

Interfacing the Standard Parallel Port

Disclaimer : While every effort has been made to make sure the information in this document is correct, the author can not be liable for any damages whatsoever for loss relating to this document. Use this information at your own risk.

Table of Contents

Introduction to Parallel Ports	Page 1
Hardware Properties	Page 2
Centronics?	Page 4
Port Addresses	Page 4
Software Registers - Standard Parallel Port (SPP)	Page 6
Bi-directional Ports	Page 8
Using The Parallel Port to Input 8 Bits.	Page 9
Nibble Mode	Page 11
Using the Parallel Port's IRQ	Page 12
Parallel Port Modes in BIOS	Page 14
Parallel Port Modes and the ECP's Extended Control Register	Page 15

Introduction to Parallel Ports

The Parallel Port is the most commonly used port for interfacing home made projects. This port will allow the input of up to 9 bits or the output of 12 bits at any one given time, thus requiring minimal external circuitry to implement many simpler tasks. The port is composed of 4 control lines, 5 status lines and 8 data lines. It's found commonly on the back of your PC as a D-Type 25 Pin female connector. There may also be a D-Type 25 pin male connector. This will be a serial RS-232 port and thus, is a totally incompatible port.

Newer Parallel Port's are standardized under the IEEE 1284 standard first released in 1994. This standard defines 5 modes of operation which are as follows,

1. Compatibility Mode.
2. Nibble Mode. (*Protocol not Described in this Document*)
3. Byte Mode. (*Protocol not Described in this Document*)
4. EPP Mode (*Enhanced Parallel Port*).
5. ECP Mode (*Extended Capabilities Port*).

The aim was to design new drivers and devices which were compatible with each other and

also backwards compatible with the Standard Parallel Port (SPP). Compatibility, Nibble & Byte modes use just the standard hardware available on the original Parallel Port cards while EPP & ECP modes require additional hardware which can run at faster speeds, while still being downwards compatible with the Standard Parallel Port.

Compatibility mode or "Centronics Mode" as it is commonly known, can only send data in the forward direction at a typical speed of 50 kbytes per second but can be as high as 150+ kbytes a second. In order to receive data, you must change the mode to either Nibble or Byte mode. Nibble mode can input a nibble (*4 bits*) in the reverse direction. E.g. from device to computer. Byte mode uses the Parallel's bi-directional feature (*found only on some cards*) to input a byte (*8 bits*) of data in the reverse direction.

Extended and Enhanced Parallel Ports use additional hardware to generate and manage handshaking. To output a byte to a printer (or anything in that matter) using compatibility mode, the software must.

1. *Write the byte to the Data Port.*
2. *Check to see if the printer is busy. If the printer is busy, it will not accept any data, thus any data which is written will be lost.*
3. *Take the Strobe (Pin 1) low. This tells the printer that there is the correct data on the data lines. (Pins 2-9)*
4. *Put the strobe high again after waiting approximately 5 microseconds after putting the strobe low. (Step 3)*

This limits the speed at which the port can run at. The EPP & ECP ports get around this by letting the hardware check to see if the printer is busy and generate a strobe and /or appropriate handshaking. This means only one I/O instruction need to be performed, thus increasing the speed. These ports can output at around 1-2 megabytes per second. The ECP port also has the advantage of using DMA channels and FIFO buffers, thus data can be shifted around without using I/O instructions.

Hardware Properties

On the next page is a table of the "Pin Outs" of the D-Type 25 Pin connector and the Centronics 34 Pin connector. The D-Type 25 pin connector is the most common connector found on the Parallel Port of the computer, while the Centronics Connector is commonly found on printers. The IEEE 1284 standard however specifies 3 different connectors for use with the Parallel Port. The first one, 1284 Type A is the D-Type 25 connector found on the back of most computers. The 2nd is the 1284 Type B which is the 36 pin Centronics Connector found on most printers.

IEEE 1284 Type C however, is a 36 conductor connector like the Centronics, but smaller. This connector is claimed to have a better clip latch, better electrical properties and is easier to assemble. It also contains two more pins for signals which can be used to see whether the other device connected,

has power. 1284 Type C connectors are recommended for new designs, so we can look forward on seeing these new connectors in the near future.

Pin No (D-Type 25)	Pin No (Centronics)	SPP Signal	Direction In/out	Register	Hardware Inverted
1	1	nStrobe	In/Out	Control	Yes
2	2	Data 0	Out	Data	
3	3	Data 1	Out	Data	
4	4	Data 2	Out	Data	
5	5	Data 3	Out	Data	
6	6	Data 4	Out	Data	
7	7	Data 5	Out	Data	
8	8	Data 6	Out	Data	
9	9	Data 7	Out	Data	
10	10	nAck	In	Status	
11	11	Busy	In	Status	Yes
12	12	Paper-Out PaperEnd	In	Status	
13	13	Select	In	Status	
14	14	nAuto-Linefeed	In/Out	Control	Yes
15	32	nError / nFault	In	Status	
16	31	nInitialize	In/Out	Control	
17	36	nSelect-Printer nSelect-In	In/Out	Control	Yes
18 - 25	19-30	Ground	Gnd		

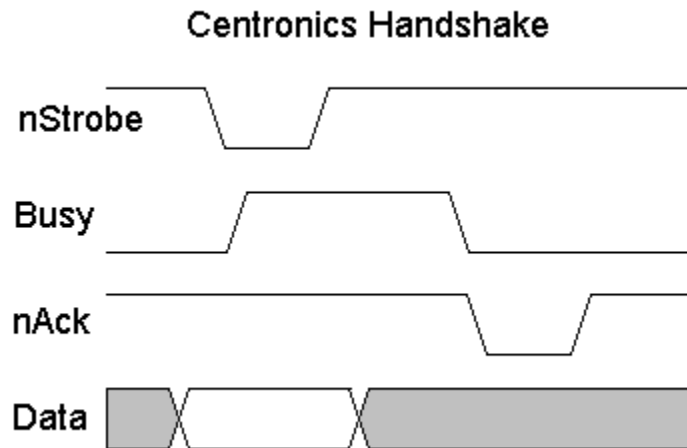
Table 1. Pin Assignments of the D-Type 25 pin Parallel Port Connector.

The above table uses "n" in front of the signal name to denote that the signal is active low. e.g. nError. If the printer has occurred an error then this line is low. This line normally is high, should the printer be functioning correctly. The "Hardware Inverted" means the signal is inverted by the Parallel card's hardware. Such an example is the Busy line. If +5v (Logic 1) was applied to this pin and the status register read, it would return back a 0 in Bit 7 of the Status Register.

The output of the Parallel Port is normally TTL logic levels. The voltage levels are the easy part. The current you can sink and source varies from port to port. Most Parallel Ports implemented in ASIC, can sink and source around 12mA. However these are just some of the figures taken from Data sheets, Sink/Source 6mA, Source 12mA/Sink 20mA, Sink 16mA/Source 4mA, Sink/Source 12mA. As you can see they vary quite a bit. The best bet is to use a buffer, so the least current is drawn from the Parallel Port.

Centronics?

Centronics is an early standard for transferring data from a host to the printer. The majority of printers use this handshake. This handshake is normally implemented using a Standard Parallel Port under software control. Below is a simplified diagram of the 'Centronics' Protocol.



Data is first applied on the Parallel Port pins 2 to 7. The host then checks to see if the printer is busy. i.e. the busy line should be low. The program then asserts the strobe, waits a minimum of $1\mu\text{S}$, and then de-asserts the strobe. Data is normally read by the printer/peripheral on the rising edge of the strobe. The printer will indicate that it is busy processing data via the Busy line. Once the printer has accepted data, it will acknowledge the byte by a negative pulse about $5\mu\text{S}$ on the nAck line.

Quite often the host will ignore the nAck line to save time. Latter in the Extended Capabilities Port, you will see a Fast Centronics Mode, which lets the hardware do all the handshaking for you. All the programmer must do is write the byte of data to the I/O port. The hardware will check to see if the printer is busy, generate the strobe. Note that this mode commonly doesn't check the nAck either.

Port Addresses

The Parallel Port has three commonly used base addresses. These are listed in table 2, below. The 3BCh base address was originally introduced used for Parallel Ports on early Video Cards. This address then disappeared for a while, when Parallel Ports were later removed from Video Cards. They has now reappeared as an option for Parallel Ports integrated onto motherboards, upon which their configuration can be changed using BIOS.

LPT1 is normally assigned base address 378h, while LPT2 is assigned 278h. However this may not always be the case as explained later. 378h & 278h have always been commonly used for Parallel Ports. The lower case h denotes that it is in hexadecimal. These addresses may change from machine to machine.

Address	Notes:
3BCh - 3BFh	Used for Parallel Ports which were incorporated in to Video Cards and now, commonly an option for Ports controlled by BIOS. - Doesn't support ECP addresses.
378h - 37Fh	Usual Address For LPT 1
278h - 27Fh	Usual Address For LPT 2

Table 2 Port Addresses

When the computer is first turned on, BIOS (Basic Input/Output System) will determine the number of ports you have and assign device labels LPT1, LPT2 & LPT3 to them. BIOS first looks at address 3BCh. If a Parallel Port is found here, it is assigned as LPT1, then it searches at location 378h. If a Parallel card is found there, it is assigned the next free device label. This would be LPT1 if a card wasn't found at 3BCh or LPT2 if a card was found at 3BCh. The last *port of call*, is 278h and follows the same procedure than the other two ports. Therefore it is possible to have a LPT2 which is at 378h and not at the expected address 278h.

