Copley Controls CAN-PCI-02

Software interface

Overview:

The CAN-PCI-02 product is a PCI based Control Area Network interface card. This card allow software running on a PC style computer to access devices connected to it through the CAN network.

The hardware of the CAN card is accessed through the use of a device driver running on the host operating system. It's Copley's intention to support a wide variety of different operating systems with this card. At the time of this writing (Spring 07) there are drivers available for: Windows NT 4.0, Windows 2k - Vista, Linux 2.4 and 2.6 kernels, and QNX real time OS. Other drivers are likely to be available in the future.

This document describes the interface between the host software and the device driver. Copley has taken pains to keep this interface as simple as possible and to keep it consistent from one OS to another. In particular, the use of DLL (Dynamic Link Libraries) has been avoided in favor of a more direct connection to the device driver. This simplifies software installation as it reduces the number of additional files that need to be installed.

A simple set of functions written in both C and C++ programming languages is available on the Copley web site which manage the interface to the CAN driver. These functions handle all the low level details of opening a connection to the driver, configuring the port, reading and writing CAN messages and closing the port. No knowledge of the following low level details of how these functions work is required in order to use them.

Connecting to the card:

Before any communication can be made to the PCI card, a *handle* to the device driver must be obtained. This is generally done through the use of the same OS support functions that would be used to open a file in the file system.

To open a handle to the driver, it's first necessary to know the name assigned to the device driver by the operating system. Since the PCI card contains two separate CAN ports, there are actually two names associated with each card; one for the first port, and one for the second port. By convention, the first port is the one closest to the PCI connector on the PCI card.

Under windows, the name used to open a handle to the device driver has the following form:

`\\.\copleycan<card>\<port>`

In this name, the `<card>` should be substituted with a two digit card number, and the `<port>` should be substituted with a single digit port number. Both the card and port numbers count from zero.

For example, on a PC with two CAN cards installed, the following four port names are available:

```
\\.\copleycan00\0
\\.\copleycan00\1
\\.\copleycan01\0
\\.\copleycan01\1
```

Under Linux and most other Unix based operating systems, the device files reside in the `/dev` sub-directory and are given names of the form `/dev/copleycan<port>` where `<port>` is the zero

based port number. Each card creates two ports starting with port zero.

Once the name of the driver file is known, the low level OS function used to open a file may be used to create a handle to the driver. In windows, this function is called `CreateFile`. In Linux / Unix this function is called simply `open`.

Example C code for opening a handle to the driver under Windows:

```
HANDLE hndl;
hndl = CreateFile( "\\\\.\\copleycan00\\0",
                   GENERIC_READ|GENERIC_WRITE,
                   FILE_SHARE_READ|FILE_SHARE_WRITE,
                   NULL, OPEN_EXISTING,
                   FILE_ATTRIBUTE_NORMAL|FILE_FLAG_OVERLAPPED,
                   NULL );
if( hndl == INVALID_HANDLE_VALUE )
{
    // handle open failure
}
```

Note that in the windows example above, the backslash characters in the file name were doubled. This is required in C code because the backslash character has special meaning in strings. The resulting file name is as described above.

Example C code for opening the device under Linux:

```
int hndl;
hndl = open( "/dev/copleycan00", O_RDWR );
if( hndl < 0 )
{
   // handle open failure
}
```

Sending commands:

Once a handle to the driver has been opened, it is possible to send commands to the driver. These commands are used to configure the card and read status information from it.

All commands to the driver are packaged as an array of one or more 32-bit integer values. All commands return an array of one or more 32-bit integer values as a response. The actual number of words of data sent to or from a particular command is defined as part of the command description.

The first word of data sent as part of a command is required for all command types. This word is formatted as follows:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								Command code								Reserved										Data words					

Bits 16-23 give the command code that defines the type of command being sent. Bits 0-5 give the number of additional 32-bit words that are passed with the command. All other bits of this word are reserved and should be set to zero.

All commands return one or more 32-bit words of data. The first word of data returned from a command is formatted as follows:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								Error code								Reserved										Data words					

Bits 16-32 give an error code which indicates the success/failure of the command. A zero in this field indicates success. Bits 0-5 give the number of additional 32-bit words of data returned by the command. All other bits are reserved and should be ignored.

Commands are sent to the driver using the I/O control system function. In windows, this function is called `DeviceIoControl`. The use of this function is somewhat complex and beyond the scope of this document. Interested readers should refer to the documentation provided by Microsoft on the Microsoft developers network web site.

A list of all command and error codes can be found at the end of this document.

