ASCB Software Library and Utilities

User Manual

The Goebel Company

Revision: 2.1-a

May 18, 2013

Purpose

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

Notice

Contact

For technical or other inquiries contact:

email: info@GoebelEtc.com

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

Table of Content

1 Introduction

The ASCB-D boards supplied by the Goebel Company are used to simulate one or more end systems, and corresponding NICs. The API provides interfaces to transmit and receive frames corresponding to the NIC's data for the frame. Transmit frames are supplied to both primary, and backup network interfaces in compliance with the ASCB-D specification. ASCB headers are produced within the board firmware. Receive frames are processed to reassemble into the NICs frame data. The NIC data is transferred to/from the card with a simple read/write protocol.

Synchronization packets are produced when simulating timing NICs. Synchronization packets are consumed when simulating non-timing NICs to transmit packets in keeping with ASCB timing specifications.

1.1 Release change log

For the most current release change log, consult the release notes provided with the release.

1.1.1 Release 0.1.0

Initial release containing:

- Support of transmit of multiple NICs.
- Support DMA by board CPU.
- Support for NIC.be_reg as well as NIC.le_reg.
- Support for multiple timing NICs.
- Support sys_id.bin for timing NIC system id.
- Supported by TIU software.

1.1.2 Release 0.1.1

Internal release only.

1.1.3 Release 0.1.2

Revision 1.2 includes the following changes to functionality.

- Correct problem with transmit data on right side bus not providing proper sequence numbers.

1.1.4 Release 0.1.3-a

Revision 1.3-a includes the following changes to functionality.

- Add jitter and skew controls for altering NIC transmit timing from nominal. This is a Goebelyzer capability.

1.1.5 Release 0.1.4-a

Revision 1.4-a includes the following software corrections.

- Have firmware deallocate and reallocate the NIC.le_reg configuration as stored in flash memory when closing/opening the board. This allows changing NIC.le_reg files without requiring a board reset or reboot of the host computer.

1.1.6 Release 0.1.4-b

ASCB-0.1.4-b is a release of ASCB firmware to inhibit disabling of a receive channel by firmware when excess error rates are detected. Disabling of channels can mask channel specific errors. When a disabled channel gets reenabled, due to excess errors on the opposite channel, users can see a spike in errors and the appearance of erratic bus behaviour.

1.1.7 Release 0.1.4-c

Revision 1.4-c includes the following software changes.

- Fix read of bad PCI address that sometimes causes host hang on Windows systems.
- Add control pingpong=[off|on] to force ping pong in spite of channel errors. When pingpong=off the behavior is the same as version ascb-0.1.4-b.

1.1.8 Release 0.1.4-d

Revision 1.4-d includes the following software changes.

- Protect against bad ascb message id causing firmware hang.

1.1.9 Release 0.1.4-e

Revision 1.4-e is an internal revision only and not for general use.

1.1.10 Release 0.1.4-f

Revision 1.4-f includes the following software changes.

- Add support for Enhanced ASCB.
- Correct Sync packet transmission content and Master determination.
- Correct channel selection on rx bus.

1.1.11 Release 0.2.1-a

Revision 2.1-a includes the following software changes.

- Add support for FPGA load via software on rev 1203 boards or newer.

1.2 ASCB-D concepts

ASCB-D is a proprietary protocol used in Honeywell's EPIC line of commercial avionics. As a Honeywell proprietary product we defer to Honeywell the description of ASCB-D protocols and reliability assurance algorithms.

1.2.1 Redundant busses

ASCB-D utilizes redundant busses for enhanced reliability and fault tollerance. The Goebel Company's ASCB-D test interface supports Honeywell's redundant bus protocols, with additional controls over bus utilization for fault diagnosing and isolation purposes.

1.2.2 Time scheduling

ASCB-D utilizes a time scheduled protocol to share the ASCB-D bus resource. The timing schedule is defined in the NIC.le_reg, or NIC.be_reg files. Either one can be loaded on the the Goebel NIC.

1.2.3 End System/NIC

And End System transmits and receives on the ASCB-D bus via what is termed a Network Interface Controller or NIC. The NIC is responsible for transmission of data according to the timing schedule as well as reception of data from all other NICs.