What can make this even confusing, is that some manufacturers of Parallel Port Cards, have jumpers which allow you to set your Port to LPT1, LPT2, LPT3. Now what address is LPT1? - On the majority of cards LPT1 is 378h, and LPT2, 278h, but some will use 3BCh as LPT1, 378h as LPT1 and 278h as LPT2. *Life wasn't meant to be easy.*

The assigned devices LPT1, LPT2 & LPT3 should not be a worry to people wishing to interface devices to their PC's. Most of the time the base address is used to interface the port rather than LPT1 etc. However should you want to find the address of LPT1 or any of the **Line PrinTer** Devices, you can use a lookup table provided by BIOS. When BIOS assigns addresses to your printer devices, it stores the address at specific locations in memory, so we can find them.

Start Address	Function
0000:0408	LPT1's Base Address
0000:040A	LPT2's Base Address
0000:040C	LPT3's Base Address
0000:040E	LPT4's Base Address (Note 1)

Table 3 - LPT Addresses in the BIOS Data Area

Note 1 : Address 0000:040E in the BIOS Data Area may be used as the Extended Bios Data Area in PS/2 and newer Bioses, and thus this field may be invalid.

The above table, table 3, shows the address at which we can find the Printer Port's addresses in the BIOS Data Area. Each address will take up 2 bytes. The following sample program in C, shows how you can read these locations to obtain the addresses of your printer ports.

```
#include <stdio.h>
#include <dos.h>

void main(void)
{
    unsigned int far *ptraddr; /* Pointer to location of Port Addresses */
    unsigned int address;      /* Address of Port */
    int a;

    ptraddr=(unsigned int far *)0x00000408;

    for (a = 0; a < 3; a++)
    {
        address = *ptraddr;
        if (address == 0)
            printf("No port found for LPT%d \n",a+1);
        else
            printf("Address assigned to LPT%d is %Xh\n",a+1,address);
        *ptraddr++;
    }
}
```

Software Registers - Standard Parallel Port (SPP)

Offset	Name	Read/Write	Bit No.	Properties
Base + 0	Data Port	Write (Note-1)	Bit 7	Data 7 (Pin 9)
			Bit 6	Data 6 (Pin 8)
			Bit 5	Data 5 (Pin 7)
			Bit 4	Data 4 (Pin 6)
			Bit 3	Data 3 (Pin 5)
			Bit 2	Data 2 (Pin 4)
			Bit 1	Data 1 (Pin 3)
			Bit 0	Data 0 (Pin 2)

Table 4 Data Port

Note 1 : If the Port is bi-directional then Read and Write Operations can be performed on the Data Register.

The base address, usually called the Data Port or Data Register is simply used for outputting data on the Parallel Port's data lines (Pins 2-9). This register is normally a write only port. If you read from the port, you should get the last byte sent. However if your port is bi-directional, you can receive data on this address. See *Bi-directional Ports* for more detail.

Base + 1	Status Port	Read Only	Bit 7	Busy
			Bit 6	Ack
			Bit 5	Paper Out
			Bit 4	Select In
			Bit 3	Error
			Bit 2	IRQ (Not)
			Bit 1	Reserved
			Bit 0	Reserved

Table 5 Status Port

The Status Port (base address + 1) is a read only port. Any data written to this port will be ignored. The Status Port is made up of 5 input lines (Pins 10,11,12,13 & 15), a IRQ status register and two reserved bits. Please note that Bit 7 (Busy) is a active low input. E.g. If bit 7 happens to show a logic 0, this means that there is +5v at pin 11. Likewise with Bit 2. (nIRQ) If this bit shows a '1' then an interrupt has **not** occurred.

Base + 2	Control Port	Read/Write	Bit 7	Unused
			Bit 6	Unused
			Bit 5	Enable bi-directional Port
			Bit 4	Enable IRQ Via Ack Line
			Bit 3	Select Printer
			Bit 2	Initialize Printer (Reset)
			Bit 1	Auto Linefeed
			Bit 0	Strobe

Table 6 Control Port

The Control Port (base address + 2) was intended as a write only port. When a printer is attached to the Parallel Port, four "controls" are used. These are Strobe, Auto Linefeed, Initialize and Select Printer, all of which are inverted except Initialize.

The printer would not send a signal to initialize the computer, nor would it tell the computer to use auto linefeed. However these four outputs can also be used for inputs. If the computer has placed a pin high (e.g. +5v) and your device wanted to take it low, you would effectively short out the port, causing a conflict on that pin. Therefore these lines are "open collector" outputs (*or open drain for CMOS devices*). This means that it has two states. A low state (0v) and a high impedance state (open circuit).

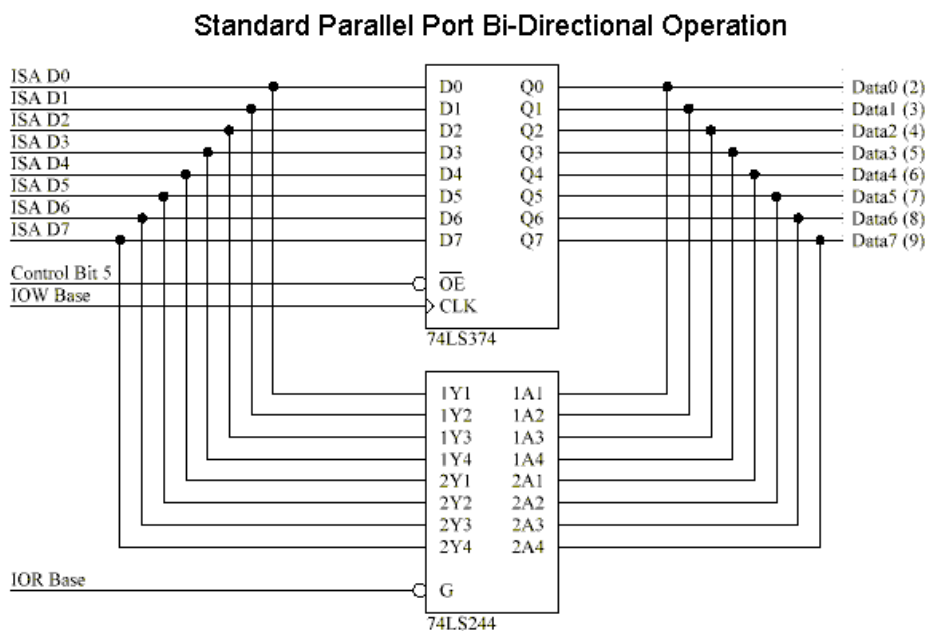
Normally the Printer Card will have internal pull-up resistors, but as you would expect, not all will. Some may just have open collector outputs, while others may even have normal totem pole outputs. In order to make your device work correctly on as many Printer Ports as possible, you can use an external resistor as well. Should you already have an internal resistor, then it will act in Parallel with it, or if you have Totem pole outputs, the resistor will act as a load.

An external 4.7k resistor can be used to pull the pin high. I wouldn't use anything lower, just in case you do have an internal pull up resistor, as the external resistor would act in parallel giving effectively, a lower value pull up resistor. When in high impedance state the pin on the Parallel Port is high (+5v). When in this state, your external device can pull the pin low and have the control port change read a different value. This way the 4 pins of the Control Port can be used for bi-directional data transfer. However the Control Port must be set to xxxx0100 to be able to read data, that is all pins to be +5v at the port so that you can pull it down to GND (logic 0).

Bits 4 & 5 are internal controls. Bit four will enable the IRQ (*See Using the Parallel Ports IRQ*) and Bit 5 will enable the bi-directional port meaning that you can input 8 bits using (DATA0-7). This mode is only possible if your card supports it. Bits 6 & 7 are reserved. Any writes to these two bits will be ignored.

Bi-directional Ports

The schematic diagram below, shows a simplified view of the Parallel Port's Data Register. The original Parallel Port card's implemented 74LS logic. These days all this is crammed into one ASIC, but the theory of operation is still the same.



The non bi-directional ports were manufactured with the 74LS374's output enable tied permanent low, thus the data port is always output only. When you read the Parallel Port's data register, the data comes from the 74LS374 which is also connected to the data pins. Now if you can overdrive the '374 you can effectively have a Bi-directional Port. (*or a input only port, once you blow up the latches output!*)

What is very concerning is that people have actually done this. I've seen one circuit, a scope connected to the Parallel Port distributed on the Internet. The author uses an ADC of some type, but finds the ADC requires transistors on each data line, to make it work! No wonder why. Others have had similar trouble, the 68HC11 cannot sink enough current (30 to 40mA!)

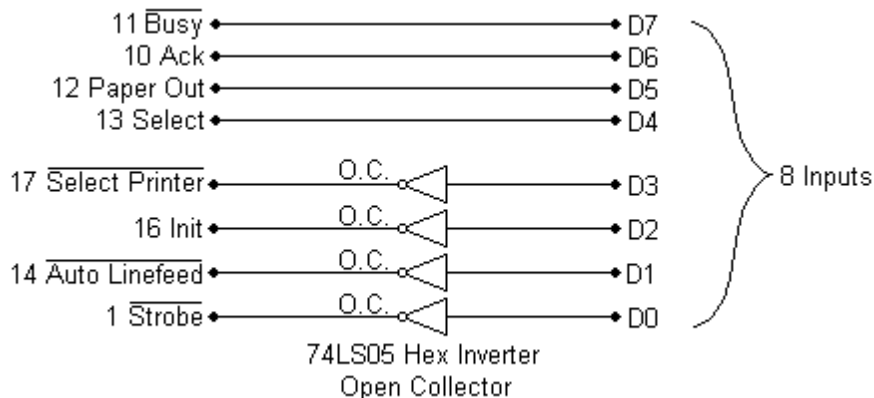
Bi-directional ports use Control Bit 5 connected to the 374's OE so that it's output drivers can be turned off. This way you can read data present on the Parallel Port's Data Pins, without having bus conflicts and excessive current drains.

