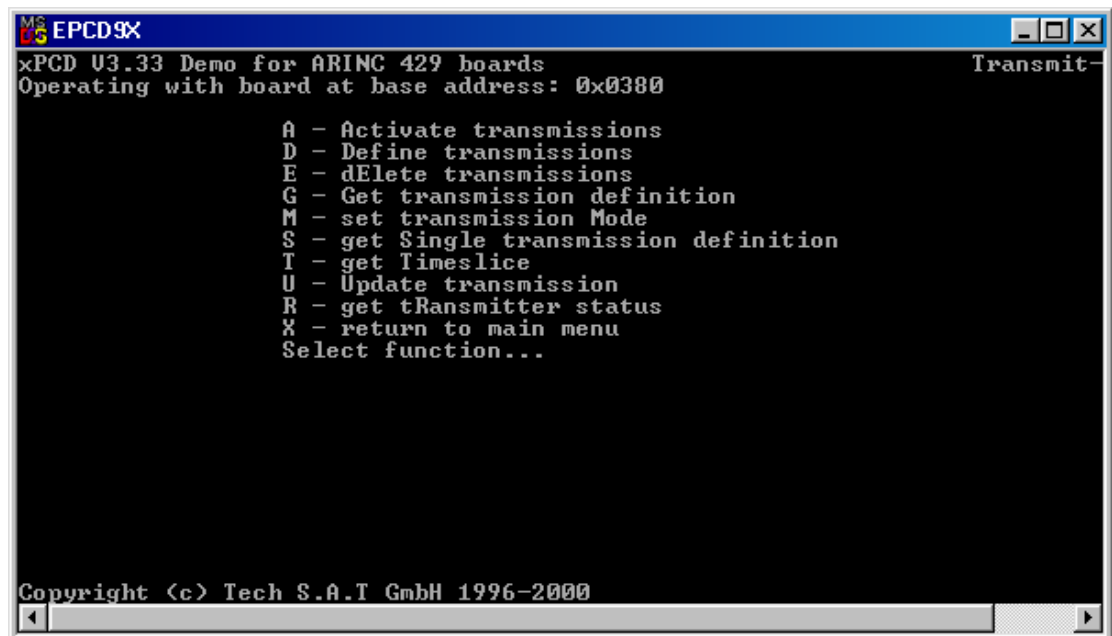
The screenshot shows a Windows-style window titled "EPCD9X". The main text area displays the following menu options for the 'Receive' function:

```
xPCD V3.33 Demo for ARINC 429 boards
Operating with board at base address: 0x0380

A - Activate receiver
D - get receiver Data ready
G - Get receiver mode
R - Read receiver buffer
S - Set receiver buffer size
B - get receiver Buffer count
E - get receiver data Exist
L - get receiver data Lost
T - reset receiver data lost
O - set timestamp resolution
H - set Highres timestamp resolution
I - get timestamp resolution
X - return to main menu
Select function...
```

At the bottom of the window, the copyright notice reads: "Copyright (c) Tech S.A.T GmbH 1996-2000". The window title bar includes standard minimize, maximize, and close buttons.

T – Transmit



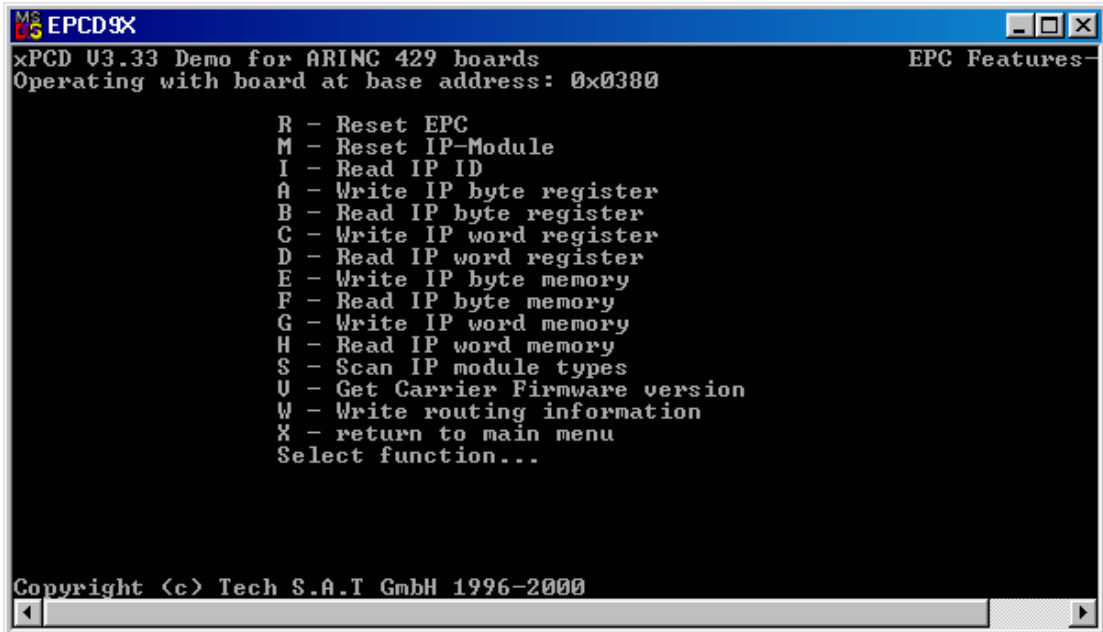
The screenshot shows the same "EPCD9X" window, but with the 'Transmit' menu options displayed:

```
xPCD V3.33 Demo for ARINC 429 boards
Operating with board at base address: 0x0380

A - Activate transmissions
D - Define transmissions
E - delete transmissions
G - Get transmission definition
M - set transmission Mode
S - get Single transmission definition
T - get Timeslice
U - Update transmission
R - get tRansmitter status
X - return to main menu
Select function...
```