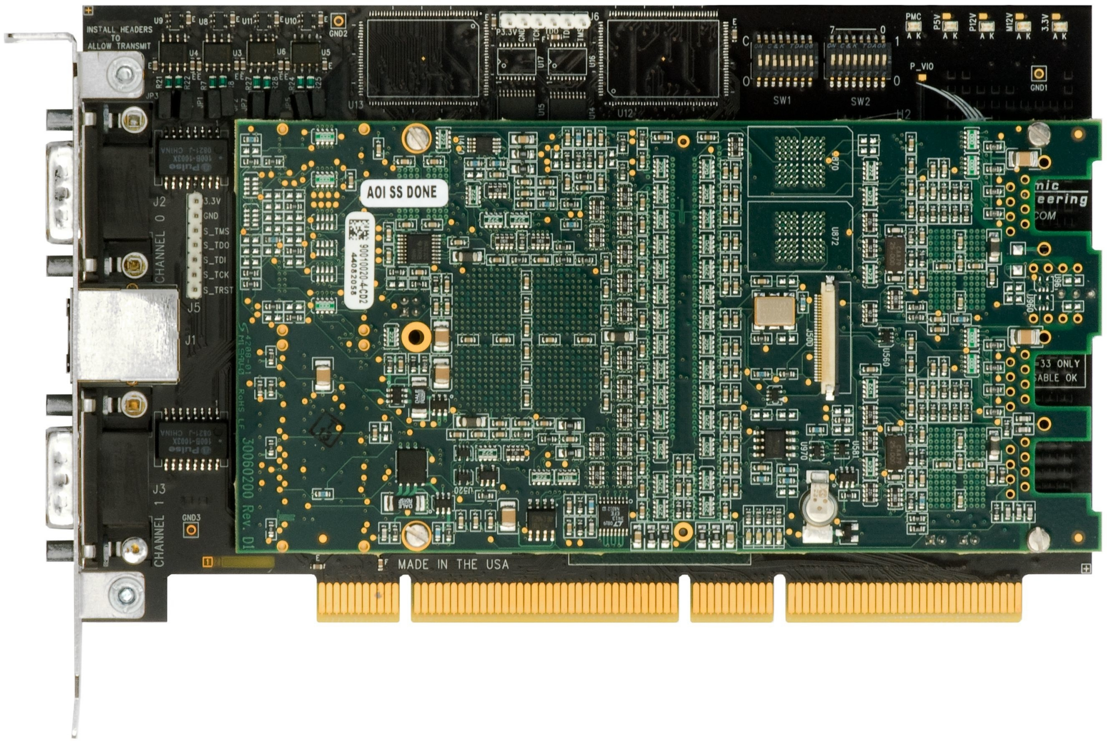
1.2.4 Enhanced ASCB-D

Honeywell's enhanced ASCB provides increased data capacity on the bus compared to previous versions. The Goebel ASCB-D simulation interface card supports enhanced ascb in release ASCB-0.1.4-f and later versions.

Detection of Enhanced ASCB is transparent to the user. Loading the proper NIC.le_reg file is all that is required to insure proper use of Enhanced mode.

2 Hardware Architecture

ASCB-D hardware consists of a PCI card with PMC slot for a CPU. The PCI card contains the ASCB-D IO electronics, while the CPU PMC contains the firmware which drives the ASCB-D protocol. The ASCB-D PCI card is a universal card, meaning it is compatible with legacy 5V PCI as well as 3.3v 33/66 Mhz PCI or PCIX. Note, for insertion on 32 bit slots, insure the mother board does not have components which interfere with the rear tab of PCI-64 connections.



Picture 1: ASCB-D PCI Card

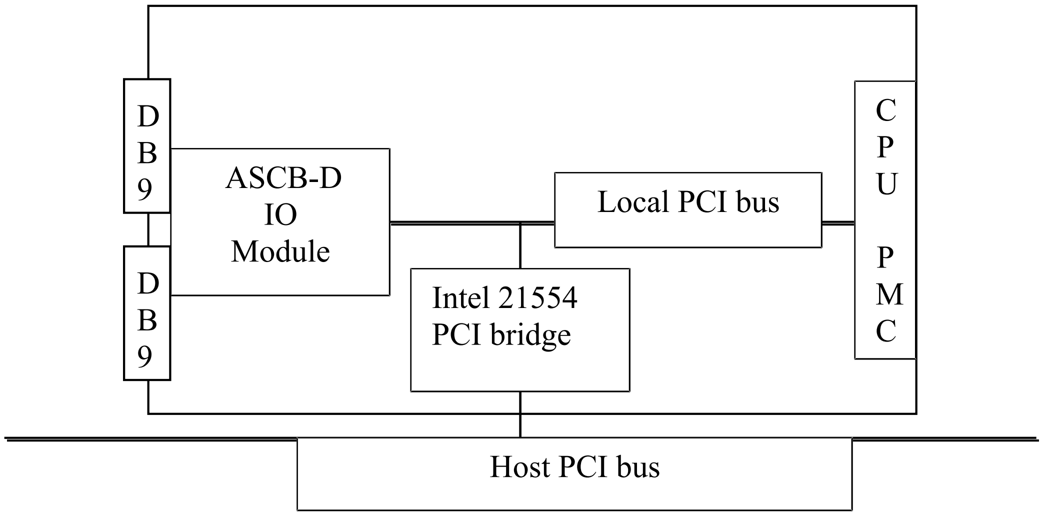


Figure 1 ASCB-D block diagram

3 ASCB-D Application Programming Interface

3.1 Board Control Functions

3.1.1 ascb_open

Synopsis

```
#include "ascb_lib.h"

ascb_hdl_t
ascb_open(int slot, unsigned options);
```

Description

This function establishes the connection to an ASCB board identified by **slot** number. A single board is typically all that is required, so this option would normally be 1. An application will normally call ascb_open with the **options** ASCB_OPEN_BUFFERED | ASCB_OPEN_READ_BLOCKING. The ASCB_OPEN_BUFFERED option specifies that buffers be allocated for read and write operations. This allows the transmit of receive of multiple NIC frames per read or write to the board. The ASCB_OPEN_READ_BLOCKING option indicates that reads block waiting on the frame tick. This allows the application to be synchronized with the frame tick on the ASCB bus.

The ASCB transmit and receive processing is stopped after a successful open. Each board is limited to a single open for transmit and receive.