Bit 5 of the Control Port enables or disables the bi-directional function of the Parallel Port. This is only available on true bi-directional ports. When this bit is set to one, pins 2 to 9 go into high impedance state. Once in this state you can enter data on these lines and retrieve it from the Data Port (base address). Any data which is written to the data port will be stored but will not be available at the data pins. To turn off bi-directional mode, set bit 5 of the Control Port to '0'.

However not all ports behave in the same way. Other ports may require setting bit 6 of the Control Port to enable Bi-directional mode and setting of Bit 5 to dis-enable Bi-directional mode, Different manufacturers implement their bi-directional ports in different ways. If you wish to use your Bi-directional port to input data, test it with a logic probe or multimeter first to make sure it is in bi-directional mode.

Using The Parallel Port to Input 8 Bits.

If your Parallel Port doesn't support bi-directional mode, don't despair. You can input a maximum of 9 bits at any one given time. To do this you can use the 5 input lines of the Status Port and the 4 inputs (open collector) lines of the Control Port.



The inputs to the Parallel Port has be chosen as such, to make life easier for us. Busy just happens to be the MSB (Bit 7) of the Status Port, then in ascending order comes Ack, Paper Out and Select, making up the most significant nibble of the Control Port. The Bars are used to represent which inputs are Hardware inverted, i.e. +5v will read 0 from the register, while GND will read 1. The Status Port only has one inverted input.

The Control port is used to read the least significant nibble. As described before, the control port has open collector outputs, i.e. two possible states, high impedance and GND. If we connect our inputs directly to the port (*For example an ADC0804 with totem pole outputs*), a conflict will result if the input is high and the port is trying to pull it down. Therefore we use open collector inverters.

However this is not always entirely necessary. If we were connecting single pole switches to the port with a pull up resistor, then there is no need to bother with this protection. Also if your software initializes the control port with xxxx0100 so that all the pins on the control port are high,

then it may be unnecessary. If however you don't bother and your device is connected to the Parallel Port before your software has a chance to initialize then you may encounter problems.

Another problem to be aware of is the pull up resistors on the control port. The average pull-up resistor is 4.7k. In order to pull the line low, your device will need to sink 1mA, which some low powered devices may struggle to do. Now what happens if I suggest that some ports have 1K pull up resistors? Yes, there are such cards. Your device now has to sink 5mA. More reason to use the open collector inverters.

Open collector inverters were chosen over open collector buffers as they are more popular, and thus easier to obtain. There is no reason, however why you can't use them. Another possibility is to use transistors.

The input, D3 is connected via the inverter to Select Printer. Select Printer just happens to be bit 3 of the control port. D2, D1 & D0 are connected to Init, Auto linefeed and strobe, respectively to make up the lower nibble. Now this is done, all we have to do is assemble the byte using software. The first thing we must do is to write xxxx0100 to the Control Port. This places all the control port lines high, so they can be pulled down to input data.

```
outportb(CONTROL, inportb(CONTROL) & 0xF0 | 0x04);
```

Now that this is done, we can read the most significant nibble. This just happens to be the most significant nibble of the status port. As we are only interested in the MSnibble we will AND the results with 0xF0, so that the LSnibble is clear. Busy is hardware inverted, but we won't worry about it now. Once the two bytes are constructed, we can kill two birds with one stone by toggling Busy and Init at the same time.

```
a = (inportb(STATUS) & 0xF0); /* Read MSnibble */
```

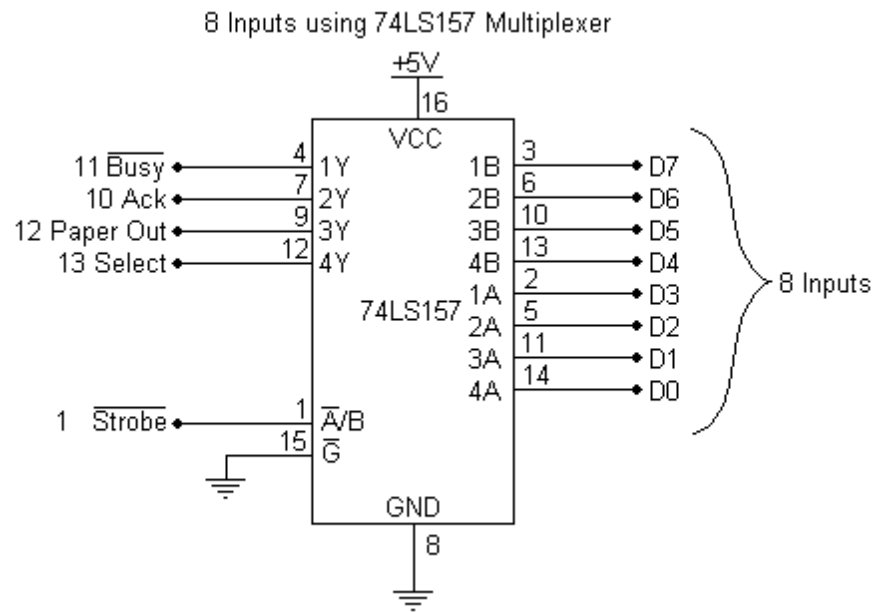
We can now read the LSnibble. This just happens to be LSnibble of the control port - How convenient! This time we are not interested with the MSnibble of the port, thus we AND the result with 0x0F to clear the MSnibble. Once this is done, it is time to combine the two bytes together. This is done by OR'ing the two bytes. This now leaves us with one byte, however we are not finished yet. Bits 2 and 7 are inverted. This is overcome by XOR'ing the byte with 0x84, which toggles the two bits.

```
a = a |(inportb(CONTROL) & 0x0F); /* Read LSnibble */  
a = a ^ 0x84; /* Toggle Bit 2 & 7 */
```

Note: Some control ports are not open collector, but have totem pole outputs. This is also the case with EPP and ECP Ports. Normally when you place a Parallel Port in ECP or EPP mode, the control port becomes totem pole outputs only. Now what happens if you connect your device to the Parallel Port in this mode? Therefore, in the interest of portability I recommend using the next circuit, reading a nibble at a time.

Nibble Mode

Nibble mode is the preferred way of reading 8 bits of data without placing the port in reverse mode and using the data lines. Nibble mode uses a Quad 2 line to 1 line multiplexer to read a nibble of data at a time. Then it “switches” to the other nibble and reads its. Software can then be used to construct the two nibbles into a byte. The only disadvantage of this technique is that it is slower. It now requires a few I/O instructions to read the one byte, and it requires the use of an external IC.



The operation of the 74LS157, Quad 2 line to 1 line multiplexer is quite simple. It simply acts as four switches. When the A/B input is low, the A inputs are selected. E.g. 1A passes through to 1Y, 2A passes through to 2Y etc. When the A/B is high, the B inputs are selected. The Y outputs are connected up to the Parallel Port’s status port, in such a manner that it represents the MSnibble of the status register. While this is not necessary, it makes the software easier.

To use this circuit, first we must initialize the multiplexer to switch either inputs A or B. We will read the LSnibble first, thus we must place A/B low. The strobe is hardware inverted, thus we must set Bit 0 of the control port to get a low on Pin 1.

```
outportb(CONTROL, inportb(CONTROL) | 0x01); /* Select Low Nibble (A)*/
```

Once the low nibble is selected, we can read the LSnibble from the Status Port. Take note that the Busy Line is inverted, however we won’t tackle it just yet. We are only interested in the MSnibble of the result, thus we AND the result with 0xF0, to clear the LSnibble.

```
a = (inportb(STATUS) & 0xF0); /* Read Low Nibble */
```

Now it’s time to shift the nibble we have just read to the LSnibble of variable a,

```
a = a >> 4; /* Shift Right 4 Bits */
```

We are now half way there. It’s time to get the MSnibble, thus we must switch the multiplexer to select inputs B. Then we can read the MSnibble and put the two nibbles together to make a byte,

Interfacing the Standard Parallel Port

```
    outportb(CONTROL, inportb(CONTROL) & 0xFE); /* Select High Nibble (B)*/
    a = a |(inportb(STATUS) & 0xF0); /* Read High Nibble */
    byte = byte ^ 0x88;
```

The last line toggles two inverted bits which were read in on the Busy line. It may be necessary to add delays in the process, if the incorrect results are being returned.

Using the Parallel Port's IRQ

The Parallel Port's interrupt request is not used for printing under DOS or Windows. Early versions of OS-2 used them, but don't anymore. Interrupts are good when interfacing monitoring devices such as high temp alarms etc, where you don't know when it is going to be activated. It's more efficient to have an interrupt request rather than have the software poll the ports regularly to see if something has changed. This is even more noticeable if you are using your computer for other tasks, such as with a multitasking operating system.

