

# A429-USB Device

## User Manual



## Imprint

<b>Document Title</b>	A429-USB Device User Manual
<b>Document ID</b>	702096_A429-USB-USM_1000
<b>Issue</b>	1000
<b>Classification</b>	Public
<b>Project</b>	
<b>Released by</b>	TechSAT GmbH, Poing
<b>Release Date</b>	January 16, 2008

## Copyright Notice

This document is the property of TechSAT GmbH.  
Its content may not be copied or distributed to third parties  
without the written approval of TechSAT GmbH.

The document has been checked carefully and is thought  
to be entirely reliable. However, no responsibility is assumed  
in case of inaccuracies.

TechSAT GmbH reserves the right to make changes  
without further notice.

© Copyright 2008 TechSAT GmbH • All rights reserved.

## Trademark Notice

Product and other trademark names referred to within this  
manual are the property of their respective trademark holders.



# Table of Contents

<b>WELCOME TO THE A429-USB DEVICE.....</b>	<b>5</b>
ABOUT THIS MANUAL .....	5
<i>Scope</i> .....	5
<i>Related Documents</i> .....	5
<i>List of Incorporated Changes</i> .....	5
LIST OF DELIVERY .....	5
OVERVIEW .....	6
<b>INSTALLATION AND STARTUP .....</b>	<b>7</b>
<i>Non-Human Interface Device (HID)</i> .....	7
A429-USB INTERFACE CONNECTOR.....	8
DISCRETE I/O-SIGNALS .....	9
<b>SOFTWARE/FIRMWARE DESCRIPTION.....</b>	<b>11</b>
FIRMWARE RELEASE HISTORY .....	11
ARINC 429 DATA TRANSMISSION.....	11
<i>Transmission Modes</i> .....	11
<i>Update Rate</i> .....	12
<i>Transmit Data Organization</i> .....	13
<i>Conditional Transmission, Trigger State Table</i> .....	14
ARINC 429 DATA RECEPTION .....	14
<i>Storage Mode</i> .....	14
<i>Masked Mode</i> .....	15
<i>Trigger Mode</i> .....	15
<b>A429-USB COMMAND INTERFACE.....</b>	<b>17</b>
CONFIGURATION FUNCTIONS .....	18
<i>configure transmitter</i> .....	18
<i>set parity mode</i> .....	19
<i>get transmitter set</i> .....	19
<i>configure receiver</i> .....	20
TRANSMITTER FUNCTIONS .....	20
<i>set transmitter mode</i> .....	20
<i>define transmit data</i> .....	20
<i>activate transmit data</i> .....	21
<i>delete transmit data</i> .....	22
<i>update transmit data</i> .....	22
<i>get all transmit data definitions</i> .....	22
<i>get single transmit data definitions</i> .....	22
<i>get transmitter timeslice</i> .....	22
RECEIVER FUNCTIONS.....	23
<i>get receiver mode</i> .....	23
<i>set receiver timestamp resolution</i> .....	23
<i>get receiver timestamp resolution</i> .....	23
<i>set timestamp high resolution mode</i> .....	23
<i>set receiver buffer size</i> .....	24
<i>get receiver buffer count</i> .....	24
<i>activate receiver</i> .....	24
<i>read receiver buffer</i> .....	25
<i>get receiver data ready</i> .....	25
<i>get receiver data exist</i> .....	25
<i>get receiver data lost</i> .....	25
<i>reset data lost flag</i> .....	26
EVENT SYSTEM & TRIGGER FUNCTIONS .....	26

## Contents

<i>define trigger event</i> .....	26
<i>set trigger state</i> .....	26
<i>get trigger state</i> .....	27
DISCRETE I/O FUNCTIONS .....	27
<i>get state of input lines</i> .....	27
<i>set state of output line OUTPCC</i> .....	27
MISCELLANEOUS FUNCTIONS.....	27
<i>perform reset/selftest</i> .....	27
<i>get status word</i> .....	28
<i>establish loop connections</i> .....	28
<i>reset card system clock</i> .....	28
<i>get board/card timer value</i> .....	29
<i>get A429 board/card type</i> .....	29
<i>get firmware version</i> .....	29

# Welcome to the A429-USB Device

---

## About this Manual

### Scope

This document contains information about the installation and the operation of the **A429-USB** ARINC 429 USB device. It assumes that the reader is familiar with both the ARINC 429 bus concept and IBM-type personal computers. A basic idea regarding the USB (Universal Serial Bus) standard is assumed.