Other values of **options** exist on ascb_open but are not typically used by user applications. Various ascb utility programs use these other options. Opening a board with the ASCB_OPEN_STATUS option is always possible, regardless of an existing connection for transmit/receive. This option is used by the **afscb** utility to retrieve status information from a board. The ASCB_OPEN_FLASH option is used by the **ascbload** utility to load firmware.

Parameters

slot	the instance number of the ASCB board starting with 1.

options:

ASCB_OPEN_READ_BLOCKING	ascb_read calls block waiting for frame tick
ASCB_OPEN_READ_BUFFERED	for buffered read capability.
ASCB_OPEN_WRITE_BUFFERED	for buffered write capability.
ASCB_OPEN_BUFFERED	for buffered read/write capability.
ASCB_OPEN_STATUS	for opening status device.
ASCB_OPEN_FLASH	for opening firmware device.

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

Returns

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

Errors

EINVAL:
slot or **options** is invalid

EBUSY:
The device with the given **slot** number is already open

ENODEV:
No operational device was found with the given **slot** number

3.1.2 ascb_close

Synopsis

```
#include "ascb_lib.h"

int
ascb_close(ascb_hdl_t hdl);
```

Description

This closes a connection to a board. The boards stops transmitting, and receive processing stops, and the connection to the board is closed.

Parameters

hdl	value returned by ascb_open

Returns

0 on success, ASCB_CALL_FAILED on error.

Errors

EINVAL:
hdl does not describe a board previously opened.

3.1.3 ascb_bit

Synopsis

```
#include "ascb_lib.h"

int
ascb_bit(ascb_hdl_t hdl, unsigned options);
```

Description

This function runs on-board diagnostics according to the specified **options**. Note diagnostics run via API call are non-invasive, in that they are limited to tests which can be performed without reset of the card. A card reset performs a more exhausitive set of tests, and is initiated by the ascb_reset API described in the following section.

Parameters

options identifys the tests to perform. The following options can be added together.
ASCB_BIT_I2CPerform I2C bus check.

ASCB_BIT_CACHE	Perform processor cache check.
ASCB_BIT_MEM	Perform memory check.
ASCB_BIT_PCI	Perform PCI bus interface check.
ASCB_BIT_ETH	Perform ethernet interface check.
ASCB_BIT_PHY0	Perform PHY 0 check.
ASCB_BIT_FLASH	Perform flash memory check.

Returns

Zero on success, or a bit mask of failing tests. The bits returned are the same as the option definitions above. In the case that the bit test call is invalid or the card not accessible, ASCB_CALL_FAILED is returned.

Errors

EINVAL:

hdl is not a valid board handle.

ENODEV

The board is not present.

EIO

The board is not responding.

3.1.4 ascb_config

Synopsis

#include <ascb_lib.h>

int
ascb_config(ascb_hdl_t **hdl**, char ***string**, int **length**);

Description

This function allows configuring board parameters and controls with text commands passed to the firmware. Text commands are used to allow an extensible set of board controls without having to change the library API. The command is contained in **string**. A **length** of string must be specified to prevent buffer overflows. Commands are case insensitive.

Parameters

hdl is the value returned by ascb_open, ascb_nic_handle

string is a set of commands which can include:

```
left(rxbus=off|primary|backup|both,txbus=off|primary|backup|both)
right(rxbus=off|primary|backup|both,txbus=off|primary|backup|both)
nic[1-31,33-63](txbus=off|primary|backup|both)
read_stats=0|1
```

length is the length of **string** in bytes.

Returns

0 on success, ASCB_CALL_FAILED on failure

Errors

EINVAL:

hdl is not a valid board or NIC.

Examples

"left (rxbus=primary)"	selects primary bus for receive on left channel.
"nic1(txbus=backup)"	selects transmit for nic1 on backup channel only.
"read_stats=1"	selects read statistics to be returned with read data.

3.1.5 ascb_config_file

Synopsis

```
#include <ascb_lib.h>

int
ascb_config(ascb_hdl_t hdl, char *filename);
```

Description

This function allows configuring board parameters and controls with text commands passed to the firmware from a file. Text commands are described in the previous section. The text file may include comments which are all characters following a # sign in a line.

Parameters

hdl	is the value returned by ascb_open, ascb_nic_handle
filename	is the name of the file including text command.

Example config file contents

```
#
# Test configuration where timing NICs only transmit on primay bus.
#
nic1 (txbus=primary)
nic2 (txBus=Primary)    # Note case insensitive parameters/values
nic33(TxBus=primary)
nic34(txBus=primary)
```

3.1.6 ascb_reset

Synopsis

```
#include "ascb_lib.h"

ascb_hdl_t
ascb_reset(int slot);
```

Description

This function resets an ASCB board identified by **slot** number. A reset consists of a complete card reset, diagnostics and firmware reboot that takes approximately 20 seconds during which time no application access is allowed. Any existing application connections should be terminated prior to reset. An application which does not terminat it's connection prior to reset will loose functionality on the connection.

Parameters

slot the instance number of the ASCB board starting with 1.

Returns

Zero on success or ASCB_CALL_FAILED in case of an error.