The copyright notice at the bottom remains the same: "Copyright (c) Tech S.A.T GmbH 1996-2000". The window title bar and standard buttons are also visible.

E – EPC features



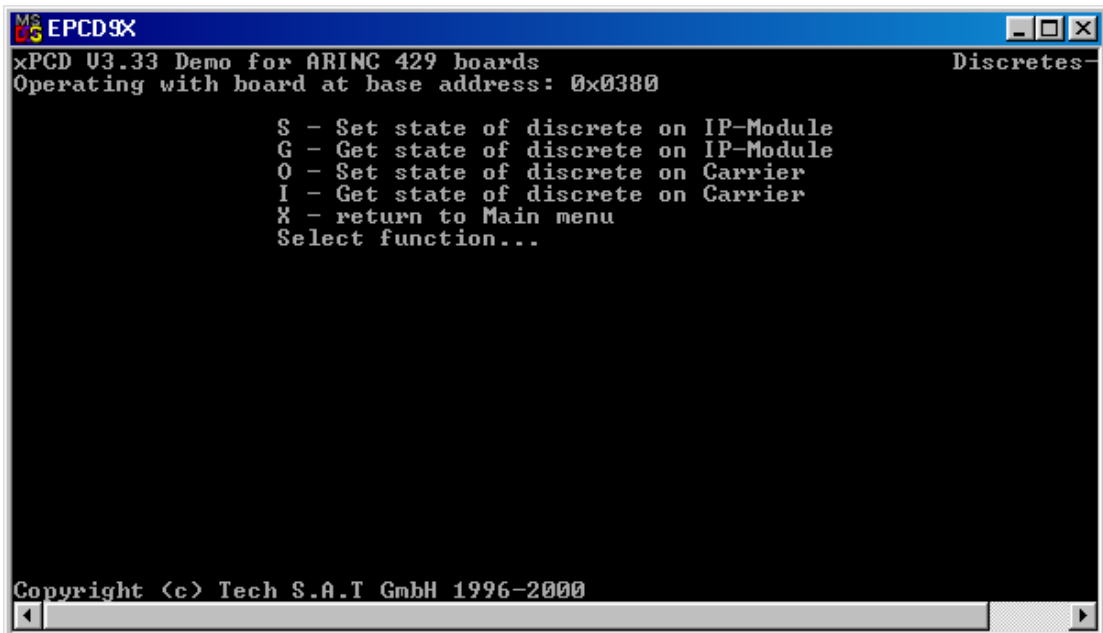
The screenshot shows a Windows-style window titled "EPCD9X" with a blue header bar. The main content area has a black background with white text. At the top, it says "xPCD V3.33 Demo for ARINC 429 boards" and "Operating with board at base address: 0x0380". On the right side, the text "EPC Features-" is visible. The central part of the window contains a list of functions: R - Reset EPC, M - Reset IP-Module, I - Read IP ID, A - Write IP byte register, B - Read IP byte register, C - Write IP word register, D - Read IP word register, E - Write IP byte memory, F - Read IP byte memory, G - Write IP word memory, H - Read IP word memory, S - Scan IP module types, U - Get Carrier Firmware version, W - Write routing information, and X - return to main menu. Below the list is the prompt "Select function...". At the bottom, a copyright notice reads "Copyright (c) Tech S.A.T GmbH 1996-2000". A scrollbar is visible at the bottom of the text area.

```
MS-DOS EPCD9X
xPCD V3.33 Demo for ARINC 429 boards
Operating with board at base address: 0x0380

R - Reset EPC
M - Reset IP-Module
I - Read IP ID
A - Write IP byte register
B - Read IP byte register
C - Write IP word register
D - Read IP word register
E - Write IP byte memory
F - Read IP byte memory
G - Write IP word memory
H - Read IP word memory
S - Scan IP module types
U - Get Carrier Firmware version
W - Write routing information
X - return to main menu
Select function...

Copyright (c) Tech S.A.T GmbH 1996-2000
```