The Parallel Port's interrupt request is normally IRQ5 or IRQ7 but may be something else if these are in use. It may also be possible that the interrupts are totally disabled on the card, if the card was only used for printing. The Parallel Port interrupt can be disabled and enabled using bit 4 of the control register, *Enable IRQ Via Ack Line*. Once enabled, an interrupt will occur upon a low to high transition (rising edge) of the nACK. However like always, some cards may trigger the interrupt on the high to low transition.

The following code is an Interrupt Polarity Tester, which serves as two things. It will determine which polarity your Parallel Port interrupt is, while also giving you an example for how to use the Parallel Port's Interrupt. It checks if your interrupt is generated on the rising or falling edge of the nACK line. To use the program simply wire **one of** the Data lines (Pins 2 to 9) to the Ack Pin (Pin 10). The easiest way to do this is to bridge some solder from DATA7 (Pin 9) to ACK (Pin 10) on a male DB25 connector.

```
/* Parallel Port Interrupt Polarity Tester                               */
/* 2nd February 1998                                                  */
/* Copyright 1997 Craig Peacock                                       */
/* WWW      - http://www.senet.com.au/~cpeacock                       */
/* Email    - cpeacock@senet.com.au                                   */

#include <dos.h>

#define PORTADDRESS 0x378 /* Enter Your Port Address Here */
#define IRQ 7           /* IRQ Here */

#define DATA PORTADDRESS+0
#define STATUS PORTADDRESS+1
#define CONTROL PORTADDRESS+2

#define PIC1 0x20
#define PIC2 0xA0

int interflag; /* Interrupt Flag */
int picaddr;   /* Programmable Interrupt Controller (PIC) Base Address */
```

Interfacing the Standard Parallel Port

```
void interrupt (*oldhandler)();

void interrupt parisr() /* Interrupt Service Routine (ISR) */
{
    interflag = 1;
    outportb(picaddr,0x20); /* End of Interrupt (EOI) */
}

void main(void)
{
    int c;
    int intno; /* Interrupt Vector Number */
    int picmask; /* PIC's Mask */

    /* Calculate Interrupt Vector, PIC Addr & Mask. */
    if (IRQ >= 2 && IRQ <= 7) {
        intno = IRQ + 0x08;
        picaddr = PIC1;
        picmask = 1;
        picmask = picmask << IRQ;
    }
    if (IRQ >= 8 && IRQ <= 15) {
        intno = IRQ + 0x68;
        picaddr = PIC2;
        picmask = 1;
        picmask = picmask << (IRQ-8);
    }
    if (IRQ < 2 || IRQ > 15)
    {
        printf("IRQ Out of Range\n");
        exit();
    }

    outportb(CONTROL, inportb(CONTROL) & 0xDF); /* Make sure port is in Forward Direction */
    outportb(DATA,0xFF);
    oldhandler = getvect(intno); /* Save Old Interrupt Vector */
    setvect(intno, parisr); /* Set New Interrupt Vector Entry */
    outportb(picaddr+1,inportb(picaddr+1) & (0xFF - picmask)); /* Un-Mask Pic */
    outportb(CONTROL, inportb(CONTROL) | 0x10); /* Enable Parallel Port IRQ's */

    clrscr();
    printf("Parallel Port Interrupt Polarity Tester\n");
    printf("IRQ %d : INTNO %02X : PIC Addr 0x%X : Mask 0x%02X\n",IRQ,intno,picaddr,picmask);
    interflag = 0; /* Reset Interrupt Flag */
    delay(10);
    outportb(DATA,0x00); /* High to Low Transition */
    delay(10); /* Wait */
    if (interflag == 1) printf("Interrupts Occur on High to Low Transition of ACK.\n");
    else
    {
        outportb(DATA,0xFF); /* Low to High Transition */
        delay(10); /* wait */
        if (interflag == 1) printf("Interrupts Occur on Low to High Transition of ACK.\n");
        else printf("No Interrupt Activity Occurred. \nCheck IRQ Number, Port Address "
            "and Wiring.");
    }

    outportb(CONTROL, inportb(CONTROL) & 0xEF); /* Disable Parallel Port IRQ's */
    outportb(picaddr+1,inportb(picaddr+1) | picmask); /* Mask Pic */
    setvect(intno, oldhandler); /* Restore old Interrupt Vector Before Exit */
}
```

At compile time, the above source may generate a few warnings, *condition always true*, *condition always false*, *unreachable code* etc. These are perfectly O.K. They are generated as some of the condition structures test which IRQ you are using, and as the IRQ is defined as a constant some outcomes will never change. While they would of been better implemented as a preprocessor directive, I've done this so you can cut and paste the source code in your own programs which may use command line arguments, user input etc instead of a defined IRQ.

To understand how this example works, the reader must have an assumed knowledge and understanding of Interrupts and Interrupt Service Routines (ISR). If not, see *Using Interrupts*¹ for a quick introduction.

The first part of the mainline routine calculates the Interrupt Vector, PIC Addr & Mask in order to use the Parallel Port's Interrupt Facility. After the Interrupt Service Routine (ISR) has been set up and the Programmable Interrupt Controller (PIC) set, we must enable the interrupt on the Parallel Port. This is done by setting bit 4 of the Parallel Port's Control Register using `outportb(CONTROL, inportb(CONTROL) | 0x10);`

Before enabling the interrupts, we wrote 0xFF to the Parallel Port to enable the 8 data lines into a known state. At this point of the program, all the data lines should be high. The interrupt service routine simply sets a flag (*interflag*), thus we can determine when an IRQ occurs. We are now in a position to write 0x00 to the data port, which causes a high to low transition on the Parallel Port's Acknowledge line as it's connected to one of the data lines.

If the interrupt occurs on the high to low transition, the interrupt flag (*interflag*) should be set. We now test this, and if this is so the program informs the user. However if it is not set, then an interrupt has not yet occurred. We now write 0xFF to the data port, which will cause a low to high transition on the nAck line and check the interrupt flag again. If set, then the interrupt occurs on the low to high transition.

However if the interrupt flag is still reset, then this would suggest that the interrupts are not working. Make sure your IRQ and Base Address is correct and also check the wiring of the plug.

Parallel Port Modes in BIOS

Today, most Parallel Ports are mulimode ports. They are normally software configurable to one of many modes from BIOS. The typical modes are,

Printer Mode (Sometimes called Default or Normal Modes)

Standard & Bi-directional (SPP) Mode

EPP1.7 and SPP Mode

EPP1.9 and SPP Mode

ECP Mode

ECP and EPP1.7 Mode

ECP and EPP1.9 Mode

Printer Mode is the most basic mode. It is a Standard Parallel Port in forward mode only. It has no bi-directional feature, thus Bit 5 of the Control Port will not respond. *Standard & Bi-directional (SPP) Mode* is the bi-directional mode. Using this mode, bit 5 of the Control Port will reverse the direction of the port, so you can read back a value on the data lines.

EPP1.7 and SPP Mode is a combination of EPP 1.7 (Enhanced Parallel Port) and SPP Modes. In this mode of operation you will have access to the SPP registers (Data, Status and Control) and access to the EPP Registers. In this mode you should be able to reverse the direction of the port using bit 5 of the control register. EPP 1.7 is the earlier version of EPP. This version, version 1.7, may not have the time-out bit. See *Interfacing the Enhanced Parallel Port*² for more information.

EPP1.9 and SPP Mode is just like the previous mode, only it uses EPP Version 1.9 this time. As in the other mode, you will have access to the SPP registers, including Bit 5 of the control port. However this differs from EPP1.7 and SPP Mode as you should have access to the EPP Timeout bit.

ECP Mode will give you an Extended Capabilities Port. The mode of this port can then be set using the ECP's Extended Control Register (ECR). However in this mode from BIOS the EPP Mode (100) will not be available. We will further discuss the ECP's Extended Control Register in this document, but if you want further information on the ECP port, consult *Interfacing the Extended Capabilities Port*³.

ECP and EPP1.7 Mode & ECP and EPP1.9 Mode will give you an Extended Capabilities Port, just like the previous mode. However the EPP Mode in the ECP's ECR will now be available. Should you be in *ECP and EPP1.7 Mode* you will get an EPP1.7 Port, or if you are in *ECP and EPP1.9 Mode*, an EPP1.9 Port will be at your disposal.

The above modes are configurable via BIOS. You can reconfigure them by using your own software, but this is **not recommended**. These software registers, typically found at 0x2FA, 0x3F0, 0x3F1 etc are only intended to be accessed by BIOS. There is no set standard for these configuration registers, thus if you were to use these registers, your software would not be very portable. With today's multitasking operating systems, its also not a good idea to change them when it suits you.

A better option is to select *ECP and EPP1.7 Mode* or *ECP and EPP1.9 Mode* from BIOS and then use the ECP's Extended Control Register to select your Parallel Port's Mode. The EPP1.7 mode had a few problems in regards to the Data and Address Strokes being asserted to start a cycle regardless of the wait state, thus this mode if not typically used now. Best set your Parallel Port to *ECP and EPP1.9 Mode*.

Parallel Port Modes and the ECP's Extended Control Register

As we have just discussed, it is better to set the Parallel Port to *ECP and EPP1.9 Mode* and use the ECP's Extended Control Register to select different modes of operation. The ECP Registers are standardized under Microsoft's **Extended Capabilities Port Protocol and ISA Interface Standard**, thus we don't have that problem of every vendor having their own register set.

When set to ECP Mode, a new set of registers become available at Base + 0x400h. A discussion of these registers are available in *Interfacing the Extended Capabilities Port*³. Here we are only interested in the Extended Control Register (ECR) which is mapped at Base + 0x402h. It should be stated that the ECP's registers are not available for port's with a base address of 0x3BCh.

