USB DAQ & Relay Card Products

Using Linux to control EasyDAQ USB Products

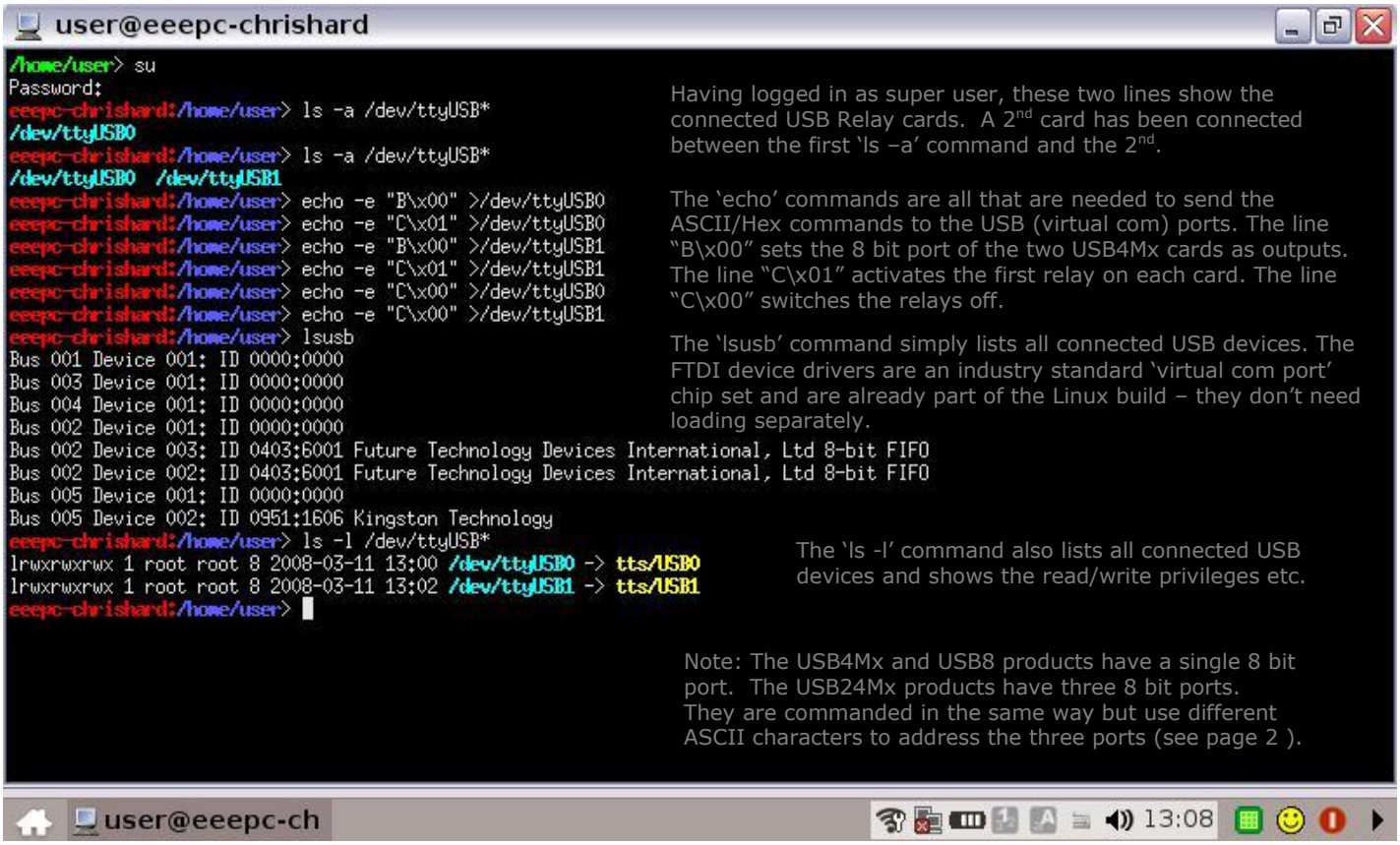
Product Datasheet 34

Description

We have many customers, worldwide, using our USB products in Linux based applications and designs.

All of our USB products (except the USBDAQ1208LS) use the industry standard 'FTDI virtual com port' chipset and offer a simple text string command protocol. Any programming language which offers COM port output (i.e. Serial/RS232 string output) can be used (see page 2 for details of the command syntax). You can send commands to the card using simple 'echo' command strings or script files. All Linux builds (to our knowledge) already include the low level device drivers so there is no need to install any additional drivers.