D – Discrete Signals



The screenshot shows a Windows-style window titled "EPCD9X" with a blue header bar. The main content area has a black background with white text. At the top, it says "xPCD V3.33 Demo for ARINC 429 boards" and "Operating with board at base address: 0x0380". On the right side, the text "Discretes-" is visible. The central part of the window contains a list of functions: S - Set state of discrete on IP-Module, G - Get state of discrete on IP-Module, O - Set state of discrete on Carrier, I - Get state of discrete on Carrier, and X - return to Main menu. Below the list is the prompt "Select function...". At the bottom, a copyright notice reads "Copyright (c) Tech S.A.T GmbH 1996-2000". A scrollbar is visible at the bottom of the text area.

```
MS-DOS EPCD9X
xPCD V3.33 Demo for ARINC 429 boards
Operating with board at base address: 0x0380

S - Set state of discrete on IP-Module
G - Get state of discrete on IP-Module
O - Set state of discrete on Carrier
I - Get state of discrete on Carrier
X - return to Main menu
Select function...

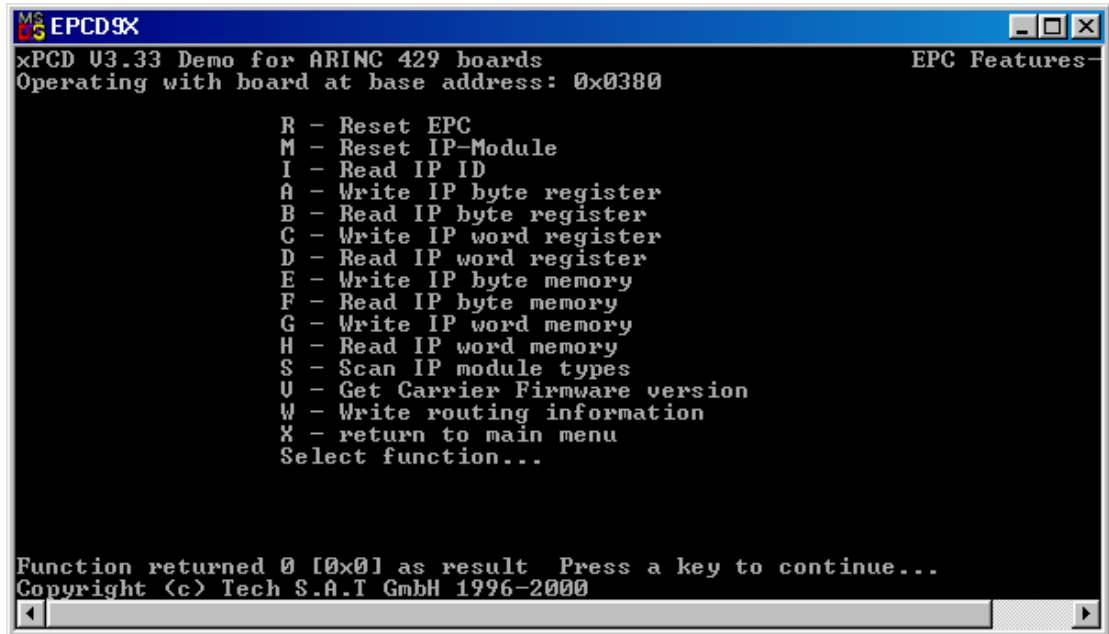
Copyright (c) Tech S.A.T GmbH 1996-2000
```