### Related Documents

#	Document	Company
1.	MARK 33 Digital Information Transfer System (DITS) ARINC Specification 429-x	AERONAUTICAL RADIO, Inc. 2551 Riva Road Annapolis Maryland 21401
2.	C-Interface Description A429xPC ARINC 429 intelligent Interface Boards Doc#: IFC-XPC-G-xyz, Rev.x Vx	TechSAT GmbH Gruber Strasse 46 b 85586 Poing, Germany
3.	Installation Guidelines (DOS/Windows9x/ Windows NT/ Linux) Driver and Interface Libraries A429xPC ARINC 429 intelligent Interface Boards Doc#: CIG-XPC-G-xyz, Rev.x Vx	TechSAT GmbH Gruber Strasse 46 b 85586 Poing, Germany

### List of Incorporated Changes

Revision	Description	Date
A	A429USB V1.0 – initial release	20-Oct-06
B	Transition from HID Device to non-HID Device (performance improvement) – Firmware V2.1; Driver 1.0.1.10; IF-SW: V3.64	18-Dec-06
C	New Windows 2000/XP Driver: V1.2.0.0. (27.1.07)	31-Jan-07
1000	QM and CI layout adjustment	16-Jan-08

## List of Delivery

- **A429-USB** USB device incl. USB 2.0 to Mini-USB cable
- CD with the ZIP archive file **A429USB.zip** containing the following:
  - × DLL **usb32nh.dll** (**usb32nh.lib**) for Windows 2000 and Windows XP
  - × Driver installation directory with files **A429USBdrv.inf**, **A429USBdrv.sys** and Installer-DLL **wdfCoInstaller01001.dll**
  - × User Documentation (C-Interface Description and Installation Guidelines as PDF files) & Release Notes
  - × Header files **a429xpcd.h** and **a429usb.h**
  - × Test/sample source **xpcd.c**, and executables **usb32bit.exe** and **usb32d.exe**

- A429xPC ARINC 429 intelligent Interface Boards C-Interface description (DOS/Windows9x/WindowsNT) A429USB | A429PCC | A429EPC | A429PCI | A429PC104 | A429VME | A429VXI | A429IPM [Doc#: CID-XPC-G-xyz].
- Installation Guidelines A429xPC (DOS/Windows9x/WindowsNT) ARINC 429 intelligent Interface Boards C-Interface description A429USB | A429EPC | A429PCI | A429PC104 | A429VME | A429VXI [Doc#: CIG-XPC-G-xyz].
- A429-USB Device User Manual (this manual)

## Overview

The **A429-USB** ARINC 429 USB device is a microprocessor controlled subsystem for any modern personal computer (PC) that is equipped with a USB V1.1 or V2.0 interface. The A429-USB's power is supplied by the USB bus. The A429-USB must be operated with an active USB hub. It is not recommended, or may not even work, to operate the A429-USB with a passive USB hub. If more channels are required than provided by a single device, multiple cards can be operated simultaneously.

---

**WARNING** The DC power connector next to the Mini-USB connector is only for manufacturer use. The device may be damaged if external power is connected.

---

Modern notebook PCs and desktop PCs are usually equipped with several USB interfaces. A429-USB is capable to serve up to four ARINC 429 input channels and two ARINC 429 output channels.

---

<b>Note</b>	If more channels are required than provided by a single device, multiple cards can be operated simultaneously.
-------------	--

---

Additionally, four discrete outputs are provided. Each output can be individually switched to one of two signal pins on the 25-pin D-sub connector.

---

<b>Note</b>	The hardware allows for various other combinations of switching options. Contact the manufacturer if you have specific requirements.
-------------	--

---

The A429-USB also provides six 28V/Open sensing and six GND/Open sensing discrete inputs.

The A429-USB is meant to be a testing device rather than a mere ARINC 429 device. For this purpose, ARINC 429 operating characteristics are programmable to simulate a marginal environment and to check the behavior of ARINC devices under test.

Technical characteristics of the A429-USB:

- on-card 36 MHz micro-controller SAB C165UTAH
- 2 Mbyte SRAM, 8 Mbyte Flash-EEPROM
- 2 ARINC 429 transceiver
- chips (each 1Tx/2Rx) with over-voltage/lightning protection
- opto-electronic Tx-Rx wrap-around relays
- programmable logic devices

# Installation and Startup

---

The A429-USB System Software CD contains the **A429USB.zip**

Upon unzipping the file, the following directory structure is created:

File	Description
\COMMON\	
A429USB.H	Header file for A429-USB USB device. Refer to C-Interface Description for proper <b>#defines</b> .
A429XPC.H	General header file for all TechSAT ARINC 429 interface boards. This file is always included by the board-specific header file.
\DRV	
A429USBDrv.inf A429USBDrv.sys WdfCoInstaller01001.dll	Driver files for Windows 2000 and XP.
\DLL\	
usb32nh.dll	Microsoft C V6.0 DLL.
usb32nh.lib	Corresponding lib file - to be linked to application.
usb32d.exe	Test and demonstration program for A429-USB. This <b>.exe</b> file is generated from the sample source <b>xpcd.c</b> , and contains a basic menu-based user interface which allows executing all single functions of the A429-USB individually.
usb32bit.exe	Test program for A429-USB, which allows executing complete BITEs for the A429-USB.
\DOC\	
CIFDESC_abc.PDF	Current version of the C-Interface Description.
INSTALL_abc.PDF	Current version of the Installation Guidelines
702096_USB-USM_<issue>.pdf	This A429-USB User Manual.
\SAMPLES\	
XPCD.C	Sample source code. From this source all sample executables for all TechSAT ARINC 429 interface boards are generated.