Errors

EINVAL:
slot is invalid
EBUSY:
The device with the given **slot** number is already open
ENODEV:

3.1.7 ascb_start

Synopsis

#include <ascb_lib.h>

int
ascb_start(ascb_hdl_t **hdl**, int **options**);

Description

This function starts the object identified by **hdl**. This can be a board (**hdl** from ascb_open) or a NIC (**hdl** from ascb_nic_handle). NICs are started to enable transmit processing. NICs which are not started do not transmit, but will return received frames.

Starting the board starts the transmit and receive processing. As soon as a frame tick is detected read calls should start returning NIC frames.

Each of the objects which can be started, have statistics counters associated with them. To reset these counters, include ASCB_COUNTER_RESET in the options flags.

Parameters

hdl is the value returned by ascb_open, ascb_nic_handle

Options is a set of flags which can include:

ASCB_FLUSH	flush read data in the board or NIC.
ASCB_COUNTER_RESET	zero the statistics counters for the board or NIC.

Returns

0 on success, ASCB_CALL_FAILED on failure

Errors

EINVAL:
hdl is not a valid board or NIC.

hdl is already started.

3.1.8 ascb_stop

Synopsis

#include <ascb_lib.h>

int
ascb_stop(ascb_hdl_t **hdl**);

Description

This function stops the object identified by **hdl**. This can be a board (**hdl** from ascb_open) or a NIC (**hdl** from ascb_nic_handle).

If **hdl** is for a board the board stops to receive and transmit data. If **hdl** is for a NIC then the NIC stops transmitting. Note that any currently scheduled frame will be transmitted after the stop call.

Parameters

hdl is the value returned by ascb_open or ascb_nic_handle.

Returns

0 on success, ASCB_CALL_FAILED on failure

Errors

EINVAL:
hdl is not a valid board or NIC handle.
hdl is already stopped.

3.1.9 ascb_time_config

Synopsis

#include <ascb_lib.h>

int
ascb_time_config (ascb_hdl_t **hdl**, unsigned long **secs**, unsigned long **usecs**);

Description

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

To specify host clock time specify ~0 for **secs**, and **usecs**. By doing so the driver captures the system clock immediately before passing the time to the card. This results in the smallest latency and highest accuracy of board clock. To maintain synchronization of board and host time, make this call once a second.

Parameters

hdl	The handle of the object being configured.
secs	Greenwich time is secs plus usecs since January 1 1970.
usecs	Greenwich time is secs plus usecs since January 1 1970.

Returns

0 on success, ASCB_CALL_FAILED on failure

Errors

EINVAL:

hdl is not a valid board handle.

3.2 *General Functions*

3.2.1 ascb_fw_version

Synopsis

```
#include <ascb_lib.h>

const char *
ascb_fw_version(ascb_hdl_t hdl);
```

Description

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

Parameters

hdl	the value returned by ascb_open

Returns

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

Errors

EINVAL

hdl is not a valid handle returned from ascb_open

3.2.2 ascb_lib_version

Synopsis

```
#include <ascb_lib.h>

const char *
ascb_lib_version();
```

Description

This function retrieves the ASCB_ interface library version. The current version described by this manual is: "ASCB library rev 0.1.0 Copyright The Goebel Company 2009"

Parameters

None

Returns

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

Errors

None

3.2.3 ascb_errno

Synopsis

```
#include <ascb_lib.h>

int
ascb_errno();
```

Description

This function returns the error code for the last failed call. Note: for systems with errno this call is equivalent to referencing errno.

Parameters

None

Returns

The error code for the last failed call.

Errors

None

3.2.4 ascb_strerror

Synopsis

```
#include <ascb_lib.h>

int
ascb_strerror(int error);
```

Description

This function returns a string describing the error number given as a parameter. Note: for systems with strerror() this call is equivalent.

Parameters

error is an error number returned by ascb_errno() after a function failed.

Returns

Pointer to error string, or the string "unknown error" if the error number passed is invalid.

Errors

None

3.3 NIC Functions

This group of functions deals with creating, configuring and deleting transmit and receive virtual links.

3.3.1 ascb_nic_handle

Synopsis

```
#include <ascb_lib.h>

ascb_hdl_t
ascb_nic_handle(
        ascb_hdl_t      hdl,
        unsigned int    nic_id);
```

Description

This function returns a handle for use by functions which apply to a specific NIC.

Parameters

hdl	value returned by ascb_open
nic_id	nic_id is an integer 1-31 or 33-63.

Returns

0 on success, ASCB_CALL_FAILED on error

Errors

EINVAL

hdl is not a valid handle returned from ascb_open
The nic identified by **nic_id** is not in the schedule.

ENOMEM

A resource limit prevented access to the nic.

3.3.2 ascb_start

See the description under Board Control Functions

3.3.3 ascb_stop

See the description under Board Control Functions

3.3.4 ascb_close

Synopsis

```
#include <ascb_lib.h>

int
```

ascb_close(ascb_hdl_t **hdl**);

Description

This function closes the nic associated with **hdl.**

Parameters

hdl	the handle returned by ascb_nic_handle.

Returns

0 on success, ASCB_CALL_FAILED on error

Errors

EINVAL:

The board is still in a started state.
The NIC associated with hdl does not exist.

3.4 Data Transfer Concepts

3.4.1 NIC schedule

The transmit schedule is defined in the NIC.le_reg, or NIC.be_reg files. This file indicates the timing and amount of data transmitted by each NIC on each frame. In order to perform any transmit on the ASCB bus this schedule must be loaded into the card.

