Goebel ARINC 429
Software Library and Utilities

User Manual

The Goebel Company

Revision: 2.1.0 **August 27, 2015**

Purpose

This manual describes the software for Arinc 429 PCI interface boards offered by The Goebel Company. This includes application programming interface library and management applications.

Notice

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

Contact

For technical or other inquiries contact:

email: support@GoebelEtc.com

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

Table of Content

1 Introduction

Goebel supplies 32 and 16 channel Arinc 429 PCI boards with configurable transmit/receive channels.

This manual describes the API for part numbers GIO-A429-P32 (32 channel) and GIO-A429-P16 (16 channel) boards. These are universal PCI cards, meaning they operate in legacy 5v 32/64 bit PCI, 3.3v 32/64 bit PCI and PCI-X slots at 33 MHz. All channels are accessed by an HD68 connector on the front panel of the card. Cards are numbered via dip switch for positive identification.

1.1 Advanced Features

Goebel provides features in our 429 card exceeding that available in the industry. Below are highlighted the main features of

- up to 32 channels configurable as transmit, receive or both
- able to receive own transmittion for selftest mode
- card ID via dip switch for positive idendtification
- shared software access to card via Advanced Driver Paradigm
- Error detection or injection of parity, and gap time errors
- Scheduled labels up to 255 per channel for all 32 channels

1.1.1 Configurable transmit receive

Channels which are configurable as to transmit receive function provide the following benefits:

- Switching between real and simulated mode for an LRU can be accomplished without muxing seperate channels. In the sim case the channel is configured for transmit, and in the real case the channel is configured for receive.
- Channels can self check as they are able to receive their own transmission. Receive errors of ones own transmition can be used to detect another transmitter on the bus.
- Fewer card are needed to support different mixes of channels.

1.1.2 Card id

There is a dip switch block that is configured with a card id, in the case of multiple cards. The benefit is that individual cards are positively identified. A card not seen on the bus can be poitively identified with it's card id.

1.1.3 Adjustable Baud Rate

The baud rate is continuously adjustable from 10Kb to 500Kb. Slew rate change over from low speed setting to high speed setting automatic at 25Kb.

1.1.4 Advanced Scheduling

Each channel has up to 255 hardware schedulable labels. The scheduler automatically adjusts for non-standard baud rate setting.

1.1.5 Advanced Driver Paradigm

The driver has a number of advanced features:

- Generation of data.

- Repeating a receive channel onto one or more transmit channels.
- Override of fields of repeated, receive or transmit data.
- Parity error generation on an individual word basis.

1.2 Change log

1.2.1 Revision 2.0.0-a

This is the initial revision for Windows and Linux drivers. The driver name is ga429 and is avalable as Linux RPM distributions, and Windows installers.

2 GA429 Interface Overview.

The API interface has two layers, one which makes use of standard OS calls, and a host independent layer for generating code transportable between Windows and Linux. The host independent layer handles host naming differences.

The API is entirely accessible with only the ga429.h include file. No DLL is required for interfacing. The following table show the calls based on OS type. Note Windows supports stdio calls, open, close, read, write, ioctl.

Host independent	Windows	Linux
ga429_open	CreateFile or open	open
ga429_close	CloseHandle or close	close
ga429_cmd	DeviceIoControl or ioctl	ioctl
ga429_read	FileRead or read	read
ga429_write	FileWrite or write	write

2.1 Devices

There are multiple devices associated with a card. Each device has different purposes in accessing the card. The following is a list of device types and their pupose.

2.1.1 board

This board device allows access to all channels by opening just one device. When writing to the device one supplies a header to each word or block, that can specify timing and error parameters as well as data. Reading returns a header as well as data. The header supplies channel, timing and error information.

2.1.2 channel

There is one device for each channel. Opening a channel device provides a way to access the channel by itself. The channel device is the only device providing RAW mode. That is the mode where you supply or receive a429 words without any formating overhead. RAW mode is the default for channel devices, while PACKET mode can be configured if desired.

2.2 Channel modes

2.3 Queued mode

In Queued mode, data writen to the channel will be transmitted as soon as possible and in the order written. For receive channels data is automatically queued.

2.4 Scheduled mode

Transmit on a channel can be scheduled, whereby the card determines when to transmit labels in a periodic fashion. To set up scheduled mode one issues commands to define the transmit rate for a label – sdi combination. When writing a scheduled label to a channel it will be transmitted the next time that

label is scheduled to transmit. Previously written data for that label will be overwritten and the new data used even if the previous data has never been transmitted.