Bit	Function	
7:5	<i>Selects Current Mode of Operation</i>	
	000	Standard Mode
	001	Byte Mode
	010	Parallel Port FIFO Mode
	011	ECP FIFO Mode
	100	EPP Mode
	101	Reserved
	110	FIFO Test Mode
	111	Configuration Mode
4	ECP Interrupt Bit	
3	DMA Enable Bit	
2	ECP Service Bit	
1	FIFO Full	
0	FIFO Empty	

Table 7 ECR - Extended Control Register

The table above is of the Extended Control Register. We are only interested in the three MSB of the Extended Control Register which selects the mode of operation. There are 7 possible modes of operation, but not all ports will support all modes. The EPP mode is one such example, not being available on some ports.

Modes of Operation

- Standard mode Selecting this mode will cause the ECP port to behave as a Standard Parallel Port, without bi-directional functionality.

- Byte Mode / PS/2 mode Behaves as a SPP in bi-directional mode. Bit 5 will place the port in reverse mode.

- Parallel Port FIFO mode In this mode, any data written to the Data FIFO will be sent to the peripheral using the SPP Handshake. The hardware will generate the handshaking required. Useful with non-ECP devices such as printers. You can have some of the features of ECP like FIFO buffers and hardware generation of handshaking but with the existing SPP handshake (Centronics) instead of the ECP Handshake.

- ECP FIFO mode Standard mode for ECP use. This mode uses the ECP Handshake described in *Interfacing the Extended Capabilities Port*³

When in ECP Mode though BIOS, and the ECR register is set to ECP FIFO Mode (011), the SPP registers may disappear.

- EPP mode/Reserved This will enable EPP Mode, if available. Under BIOS, if *ECP mode* is set then it's more than likely, this mode is not an option. However if BIOS is set to *ECP and EPP1.x Mode*, then EPP 1.x will be enabled.

Under Microsoft's Extended Capabilities Port Protocol and ISA Interface Standard this mode is Vendor Specified.

Interfacing the Standard Parallel Port

Reserved	Currently Reserved. <i>Under Microsoft's Extended Capabilities Port Protocol and ISA Interface Standard this mode is Vendor Specified.</i>
FIFO Test Mode	While in this mode, any data written to the Test FIFO Register will be placed into the FIFO and any data read from the Test FIFO register will be read from the FIFO buffer. The FIFO Full/Empty Status Bits will reflect their true value, thus FIFO depth, among other things can be determined in this mode.
Configuration Mode	In this mode, the two configuration registers, cnfgA & cnfgB become available at their designated Register Addresses.

If you are in *ECP Mode* under BIOS, or if your card is jumpered to use ECP then it is a good idea to initialize the mode of your ECP port to a pre-defined state before use. If you are using SPP, then set the port to Standard Mode as the first thing you do. Don't assume that the port will already be in Standard (SPP) mode.

Under some of the modes, the **SPP registers may disappear or not work correctly**. If you are using SPP, then set the ECR to Standard Mode. This is one of the most common mistakes that people make.

Notes

Note¹ **Using Interrupts** is available in PDF from
<http://www.geocities.com/SiliconValley/Bay/8302/interrupt.pdf> (62kb)

Note² **Interfacing the Enhanced Parallel Port** is available in PDF from
<http://www.geocities.com/SiliconValley/Bay/8302/epp.pdf> (33kb)

Note³ **Interfacing the Extended Capabilities Port** is available in PDF from
<http://www.geocities.com/SiliconValley/Bay/8302/ecp.pdf> (53kb)

Craig Peacock's Interfacing the PC

<http://www.senet.com.au/~cpeacock>

<http://www.geocities.com/SiliconValley/Bay/8302/>

Copyright February 1998 Craig Peacock.

Any errors, ideas, criticisms or problems, please contact the author at cpeacock@senet.com.au

Interfacing the Enhanced Parallel Port

Table of Contents

EPP - Enhanced Parallel Port	Page 1
EPP Hardware Properties	Page 1
The EPP Handshake	Page 2
EPP Data Write Cycle	Page 3
EPP Address Write Cycle	Page 3
EPP Data Read Cycle	Page 4
EPP Address Read Cycle	Page 4
The EPP's Software Registers	Page 4
EPP Programming Considerations	Page 6

EPP - Enhanced Parallel Port

The Enhanced Parallel Port (EPP) was designed in a joint venture between Intel, Xircom & Zenith Data Systems. EPP Ports were first specified in the EPP 1.7 standard, and then later included in the IEEE 1284 Standard released in 1994. EPP has two standards, EPP 1.7 and EPP 1.9. There are differences between the two standards which may affect the operation of devices. This is further discussed latter. EPP has a typical transfer rate in the order of 500KB/S to 2MB/S. This is achieved by allowing the hardware contained in the port to generate handshaking, strobing etc, rather that have the software do it, which was the case with Centronics.

For the hobbyist, EPP is more commonly used than ECP. EPP differs from ECP by the fact that the EPP Port generates and controls all the transfers to and from the peripheral. ECP on the other hand requires the peripheral to negotiate a reverse channel and control the handshaking. This is harder to achieve with common glue logic, thus really requires a dedicated controller or ECP Peripheral Chip.

EPP Hardware Properties

When using EPP mode, a different set of tasks and labels are assigned to each line. These are listed below in Table 4. It's very common to see both the SPP and EPP names interchanged in Parallel Port Data Sheets and Literature. This can make it very hard to focus on what is exactly happening. Therefore all the documentation here will use the EPP names.

Pin	SPP Signal	EPP Signal	In/Out	Function
1	Strobe	Write	Out	A low on this line indicates a Write, High indicates a Read
2-9	Data 0-7	Data 0-7	In-Out	Data Bus. Bi-directional
10	Ack	Interrupt	In	Interrupt Line. Interrupt occurs on Positive (Rising) Edge.
11	Busy	Wait	In	Used for handshaking. A EPP cycle can be started when low, and finished when high.
12	Paper Out / End	Spare	In	Spare - Not Used in EPP Handshake
13	Select	Spare	In	Spare - Not Used in EPP Handshake
14	Auto Linefeed	Data Strobe	Out	When Low, indicates Data transfer
15	Error / Fault	Spare	In	Spare - Not used in EPP Handshake
16	Initialize	Reset	Out	Reset - Active Low
17	Select Printer	Address Strobe	Out	When low, indicates Address transfer
18-25	Ground	Ground	GND	Ground

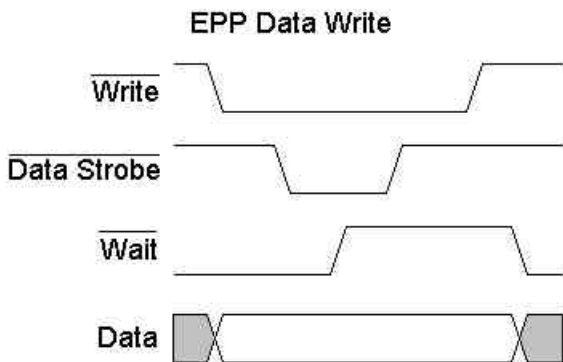
Table 1. Pin Assignments For Enhanced Parallel Port Connector.

Paper Out, Select and Error are not defined in the EPP handshake. These lines can be utilised in any way by the user. The status of these lines can be determined at anytime by viewing the SPP Status Register. Unfortunately there are no spare output's. This can become a hassle regularly.

The EPP Handshake

In order to perform a valid exchange of data using EPP we must follow the EPP handshake. As the hardware does all the work, this handshake only requires to be used for your hardware and not for software as the case with SPP. To initiate an EPP cycle your software needs to perform only one I/O operation to the relevant EPP Register. Details on this, later.

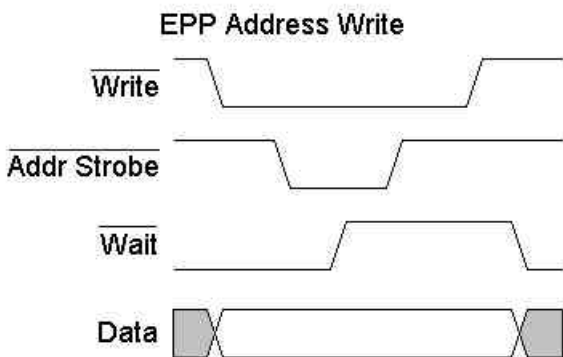
EPP Data Write Cycle



1. **Program writes to EPP Data Register. (Base + 4)**
2. **nWrite** is placed low. (Low indicates write operation)
3. **Data is placed on Data Lines 0-7.**
4. **nData Strobe** is asserted if **Wait** is Low (O.K. to start cycle)
5. **Host waits for Acknowledgment by nWait going high (O.K. to end cycle)**
6. **nData Strobe** is de-asserted.
7. **EPP Data Write Cycle Ends.**

Figure 1. Enhanced Parallel Port Data Write Cycle.

EPP Address Write Cycle



1. **Program writes address to EPP's Address Register (Base + 3)**
2. **Write** is placed low. (Low indicates write operation)
3. **Address is placed on Data Lines 0-7.**
4. **nAddress Strobe** is asserted if **Wait** is Low (O.K. to start cycle)
5. **Host waits for Acknowledgment by wait going high (O.K. to end cycle)**
6. **nAddress Strobe** is De-asserted.
7. **EPP Address Write Cycle Ends.**

Figure 2. Enhanced Parallel Port Address Write Cycle.