3.4.2 NIC Frames

The set of data read or written for a NIC for a single frame is referred to as the NIC frame. If you are familiar with the Gamma NIC and BIC buffer from flight hardware a NIC frame corresponds to the portion of data in the BIC buffer for the NIC. The following illustration shows the BIC buffers and the corresponding NIC frames. Note that the position and length of a NIC frame is typically different for each frame.

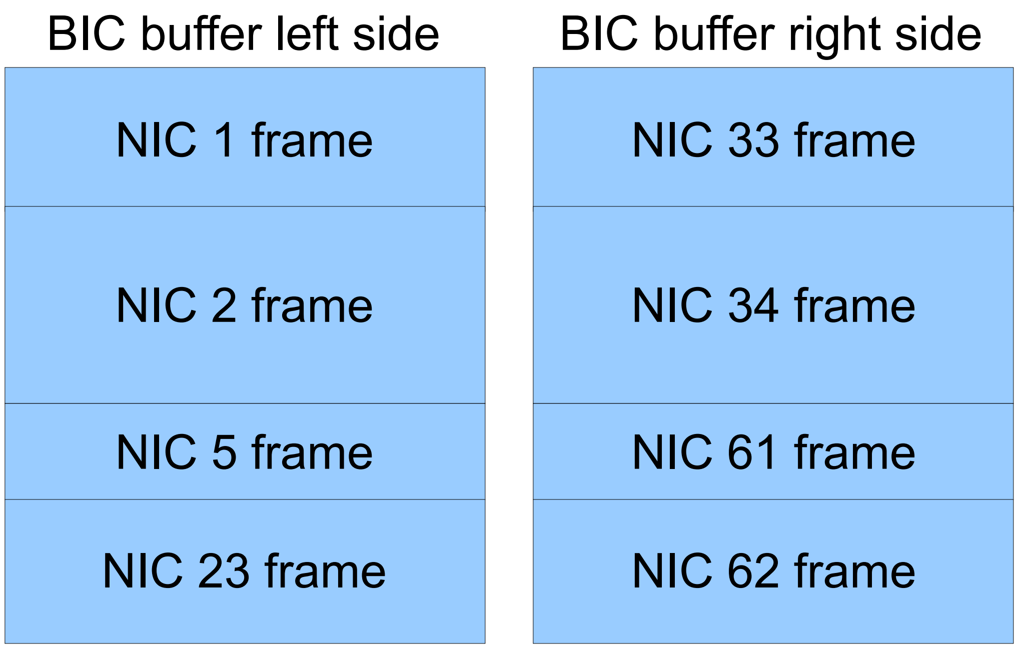


Illustration 1: NIC frames in BIC buffer

3.4.3 Buffered mode

Due to the nature of ASCB, buffered mode is used exclusively to read and write an entire frame worth of data at once. On write, buffering is done within the API, to combine all NIC frames for transfer to the board in a single DMA operation. As far as the application is concerned, the data is generated and written independently for each NIC frame. When all NICs are completed a write flush is done to transfer all NIC frames to the board.

For buffered mode reads all NIC frames received on the prior frame are transferred in a single DMA to the API. Each NIC frame is then read individually through the API until the end of buffer is reached. NIC frames for both sides are available and transferred in a single dma to the read buffer.

When using buffered mode one can read or write directly to the buffer allocated in the API. Using this technique avoids any extra data copy to pass the data between the application and API. The following example shows a buffered mode read.

```
Struct ascb_buffer *buffer;
ascb_hdl_t      board_hdl;

board_hdl       = ascb_open(1, ASCB_OPEN_BUFFERED|ASCB_OPEN_READ_BLOCKING);

/* Get a NIC frame buffer, buffer points into the API allocated buffer holding all NIC data */
buffer  = ascb_get_read_buffer(board_hdl);
```

On writes we get a pointer to buffer space in the API allocated buffer, and then build the NIC frame in this buffer. This eliminates data transfers from the application to the API.

```
Buffer  = ascb_get_write_buffer(board_hdl, size);

/* Build the NIC frame in the API buffer area */
status  = ascb_put_write_buffer(board_hdl, buffer, size);
```

3.4.4 Blocking read

Blocking reads wait within the driver and firmware for the frame tick to complete the read. This allows the application to be synchronized with the ASCB frames. To select blocking reads use the option ASCB_OPEN_READ_BLOCKING on ascb_open.

```
board_hdl       = ascb_open(1, ASCB_OPEN_BUFFERED|ASCB_OPEN_READ_BLOCKING);
```

3.4.5 ascb_buffer

The structure used to hold a NIC frame is the ascb_buffer. The buffer consists of a header, followed by the NIC frame payload. This structure is defined as follows:

```
struct ascb_buf_hdr {
  unsigned short nic_id;        /* NIC number 1-63 */
  unsigned short length;        /* length in bytes of payload, not including ascb_buf_hdr */
  unsigned       error;         /* error flags */
  unsigned       frame;         /* frame number 1-2**32-1 */
  unsigned       bicOffset;     /* position withing BIC buffer of NIC frame */
  unsigned       start_time;    /* time in frame, 12.5 MHz clock */
  unsigned       stop_time;     /* time in frame, 12.5 MHz clock */
  int            secs;          /* seconds since January 1 1970 GMT */
  int            usec;          /* microseconds for above */
};
```

length is the total number of bytes in the NIC frame.