example: loop back Tx-Data

Step 1: Reset A429EPC

E: EPC features

R: Reset EPC



The screenshot shows a DOS-style window titled "EPCD9X". The text inside reads: "xPCD U3.33 Demo for ARINC 429 boards" and "Operating with board at base address: 0x0380". On the right side, it says "EPC Features-". A list of functions is displayed in the center: R - Reset EPC, M - Reset IP-Module, I - Read IP ID, A - Write IP byte register, B - Read IP byte register, C - Write IP word register, D - Read IP word register, E - Write IP byte memory, F - Read IP byte memory, G - Write IP word memory, H - Read IP word memory, S - Scan IP module types, U - Get Carrier Firmware version, W - Write routing information, and X - return to main menu. Below the list is the prompt "Select function...". At the bottom, it says "Function returned 0 [0x0] as result Press a key to continue..." and "Copyright (c) Tech S.A.T GmbH 1996-2000".

```
MS-DOS EPCD9X
xPCD U3.33 Demo for ARINC 429 boards
Operating with board at base address: 0x0380
EPC Features-

R - Reset EPC
M - Reset IP-Module
I - Read IP ID
A - Write IP byte register
B - Read IP byte register
C - Write IP word register
D - Read IP word register
E - Write IP byte memory
F - Read IP byte memory
G - Write IP word memory
H - Read IP word memory
S - Scan IP module types
U - Get Carrier Firmware version
W - Write routing information
X - return to main menu
Select function...

Function returned 0 [0x0] as result Press a key to continue...
Copyright (c) Tech S.A.T GmbH 1996-2000
```

Step 2: Configuration of Transmitter and Receiver

C: Configuration

T: Configure transmitter

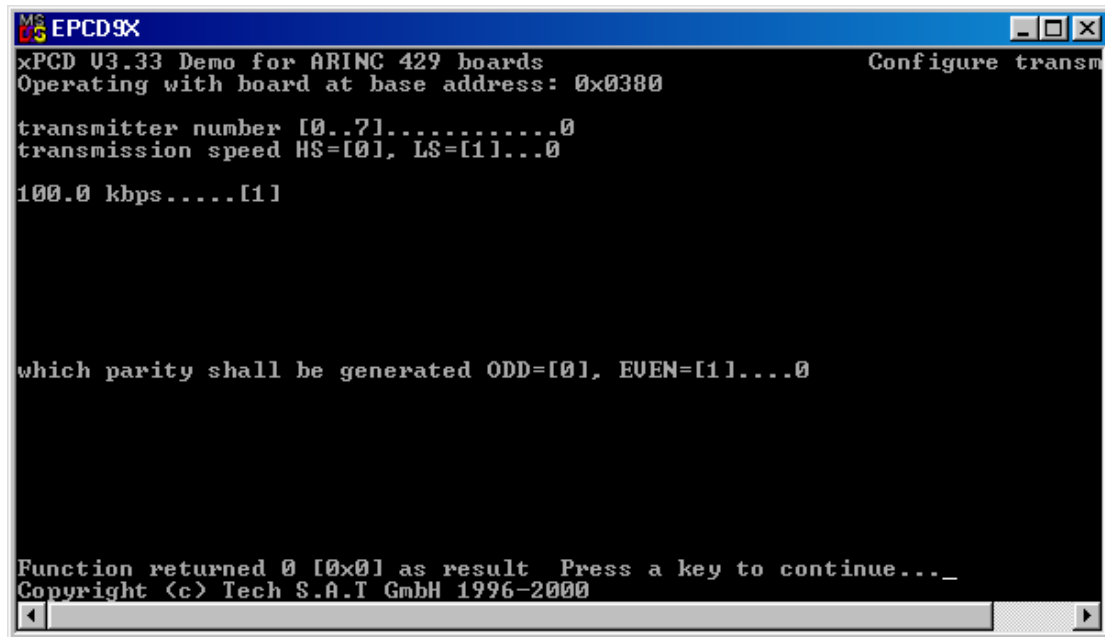
set Tx-No. [0..7]

where

- 0 is the first Tx of Module 1,
- 1 is the second Tx of Module 1,
- 2 is the first Tx of Module 2,
- 3 is the second Tx of Module 2,
- 4 is the first Tx of Module 3,
- 5 is the second Tx of Module 3,
- 6 is the first Tx of Module 4,
- 7 is the second Tx of Module 4.