Here is a screenshot of how it is done. We used an Asus EeePC, which runs a version of Linux called Xandros:



General information about the USB/serial interface (Virtual Com Port):

Serial Port settings

Baud rate:	9600	Parity:	0
Data:	8 bits	Stop bits:	1
Handshaking:	None		

USB - Auto detection & com port assignment

This product uses the industry standard, FTDI Virtual Com Port chipset. When you connect it the USB port of a PC for the first time, it will be auto-detected and ask you to install the appropriate 32 or 64 bit drivers (downloadable from the 'downloads' section of our website). After installation, the card will appear as a 'Virtual' COM port and be automatically assigned a COM port number by your OS. Following installation, the COM port number can be manually re-assigned via the control panel if required. Following reboots or disconnects of the USB card, the same COM port number will be assigned. You can find the assigned COM port number via the Com Port settings of the Device Manager (under your PC's Control Panel). Note: Mac & Linux OS's do not require the VCP drivers to be installed; they are already installed as part of the standard OS build.

Command format

The card is commanded via simple single ASCII characters (+ status byte). These are commands that address each port of the PIC device (Hex equiv shown in brackets). The card can also be commanded via HyperTerminal – see below. If writing your own code, and not using one of our downloaded example programs, you need to ensure that you allow approximately 10mS between successive commands. After issuing a 'read' command, you subsequently need to 'poll' the USB/serial port in order to read the data transmitted by the USB Relay/DIO card (from the card to the USB/Serial port).

Command summary:

ASCII A x (41H) = Read Port B (Chans 1-8)
ASCII B x (42H) = Configure Port B (Chans 1-8)
ASCII C x (43H) = Write Port B (Chans 1-8)

ASCII D x (44H) = Read Port C (Chans 9-16)
ASCII E x (45H) = Configure Port C (Chans 9-16)
ASCII F x (46H) = Write Port C (Chans 9-16)

ASCII G x (47H) = Read Port D (Chans 17-24)
ASCII H x (48H) = Configure Port D (Chans 17-24)
ASCII J x (4AH) = Write Port D (Chans 17-24)

Port B (Channels 1-8) command detail (Port A is not configurable – it handles the data comms):

ASCII 'B' (42H), X Initialises the card (sets the port & channel I/O directions). Set direction of Port B, 1=Input, 0= output. (i.e. where X=10111111 (AFH) = sets bit 7 as an output, the rest as inputs).
ASCII 'A' (41H), X Read Port B (Char X=don't care. Device sends 1 byte of returned data).
ASCII 'C' (43H), X Write data X to Port B (i.e. X=00000001 (01H), sets channel 1 to active). Valid data bytes are latched by the card until a further valid data byte is written to it.

Port C (Channels 9-16) command detail:

ASCII 'E' (45H), X Initialises the card (sets the port & channel I/O directions). Set direction of Port C, 1=Input, 0= output. (i.e. where X=10111111 (AFH) = sets bit 7 as an output, the rest as inputs).
ASCII 'D' (44H), X Read Port C (Char X=don't care. Device sends 1 byte of returned data).
ASCII 'F' (46H), X Write data X to Port C (i.e. X=00000001 (01H), sets channel 1 to active). Valid data bytes are latched by the card until a further valid data byte is written to it.

Port D (Channels 17-24) command detail:

ASCII 'H' (48H), X Initialises the card (sets the port & channel I/O directions). Set direction of Port D, 1=Input, 0= output. (i.e. where X=10111111 (AFH) = sets bit 7 as an output, the rest as inputs).
ASCII 'G' (47H), X Read Port D (Char X=don't care. Device sends 1 byte of returned data).
ASCII 'J' (4AH), X Write data X to Port D (i.e. X=00000001 (01H), sets channel 1 to active). Valid data bytes are latched by the card until a further valid data byte is written to it.

Using Windows HyperTerminal

In order to test operation, the card can be connected to a serial port and controlled from Windows HyperTerminal. Ensure port configuration is set as shown above, type (ASCII) characters shown above to achieve port direction and read or write command/data. Please see the following screenshots for configuring HyperTerminal for use with our serial or USB connected relay/DIO cards:

Commanding our USB relay & DIO cards and writing your own software:

- In general our USB and serial port cards are designed to be easy to use. There are example programs (available from the 'downloads' area of our website) in LabVIEW, Visual Basic, Visual C, Delphi & Agilent VEE. If you are not using/adapting one of our programming examples, and wish to write your own software, the cards can be commanded by simple ASCII/Hex based command characters. All of our USB/serial products use the command format so you should find it easy to port your software from one product or application to another.