## Non-Human Interface Device (HID)

The **A429-USB** device was initially developed as a Human Interface Device (HID) completely handled by the USB and HID-related drivers of the Windows operating system without the need of additional device drivers.

USB keyboards or USB mice are typical HID devices.

HID devices cannot use the entire bandwidth that USB offers. Therefore, TechSAT decided to switch from HID to non-HID. A dedicated driver for both Windows 2000 and Windows XP is required. The installation of the driver follows the standard Windows driver installation procedure(s).

<b>Note</b>	For non-HID operation, firmware version V2.1 or higher is required. The current driver version is V2.1.
-------------	--

## A429-USB Interface Connector

The A429-USB interface provides four receiver channels (Rx0-Rx3) and two transmitter channels (Tx0 and Tx1). These are brought out on a female D-type connector (SUBD25S, J1).

Additionally to the ARINC 429 signal lines, the discrete I/O lines are brought out on J1.

Pins 24 and 25 can be used for external shielding.

<b>Note</b>	The J1 connector layout of the A429-USB is compatible (in terms of ARINC 429 Tx and Rx lines) to the one of TechSAT's legacy A429PCC ARINC 429 PCMCIA-card [not manufactured any longer].
-------------	---

### Connection Diagram (J1)

Pin Numbers	Signal Description
• [1] —————	Receiver Rx3 B-line
• [14] —————	Open/GND sensing Input #6
• [2] —————	Receiver Rx3 A-line
• [15] —————	Open/GND sensing Input #5
• [3] —————	Receiver Rx2 B-line
• [16] —————	Open/GND sensing Input #4
• [4] —————	Receiver Rx2 A-line
• [17] —————	Open/GND sensing Input #3
• [5] —————	Receiver Rx1 B-line
• [18] —————	Open/GND sensing Input #2
• [6] —————	Receiver Rx1 A-line
• [19] —————	Open/GND sensing Input #1
• [7] —————	Receiver Rx0 B-line
• [20] —————	GND
• [8] —————	Receiver Rx0 A-line
• [21] —————	28V/Open sensing Input #6; A615-3: FD 4 (with Pin 9)
• [9] —————	28V/Open sensing Input #5; A615-3: AC Ground
• [22] —————	28V/Open sensing Input #4; A615-3: FD 3 (with Pin 9)
• [10] —————	Transmitter Tx0 B-line
• [23] —————	28V/Open sensing Input #3; A615-3: FD 2 (with Pin 24)
• [11] —————	Transmitter Tx0 A-line
• [24] —————	28V/Open sensing Input #2; A615-3: AC Ground
• [12] —————	Transmitter Tx1 B-line
• [25] —————	28V/Open sensing Input #1; A615-3: FD 1 (with Pin 24)
• [13] —————	Transmitter Tx1 A-line



## Discrete I/O-Signals

The A429-USB is provided with the following sensing inputs:

- Six 28V/Ground
- Six Open/Ground

Six of these input pins have special functions. Four pins can be used as discrete outputs. They can be individually switched via software-controlled relays to connect them to one of two other pins (#9 or #24; see “Connection Diagram”).

The special background for this functionality is the ARINC 615-3/-4 data loading specification. In this specification so called *Function Discretes* are defined. A load session can use the *Function Discretes* in order to control the LRU to load. The specification defines the *Function Discretes* to be active if connected to AC Ground ( $< 10 \text{ Ohm}$ ) and to be inactive in Open state ( $> 100 \text{ kOhm}$ ).



# Software/Firmware Description

---

## Firmware Release History

Version	Problems discovered in	Enhancements to previous version
1.1 (Oct 2006)	initial version (HID)	
2.1 (Dec 2006)	non-HID firmware	

## ARINC 429 Data Transmission

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)

**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.

### 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. These flags are called **trigger flags**, and they 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 trigger ID, which is the index of the trigger state table. More than one ARINC 429 word can refer to the same trigger ID. In one-shot mode, 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** 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 D
-------------	--------	-------------	--------	-------------	--------	-------------	--------	-------------	--------	-------------	--------

## 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 1 msec, i.e. minimum transmit interval is 1 msec. 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

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

---

**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 at all.

---

A special situation arises, when one-shot data definitions (characterized by an update-rate of zero 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 transmission ID. There are 256 transmission IDs (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.