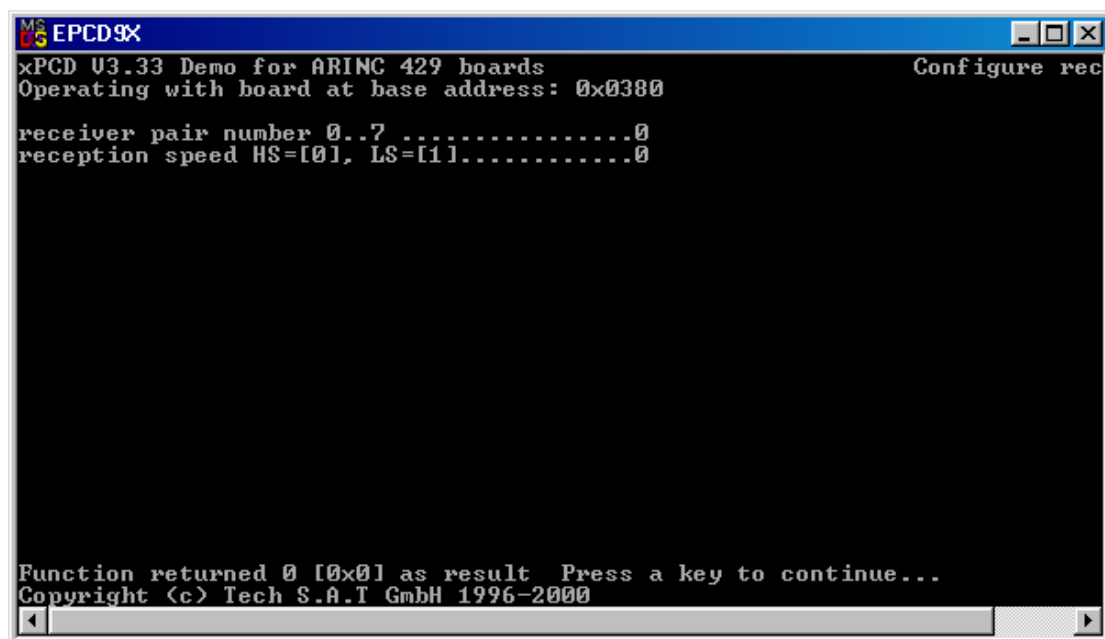
set Speed [low..high]

set Parity [odd..even]



R: Configure receiver

Rx-pair-No. [0..7]
 where 0 is Rx 0 and Rx 1 of Module 1
 1 is Rx 2 and Rx 3 of Module 1
 2 is Rx 4 and Rx 5 of Module 2
 3 is Rx 6 and Rx 7 of Module 2
 4 is Rx 8 and Rx 9 of Module 3
 5 is Rx10 and Rx11 of Module 3
 6 is Rx12 and Rx13 of Module 4
 7 is Rx14 and Rx15 of Module 4
 Speed [low..high]



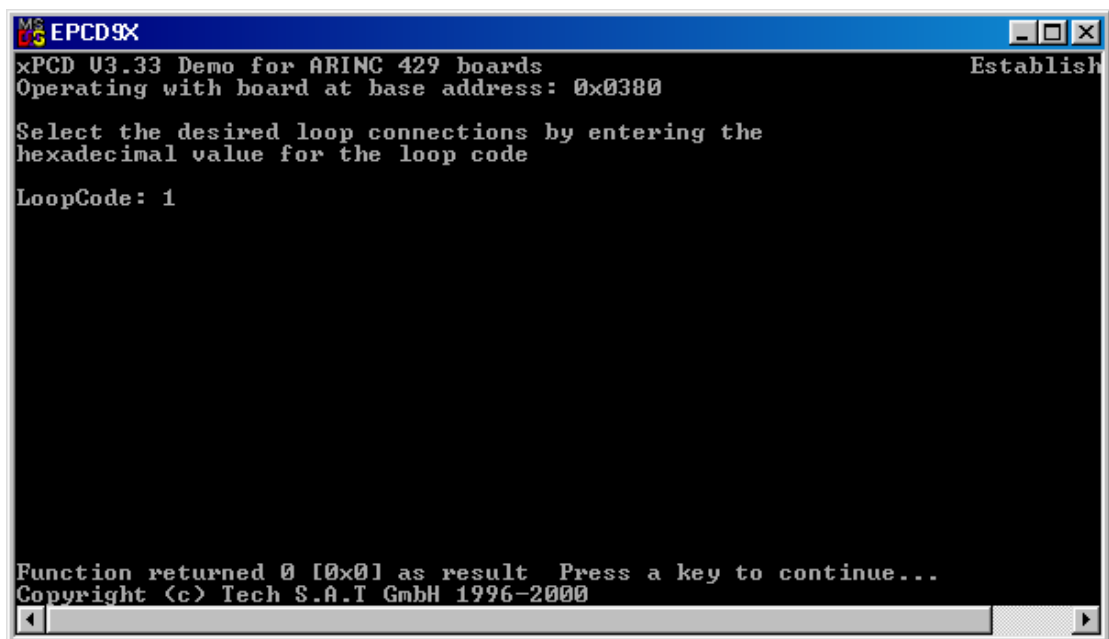
X: Return to main menu

Step 3: establish Tx-Rx Loop

M: Miscellaneous

E: Establish loop

Loop Code (in Hex):	Module 1:	Loops
	1	Tx0 to Rx0
	2	Tx0 to Rx2
	4	Tx1 to Rx1
	8	Tx1 to Rx3
	Module 2:	Loops
	10	Tx2 to Rx4
	20	Tx2 to Rx6
	40	Tx3 to Rx5
	80	Tx3 to Rx7
	Module 3:	Loops
	100	Tx4 to Rx8
	200	Tx4 to Rx10
	400	Tx5 to Rx9
	800	Tx5 to Rx11
	Module 4:	Loops
	1000	Tx6 to Rx12
	2000	Tx6 to Rx14
	4000	Tx7 to Rx13
	8000	Tx7 to Rx15