- The cards have either a single 8 bit port (USB4/8 = Port B), two 8 bit ports (USB16 = Ports B & C) or three 8 bit ports (USB24 = Ports B, C & D). In the case of the USB4Mx cards, the lower 4 bits are used to control the relays (therefore they need to be set as outputs) and the upper 4 connect to the four general purpose (uncommitted) DIO channels. These can be configured as either inputs or outputs, depending on your requirements.

- **Steps required to command/control the cards:**

1) Configuring the 8 bit port (set the port direction):

- **ASCII B (42H) (Config Port B),** followed by the hex character 00H (all zeroes) – this sets all 8 bits of the port as outputs (i.e. to control the relays). Your code only needs to do this once during initialization (only needs to be done following power up - if card is unplugged or powered down it will loose it's config settings).

- (ASCII B (42H) followed by the hex character F0H will set the lower four channels as outputs and the upper four as inputs).

2) Setting/reading individual channels:

- **ASCII C (43H) (Write Port B),** followed by the individual bit status for whichever relay you want to switch on: 43H01H will switch on relay 1, 43H02H will switch on 2nd relay, 43H03H will switch on the 1st and 2nd relay at the same time and so on.

- **ASCII A (41H) (Read Port B),** followed by a 'don't care' character will perform a port read function. If you send this character sequence to the card, it will perform a read of the current status of 8 bit port (even if channels are set as outputs - therefore, your software does not need to 'remember' what you previously sent to the card). When you perform a port read function, you will also need to immediately follow the command by a serial port read in order to read the characters that are transmitted back from the card to your PC.

- 16 and 24 channel cards work in exactly the same way, but use the following additional ports:

- **Port C = ASCII Exx (45H) (Config Port C), Fxx (46H) (Write Port C), & Dxx (44H) (Read Port C).**

- **Port D= ASCII Hxx (48H) (Config Port H), Jxx (4AH) (Write Port D), & Gxx (47H) (Read Port D).**

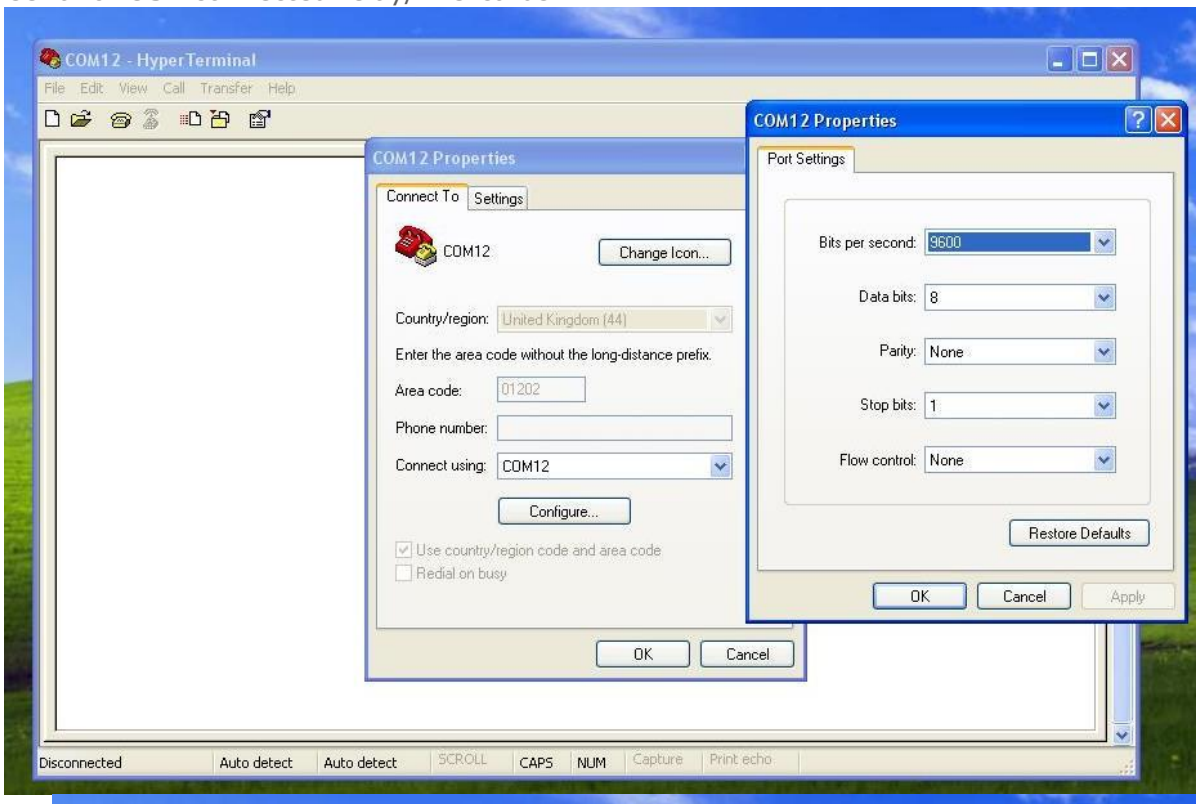
- xx = 8 bit hex number (00H to FFH, where 00H = all channels off, FFH = all channels on)