MSB	LSB
assigned trigger-ID	adr (n)
ARIC 429 (low word)	adr (n+2)
ARIC 429 (high word)	adr (n+4)
transmit interval [msec]	adr (n+6)
transmit interval [timeslice ticks]	adr (n+8)
timeslice ticks until transmission	adr (n+10)
flags for one-shot mode	adr (n+12)
spare	adr (n+14)

*Layout of the data definition list*

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 (1 msec) 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 (trigger ID). 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 insures 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, signaling 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:

- the fulfillment of a receiver matching criterion
- explicit programming by the host
- signal transition on discrete input INPCC

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 trigger ID. Two trigger IDs have a special meaning. They are not meant to be used for conditional transmission.

- trigger ID 0 means 'never transmit'  
This trigger table entry is always reset and cannot be modified.
- trigger ID 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

An ARINC 429 receiver may operate in three fundamental modes and combinations thereof:

- Storage mode
- Masked mode
- Trigger mode

### Storage Mode

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 A429-USB 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, but depends on the selected size of the receive buffers, which is determined by the interface function `_set_rec_buf_size`.

The maximum buffer size is 252 data items (32-bit data + 32-bit timestamp). This limitation is due to the size of the on-card Dual Port RAM, which is used for data exchange with the PC.

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.

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.

**Masked Mode**

In Masked mode, a received ARINC 429 word is processed only if it fulfills a matching criterion defined by the host. Otherwise, it is ignored. The matching criterion consists of a 32-bit mask pattern and a 32-bit 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.

**Trigger Mode**

In Trigger mode, additional comparisons are performed. An ARINC 429 word is further investigated whether it should cause an internal event. Depending on the result, transmission of selected ARINC 429 words can be started or stopped.





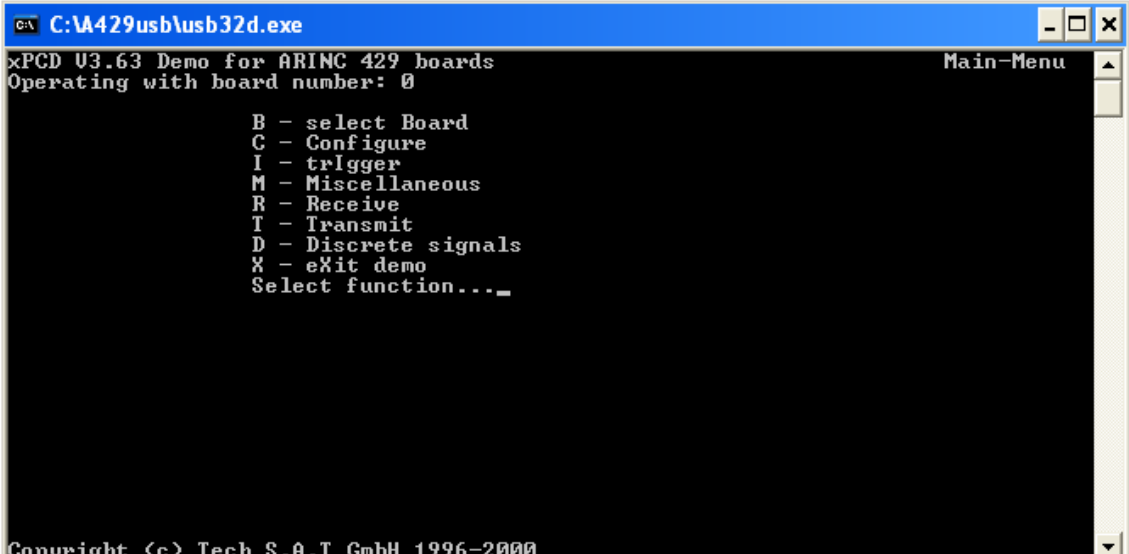
# A429-USB Command Interface

This chapter describes the subset of commands that can be executed with A429-USB. For each command there is a corresponding function in the DLL. Also, each interface function has its counterpart inside the A429-USB firmware.

The C-Interface Description document explains in detail the usage of each function and the parameters required by it.

Additionally, the sample source file **xpcd.c** can serve as a reference on how to use the functions. The **xpcd.c** file is the central source for a basic menu-driven user interface for all of TechSAT's intelligent ARINC 429 boards. Practicing with the resulting USB executable **usb32d.exe** helps you acquire knowledge on how to operate the A429-USB in a fast and convenient way.

Here are some sample screen shots.