Note the error field is primarily for analyzer use. NIC frames containing errors would not normally be passed to the API.

error is a mask of the following:

ERR_CHECKSUM	sync packet checksum error
ERR_CRC	packet with CRC error
ERR_BICLEN	frame size does not match schedule
ERR_SEQ	packet with sequence number error
ERR_NIC	nic id does not match schedule
ERR_MAN	Manchester error
ERR_BITLEN	packet ends on partial word
ERR_HDR	packet with header error
ERR_LENGTH	packet length does not match header length
ERR_OPMODE	packet with invalid operational mode
ERR_FRAME	sync packet with invalid frame

```
#define ASCB_MAX_PAYLOAD 8192
struct ascb_buffer {
  ascb_buf_header_t  hdr;               /* Header describing packet */
  unsigned char      data[ASCB_MAX_PAYLOAD];     /* frame data */
} ascb_rx_packet_t;
```

3.5 Data transfer Functions

3.5.1 ascb_get_read_buffer

Synopsis

```
#include <ascb_lib.h>

struct ascb_buffer *
ascb_get_read_buffer(ascb_hdl_t hdl);
```

Description

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

Example

```
Struct ascb_buffer *pbuf;

pbuf        = ascb_get_read_buffer(board_hdl);

if (pbuf == NULL) {
  /* No more NICs in frame, do write processing and wait for next frame */
}

if (pbuf == ASCB_CALL_FAILED) {
  /* Not in sync */
  return errno;
}

length      = pbuf->hdr.length;
if (length == 0) {
  /* No NIC frame data, but frame number can be found in header. */
  frame     = pbuf->hdr.frame;
}

switch (NIC_TYPE(pbuf->hdr.nic_id)) {
case NIC_FRAME:
  /* process NIC frame data */
  break;
case NIC_STATS:
  /* process NIC stats if selected via ascb_config() */
  break;
}
```

Parameters

hdl board handle returned by ascb_open with ASCB_OPEN_READ_BUFFERED or ASCB_OPEN_BUFFERED selected.

Returns

The function returns a pointer to the ascb_buffer, or NULL if the end of read data is reached.

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

Errors

EINVAL
hdl is not a valid handle returned from ascb_open
EBUSY
Frame sync not established

3.5.2 ascb_get_write_buffer

Synopsis

```
#include <ascb_lib.h>

struct ascb_buffer *
ascb_get_write_buffer(ascb_hdl_t hdl,int size);
```

Description

This function returns a pointer to an ascb_buffer to in which to build write data for a NIC frame. Note BUFFERED mode must be selected when using this function. By using this function the copy of NIC data from the user application to the write buffer can be avoided. When the write buffer is populated with NIC frame data, call ascb_put_write_buffer to let the API know the buffer is complete.

Example

```
Struct ascb_buffer *pbuf;

pbuf      = ascb_get_write_buffer(board_hdl, size);

if (pbuf == ASCB_CALL_FAILED) {
  /* An error in handle of size detected */
  return errno;
}

/* Build NIC frame data in ascb_buffer. */
  frame   = pbuf->hdr.frame;
}

status    = ascb_put_write_buffer(board_hdl, size);
```

Parameters

hdl	board handle returned by ascb_open with ASCB_OPEN_WRITE_BUFFERED or ASCB_OPEN_BUFFERED selected.

Returns

The function returns a pointer to the ascb_buffer for building a NIC frame.

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

Errors

EINVAL
  hdl is not a valid handle returned from ascb_open
  size is too big

3.5.3 ascb_put_write_buffer

Synopsis

```
#include <ascb_lib.h>

int
ascb_put_write_buffer(ascb_hdl_t hdl,int size);
```

Description

This function passes an ascb_buffer containing write data for a NIC frame to the API. Note BUFFERED mode must be selected when using this function. By using this function the copy of NIC data from the user application to the write buffer can be avoided.

Example

```
Struct ascb_buffer *pbuf;

pbuf      = ascb_get_write_buffer(board_hdl, size);

if (pbuf == ASCB_CALL_FAILED) {
  /* An error in handle of size detected */
  return errno;
}

/* Build NIC frame data in ascb_buffer. */
  frame   = pbuf->hdr.frame;
}

status    = ascb_put_write_buffer(board_hdl, size);
```

Parameters

hdl board handle returned by ascb_open with ASCB_OPEN_WRITE_BUFFERED or ASCB_OPEN_BUFFERED selected.

Returns

The function returns a pointer to the ascb_buffer for building a NIC frame.

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

Errors

EINVAL
hdl is not a valid handle returned from ascb_open
size is too big

3.5.4 ascb_read

Synopsis

```
#include <ascb_lib.h>

int
ascb_read(ascb_hdl_t hdl, void *buf, int leng);
```

Description

This function is primarily for ASCB utilities and not normally user programs. This function reads data from **hdl**, where the payload returned is dependent on the type of handle.

Parameters

hdl	board handle returned by ascb_open
buf	pointer to a buffer of length **leng**.
leng	total length of data returned.

Returns

The function returns the length of the data returned to the buffer, including any header data which may be selected. Zero is returned if no data is available.

If an error occurred, ASCB_CALL_FAILED is returned and ascb_errno() will return an error code.

