How to use the parallel and I/O port peek poke driver for Windows NT

Introduction

This package of an NT device driver, driver loader utility and a library of functions allows you a direct access PC's parallel and other I/O port. This file describes the steps necessary to set up your PC for this task. This package is supplied as is with no support or maintenance provided. The user is expected to have a good working knowledge of how to configure a PC NT workstation. Further it is expected that the user have the knowledge of PC printer and I/O, and how they are used. Please do not use this package if you are not sure what you are doing. Accessing a wrong I/O port can result in an unrecoverable system crash or hardware damage. Windows NT 4.0 and 3.51 are supported.
I would like to emphasize that this driver is of very simple design. It is primarily intended for those who would like to prototype and experiment with devices that can be connected to the parallel ports. It is not intended for production software/hardware.

The package

The self-extracting archive (pp0000.exe) contains the following.
Ppppdvr.sys: NT kernel driver that talks to PC's I/O port space.
Dvrload.exe: interactive driver loader that loads the NT kernel driver.
Ppppapi.dll: Dynamic link library that contains functions that you can call from your application to talk to the driver and therefore to PC's I/O ports.
Ppppapi.lib: To link Ppppapi.dll into you C/C++ applications at link time.
Veeppapi.h: Header file to use when you are importing ppppapi.dll to VEE.
Cppapi.h: header file to include in your C/C++ applications for dynamic linking.
Apicdecl.h: Header file to include in your C/C++ applications for compile-time linking.
D2Acurve.VEE, BlockOut.VEE: example VEE programs.
Sine1, sine2, sine4, sine8, sine16: data files to go with BlockOut.VEE.
d2adev.bmp: schematic of a very simple D to A converter.
Readme.doc: This file.

Installation

1. Copy pp0000.exe to a temporary file and execute it. It will extract the components listed above.
2. Copy the files to the VEE install directory if you have install VEE. Otherwise you can copy it to any local (not on another PC on LAN) directory. However, I would not recommend installing it in the Windows System32 directory where there is a possibility of a name collision.
3. To install the ppppdvr.sys kernel driver run the dvrload.exe program. This program works only in Windows NT and you must have the system administrator privilege to install this kernel driver. It will look for the driver file first in the VEE install directory. If it's not there you need to press [Browse.] to search for it. You specify the directory where you copied the driver to. After the driver file has been located press [Install] to

install the driver. You can use the dvrload program to uninstall the driver, also. You don't have to reboot your machine to use this driver.

4. From VEE Device->Function->Import Library obtain an Import Library object. Select Compiled function for Function Type. Select the path names of ppppapi.dll for File name and veeppapi.h for Definition File. To compile C/C++ programs include cppapi.h file in your source file.
5. If you are peeking and poking to LPT1 or LPT2 port before you run your program unconfigure it. NT pings LPT ports even if there is no printer attached as long as LPT port(s) are set up as printer ports. This will cause the driver call to access the parallel port to fail unpredictably.
6. If multiple processes access the driver the outcome is not predictable.

Suggestion on use of LPT ports.
If you are going to use LPT port(s) to test your own piece of equipment I highly recommend that you buy a plug-in parallel printer card. You can buy a regular one for less than $10 and an enhanced card for less than $20. It is certainly worth it. If something goes wrong you can throw it away but you can't do the same with the built-in parallel port.

Notes on the ppppdvr NT device driver

After you install this driver with the dvrload utility the driver is up and running. It is NOT necessary to reboot your NT workstation. When you are done with the driver you can run the same utility to unload it.

If you reboot your NT workstation without uninstalling the driver it is not active when NT comes up next. You need to manually start it to use the driver. To start the driver select Devices in the Control Panel. Look for ppppdvr entry in the Device list and highlight it. It says "manual" in the Startup column. If the value in the Status column is blank, push the [Start] button to start the driver. Now you are ready to access the driver and also you can uninstall it by the dvrload utility.

If nothing seems to work, i.e., you can't install or uninstall the driver, or access it from your program, you can take a more drastic action. Run the regedit program and go to HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\ppppdvr and delete the ppppdvr subtree. Make sure you don't delete any other subtree by mistake. The reboot the machine and reinstall the driver.