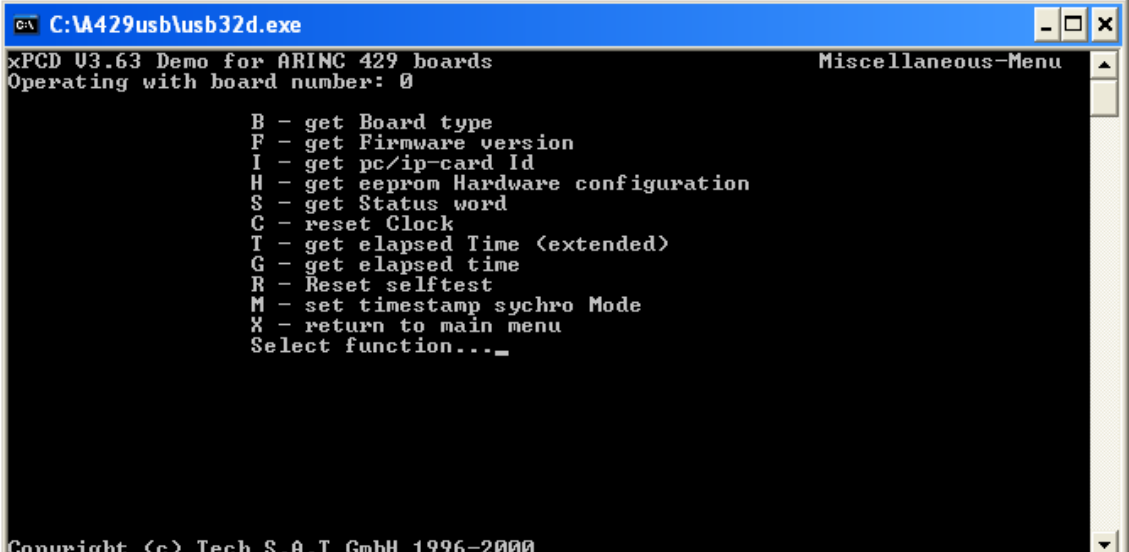
The screenshot shows a Windows command prompt window titled "C:\A429usb\usb32d.exe". The text inside the window reads: "xPCD U3.63 Demo for ARINC 429 boards", "Operating with board number: 0", and "Main-Menu". A list of functions is displayed: B - select Board, C - Configure, I - trigger, M - Miscellaneous, R - Receive, T - Transmit, D - Discrete signals, X - eXit demo, and "Select function...". The copyright notice "Copyright (c) Tech S.A.T GmbH 1996-2000" is at the bottom.

```
C:\A429usb\usb32d.exe
xPCD U3.63 Demo for ARINC 429 boards
Operating with board number: 0
Main-Menu

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

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

*XPCD-Fig.1 Main Menu*



The screenshot shows the same Windows command prompt window, but now displaying the "Miscellaneous-Menu". The list of functions includes: B - get Board type, F - get Firmware version, I - get pc/ip-card Id, H - get eeprom Hardware configuration, S - get Status word, C - reset Clock, T - get elapsed Time (extended), G - get elapsed time, R - Reset selftest, M - set timestamp sychro Mode, and X - return to main menu. The copyright notice "Copyright (c) Tech S.A.T GmbH 1996-2000" is at the bottom.

```
C:\A429usb\usb32d.exe
xPCD U3.63 Demo for ARINC 429 boards
Operating with board number: 0
Miscellaneous-Menu

B - get Board type
F - get Firmware version
I - get pc/ip-card Id
H - get eeprom Hardware configuration
S - get Status word
C - reset Clock
T - get elapsed Time (extended)
G - get elapsed time
R - Reset selftest
M - set timestamp sychro Mode
X - return to main menu
Select function...

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

*XPCD-Fig.2 Sub-Menu: Miscellaneous*

```

C:\A429usb\usb32d.exe
xPCD U3.63 Demo for ARINC 429 boards
Operating with board number: 0
Reset and Selftest

Selftest-Result:

DPRAM check.....o.k.
SRAM check.....o.k.
Transceiver A <Tx0,Rx0,Rx1>.....o.k.
Transceiver B <Tx1,Rx2,Rx3>.....o.k.
External loop test.....o.k.

1. external loop analysis T0-R0.....o.k.
2. external loop analysis T1-R1.....o.k.
3. external loop analysis T0-R2.....o.k.
4. external loop analysis T1-R3.....o.k.

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

```

XPCD-Fig.3 Reset Selftest Result

```

C:\A429usb\usb32d.exe
xPCD U3.63 Demo for ARINC 429 boards
Operating with board number: 0
Transmit-Menu

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..._

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

```

XPCD-Fig.4 Sub-Menu: Transmit

## Configuration Functions

### configure transmitter

This command sets the basic operating characteristics of a transmitter, including:

- the transmit speed : low speed (LS) or high speed (HS)
- the parity (odd or even) to be generated by the transmitter channel

As soon as the transmit speed is selected, the transmit waveform slope is automatically adjusted: 5V/μsec for HS and 1V/μsec for LS.