Errors

EINVAL
hdl is not a valid handle

3.5.5 ascb_write

Synopsis

```
#include <ascb_lib.h>

int
ascb_write(ascb_hdl_t hdl, void *buf, int leng);
```

Description

This function is primarily for ASCB utilities and not normally user programs. This function writes data to **hdl**, where the payload written is dependent on the type of handle.

Parameters

hdl	board handle returned by ascb_open
buf	pointer to a buffer of length **leng**.
leng	total length of data returned.

Returns

The function returns the length of the data returned to the buffer, including any header data which may be selected. Zero is returned if no data is available.

If an error occurred, ASCB_CALL_FAILED is returned and ascb_errno() will return an error code.

Errors

EINVAL
hdl is not a valid handle

3.6 Status Functions

3.6.1 ascb_get_counter

Synopsis

#include <ascb_lib.h>

int
ascb_get_counter(ascb_hdl_t **hdl**, struct ascb_counter ***counter**, int **length**);

Description

This function retrieves statistic counters for the ASCB board (**hdl** from ascb_open) or NICs (**hdl** from ascb_nic_handle)

Parameters

Hdl	is a board or NIC handle.
counter	is a pointer to a structure of type ascb_counter_t. See ascb.h for an definition.

Returns

On success 0 is returned, on error 1 is returned and ascb_errno() will return an error code.

Errors

None

3.6.2 ascb_reset_counter

Synopsis

#include <ascb_lib.h>

int
ascb_reset_counter(ascb_hdl_t **hdl**);

Description

This function resets the statistic counter for the ASCB board or NIC identified by hdl.

Parameters

hdl is the value returned by ascb_open or ascb_nic_handle.

Returns

On success 0 is returned, on error 1 is returned and ascb_errno() will return an error code.

Errors

None

3.7 Debugging Functions

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

3.7.1 ascb_debug

Synopsis

```
#include <ascb_lib.h>

int
ascb_debug(ascb_hdl_t hdl, unsigned long flags);
```

Description

Debug messages can be enabled in the driver software and board firmware. The driver outputs debug messages to the system log file, /usr/adm/SYSLOG, for Unix systems, /var/log/messages for Linux systems. For Windows systems a special program must be run to see kernel debug messages. Download dbgmon to see these messages under Windows.

The firmware outputs debug messages to an internal memory buffer. Debug output written to the memory buffer can be read from the console device (ascb_open(lbn, ASCB_OPEN_CONSOLE)).

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

To see the information routed to the console device enter "ascb console" or on Linux systems do the following:
LINUX "cat /dev/ascb/1/console"

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

Parameters

hdl is the value returned by ascb_open.

flags can be one or more of the following:

ASCB_CONFIG_DEBUG	Shows configuration related information mainly during board boot.
ASCB_OPEN_DEBUG	Shows information when opening or creating boards, Vls, or ports.
ASCB_READ_DEBUG	Shows information about frames received.
ASCB_WRITE_DEBUG	Shows information about frames transmitted.
ASCB_COMMAND_DEBUG	Shows information about commands issued to a board.
ASCB_INTR_DEBUG	Shows information during interrupts (requires debug firmware).
ASCB_VERBOSE_DEBUG	May show more detailed information for the other options.
ASCB_ERROR_DEBUG	Shows information about user call errors.
ASCB_FATAL_DEBUG	Shows information about serious firmware conditions.
ASCB_CONSOLE_DEBUG	Shows information on the serial port as well as console device.

ASCB_TIMING_DEBUG Enables timing calculations returned by ascb_get_counter.

By default the following flags are selected: ASCB_CONFIG_DEBUG, ASCB_ERROR_DEBUG, ASCB_FATAL_DEBUG.

Returns

On success 0 is returned, on error ASCB_CALL_FAILED is returned and ascb_errno() will return an error code.

Errors

None

3.7.2 ascb_config

Synopsis

#include "ascb_lib.h"

int
ascb_config(ascb_hdl_t **hdl**, char ***string**, int **length**);

Description

Various debug options are accessible through the ascb_config interface. While not normally utilized by users, the facility is documented here for information only.

Parameters

hdl is the value returned by ascb_open.

Returns

On success 0 is returned, on error 1 is returned and ascb_errno() will return an error code.

Errors

None

4 ASCB utilities

4.1 Control Panel

With version 0.1.1 of ASCB we are providing GUI based control of our ASCB utilities. This information is provided under a separate document.

4.2 ASCB firmware load utility

The ascbload utility provides for loading of firmware, NIC.le_reg, NIC.be_reg and sys_id.bin files.

When firmware updates are released they are loaded into flash memory via the **ascbload** utility. The firmware resides in the /usr/local/ascb directory. This directory contains the currently installed firmware. It may also contain previous versions of firmware. A link named “firmware” points to the most recently installed firmware.

When the **ascbload** program is run without specifying any parameters, the current firmware from /usr/local/ascb is loaded in all ASCB boards which are present. If the version of the firmware to load matches that which is running, no firmware load is done. After firmware is loaded, a card reset must be done to instantiate the newly loaded firmware. Use the -r parameter to reset after the load is complete.

Synopsis

ascbload [-rnqvF] -f file_name] [1-4]

-r	reset board upon completion of firmware load.
-n	do not load firmware, but print revisions of old and new firmware.
-q	quiet mode, does not print some messages.
-v	verbose mode prints more messages.
-F	Force firmware load even if versions match.