Receiving a CAN message:

Receiving a CAN message from the CAN card driver is also handled through the I/O control system call. The structure passed to this system call is defined as follows:

```
typedef struct
{
   int32_t  timeout;      // Reserved for use by driver
   uint32_t timestamp;    // Timestamp (microseconds)
   uint32_t id;           // CAN message ID
   uint32_t flags;        // Various flags
   uint8_t  data[8];      // CAN message data
} CANCARD_MSG;
```

The first member of this structure gives a timeout value in milliseconds. The I/O control function will return as soon as a CAN message has been received, or this timeout expires. If the timeout is set to zero, then the CAN driver will return immediately if no data is available. If the timeout is set to a negative number, then the driver will wait for a CAN message with no timeout.

If a CAN message is successfully received by the driver before the timeout expires, then the message structure will be filled in. The flags member is a bit-mapped value which gives some additional information about the CAN message received. The following bits are defined:

31-7	6	5	4	3	2	1	0
Reserved	Extended	Notify	RTR	Length			

Length gives the number of bytes of CAN data passed with the message. It's legal range is zero to eight.

RTR will be set for remote request messages. It is clear for standard data messages.

Notify will be set if the CAN port is configured to echo transmitted messages back to itself. This is mostly useful for CAN bus monitoring and is disabled by default.

Extended is set if this CAN message uses a 29 bit extended CAN ID. If set, then the CAN ID value occupies bits 0-28 of the ID field. If clear, then the CAN ID is a standard 11-bit value and occupies bits 0-10 of the ID field.

Sending CAN messages:

CAN messages are also sent through the use of an I/O control function. The structure that is used to pass the CAN message information to the driver is exactly the same as the structure used to receive CAN messages.

List of command codes:

The following command codes may be passed to the driver.

Code	*Data to driver*	*Data from driver*	*Description*
1	0	0	Open the CAN port
2	0	0	Close the CAN port
3	1	0	Set bit rate. The passed word of data gives an identifier in the it's lower 8 bits. This identifier defines the bit rate to select. The following bit-rates are supported: 1 1,000,000 bits/sec 2 800,000 bits/sec 3 500,000 bits/sec 4 250,000 bits/sec 5 125,000 bits/sec 6 100,000 bits/sec 7 50,000 bits/sec 8 20,000 bits/sec
4	0	2	Get bit rate. The bit rate code is returned in the lower 8 bits of the first word of data returned. The other data returned should be ignored.
9	2	0	Set a parameter. The first passed word of data gives the parameter number, the second word gives the value. See the list of parameters below
10	1	1	Read a parameter. The passed word of data gives the parameter number. The returned data gives the parameter value.
249	1	0	Set the receive buffer interrupt threshold. This defines how many CAN messages must be stored in the cards on-board receive buffer before the host is interrupted. This should normally be left at the default value of zero.
252	1	0	If the passed parameter is non-zero, then all transmitted CAN messages will be fed back to the receive buffer with valid time stamps. If the passed parameter is zero, then this feature will be disabled (default).
255	0	0	Reset the CAN card.

List of CAN error codes:

Code	*Description*
0	No error
1	Unknown command passed
2	Illegal parameter passed
3	CAN port is already open
4	CAN port is not open
5	Command already in progress.
6	Internal device failure
7	Timeout waiting on command.
8	Signal received by driver
9	Not enough data passed with command
10	Command mutex held (driver error)
11	Invalid queue head/tail pointer
12	Failed to erase program/data flash memory
13	Attempt to write firmware before erasing flash
14	Too much data sent
15	Unknown I/O control code passed
16	Command passed without required header
17	Too much command data passed
18	Command already in progress on card
19	More then 8 bytes of data sent with CAN message
20	Transmit queue is full
21	Receive queue is full
22	Parameter is read only
23	Memory read/write test failure
24	Memory allocation failure
25	Reserved for use internal to driver
26	Generic driver error code

List of CAN card parameters:

There are several parameters which can be accessed to get information about the CAN card.

Parameter ID	***Name***	***Description***
0	Serial number	Card serial number
1	Manufacturing information	Reserved for use by Copley Controls.
2	PCI bus voltage	Voltage of PCI bus scaled by 64k/5V.
3	3.3V supply	Voltage of the on-board 3.3V supply scaled by 32k/5V.
4	2.5V supply	Voltage of the on-board 2.5V supply scaled by 32k/5V.
5	CAN port status	Bits Meaning 0-7 Transmit error count 8-15 Receive error count 16 Set if port is open 17 Set if port is synchronized to bus 18-19 Transmit status (normal, warning, error, bus-off) 20-21 Receive status (normal, warning, error, bus-off) 22-31 Reserved.
10	Interrupt inhibit timer	Gives the maximum time in microseconds to delay a receive interrupt in the receive interrupt threshold has not been met.