2.4.1 Schedule priority

When scheduling labels, higher rate labels naturally take precedence over lower rate labels. In some cases this is not desired for a certain lower rate label that must be transmitted at the specific rate. In this case the lower rate label is given a priority that makes it take precedence and gurantees it's transmision at the requested rate.

2.5 Multi user interface

Multiple programs or threads can access board or channel devices simultaneously. This means multiple programs can run independently accessing only the channels or labels they are interested in. Any program can read any or all channels of receive data. If two programs are reading a receive channel both receive all data . If two programs are reading a board device both will receive all data from all receive channels.

If multiple programs are writing to a transmit channel, they would be producing different labels. The labels are either queued or scheduled. For queued labels, the data is transmitted in the order written by the independent programs. For scheduled labels the data is transmitted according to the schedule.

2.6 Raw vs packetized data

The simplest form of interfacing is to read or write 429 words to an open channel device. This method is called RAW mode. In this case only the data itself is read or written. No timing information is available on read, and no error injection is possible on write.
Alternately packetized mode allows reading or writing data with header information for each word. This header information would include timing, and error information. Timing data includes time of day, as well as gap time (time preceeding the word).
An open of a channel device defaults to raw mode but can be configured to packet mode via the config command.

3 A429 Application Programming Interface

The driver interface is largely the same whether the board device or channel device is opened.

3.1 Control Functions

3.1.1 ga429_open

Synopsis

```
#include "ga429.h"

ga429_hdl_t
ga429_open(unsigned lbn, unsigned options);
```

Description

This function establishes the connection to a A429 board identified by **lbn** number which is configured in the boards dip switch set at install time. Boards are shipped as lbn 1 and need not be changed unless multiple boards are present.

Parameters

lbn: logical board number in range 1 to 16.
options: GA429_OPEN_READ | GA429_OPEN_WRITE | GA429_OPEN_RW.

Returns

Handle **hdl** or GA429_CALL_FAILED in case of an error.

Errors

EINVAL:
lbn or **options** is invalid. The **lbn** must be a value between 1 and 16.
ENOMEM:
Memory can't be allocated for buffers.
ENODEV:
No operational device was found with the given **lbn**.

3.1.2 ga429_ch_open

Synopsis

```
#include "ga429.h"

ga429_hdl_t
ga429_ch_open(unsigned lbn, unsigned channel, unsigned options);
```

Description

This function establishes the connection to a channel on a Goebel A429 board identified by **lbn.** **Lbn** the number which is configured in the boards dip switch set at install time. Boards are shipped as lbn 1 and need not be changed unless multiple boards are present.

Parameters

lbn: logical board number in range 1 to 16.
channel: channel number in range 1 to 32.

options: GA429_OPEN_READ | GA429_OPEN_WRITE | GA429_OPEN_RW.

Returns

Handle **hdl** or GA429_CALL_FAILED in case of an error.

Errors

EINVAL:
lbn channel or **options** is invalid. The **lbn** must be a value between 1 and 16.
ENOMEM:
Memory can't be allocated for buffers.
ENODEV:
No operational device was found with the given lbn**.**

3.1.3 ga429_close

Synopsis

```
#include "ga429.h"

void
a429_close(ga429_hdl_t hdl);
```

Description

This closes the connection to a board or channel. Transmit is stopped on a channel if this is the last open of the channel in transmit (GA429_OPEN_WRITE) mode.

Parameters

hdl value returned by ga429_open or ga429_ch_open

Returns

none.

Errors

None.

3.1.4 ga429_cmd

Synopsis

```
#include "ga429.h"

ga429_hdl_t
ga429_cmd(ga429_hdl_t hdl, char *fmt, ...);
```

Description

This function passes command strings to the driver for a variety of device controls. See section 4 for device controls. The command strings are in an XML element like format. This means the format string results in the following general structure:

<command param1=value1 param2=value2 … paramN=valueN />

The handle returned by ga429_board_open or ga429_ch_open is used to identify the connection to apply the command to.

Parameters

hdl: value returned by ga429_open or ga429_ch_open.
fmt See command strings from section 4.

Returns

GA429_SUCCESS or GA429_FAILED in case of an error.

Errors

EINVAL:
hdl or command is invalid.
ENOMEM:
Resources can't be allocated for command.