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, the default transmitter settings are:

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

This command is realized with the function `_config_trans()`.

## set parity mode

This command disables or enables the automatic parity generation of a transmitter channel.

Any transmission activities are stopped and any transmission definitions are lost.

The disable parity mode forces the firmware to program bit 4 (PAREN) of control register to 0. When parity generation is enabled (odd or even parity according to `_config_trans`), PAREN has to be 1.

If parity generation is disabled, bit 32 of each ARINC 429 word will be transmitted as defined by `_define_trans/_update_trans`, and not automatically overridden so that the whole word has odd or even parity.

If parity generation is enabled, bit 32 of each ARINC 429 word will be adjusted to result in odd or even parity for the word as configured in `_config_trans`.

Default after reset is parity generation enabled.

This command is realized with the function `_set_parity_mode()`.

## get transmitter set

This command reports the current configuration for a particular transmitter. The following information is reported:

- Transmit speed: low speed or high speed
- Precise bit rate of the transmit waveform
- Parity: odd or even
- Transmission mode: burst or time-optimized
- Slope of the transmitter signal
- Amplitude of the transmitter signal

Except for precise bit rate, slope, and amplitude, all other parameters can be modified using the appropriate commands. The values for precise bit rate, slope, and amplitude are hard-coded internally.

Amplitude is expressed in multiples of 0.1mV, and represents the output voltage swing from A-line to Zero-line and B-line to Zero-line, respectively.

This command can be executed at any time without affecting any on-board activities.

This command is realized with the function `_get_transmitter_set()`.

## configure receiver

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. Due to hardware reasons (ARINC 429 transceiver chip) it is not possible to select the speed of a single receiver.

Data reception will be disabled upon execution of the command, until being re-enabled by appropriate instructions. The receivers are now inactive. Any data contained in the receiver buffers is lost. The size of the receiver buffers is reset to zero.

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

This command is realized with the function `_config_rec`.

## Transmitter Functions

### set transmitter mode

This command is used to set a transmitter either to time-optimized or burst transmit mode. It can be executed at any time, and has no impact on any transmit data definitions or transmit activities (except to 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 429 words with the same update rate until all time slices are equally loaded, but still maintaining the update rates.

---

<b>Note</b>	The optimized transmit mode will only work satisfactorily if the resulting timeslice 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 “Transmission Modes” on page 11).
-------------	---

---

This command is realized with the function `_set_tx_mode()`.

### define transmit data

Transmit data definition tells the controller what data to send and how often (repetition rate). This set of information is assigned a unique number between 0 and 255, referred to as transmission ID. Each transmitter has its own set of 256 transmission IDs. If more than one definition is assigned the same transmission ID, 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 a 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:

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

$$\text{Number of ARINC429 words within time slice} = \frac{\text{time slice} [\mu\text{s}]}{\text{word transmission time} [\mu\text{s}]}$$

When entering data definitions, the USB on-card CPU checks whether above conditions are met. It does so by adding up the word transmission times of all data definitions, ignoring one-shot data.

For this calculation, the real word transmission times of the transmit data are used with respect to speed. This sum should be less than or equal to the timeslice value. Special care should be exercised when mixing one-shot data and continuous data. The controller tries to send one-shot data whenever it has time to do so. This is always the case for repetitive data defined with different update rates, because from time to time there will be a timeslice that is not completely occupied. If all repetitive data has 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. Any already defined transmission ID of the transmitter will be assigned the trigger ID zero, thus inhibiting transmission.

#### **Important :**

If the number of transmission definitions does not fit the calculated timeslice (i.e. too many definitions for the timeslice making it impossible to guarantee the defined update rates), it is absolutely necessary to delete transmission definitions (see **delete transmit data** command) until their number fits again.

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

This command is realized with the function **\_define\_trans()**.

### **activate transmit data**

Any data defined will not be transmitted until explicitly activated for transmission. A previously defined data entry (identified by its transmission ID) is activated by assigning a trigger ID between 0 and 255 to it. The default trigger ID zero (0) (assigned automatically when defined) deselects (inhibits) data transmission. A trigger ID of 255 selects it for unconditional transmission. Any other trigger ID 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 IDs can be assigned to one trigger ID.

---

<b>Note</b>	The attempt to activate a transmission ID, for which no data definition exists, is ignored. Re-selection of a one-shot data definition causes this data to be sent once again.
-------------	--

---

This command is realized with the function **\_activate\_trans()**.

## delete transmit data

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 IDs 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.

---

<b>Note</b>	Any transmission activities on the affected channel will be immediately terminated.
-------------	---

---

This command is realized with the function `_delete_trans()`.