Files

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

4.3 ASCB test program

This is an internal test program that demonstrates various features of the ASCB 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.

4.3.1 ascb firmware

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

Sample output

```
    5 goebelyzer1:/home/user % ascb firmware
    ascb1 firmware:
    The Goebel Company, ASCB-D firmware Rev 0.1.0, May 25 2009 10:47:50
```

4.3.2 ascb sn

To see the serial numbers of the ASCB board, enter "ascb sn". A string such as the following will be shown:

Sample output
```
6 goebelyzer1:/home/user % ascb sn
ASCB board sn 34063010
```

4.3.3 ascb id

To show the FPGA revision id.

```
7 goebelyzer1:/home/user % ascb id
ASCB board 1    fpga id 11
```

4.3.4 ascb count

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

```
     > ascb count
1407 left  pri: packets received
          1220 left  pri: sync packets sent
           400 left  pri: sync packets received
           800 left  pri: bus selected for receive
          1410 left  sec: packets sent
          2020 left  sec: packets received
           400 left  sec: sync packets sent
           800 left  sec: sync packets received
          1287 left  sec: packets with CRC error
          1000 right pri: packets sent
           400 right pri: packets received
           800 right pri: sync packets sent
          1290 right pri: packets with Manchester error
          1780 right pri: packets with CRC error
           400 right sec: packets sent
           800 right sec: packets received
           806 emac: frame longer than maximum, 1500 bytes
           800 emac: Ethernet interrupt with no rfa
             1 frame counter
         12500 frames where sync received
         13499 frames where no sync received
         12499 frames where no sync received for 2 frames
          3800 frames where no sync received for 3 frames
           400 measured frame time of sync source
            30 max frame time in usecs
             1 min frame time in usecs
          5394 total packets sent
          1578 tx packet dropped
             1 tx write missing for frame
           399 read missing for frame
```

```
   399 packets dropped, stopped state
  1201 packets with no nic in schedule
   400 receive errors
  1600 nic frames written to wire
  1600 null nic frames written to wire
 11505 transmit length violation
 10502 transmit queue full
 11638 total sync packets sent
   936 total sync packets received
    48 time in frame tx to FPGA done
  1095 time in frame rx from FPGA done
    69 maximum time in frame tx to FPGA done
 11921 maximum time in frame of write
    70 minimum time in frame of write
    27 maximum time in frame of write done
   901 minimum time in frame of write done
    23 maximum time in frame of read
 13238 minimum time in frame of read
 11474 maximum time in frame of read interrupt
 11918 maximum time in frame of read done
    67 minimum time in frame of read done
```

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

Additional options can be explored by simply entering:

```
> ascb
enter test type:
           console   Print ascb console
             count   Print ascb internal counters
        countreset   Reset ascb internal counters
              dump   Dump ascb memory to ascb_dump
          firmware   Show firmware revision
                id   Show board id
           library   Show library revision
               mem   Print ascb memory
           release   Show board release
             reset   Reset board
                sn   Show board serial number
              test   test board
              tnic   test nic transmit
               all   test all nics in schedule
             sched   show ascb schedule
              show   show ascb stats on console
           library   Show library version
             debug   Set debug logging options
```

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>.

Again not all options may be functional, and some options require the presence of data files.

As a programming example, the source is included in /usr/local/ascb.

5 Installation

5.1 Linux

Linux software distributions consist of rpm or srpm files. The ascb package requires the rdc (remote driver call) package version 2.1 or greater as a prerequisite. To install from rpm:

```
rpm –i rdc-<version>.rpm
rpm –i ascb-<version>.rpm
```

example

```
rpm –i rdc-0.2.1.rpm
rpm –i ascb-0.1.0.rpm
```

Software is installed by default in:

/usr/local/ascb	firmware
/usr/local/bin	ascbload and test programs.
/usr/local/lib	libascb.a
/usr/local/include/goebel	include files

5.2 Windows

The ASCB software for Windows is currently released as a set of zip files containing installers. To install unzip the files and run the installer programs.

5.2.1 ASCB software install

The software required for interfacing with Goebel ASCB boards, consists of

1) driver software.
2) API in form of DLL and LIB files.
3) ascbload program and firmware for loading on cards.
4) ControlPanel.
5) manuals and documentation.

files

GoebelASCBBase.zip

To install unzip each of the above files in order and execute the installer.

5.2.2 Driver install

A driver install will be required for the ASCB card. Once installed it is currently noted as P2P/AFDX/Arinc 664/ASCB. Follow the install procedure below to install the driver. The driver is located in (\usr\local\drivers).

A driver install dialog box should appear upon booting with the ASCB card installed. From this dialog box select the option to install from specific places.

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

Select the software location c:\usr\local\drivers.

Start the install.

5.2.3 Configuration loading

A NIC configuration file must be loaded prior to interfacing with an ASCB bus. When using TIU software this file is loaded automatically at startup. To load the NIC configuration, execute the following command:

```
ascbload -f <directory location>NIC.le_reg
```

5.2.4 Installation verification

Verification of installation is accomplished by running a set of tests with the ascb program (found in \usr\local\bin). Use the following tests without cables connected to verify software functionality.

ascb firmware

```
ascb info
ascb sn
ascbload -f <location>NIC.le_reg
ascb all
```