General information on ppppapi.dll

All peek/poke functions take as the first two parameters an index to a table and offset. This dll keeps a table of I/O addresses. The caller to the peek/poke functions passes an index to this table and the DLL does a table look up to come up with an I/O port base address. The first two table entries (indeces 0 and 1) are prefilled with LPT1 and LPT2 port addresses. For example if you make a call to
Poke8(0, 0, 0xff, &myStatus);

You are sending data value 0xff to LPT1 port, i.e., 0x378. And if you make a call to
Inval = Peek8(1, 1, &myStatus);
You are reading LPT2's status port. LPT2's base address is 0x278 and you are adding offset of 1, therefore you are reading I/O port 0x379. The value of offset can range from 0 to 15. You can peek/poke up to consecutive 16 I/O ports from the base address.

The mapping table size is 20 and you can add or change (including the first two entries) mapping using the mapDeviceToPort function. For example,

mapDeviceToPort(2, 0x478, 8);

Will add a third entry in the table and Will enable you to access I/O ports 0x478 and consecutive 8 ports using the index value 2. So after this call you can do:

Poke8(2, 2, 0x0f, &myStatus);

To write a value 0xf to I/O port 0x47A (==base address (0x478) + offset(2)). You must exercise extreme caution when you extend the I/O address mapping table. You must be sure you know the I/O ports you are accessing. Accessing wrong I/O ports can result in a system crash or hardware damage.

The getPortAddress(int portNum) function returns the base address of a mapped port. In the above example,

getPortAddress(2);

will return 0x478.

Some functions take a delay parameter (time in milliseconds between consecutive peeks or pokes). Even though the unit of time is milliseconds probably you get a minimum resolution of about 50 milliseconds. You can get a better resolution by peeking/poking one data item at a time and inserting your own delay.
Some functions take an invertmask parameter. Output data are bit-wise XOR'ed before written to the port and input data are bit-wise XOR'ed before stored in the input buffer. If you don't want any bit of data to be inverted supply the value of 0.
Some block poke functions take an repeat count parameter. Output buffer is repeatedly output to the port. These functions are provided so that you can, for example, output digital wave forms to a D to A converter to be able to construct an inexpensive function generator. Don't put a large repeat count until you know how long it takes to output the buffer contents once. Your computer may seem to hang if your buffer is large and/or delay time is long and/or the repeat count is large.

Exported functions in the Ppppapi.dll
int mapDeviceToPort(int portNum, USHORT baseAddr, int portRange);
Explained above.
USHORT getPortAddress(int portNum);
Explained above.
In all following functions the last parameter is a pointer to an int variable where status of the function call is returned. If the function completes successfully its value is 0 otherwise non-zero.

USHORT Peek8(USHORT portIndex, USHORT offset, int *status);
USHORT Peek16(USHORT portIndex, USHORT offset, int *status);
ULONG Peek32(USHORT portIndex, USHORT offset, int *status);

Theses are peek functions that peek 1-byte, 2-byte or 4-byte wide ports starting with the base address pointed to in the table plus offset.

int Poke8(USHORT portIndex, USHORT offset, USHORT data, int *status);
int Poke16(USHORT portIndex, USHORT offset, USHORT data, int *status);
int Poke32(USHORT portIndex, USHORT offset, ULONG data, int *status);

Theses are poke functions that poke the data value into the 1-byte, 2-byte or 4-byte wide port starting with the base address pointed to in the table plus offset.

USHORT Peek8M(USHORT portIndex, USHORT offset, USHORT invertmask, int *status);
USHORT Peek16M(USHORT portIndex, USHORT offset, USHORT invertmask, int *status);
ULONG Peek32M(USHORT portIndex, USHORT offset, ULONG invertmask, int *status);

Theses are enhanced peek functions that peek a value from the specified port and return a value after XOR'ing it with the invertmask. Therefore the returned value has been bit-wise inverted by the invertmask parameter .

int Poke8M(USHORT portIndex, USHORT offset, USHORT data, USHORT invertmask, int *status);

int Poke16M(USHORT portIndex, USHORT offset, USHORT data, USHORT invertmask, int *status);

int Poke32M(USHORT portIndex, USHORT offset, ULONG data, ULONG invertmask, int *status);

Theses are enhanced poke functions that poke the specified value to the specified port after XOR'ing it with the invertmask. Therefore the value written to the port has been bit-wise inverted by the invertmask parameter.