3.2 Commands for ga429_cmd

This section describes the command strings passed to ga429_cmd. Command strings are used to provide a flexible method of parameter passing to the driver.

3.2.1 start

Synopsis

<start [[[channel=C] label=L] sdi=S] />

Description

This function starts either the board as a whole if no channel is specified, the channel as a whole if no label is specified, or a channel/label/sdi combination if all are specified.

Parameters

channel=C **C** is a value between 1 and 32.
label=L **L** is a label number if starting a scheduled label/sdi combination.
sdi=S **S** is the sdi value between 0 and 3 if starting a scheduled label/sdi combination.

Errors

EINVAL:
if already started

3.2.2 stop

Synopsis

<stop [[[channel=C] label=L] sdi=S] />

Description

This function stops either the board as a whole if no channel is specified, the channel as a whole if no label is specified, or a channel/label/sdi combination if all are specified.

Parameters

channel=C **C** is a value between 1 and 32.
label=L L is a label number if starting a scheduled label/sdi combination.
sdi=S **S** is the sdi value between 0 and 3 if starting a scheduled label/sdi combination.

Errors

EINVAL:
if already stopped

3.2.3 config

Synopsis

<config channel=**C** mode=rx|tx|rxtx speed=low|hi parity=odd|even|disable baud=B gap=G rxformat=raw|packet txformat=raw|packet />

Description

This command configures the channel parameters.

In configuring mode the default if not specified is rx only. When transmitting (tx mode), the channel can be configured to also receive the transmit data (rxtx mode).

Speed is normally configured to either hi (100 Kb) or low (12.5 Kb), with hi being the default if not specified. Custom speeds can be configured with the baud setting.

Rxformat or txformat says if the data includes headers. The packet format has a header prepended to the data, while raw format is the 32 bit word format. Raw format is default for channels. The board device, being a mixture of channels, only allows the packet format, as this format is the only one which provides channel specification.

Parameters

channel=C **C** is a value between 1 and 32.
mode=rx|tx|rxtx The mode indicates the direction of the channel.
speed=low|hi speed of channel, low is 12.5Kb hi is 100 Kb.
parity=odd|even|disable parity is odd by default, while even or diable can be specified. If disabled, the application would normally supply the parity.
baud=B B is a number between 10,000 and 500,000; the baud rate in bits per second.
gap=G gap is the time between words in units of 1/8 bit time. The nominal gap time is 4 bit times or 32 which is the default if gap is not specified.

Errors

EINVAL:
If any parameters are invalid or inconsistent.

3.2.4 Schedule

Synopsis

<schedule channel=**C** label=**L** sdi=**S** msec=**M** hertz=**H** words=**W** value=**V** data=**D** priority=**P** />

Description

This command schedules a label-sdi combination at a periodic rate. Labels which are not scheduled are transmitted when written to the channel. A scheduled label will have a transmission rate expressed in milliseconds, hertz or word times. One of these must be specified the label is to be periodically scheduled. Unscheduled labels are transmitted when written to the channel.

In cases where a label must be scheduled at a precise rate the priority should be specified. Normally the priority is determined by the schedule rate, with higher rate packets taking precedence over

lower rate packets. When the precision of a lower rate packet is more important a high priority should be specified.

When scheduling a label, an initial value for the can be given either for just the value portion, or bits 0x1ffffc00, or the entire data word, bits 0x7fffffff. If the initial value or data is not specified, no data is transmit until the label-sdi is written.

Parameters

channel=C:	channel number.
label=L:	label number.
sdi=S:	sdi number.
msec=M:	rate in milli seconds.
hertz=H:	rate in hertz or words per second.
word=W:	rate in word times, ie schedule every **W** word times.
value=V:	initial value of data for transmit, data resides in 0x1fffffc0 bits of word.
data=D:	initial value of word for transmit.
priority=P:	priority for transmit, a value 0 – 100, higher takes precedence.

Returns

0 on success, GA429_CALL_FAILED in case of an error

Errors

EBUSY:
The hdl has been started.
EINVAL:
channel is not of type tx.
ENOMEM
There are not sufficient resources to create memory for operation.

3.3 Data Format

The simplest form of interfacing is to read or write 429 words to an open channel device. This method is called RAW mode. In this case only the data itself is read or written. No timing information is available on read, and no error injection is possible on write.

Alternately reading or writing data can include header information for each word. This header information would include timing, and error information. Timing data includes time of day, as well as gap time (time preceeding the word). The header format is described below.

An open of a channel device defaults to raw mode but can be configured to packet mode via the config command.