EPP Data Read Cycle

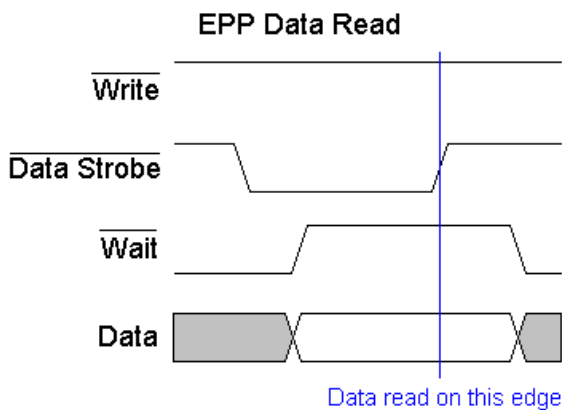


Figure 3. Enhanced Parallel Port Data Read Cycle.

1. **Program reads EPP Data Register. (Base + 4)**
2. nData Strobe is asserted if Wait is Low (O.K. to start cycle)
3. **Host waits for Acknowledgment by nWait going high**
4. Data is read from Parallel Port Pins.
5. **nData Strobe is de-asserted.**
6. EPP Data Read Cycle Ends.

EPP Address Read Cycle

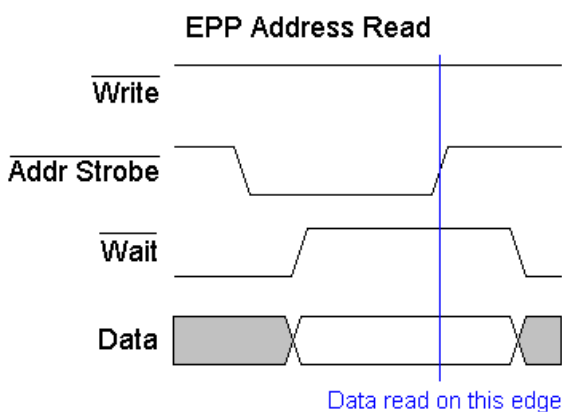


Figure 4. Enhanced Parallel Port Address Read Cycle.

1. **Program reads EPP Address Register. (Base + 3)**
2. nAddr Strobe is asserted if Wait is Low (O.K. to start cycle)
3. **Host waits for Acknowledgment by nWait going high**
4. Data is read from Parallel Port Pins.
5. **nAddr Strobe is de-asserted.**
6. EPP Address Read Cycle Ends.

Note If implementing EPP 1.7 Handshake (Pre IEEE 1284) the Data and Address Strobes can be asserted to start a cycle regardless of the wait state. EPP 1.9 will only start a cycle once wait is low. Both EPP 1.7 and EPP 1.9 require the wait to be high to finish a cycle.

The EPP's Software Registers.

The EPP Port also has a new set of registers. However 3 of them have been inherited from the Standard Parallel Port. On the next page is a table showing the new and existing registers.

Address	Port Name	Read/Write
Base + 0	Data Port (SPP)	Write
Base + 1	Status Port (SPP)	Read
Base + 2	Control Port (SPP)	Write
Base + 3	Address Port (EPP)	Read/Write
Base + 4	Data Port (EPP)	Read/Write
Base + 5	Undefined (16/32bit Transfers)	-
Base + 6	Undefined (32bit Transfers)	-
Base + 7	Undefined (32bit Transfers)	-

Table 2 EPP Registers

As you can see, the first 3 addresses are exactly the same than the Standard Parallel Port Register and behave in exactly the same way. Therefore if you used a Enhanced Parallel Port, you can output data to Base + 0 in exactly the same fashion than you would if it was a Standard Parallel Port (SPP). If you were to connect a printer, and use compatibility mode then you would have to check to see if the port is busy and then assert & de-assert the strobe using the Control and Status Port, then wait for the Ack.

If you wish to communicate with a EPP compatible device then all you have to do, is place any data you wish to send in the EPP Data Register at Base + 4 and the card will generate all the necessary handshaking required. Likewise if you wish to send an address to your device, then you use the EPP Address Register at offset +3.

Both the EPP Address Register and the EPP Data Register are read / write, thus to read data from your device, you can use the same registers. However the EPP Printer Card has to initiate a read Cycle as both the nData Strobe and nAddress Strobe are outputs. Your device can signal a read request via the use of the interrupt and have your ISR perform the Read Operation.

The Status Port has one little modification. Bit 0, which was reserved in the SPP register set, now becomes the EPP Time-out Bit. This bit will be set when an EPP time-out occurs. This happens when the nWait line is not deasserted within approximately 10uS (depending upon the port) of the IOW or IOR line being asserted. The IOW and IOR are the I/O Read and Write lines present on the ISA Bus.

The EPP mode is very depended of the ISA bus timing. When a read cycle is performed, the port must undertake the appropriate Read/Write handshake and return the data in that ISA cycle. Of course this doesn't occur within one ISA cycle, thus the port uses the IOCHRDY (I/O Channel Ready) on the ISA bus to introduce wait states, until the cycle completes. Now imagine if a EPP Read or Write is started with no peripheral connected? The port never gets an acknowledgment (nWait), thus keeps sending requests for wait states, and your computer locks up. Therefore the EPP implements a type of watchdog, which times out after approximately 10uS.

The three registers, Base + 5, Base + 6 and Base + 7 can be used for 16 and 32 bit read/write operations if your port supports it. This can further reduce your I/O operations. The Parallel Port can only transport 8 bits at a time, thus any 32 or 16 bit word written to the Parallel Port will be split into byte size blocks and sent via the Parallel Port's 8 data lines.

EPP Programming Considerations.

EPP only has two main registers and a Time-out Status Flag, What could there possibly be to set up?

Before you can start any EPP cycles by reading and writing to the EPP Data and Address Ports, the port must be configured correctly. In the idle state, an EPP port should have it's nAddress Strobe, nData Strobe, nWrite and nReset lines inactive, high. Some ports require you to set this up before starting any EPP Cycle. Therefore our first task is to manually initialise these lines using the SPP Registers. Writing XXXX0100 to the control port will do this.

On some cards, if the Parallel Port is placed in reverse mode, a EPP Write cycle cannot be performed. Therefore it is also wise to place the Parallel Port in forward mode before using EPP. Clearing Bit 5 of the Control Register should result in an more enjoyable programming session, without tearing your hair out.

The EPP Time-out bit we have already discussed. When this bit is set, the EPP port may not function correctly. A common scenario is always reading 0xFF from either the Address or Data Cycles. This bit should be cleared for reliable operation, and constantly checked.

Interfacing the Extended Capabilities Port

Table of Contents

Introduction to the Extended Capabilities Port	Page 1
ECP Hardware Properties	Page 1
The ECP Handshake	Page 2
ECP Forward Data Cycle	Page 3
ECP Forward Command Cycle	Page 3
ECP Reverse Data Cycle	Page 4
ECP Reverse Command Cycle	Page 4
EPP Handshake vs SPP Handshake	Page 5
RLE – Run Length Encoding	Page 5
ECP Software Registers	Page 5
ECP's Extended Control Register (ECR)	Page 6
ECP's Configuration Register A (cnfgA)	Page 7
ECP's Configuration Register B (cnfgB)	Page 8

Introduction to the Extended Capabilities Port

The Extended Capabilities Mode was designed by Hewlett Packard and Microsoft to be implemented as the *Extended Capabilities Port Protocol and ISA Interface Standard*. This protocol uses additional hardware to generate hand shaking signals etc just like the EPP mode, thus runs at very much the same speed than the EPP mode. This mode, however may work better under Windows as it can use DMA channels to move it's data about. It also uses a FIFO buffer for the sending and/or receiving of data.

Another feature of ECP is a real time data compression. It uses Run Length Encoding (RLE) to achieve data compression ratio's up to 64:1. This comes is useful with devices such as Scanners and Printers where a good part of the data is long strings which are repetitive.

The Extended Capabilities Port supports a method of channel addressing. This is not intended to be used to daisy chain devices up but rather to address multiple devices within one device. Such an example is many fax machines on the market today which may contain a Parallel Port to interface it to your computer. The fax machine can be split up into separate devices such as the scanner, modem/Fax and printer, where each part can be addresses separately, even if the other devices cannot accept data due to full buffers.

ECP Hardware Properties

While Extended Capabilities Printer Ports use exactly the same D25 connector as your SPP, ECP assigns different tasks to each of the pins, just like EPP. This means that there is also a different handshake method when using a ECP interface.

The ECP is backwards compatible to the SPP and EPP. When operating in SPP mode, the individual lines operate in exactly the same fashion than the SPP and thus are labelled Strobe, Auto Linefeed, Init, Busy etc. When operating in EPP mode, the pins function according to the method described in the EPP protocol and have a different method of hand shaking. When the port is operating in ECP mode, then the following labels are assigned to each pin.