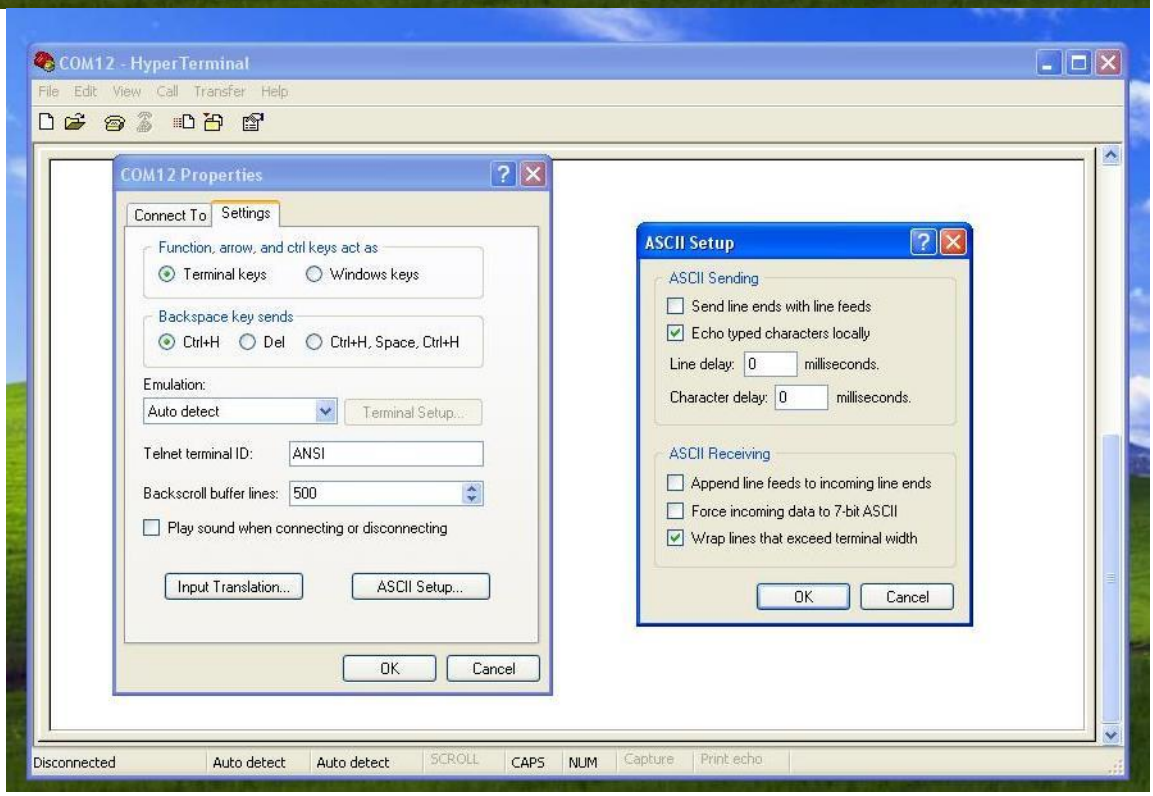
- **Note: It is recommended that you first try the card out on a PC (using our generic demo program) to ensure that you can see it running. The demo program allows you to set individual relays or auto-cycle them. It also has a numeric display window which shows the Hex characters that are being sent to the card in order to configure & control the card.**

Using Windows HyperTerminal

In order to test operation, the card can be connected to a serial (or USB) port or your PC and controlled from Windows HyperTerminal. When connected to a USB port, the card will enumerate as a 'virtual COM port' – you need to use the HW Device Manager to determine which port number has been assigned – the number should then appear in the HyperTerminal list of available COM port options.