USHORT PeekPoke8(USHORT portIndex, USHORT offset, USHORT data, USHORT mask, USHORT invert, int *status);
USHORT PeekPoke16(USHORT portIndex, USHORT offset, USHORT data, USHORT mask, USHORT invert, int *status);
ULONG PeekPoke32(USHORT portIndex, USHORT offset, ULONG data, ULONG mask, ULONG invert, int *status);

These functions perform poke after peek. They first peek a value from the specified port and mask out the unneeded bits using the mask value and OR's it with data XOR'ed with the invertmask then poke the resulting value back to the same port. If you don't want any bits to be masked out you must specify 0xff, 0xfffff or 0xffffffff depending on the port size as the mask value. If you don't want any output bits to be inverted specify 0 as the invertmask.

ULONG BlockPeek8(USHORT portIndex, USHORT offset, UCHAR *buffer, ULONG numBytes, int *status);
ULONG BlockPeek16(USHORT portIndex, USHORT offset, USHORT *buffer, ULONG numBytes, int *status);
ULONG BlockPeek32(USHORT portIndex, USHORT offset, ULONG *buffer, ULONG numBytes, int *status);

These functions perform burst mode peek of I/O port. You specify the buffer address and the number of bytes you want to read. The buffer must be of adequate size to read that many bytes. The return value is number of bytes read.

ULONG BlockPeek8L(USHORT portIndex, USHORT offset, ULONG *buffer, ULONG itemCount, int *status);
ULONG BlockPeek16L(USHORT portIndex, USHORT offset, ULONG *buffer, ULONG itemCount, int *status);
ULONG BlockPeek32L(USHORT portIndex, USHORT offset, ULONG *buffer, ULONG itemCount, int *status);

These functions are similar to plain blockpeek except that the data buffer is a pointer to an array of longs and that you specify the size of the array not in bytes but how many data items you want to read in. So regardless of the width of the port for a given item count the required buffer size is the same. These functions are provided for programming languages that do not handle byte arrays in external functions such as HP VEE. Note that these functions do not return compacted data, i.e., they don't pack four 8-bit values into a single long array element. These functions expand to long the values read regardless of the width of the port.

ULONG BlockPoke8(USHORT portIndex, USHORT offset, UCHAR *buffer, ULONG numBytes, int *status);
ULONG BlockPoke16(USHORT portIndex, USHORT offset, USHORT *buffer, ULONG numBytes, int *status);
ULONG BlockPoke32(USHORT portIndex, USHORT offset, ULONG *buffer, ULONG numBytes, int *status);

These functions perform burst mode poke of I/O port. You specify the buffer address and the number of bytes you want to write. The buffer must be of adequate size to contain the specified number of bytes. The return value is number of bytes written.

ULONG BlockPoke8L(USHORT portIndex, USHORT offset, ULONG *buffer, ULONG itemCount, int *status);
ULONG BlockPoke16L(USHORT portIndex, USHORT offset, ULONG *buffer, ULONG itemCount, int *status);
ULONG BlockPoke32L(USHORT portIndex, USHORT offset, ULONG *buffer, ULONG itemCount, int *status);

These are the L(ong) versions of BlockPoke. Note again the data buffer points to an array of longs and you specify how may data points you want to output and not the number of bytes. Regardless of the width of the port these function convert one long data into appropriate size. Note that these functions do not expect compacted data array, i.e., do not pack four 8-bit values into one long array element.

ULONG BlockPeek8X(USHORT portIndex, USHORT offset, UCHAR *buffer, ULONG numBytes, ULONG delay, int *status);
ULONG BlockPeek16X(USHORT portIndex, USHORT offset, USHORT *buffer, ULONG numBytes, ULONG delay, int *status);
ULONG BlockPeek32X(USHORT portIndex, USHORT offset, ULONG *buffer, ULONG numBytes, ULONG delay, int *status);

These functions are similar to BlockPeek. But the X(tra) versions take a delay parameter. These functions wait delay milliseconds between successive reads of data.

ULONG BlockPeek8XM(USHORT portIndex, USHORT offset, UCHAR *buffer, ULONG numBytes, ULONG delay, USHORT invertmask,int *status);
ULONG BlockPeek16XM(USHORT portIndex, USHORT offset, USHORT *buffer, ULONG numBytes, ULONG delay, USHORT invertmask, int *status);
ULONG BlockPeek32XM(USHORT portIndex, USHORT offset, ULONG *buffer, ULONG numBytes, ULONG delay, ULONG invertmask, int *status);

These functions are similar to BlockPeekX. But the M(ask) versions take an invertmask parameter. After each data item is read it is bit-wise XOR'ed before stored in the input buffer.