Pin	SPP Signal	ECP Signal	Direct'n	Function
1	Strobe	HostCLK	Out	A low on this line indicates, that there is valid data at the host. When this pin is de-asserted, the +ve clock edge should be used to shift the data into the device.
2-9	Data 0-7	Data 0-7	In/Out	Data Bus. Bi-directional
10	Ack	PeriphCLK	In	A low on this line indicates, that there is valid data at the Device. When this pin is de-asserted, the +ve clock edge should be used to shift the data into the Host.
11	Busy	PeriphAck	In	When in reverse direction a HIGH indicates Data, while a LOW indicates a Command Cycle. In forward direction, functions as PeriphAck.
12	Paper Out End	nAckReverse	In	When Low, Device acknowledges Reverse Request.
13	Select	X-Flag	In	Extensibility Flag
14	Auto Linefeed	Host Ack	Out	When in forward direction a HIGH indicates Data, while a LOW indicates a Command Cycle. In reverse direction, functions as HostAck.
15	Error Fault	PeriphRequest	In	A LOW set by the device indicates reverse data is available.
16	Initialise	NReverse Request	Out	A LOW indicates data is in reverse direction.
17	Select Printer	1284 Active	Out	A HIGH indicates Host is in 1284 Transfer Mode. Taken low to terminate.
18-25	Ground	Ground	GND	Ground.

Table 1. Pin Assignments For Extended Capabilities Parallel Port Connector.

The HostAck and PeriphAck lines indicate whether the signals on the data line are data or a command. If these lines are high then data is placed on the data lines (Pins 2-7). If a command cycle is taking place then the appropriate line will be low, i.e. if the host is sending a command, then HostAck will be low or if the device/peripheral is sending a command the PeriphAck line will be low.

A command cycle can be one of two things, either a RLE count or an address. This is determined by the bit 7 (MSB) of the data lines, i.e. pin 9. If bit 7 is a 0, then the rest of the data (bits 0-6) is a run length count which is used with the data compression scheme. However if bit 7 is a 1, then the data present on bits 0 to 6 is a channel address. With one bit missing this can only be a value from 0 to 127(DEC).

The ECP Handshake

The ECP handshake is different to the SPP handshake. The most obvious difference is that ECP has the ability at anytime to transmit data in any direction, thus additional signalling is required. Below is the ECP handshake for both the Forward and Reverse Directions.

ECP Forward Data Cycle

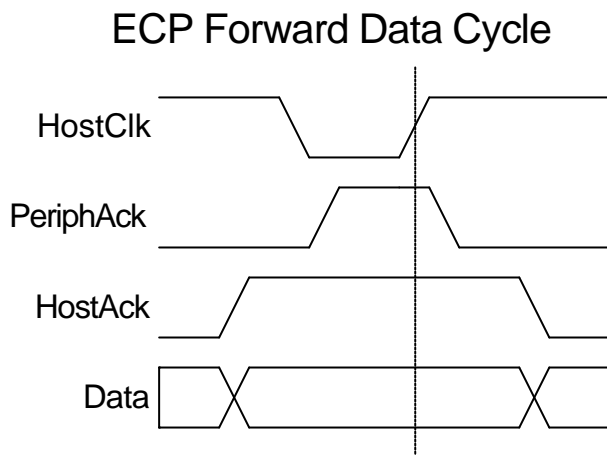


Figure 1. Enhanced Capabilities Port Forward Data Cycle.

1. **Data is placed on Data lines by Host.**
2. Host then indicates a Data Cycle will proceed by asserting HostAck.
3. **Host indicates valid data by asserting HostClk low.**
4. Peripheral sends its acknowledgment of valid data by asserting PeriphAck.
5. **Host de-asserts HostClk high. +ve edge used to shift data into the Peripheral.**
6. Peripheral sends its acknowledgment of the byte via de-asserting PeriphAck.

ECP Forward Command Cycle

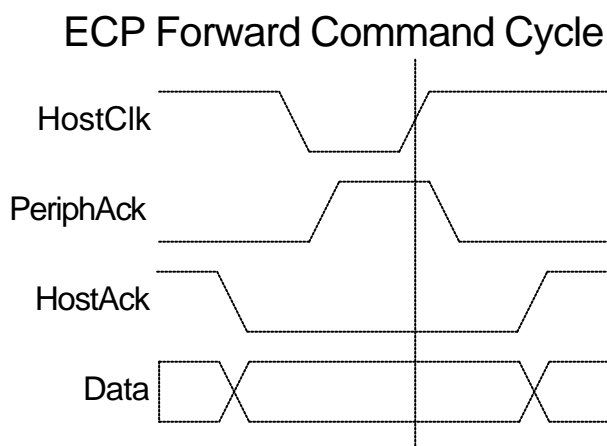


Figure 2. Enhanced Capabilities Port Forward Command Cycle.

1. **Data is placed on Data lines by Host.**
2. Host then indicates a Command cycle will proceed by de-asserting HostAck.
3. **Host indicates valid data by asserting HostClk low.**
4. Peripheral sends its acknowledgment of valid data by asserting PeriphAck.
5. **Host de-asserts HostClk high. +ve edge used to shift data into the Peripheral.**
6. Peripheral sends its acknowledgment of the byte via de-asserting PeriphAck.

ECP Reverse Data Cycle

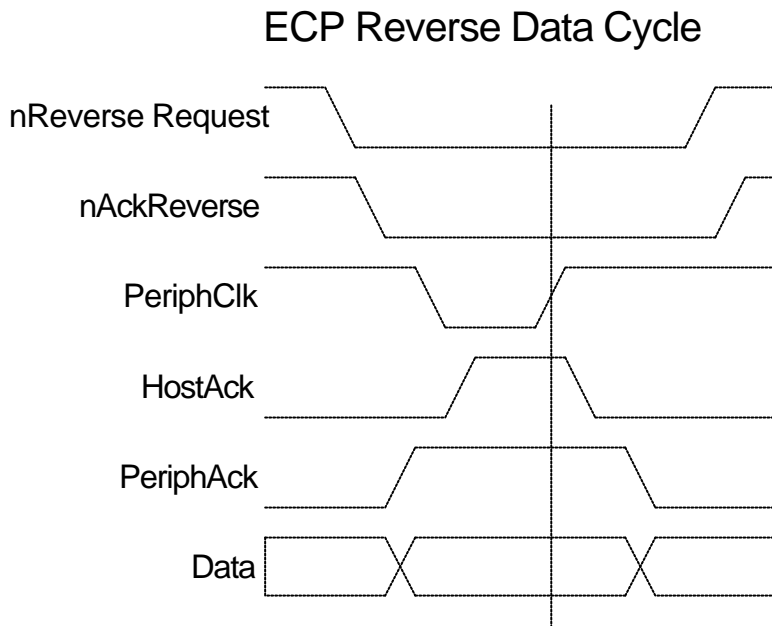


Figure 3. Enhanced Capabilities Port Reverse Data Cycle.

1. **Host sets nReverseRequest Low to request a reverse channel.**
2. Peripheral acknowledges reverse channel request via asserting nAckReverse low.
3. **Data is placed on data lines by Peripheral.**
4. Data cycle is then selected by Peripheral via PeriphAck going high.
5. **Valid data is indicated by the Peripheral setting PeriphClk low.**
6. Host sends its acknowledgment of valid data via HostAck going high.
7. **Device/Peripheral sets PeriphClk high. +ve edge used to shift data into the Host.**
8. Host sends its acknowledgment of the byte by de-asserting HostAck low.

ECP Reverse Command Cycle

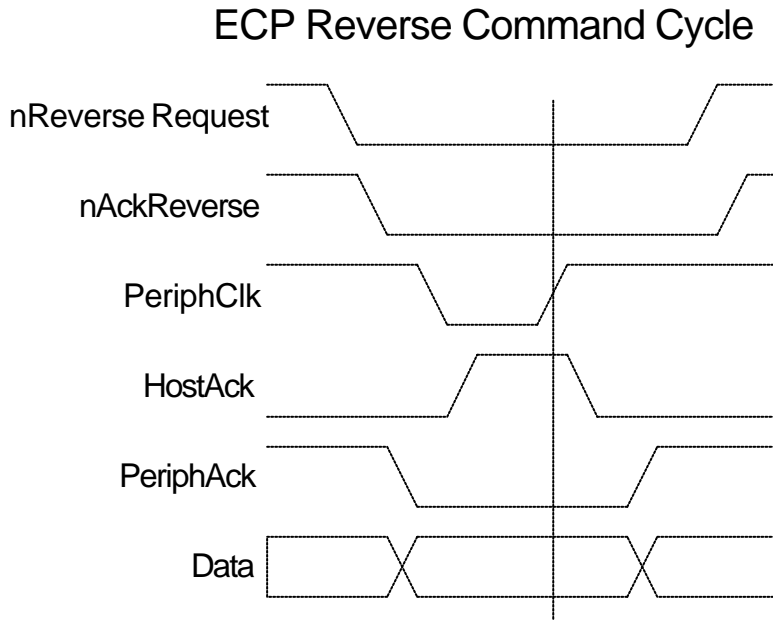


Figure 4. Enhanced Capabilities Port Reverse Command Cycle.

1. **Host sets nReverseRequest Low to request a reverse channel.**
2. Peripheral acknowledges reverse channel request via asserting nAckReverse low.
3. **Data is placed on data lines by Peripheral.**
4. Command cycle is then selected by Peripheral via PeriphAck going low.
5. **Valid data is indicated by the Peripheral setting PeriphClk low.**
6. Host sends its acknowledgment of valid data via HostAck going high.
7. **Device/Peripheral sets PeriphClk high. +ve edge used to shift data into the Host.**
8. Host sends its acknowledgment of the byte by de-asserting HostAck low.