3.3.1 RX Packet headers

All a429 packets read by the API which follow, include a packet header before payload data. The following is the format of this header:

```
typedef struct ga429_rx_hdr {
        unsigned        error:4;        /* error flags */
        unsigned        gap:6;          /* gap time in 1/8 bit times */
        unsigned        channel:6;      /* channel number 1-32 */
        unsigned        length:16;      /* bytes of data returned, not including ga429_rx_hdr_t */
        unsigned        secs;           /* seconds since January 1 1970 GMT */
        unsigned        usec;           /* microseconds for above */
} ga429_rx_hdr_t;
```

length is the total number of bytes in the packet (always 4 bytes). **secs**/**usec** are a unix style timestamp of the packet's receive time. **Flags** contains various information about the packet including errors, label_size, and protocol.

where error is a mask of the following:

```
GA429_PARITY_ERROR_MASK   1
GA429_MANCH_ERROR_MASK   2
```

3.3.2 TX Packet headers

For transmit packets, a transmit header can be supplied to the a429_write call. This is done for error injection or data replay purposes only. In this case the following header is supplied:

```
struct a429_tx_hdr {
        unsigned        error:4;        /* errors to generate */
        unsigned        gap:6;          /* gap time in 1/8 bit times */
        unsigned        channel:6;      /* channel number 1-32 */
        unsigned        length:16;      /* bytes of data returned, not including ga429_tx_hdr_t */
        unsigned        secs;           /* time for replay device only */
        unsigned        usec;           /* time for replay device only */
} a429_tx_hdr_t;
```

length is the total number of bytes in the packet.

3.3.3 Packets

Packet structures including headers are defined as follows:

```
typedef union ga429_packet {
        unsigned char  u8[GA429_PACKET_BYTES];
        unsigned short u16[GA429_PACKET_BYTES/2];
        unsigned int   u32[GA429_PACKET_BYTES/4];
} ga429_packet_t;

struct ga429_rx_packet {
        ga429_rx_hdr_t      hdr;    /* Header describing packet */
        ga429_packet_t      pkt;    /* packet data */
} ga429_rx_packet_t;

struct ga429_tx_packet {
        ga429_tx_hdr_t      hdr;    /* Header describing packet */
        ga429_packet_t      pkt;    /* packet data */
} ga429_tx_packet_t;
```

3.4 Data Transfer Functions.

3.4.1 ga429_read.

Synopsis

```
#include <ga429.h>

int
ga429_read(ga429_hdl_t hdl, void *rcv, int leng);
```

Description

This function reads data from the board or channel identified by **hdl**.

When reading from the channel in raw mode, one or more 32 bit words of 429 data are returned. In raw mode the header contianing timing, and error flags will not be present.

If reading the board, or reading in packet mode one or more packets of type ga429_rx_packet_t will be returned. Packets will have length of sizeof(ga429_rx_hdr_t) + 4. The packet header gives timing and error information.

Parameters

hdl handle returned by ga429_open
rcv pointer to a buffer of length **leng**.
leng total length of rcv buffer in bytes.

Returns

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

If an error occurred, GA429_FAILED is returned and errno will return an error code.

Errors

EINVAL

hdl is not defined.

3.4.2 ga429_write

Synopsis

```
#include <ga429.h>

int
ga429_write(ga429_hdl_t hdl, void *data, int leng);
```

Description

This function writes the given data to the channel or label identified by **hdl**. Normally, **data** points to the complete buffer including any header or label.

If the handle is a sampling label, the data is copied to a label buffer. It will be transmitted when the sampling label scheduler determines that the label is due for transmission.

If the handle is a queuing label, the data is appended to the label's queue, and will be transmitted with all other messages queued when the TX scheduler on the a429 hardware determines that the label is due for transmission. This will happen immediately if not prevented by scheduling constraints.

If the handle is a channel, the data is organized as an array of 4 byte labels or one a429_tx_packet_t structure. The a429_tx_packet_t data is used when in a replay mode. The header will contain the time value used in the replay. The size of the label array can be up to 255 words, if space is available. The size must be ored with 0x8000 to indicate that the data is a a429_tx_packet_t structure.

Parameters

hdl: value returned by a429_channel_open or a429_label_open.
data: buffer holding the data. Array of 1-X 32 label words or one a429_tx_packet_t.
leng: length of the data to be copied in bytes.