Ensure port configuration is set as shown above, type (ASCII) characters shown above to achieve port direction and read or write command/data. Please see the following screenshots for configuring HyperTerminal for use with our serial or USB connected relay/DIO cards:

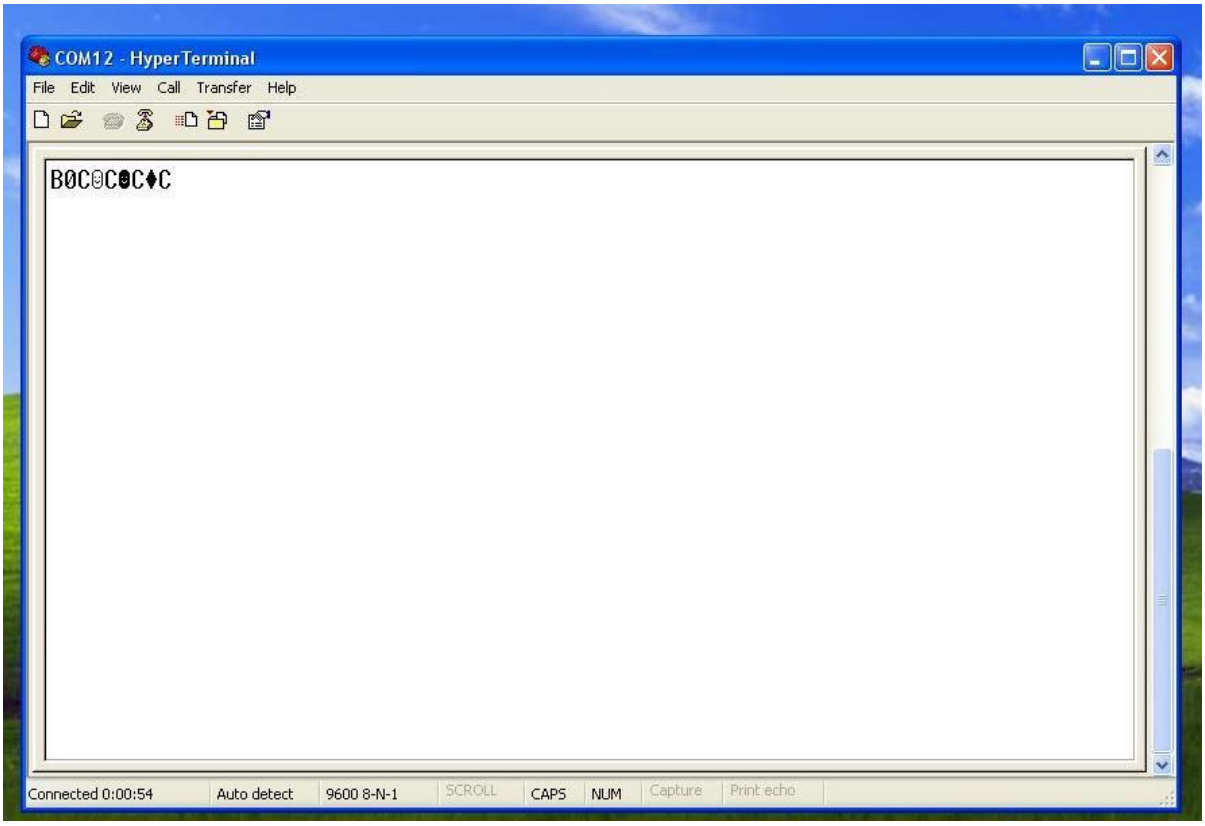




Using Windows HyperTerminal

In order to activate relays 1 to 4 of a connected serial or USB card, set the CAPS LOCK key on (i.e. capital ASCII characters) & type the following characters:

B0	(Capital B, number zero) – configures first port as outputs – see description in next section
C CtrlA	(Capital C & Ctrl+A key pressed at the same time – where Ctrl = Control key)= Relay 1 on
C CtrlB	(Capital C & Ctrl+B key pressed at the same time) = Relay 2 on
C CtrlD	(Capital C & Ctrl+D key pressed at the same time) = Relay 3 on
C CtrlH	(Capital C & Ctrl+H key pressed at the same time) = Relay 4 on