ULONG BlockPeek8XL(USHORT portIndex, USHORT offset, ULONG *buffer,ULONG itemCount,ULONG delay, int *status);
ULONG BlockPeek16XL(USHORT portIndex, USHORT offset, ULONG *buffer,ULONG itemCount,ULONG delay, int *status);
ULONG BlockPeek32XL(USHORT portIndex, USHORT offset, ULONG *buffer,ULONG itemCount,ULONG delay, int *status);

These are the L(ong) version of BlockPeekX. Regardless of the width of the I/O port the input data is stored in the array of longs unpacked.

ULONG BlockPeek8XLM(USHORT portIndex, USHORT offset, ULONG *buffer,

ULONG itemCount, ULONG delay, USHORT invert, int *status);
ULONG BlockPeek16XLM(USHORT portIndex, USHORT offset, ULONG *buffer,
ULONG itemCount, ULONG delay, USHORT invert, int *status);
ULONG BlockPeek32XLM(USHORT portIndex, USHORT offset, ULONG *buffer,
ULONG itemCount, ULONG delay, ULONG invert, int *status);

These are the M(mask) version of BlockPeekXL. Each item of data is XOR'ed with the mask value before stored in the array of longs unpacked.

ULONG BlockPoke8X(USHORT portIndex, USHORT offset, UCHAR *buffer,
ULONG numBytes, ULONG rptCount, ULONG delay, int *status);
ULONG BlockPoke16X(USHORT portIndex, USHORT offset, USHORT *buffer,
ULONG numBytes, ULONG rptCount, ULONG delay, int *status);
ULONG BlockPoke32X(USHORT portIndex, USHORT offset, ULONG *buffer,
ULONG numBytes, ULONG rptCount, ULONG delay, int *status);

These are similar to BlockPoke except that they take extra two parameters, rptCount and delay. They output the entire array of data repeatedly rptCount times with delay milliseconds between consecutive data items.

ULONG BlockPoke8XM(USHORT portIndex, USHORT offset, UCHAR *buffer,
ULONG numBytes, ULONG rptCount, ULONG delay, USHORT invertmask, int
*status);
ULONG BlockPoke16XM(USHORT portIndex, USHORT offset, USHORT *buffer,
ULONG numBytes, ULONG rptCount, ULONG delay, USHORT invertmask, int
*status);
ULONG BlockPoke32XM(USHORT portIndex, USHORT offset, ULONG *buffer,
ULONG numBytes, ULONG rptCount, ULONG delay, ULONG invertmask, int *status);

These are the M(ask) versions of BlockPokeX. The output data are XOR'ed with the invertmask before output to the I/O port.

ULONG BlockPoke8XL(USHORT portIndex, USHORT offset, ULONG *buffer,
ULONG itemCount, ULONG rptCount, ULONG delay, int *status);
ULONG BlockPoke16XL(USHORT portIndex, USHORT offset, ULONG *buffer,
ULONG itemCount, ULONG rptCount, ULONG delay, int *status);
ULONG BlockPoke32XL(USHORT portIndex, USHORT offset,ULONG *buffer,
ULONG itemCount, ULONG rptCount, ULONG delay, int *status);

These are the L(ong) versions of BlockPokeX. Each element of the long array is output as an 8-bit, 16-bit or 32-bit value.

ULONG BlockPoke8XLM(USHORT portIndex, USHORT offset, ULONG *buffer,
ULONG itemCount, ULONG rptCount, ULONG delay, USHORT invert, int *status);
ULONG BlockPoke16XLM(USHORT portIndex, USHORT offset, ULONG *buffer,

ULONG itemCount, ULONG rptCount, ULONG delay, USHORT invert, int *status);
ULONG BlockPoke32XLM(USHORT portIndex, USHORT offset, ULONG *buffer,
ULONG itemCount, ULONG rptCount, ULONG delay, ULONG invert, int *status);

These are the M(ask) versions of BlockPokeXL. Each element of the long array is output after bit-wise XOR'ed with the invert mask as an 8-bit, 16-bit or 32-bit value.