Step 4: Define and activate cyclic transmission

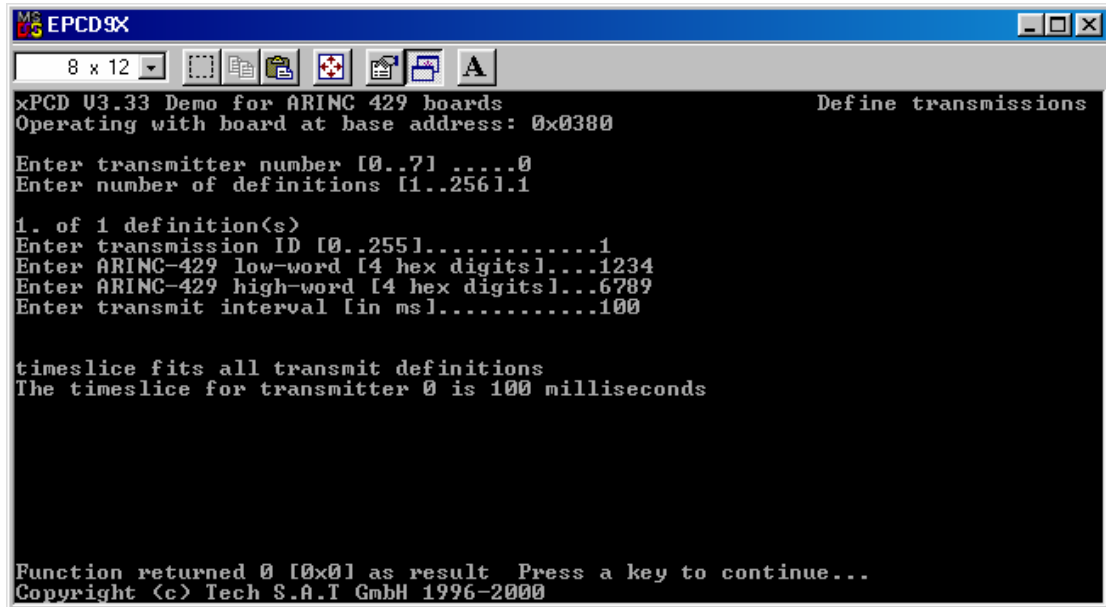
T: Transmit

D: Define Transmission

Tx-Number [0..7]

Number of Transmission-IDs (= Number of word definitions)

Define for each Transmission-ID: low-word, high-word
and the Update-time.



```
MS-DOS EPCD9X
8 x 12
xPCD U3.33 Demo for ARINC 429 boards          Define transmissions
Operating with board at base address: 0x0380

Enter transmitter number [0..7] .....0
Enter number of definitions [1..256]..1

1. of 1 definition(s)
Enter transmission ID [0..255].....1
Enter ARINC-429 low-word [4 hex digits]....1234
Enter ARINC-429 high-word [4 hex digits]...6789
Enter transmit interval [in ms].....100

timeslice fits all transmit definitions
The timeslice for transmitter 0 is 100 milliseconds

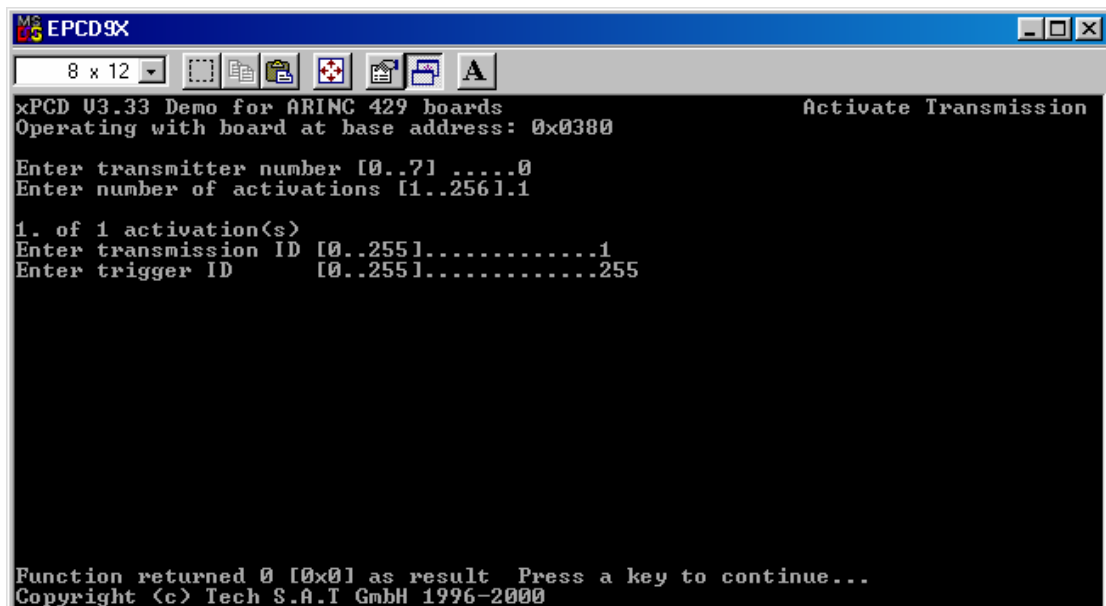
Function returned 0 [0x0] as result  Press a key to continue...
Copyright (c) Tech S.A.T GmbH 1996-2000
```