Using Windows HyperTerminal - explanation of Ctrl characters:

Binary equivalents of the above ASCII characters – you can find these in any internet search for an ASCII character table:

Ctrl@	= Binary 00000000 (All relays/DIO chans off)
CtrlA	= Binary 00000001 (Relay/DIO chan 1 on)
CtrlB	= Binary 00000010 (Relay/DIO chan 2 on)
CtrlD	= Binary 00000100 (Relay/DIO chan 3 on)
CtrlH	= Binary 00001000 (Relay/DIO chan 4 on)
CtrlP	= Binary 00010000 (Relay/DIO chan 5 on)
Space	= Binary 00100000 (Relay/DIO chan 6 on)
Shift2	= Binary 01000000 (Relay/DIO chan 7 on)

Chan 8 – there is no ASCII character equivalent for Chan 8

The above ASCII codes are shown as examples only – when you create your own code (if not using one of our example programs), you need to configure your SW to send the equivalent binary (or Hex) code sequences.

Commanding our USB relay & DIO cards and writing your own software:

- In general our USB and serial port cards are designed to be easy to use. There are example programs (available from the 'downloads' area of our website) in LabVIEW, Visual Basic, Visual C, Delphi & Agilent VEE. If you are not using/adapting one of our programming examples, and wish to write your own software, the cards can be commanded by simple ASCII/Hex based command characters. All of our USB/serial products use the command format so you should find it easy to port your software from one product or application to another.

- The cards have either a single 8 bit port (USB4/8 = Port B), two 8 bit ports (USB16 = Ports B & C) or three 8 bit ports (USB24 = Ports B, C & D). In the case of the USB4Mx cards, the lower 4 bits are used to control the relays (therefore they need to be set as outputs) and the upper 4 connect to the four general purpose (uncommitted) DIO channels. These can be configured as either inputs or outputs, depending on your requirements.

- **Steps required to command/control the cards:**

1) Configuring the 8 bit port (set the port direction):

- **ASCII B (42H) (Config Port B),** followed by the hex character 00H (all zeroes) – this sets all 8 bits of the port as outputs (i.e. to control the relays). Your code only needs to do this once during initialization (only needs to be done following power up - if card is unplugged or powered down it will loose it's config settings).

- (ASCII B (42H) followed by the hex character F0H will set the lower four channels as outputs and the upper four as inputs).

2) Setting/reading individual channels:

- **ASCII C (43H) (Write Port B),** followed by the individual bit status for whichever relay you want to switch on: 43H01H will switch on relay 1, 43H02H will switch on 2nd relay, 43H03H will switch on the 1st and 2nd relay at the same time and so on.

- **ASCII A (41H) (Read Port B),** followed by a 'don't care' character will perform a port read function. If you send this character sequence to the card, it will perform a read of the current status of 8 bit port (even if channels are set as outputs - therefore, your software does not need to 'remember' what you previously sent to the card). When you perform a port read function, you will also need to immediately follow the command by a serial port read in order to read the characters that are transmitted back from the card to your PC.

- 16 and 24 channel cards work in exactly the same way, but use the following additional ports:

- **Port C = ASCII Exx (45H) (Config Port C), Fxx (46H) (Write Port C), & Dxx (44H) (Read Port C).**

- **Port D= ASCII Hxx (48H) (Config Port H), Jxx (4AH) (Write Port D), & Gxx (47H) (Read Port D).**

- xx = 8 bit hex number (00H to FFH, where 00H = all channels off, FFH = all channels on)

- **Note: It is recommended that you first try the card out on a PC (using our generic demo program) to ensure that you can see it running. The demo program allows you to set individual relays or auto-cycle them. It also has a numeric display window which shows the Hex characters that are being sent to the card in order to configure & control the card.**

Addendum - user comments regarding binary format commanding:

If you prefer to think in binary rather than Hex & ASCII, read on….

Thinking of our USB relay cards as a purely binary system makes more sense for some users/programming language:

User comments regarding USB 4, 8, 16 & 24 chan cards (they all use the same commands):

Send it '195' and it switches on relays 1, 2, 7 and 8 (195 = 11000011). Which is 1 + 2 + 64 + 128 = 159. Whereas in hex this would be C 3, which looks nothing like the the LED display.