Returns

On success the amount of data transferred is returned, on error A429_CALL_FAILED is returned and a429_errno() will return the error code.

Errors

EINVAL
The **leng** parameter is greater than the a429 maximum packet size.
EFAULT
write operation failed.
ENOSPC:
no space on queue to write data. Queue size is 255 words.
ENODEV
hdl is not a valid transmit label.

3.5 Debugging Commands

The driver has the capability of logging debug information to the system log file. The following command initiates the debugging log messages for the driver and may be useful for debugging user programs in exceptional situations.

3.5.1 debug

Synopsis

<debug mask=**flags** />

Description

Debug messages can be enabled in the driver software. The driver outputs debug messages to the system log file, /var/log/messages, for Linux systems, or DebugView accessible messages for Windows.

This command sets the debug options specified by **flags**. Each flag turns controls a certain type of output. A flag has proven useful to user programs is **error**. It gives information about the test that failed, resulting in an API error returned by the driver. Errors detected at the library level do not result in driver calls or log into the system log.

Parameters

flags can be one or more of the following, separated by "|" character.

config Shows configuration related information mainly during board boot.
open Shows information when opening ga429 devices.
read Shows information about receive data.
write Shows information about transmitted data.
ioctl Shows information about commands issued to a board.
intr Shows information during interrupts.
verbose May show more detailed information for the other options.
error Shows information about user call errors.
fatal Shows information about serious firmware conditions.
timing Enables timing calculations returned by a429_get_counter.

By default the following flags are selected: <debug mask=config|error|fatal />.

Returns

0 on success, A429_CALL_FAILED in case of an error

Errors

None

4 GA429 Utilities

4.1 GA429 test program

This is an internal test program that demonstrates various features of a Goebel a429 PCI board. It is provided as a basic test program to validate board functionality. In addition source code is provided in the hope that it may prove useful as an example for programming. It is provided on an as-is basis, and is not intended for production use. As such this documentation is incomplete and not all features are present or functional. That said it is provided in the hope that it may prove useful for certain test uses.

One common use of this program by user's would be to check on activity of the a429 bus. To obtain a list of activity counts enter the following command:

> ga429 count

This particular example shows the counts after running the "ga429 selftest" test.

Additional options can be explored by simply entering:

```
> ga429
enter test type:
             debug Set debug logging options
             sched   Print Channel Schedule
          chaninfo   Print Channel Device Info
             count   Print Channel Counters
        countreset   Reset Channel Counters
           library   Show library revision
                tx   simple tx test
                rx   simple rx test
           connect   simple rx->tx connect
          tx_sched   tx sched test
          selftest   device selftest
```

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

All options may be functional, and some options require the presence of data files and cabling.

4.2 Documentation.

/usr/local/goebel/docs.

4.3 Example programs

Example source code can be found in /usr/local/goebel/ga429/example.

5 Installation

5.1 Linux

Linux software distributions consist of rpm or srpm files.

Software is installed by default in:

/usr/local/bin	ga429 test programs.
/usr/local/lib	libga429.a
/usr/local/include/goebel	include files
/usr/local/goebel/docs	documentation
/usr/local/a429/example	example programs

rpm –U –force a429-<version>.rpm

```
>sudo rpm -U --force ga429-2.1.0-a.centos.el7.rpm
```

Once installed, the user can verify the package was installed using the rpm command.

```
>rpm -qi ga429

Name        : ga429                    Relocations: /usr/local
Version     : 2.1.0                    Vendor: The Goebel Company <Support@GoebelEtc.com>
Release     : a                        Build Date: Wed 04 Nov 2015 05:18:59 PM MST
Install Date: Wed 04 Nov 2015 05:19:24 PM MST Build Host: goebelyzer-7.dv.goebel.aero
Group       : System Environment/Kernel Source RPM: ga429-2.1.0-a.src.rpm
Size        : 1160206                  License: Proprietary
Signature   : (none)
Packager    : The Goebel Company <Support@GoebelEtc.com>
URL         : http://support.goebeletc.com/
Summary     : Goebel a429 Linux driver
Description :
Goebel Arinc 429 (A429) driver

This package is compiled against kernel %{kverrel}.
```

5.1.1 Installation verification

Verification of installation is accomplished by running the selftest with the ga429 program (found in \usr\local\bin). Use the following tests to verify software functionality. No cable need be connected for this test.

```
>ga429 selftest
```

5.2 Windows

Windows installers are available on your companies web support area, under Window/ga429/ga429-<rev>.exe