## update transmit data

The ARINC 429 word of a transmit data definition can be modified at any time, regardless of its transmission state. The very next transmission is performed with the new ARINC 429 word. If multiple entries for the same transmission ID are given, the one encountered last 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 the trigger state, i.e. the new data is not transmitted until the associated trigger state is set.

---

<b>Note</b>	Data is ignored if no data definition was specified for it previously.
-------------	--

---

This command is realized with the function `_update_trans()`.

## get all transmit data definitions

This command allows reading back all transmit data definitions. The complete transmit data definition list is read (see “Transmit Data Organization” on page 13).

---

<b>Note</b>	As transmit data definitions are read asynchronously with respect to the transmission process, the ticks-until-transmission value is an approximate value only.
-------------	---

---



---

<b>Attention</b>	Programmers are responsible to provide enough memory for the maximum number of transmit data definitions used in their application.
------------------	---

---

This command is realized with the function `_get_tx_definitions()`.

## get single transmit data definitions

This command allows reading back a single transmit data definition identified by its transmission ID. One entry of the transmit data definition list is read (see “Transmit Data Organization” on page 13).

This command is realized with the function `_get_single_tx_definition()`.

## get transmitter timeslice

This command reports the current timeslice value used for a particular transmitter.

The returned timeslice value is zero (0) if there is no transmit data definition for the respective transmitter. If only one-shot data is defined for a transmitter, the returned timeslice value is the required transmit time for all defined one-shot data.

This command is realized with the function `_get_tx_timeslice()`.

## Receiver Functions

### get receiver mode

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.

This command is realized with the function `_get_receiver_mode()`.

### set receiver timestamp resolution

This command sets the resolution of the 32-bit timestamp of received data per receiver to either 100µsec [high-resolution timestamp] or 1 msec [low-resolution timestamp].

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

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

This command is realized with the function `_set_timestamp_res()`.

### get receiver timestamp resolution

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

This command is realized with the function `_get_timestamp_res()`.

### set timestamp high resolution mode

This command sets the resolution of the 32-bit high-resolution timestamps of received data to either 100µsec or 10µsec.

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

Default value after reset is a resolution of 100µs msec.

---

<b>Note</b>	Only receivers configured for high-resolution time-stamping (see <code>_set_timestamp_res()</code> ) are affected.
-------------	--

---

This command is realized with the function `_set_high_resolution()`.

## set receiver buffer size

This command initializes the receive buffers and determines the amount of receive items for one receiver, after which 'data ready' (see command **get receiver data ready**) is set. As a consequence of the size of the receive buffers also the number of allocated receive buffers is determined (see “ARINC 429 Data Reception” on page 14).

The maximum size of each receive buffer is 252 receive items, corresponding to 252 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 or tenths of milliseconds) since the last card reset or on-card clock reset.

When executed, this command purges any data already stored in the receiver buffer. To prevent data loss, this command should be only performed if the receiver is stopped.

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

Reading out the contents of the receiver buffers is independent 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. The **\_read\_receiver\_buffer** function always reads the buffer with the oldest data.

---

**Attention** Users must provide at least as much memory for reading out the receiver buffers (see command **read receiver buffer**) as they have set for 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 IDs (see command **define trigger events**).

This command is realized with the function **\_set\_rec\_buf\_size()**.

## get receiver buffer count

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 the 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 the receiver buffer size the data ready flag is set.

This command is realized with the function **\_get\_rec\_buf\_count()**.

## activate receiver

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

This command is realized with the function `_activate_rec()`.

## read receiver buffer

Received data is kept in internal buffers. These buffers can be read at any time. When this command is given, the buffer with the oldest data of the requested receiver is copied into the data exchange buffer (Dual Port RAM) and from there 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 maximum amount of data read at a time is the receiver buffer size, determined by the command **set receiver buffer size**. To avoid overflow, the user is responsible for providing sufficient memory for reading the receive data (see software interface description for the different languages).

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

This command is realized with the function `_read_receiver_buffer()`.

## get receiver data ready

This command delivers the status of the data ready flags for all four receivers of an A429-USB device.

If a number of data, equal to or greater than the receiver's buffers size, has been received, the respective receiver data ready flag is set.

Received data can be read at any time whether or not the data ready flag is set.

This command is realized with the function `_get_receiver_data_ready()`.

## get receiver data exist

This command returns the status of the data exist flags of all four receivers of an A429-USB device.

The flags indicate if any data is available in the receive buffers of the receivers. This is independent from the size of the receive buffers (see function `_set_rec_buf_size`).

This command is realized with the function `_get_receiver_data_exist()`.

## get receiver data lost

This command returns the status of the data lost flags of all four receivers of an A429-USB device.

The performance of the A429-USB allows guaranteeing that no data received by the transceiver chips is lost. However, it is possible that the host PC does not want to, or is not able to, retrieve data from the cards' 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.