The only confusion is that you need to, in VB at least, send the command integers as a **Binary type array**.

So I send: {66, 0} to initialise
{65, 0} to read
and {67, INT} to write

Each relay then has it's own independent binary value:

Relay:	1	2	3	4	5	6	7	8
Value:	1	2	4	8	16	32	64	128.

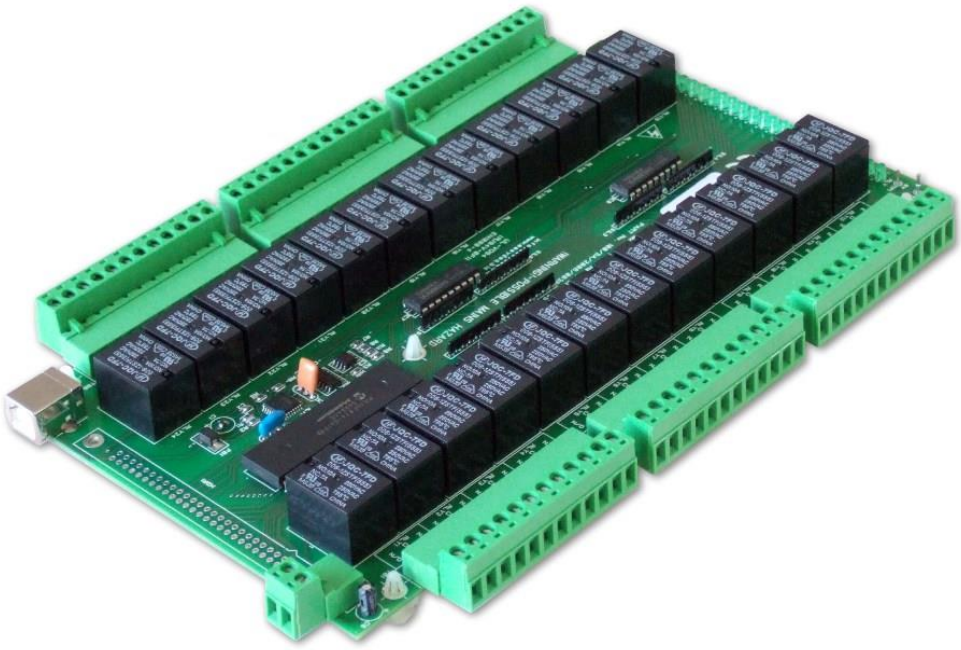
So if I now want to switch on Relay_5 I simply add it's value (16) to current display value (INT, or I can read the current display INT value) and send it:

{67, INT+16}

195 = 11000011
195 + 16 = 211 = 11001011

The only confusion I can foresee is that binary should write the highest value digit on the left (i.e. 11010011).

I think explaining in terms of binary will help people a great deal more as rather than mixing HEX, ASCII and Binary you can just stick an integer and it's direct binary representation.



Addendum - user comments & updates (various):

- Emailed update (1)….. Unfortunately that (example) code also never reads the relay state. However, it waggled the serial lines slightly differently, which I then emulated, and reads started working. After a bit of digging I found that Linux asserts DTR when a device is opened while both Windows and OSX require the programmer to assert the line. Further, the Mac toggled DTR before asserting it and that seems to be all that was needed. I cannot see why, but now that this works, I dont really care. It means we can shutdown/startup our Linux servers without having to maintain state by simply reading the initial state from the board, a big plus. So thanks for the help

- Emailed update (2)…..

Remember also the difference between the first character which is the hex representation of an ASCII character and the 2nd which is Hex/binary number.

- Hi Chris,
  That was the reason!
  I didn't modify the TTY settings, just opened /dev/ttyUSB0 and wrote my characters.
  Now I modify the port settings the same way I had to do already for the USBIO card and then it works.
  This is my added piece of code:

```
fcntl(fd, F_SETFL, 0);

 tcgetattr(fd, &options);

cfsetispeed(&options, B9600);
cfsetospeed(&options, B9600);

options.c_cflag |= (CLOCAL | CREAD);

options.c_cflag &= ~PARENB;
options.c_cflag &= ~CSTOPB;
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS8;

options.c_cflag &= ~CRTSCTS;

options.c_oflag &= ~OPOST;

tcsetattr(fd, TCSANOW, &options);
```

  The important line is possibly: options.c_oflag &= ~OPOST; This enables raw output.

  You saved my day, thank's a lot!