A: Activate transmission

Number of activations (=Number of words, which should be transmitted)

Set for each activation: Transmission ID

Trigger ID (should be set to 255 for 'transmit always')



```
MS-DOS EPCD9X
8 x 12
xPCD U3.33 Demo for ARINC 429 boards          Activate Transmission
Operating with board at base address: 0x0380

Enter transmitter number [0..7] .....0
Enter number of activations [1..256]..1

1. of 1 activation(s)
Enter transmission ID [0..255].....1
Enter trigger ID      [0..255].....255

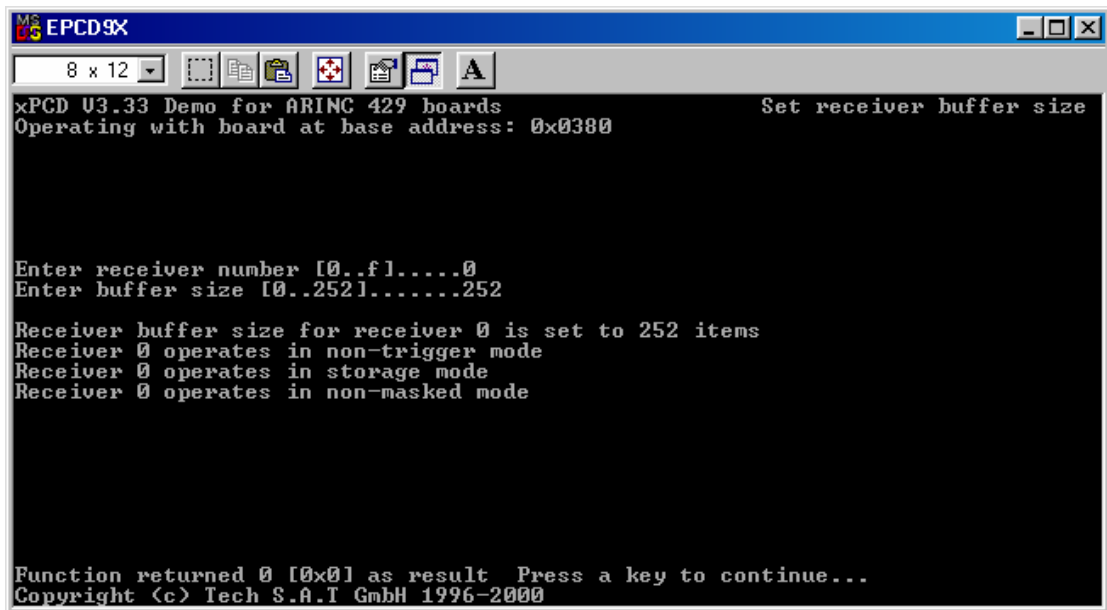
Function returned 0 [0x0] as result  Press a key to continue...
Copyright (c) Tech S.A.T GmbH 1996-2000
```

X: Return to main menu

Step 5: Set receiver buffer size and activate receiver

R: Receiver

S: Set receiver buffer size Rxi [0-f]
Buffersize [0..252]



The screenshot shows a Windows-style window titled "EPCD9X". The menu bar includes "File", "Edit", "Format", "Tools", "Window", and "Help". The status bar at the bottom indicates "8 x 12". The main text area displays the following content:

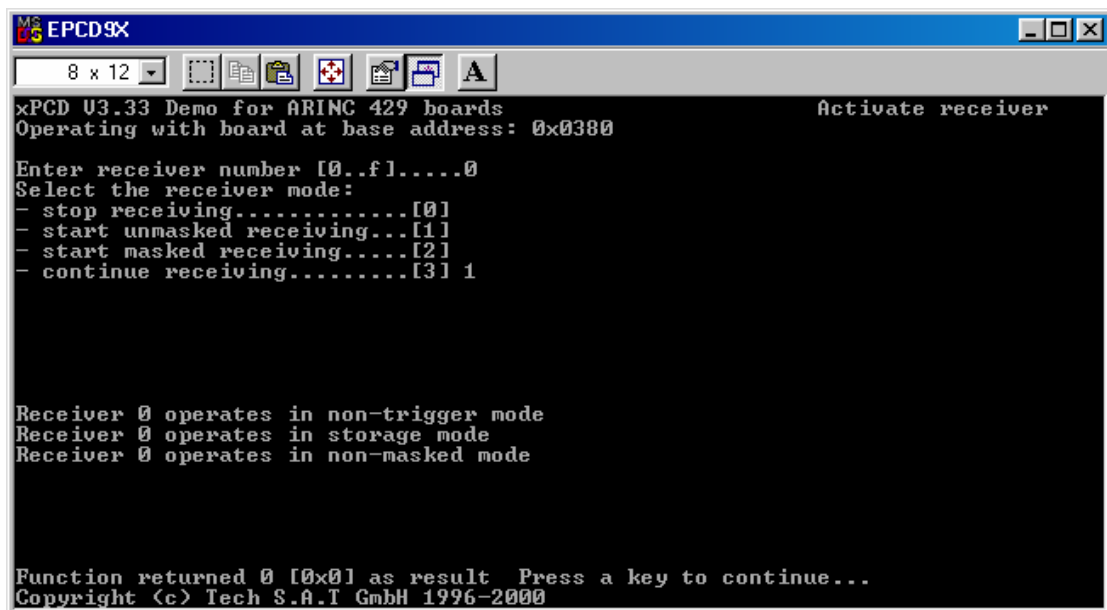
```
xPCD U3.33 Demo for ARINC 429 boards                               Set receiver buffer size
Operating with board at base address: 0x0380

Enter receiver number [0..f].....0
Enter buffer size [0..252].....252

Receiver buffer size for receiver 0 is set to 252 items
Receiver 0 operates in non-trigger mode
Receiver 0 operates in storage mode
Receiver 0 operates in non-masked mode

Function returned 0 [0x0] as result Press a key to continue...
Copyright (c) Tech S.A.T GmbH 1996-2000
```

A: Activate Receiver: Start unmasked receiving [1]



The screenshot shows the same "EPCD9X" window. The main text area displays the following content:

```
xPCD U3.33 Demo for ARINC 429 boards                               Activate receiver
Operating with board at base address: 0x0380

Enter receiver number [0..f].....0
Select the receiver mode:
- stop receiving.....[0]
- start unmasked receiving...[1]
- start masked receiving.....[2]
- continue receiving.....[3] 1

Receiver 0 operates in non-trigger mode
Receiver 0 operates in storage mode
Receiver 0 operates in non-masked mode

Function returned 0 [0x0] as result Press a key to continue...
Copyright (c) Tech S.A.T GmbH 1996-2000
```

Step 6: Read Receiver Buffer

R: Read Receiver buffer
Enter receiver number [0..F]

```
MS EPCD9X
8 x 12
xPCD U3.33 Demo for ARINC 429 boards
Operating with board at base address: 0x0380
Read receiver buffer

Enter receiver number [0..f].....0

1. entry of 103 on Rx0 data: 0x56781234 received at 600249 [ms]
2. entry of 103 on Rx0 data: 0x56781234 received at 600349 [ms]
3. entry of 103 on Rx0 data: 0x56781234 received at 600449 [ms]
4. entry of 103 on Rx0 data: 0x56781234 received at 600549 [ms]
5. entry of 103 on Rx0 data: 0x56781234 received at 600649 [ms]
6. entry of 103 on Rx0 data: 0x56781234 received at 600749 [ms]
7. entry of 103 on Rx0 data: 0x56781234 received at 600849 [ms]
8. entry of 103 on Rx0 data: 0x56781234 received at 600949 [ms]
9. entry of 103 on Rx0 data: 0x56781234 received at 601049 [ms]
10. entry of 103 on Rx0 data: 0x56781234 received at 601149 [ms]
11. entry of 103 on Rx0 data: 0x56781234 received at 601249 [ms]
12. entry of 103 on Rx0 data: 0x56781234 received at 601349 [ms]
13. entry of 103 on Rx0 data: 0x56781234 received at 601449 [ms]
14. entry of 103 on Rx0 data: 0x56781234 received at 601549 [ms]
15. entry of 103 on Rx0 data: 0x56781234 received at 601649 [ms]
16. entry of 103 on Rx0 data: 0x56781234 received at 601749 [ms]

Press a key to continue...
Copyright (c) Tech S.A.T GmbH 1996-2000
```