The data lost flags have to be explicitly reset with the function `_reset_data_lost()`.

This command is realized with the function `_get_receiver_data_lost()`.

### reset data lost flag

This command resets data lost flags of selected receivers of an A429-USB device (see command **get receiver data lost**).

This command is realized with the function `_reset_data_lost()`.

## Event System & Trigger Functions

### define trigger event

This command (re)defines up to eight trigger events for each of the four receivers. 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.

For each event definition, the action code, a selector byte, and two 32-bit patterns have to be prepared. The 'mask' is used to select the bits that should compare equal to the second pattern, referred to as 'match'. In other words: if ((ARINC 429 data AND mask) XOR match) does/does not 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 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.

---

**Caution** Specifying trigger IDs 0 or 255 is strictly forbidden. Violation of this restriction may lead to controller hang-up. The only way to recover from this situation is to reset the controller.

---

Excessive use of trigger events on receivers operating in high speed mode may cause data loss. As a rule of thumb, the sum of events on a particular high speed receiver should not exceed (4 / # of channels receiving high speed). This restriction does not apply to receivers operating in low speed mode.

This command is realized with the function `_define_trigger()`.

### set trigger state

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**).

---

<b>Note</b>	Both transmitters 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 command is realized with the function `_set_trigger_state`.

### get trigger state

This command is used to check the state of a particular trigger state table entry (trigger ID).

---

<b>Note</b>	The state for trigger ID 0 is always 'cleared', for trigger ID 255 it is always 'set'.
-------------	--

---

This command is realized with the function `_get_trigger_state()`.

## Discrete I/O Functions

### get state of input lines

This command is used to retrieve the state of the six + six discrete input lines.

This command is realized with the function `_usb_get_discretes()`.

### set state of output line OUTPCC

This command is used to set the state (connected or open) of the four discrete output lines.

This command is realized with the function `_usb_set_discretes()`.

## Miscellaneous Functions

### perform reset/selftest

The ARINC 429 USB device performs various selftest routines to ensure the integrity of its hardware and software components.

These routines are performed with this command, initiating a reset of the A429-USB device. These tests check the major A429-USB building blocks and report their result.

If a malfunction in the A429-USB memory is detected (SRAM, flash EPROM), 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. This tests the ARINC 429 driver circuitry and the receiver inputs.

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 A429-USB device 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.

---

<b>Note</b>	The error bits are useful for locating the failing chip(s) on the controller board. Any bit set denotes a failing one. They are not for customer use.
-------------	---

---

This command is realized with the functions **\_reset\_selftest** and **\_reset\_and\_selftest**.

The function **\_open\_xpc** can be used to initialize the A429-USB device without performing a selftest.

This command is realized with the functions **\_open\_xpc()**.

---

<b>Note</b>	The functions <b>_reset_selftest</b> and <b>_p_reset_selftest</b> perform tests assuming a fully configured A429-USB-R4T2. External loop tests for an A429-USB-R4 configuration deliver errors unless externally wired.
-------------	---

---

### get status word

This command retrieves the 32-bit overall status word of the A429-USB device. Please refer to the C-Interface description for interpretation of the bits.

This command is realized with the function **\_get\_status\_word()**.

### establish 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.

---

<b>Note</b>	Looping back a transmitter to a free receiver allows determining the exact 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 A429-USB applications.

The A429-USB-R4 does not support loopback of transmit channels via relays.

This command is realized with the function **\_establish\_loop()**.

### reset card system clock

This command resets the A429-USB internal system clock.

Resetting the controller clock will cause discontinuities in the outgoing ARINC 429 data stream, if any, and will void timestamps of received data items.

This command is realized with the function **\_reset\_clock()**.

### get board/card timer value

This command returns the A429-USB's timer value containing the number of elapsed milliseconds since the last card reset or A429-USB system clock reset.

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

This command is realized with the function `_get_ticks()`.

The similar function `_get_ext_ticks` returns the A429-USB timer value with the number of elapsed milliseconds and additionally the number of elapsed 100µsecs [or 10µsecs depending on the configuration of the high-resolution timestamping] since the last card reset or A429-USB system clock reset.

This command is realized with the function `_get_ext_ticks()`.

### get A429 board/card type

This command reports the type of the installed ARINC 429 card/board/device.

There are the following type assignments:

- 0 = reserved
- 1 = A429PCC ARINC 429 PCMCIA card
- 2 = all boards using the A429IPM IP module
- 3 = A429-USB device

All hardware is from TechSAT.

This command is realized with the function `_get_arinc_board_type()`.

### get firmware version

This command returns the firmware version number of the currently downloaded firmware.

It is divided into a minor and a major part of the overall number.

The current firmware version is **V1.1**.

This command is realized with the function `_get_firmware_version()`.