EPP Handshake vs SPP Handshake

If we look back at the SPP handshake you will realise it only has 5 steps,

1. *Write the byte to the Data Port*
2. *Check to see if the printer is busy. If the printer is busy, it will not accept any data, thus any data which is written will be lost.*
3. *Take the Strobe (Pin 1) low. This tells the printer that there is the correct data on the data lines. (Pins 2-9)*
4. *Put the strobe high again after waiting approximately 5 microseconds after putting the strobe low. (Step 3)*
5. *Check for Ack from Peripheral.*

and that the ECP handshake has many more steps. This would suggest that ECP would be slower than SPP. However this is not the case as all of these steps above are controlled by the hardware on your I/O control. If this handshake was to be implemented via software control then it would be a lot slower than its SPP counterpart.

RLE - Run Length Encoding

As briefly discussed earlier, the ECP Protocol includes a Simple Compression Scheme called Run Length Encoding. It can support a maximum compression ratio of 64:1 and works by sending repetitive single bytes as a run count and one copy of the byte. The run count determines how many times the following byte is to be repeated.

For example, if a string of 25 'A's were to be sent, then a run count byte equal to 24 would be sent first, followed by the byte 'A'. The receiving peripheral on receipt of the Run Length Count, would expand (Repeat) the next byte a number of times determined via the run count.

The Run Length Byte has to be distinguished from other bytes in the Data Path. It is sent as a Command to the ECP's Address FIFO Port. Bytes sent to this register can be of two things, a Run Length Count or an Address. These are distinguished by the MSB, Bit 7. If Bit 7 is Set (1), then the other 7 bits, bits 0 to 6 is a channel address. If Bit 7 is Reset (0), then the lower 7 bits is a run length count. By using the MSB, this limits channel Addresses and Run Length Counts to 7 Bits (0 - 127).

ECP Software Registers

The table below shows the registers of the Extended Capabilities Port. The first 3 registers are exactly the same than with the Standard Parallel Port registers. Note should be taken, however, of the Enable bi-directional Port bit (bit 5 of the Control Port.) This bit reflects the direction that the ECP port is currently in, and will effect the FIFO Full and FIFO Empty bits of the ECR Register, which will be explained later.

Address	Port Name	Read/Write
Base + 0	Data Port (SPP)	Write
	ECP Address FIFO (ECP MODE)	Read/Write
Base + 1	Status Port (All Modes)	Read/Write
Base + 2	Control Port (All Modes)	Read/Write
Base + 400h	Data FIFO (Parallel Port FIFO Mode)	Read/Write
	Data FIFO (ECP Mode)	Read/Write
	Test FIFO (Test Mode)	Read/Write
	Configuration Register A (Configuration Mode)	Read/Write
Base + 401h	Configuration Register B (Configuration Mode)	Read/Write
Base + 402h	Extended Control Register (Used by all modes)	Read/Write

Table 2 : ECP Registers

ECP's Extended Control Register (ECR)

The most important register with a Extended Capabilities Parallel Port is the Extended Control Register (ECR) thus we will target it's operation first. This register sets up the mode in which the ECP will run, plus gives status of the ECP's FIFO among other things. You will find the contents of this register below, in more detail.

Bit	Function
7:5	Selects Current Mode of Operation
	000 Standard Mode
	001 Byte Mode
	010 Parallel Port FIFO Mode
	011 ECP FIFO Mode
	100 EPP Mode
	101 Reserved
	110 FIFO Test Mode
	111 Configuration Mode
4	ECP Interrupt Bit
3	DMA Enable Bit
2	ECP Service Bit
1	FIFO Full
0	FIFO Empty

Table 3 ECR - Extended Control Register

The three MSB of the Extended Control Register selects the mode of operation. There are 7 possible modes of operation, but not all ports will support all modes. The EPP mode is one such example, not being available on some ports. On the next page is a table of the Modes of Operation.

Modes of Operation

Standard Mode	Selecting this mode will cause the ECP port to behave as a Standard Parallel Port, without Bi-directional functionality.
Byte Mode / PS/2 Mode	Behaves as a SPP in Bi-directional (Reverse) mode.
Parallel Port FIFO Mode	In this mode, any data written to the Data FIFO will be sent to the peripheral using the SPP handshake. The hardware will generate the handshaking required. Useful with non-ECP devices such as Printers. You can have some of the features of ECP like FIFO buffers and hardware generation of handshaking but with the existing SPP handshake instead of the ECP handshake.
ECP FIFO Mode	Standard Mode for ECP use. This mode uses the ECP Handshake, already described.
EPP Mode / Reserved	On some chipsets, this mode will enable EPP to be used. While on others, this mode is still reserved.
Reserved	Currently Reserved.
FIFO Test Mode	While in this mode, any data written to the Test FIFO Register will be placed into the FIFO and any data read from the Test FIFO register will be read from the FIFO buffer. The FIFO Full/Empty Status Bits will reflect their true value, thus FIFO depth, among other things can be determined in this mode.
Configuration Mode	In this mode, the two configuration registers, cnfgA & cnfgB become available at their designated Register Addresses.

As outlined above, when the port is set to operate in Standard Mode, it will behave just like a Standard Parallel Port (SPP) with no bi-directional data transfer. If you require bi-directional transfer, then set the mode to Byte Mode. The Parallel Port FIFO mode and ECP FIFO mode both use hardware to generate the necessary handshaking signals. The only difference between each mode is that The Parallel Port FIFO Mode uses SPP handshaking, thus can be used with your SPP printer. ECP FIFO mode uses ECP handshaking.

The FIFO test mode can be used to test the capacity of the FIFO Buffers as well as to make sure they function correctly. When in FIFO test mode, any byte which is written to the TEST FIFO (Base + 400h) is placed into the FIFO buffer and any byte which is read from this register is taken from the FIFO Buffer. You can use this along with the FIFO Full and FIFO Empty bits of the Extended Control Register to determine the capacity of the FIFO Buffer. This should normally be about 16 Bytes deep.

The other Bits of the ECR also play an important role in the operation of the ECP Port. The ECP Interrupt Bit, (Bit 4) enables the use of Interrupts, while the DMA Enable Bit (Bit 3) enables the use of Direct Memory Access. The ECP Service Bit (Bit 2) shows if an interrupt request has been initiated. If so, this bit will be set. Resetting this bit is different with different chips. Some require you to Reset the Bit, e.g. Write a Zero to it. Others will reset once the Register has been read.

The FIFO Full (Bit 1) and FIFO Empty (Bit 0) show the status of the FIFO Buffer. These bits are direction dependent, thus note should be taken of the Control Register's Bit 5. If bit 0 (FIFO Empty) is set, then the FIFO buffer is completely empty. If Bit 1 is set then the FIFO buffer is Full. Thus, if neither bit 0 or 1 is set, then there is data in FIFO, but is not yet full. These bits can be used in FIFO Test Mode, to determine the capacity of the FIFO Buffer.

ECP's Configuration Register A (cnfgA)

Configuration Register A is one of two configuration registers which the ECP Port has. These Configuration Registers are only accessible when the ECP Port is in Configuration Mode. (See Extended Control Register) CnfgA can be accessed at Base + 400h.

Bit	Function	
7	1	Interrupts are level triggered
	0	Interrupts are edge triggered (Pulses)
6:4	00h	Accepts Max. 16 Bit wide words
	01h	Accepts Max. 8 Bit wide words
	02h	Accepts Max. 32 Bit wide words
	03h:07h	Reserved for future expansion
3	Reserved	
2	<i>Host Recovery : Pipeline/Transmitter Byte included in FIFO?</i>	
	0	In forward direction, the 1 byte in the transmitter pipeline doesn't affect FIFO Full.
	1	In forward direction, the 1 byte in the transmitter pipeline is include as part of FIFO Full.
1:0	<i>Host Recovery : Unsent byte(s) left in FIFO</i>	
	00	Complete Pword
	01	1 Valid Byte
	10	2 Valid Bytes
	11	3 Valid Bytes

Table 4 ECR - Configuration Register A

Configuration Register A can be read to find out a little more about the ECP Port. The MSB, shows if the card generates level interrupts or edge triggered interrupts. This will depend upon the type of bus your card is using. Bits 4 to 6, show the buses width within the card. Some cards only have a 8 bit data path, while others may have a 32 or 16 bit width. To get maximum efficiency from your card, the software can read the status of these bits to determine the Maximum Word Size to output to the port.

The 3 LSB's are used for Host Recovery. In order to recover from an error, the software must know how many bytes were sent, by determining if there are any bytes left in the FIFO. Some implementations may include the byte sitting in the transmitter register, waiting to be sent as part of the FIFO's Full Status, while others may not. Bit 2 determines weather or not this is the case.

The other problem is that the Parallel Ports output is only 8 bits wide, and that you may be using 16 bit or 32 bit I/O Instructions. If this is the case, then part of your Port Word (Word you sent to port) may be sent. Therefore Bits 0 and 1 give an indication of the number of valid bytes still left in the FIFO, so that you can retransmit these.

ECP's Configuration Register B (cnfgB)

Configuration Register B, like Configuration Register A is only available when the ECP Port is in Configuration Mode. When in this mode, cnfgB resides at Base + 401h. On the next page you will find the make-up of the cnfgB Register.

Bit(s)	Function	
7	1	Compress outgoing Data Using RLE
	0	Do Not compress Data
6	Interrupt Status - Shows the Current Status of the IRQ Pin	
5:3	<i>Selects or Displays Status of Interrupt Request Line.</i>	
	000	Interrupt Selected Via Jumper
	001	IRQ 7
	010	IRQ 9
	011	IRQ 10
	100	IRQ 11
	101	IRQ 14
	110	IRQ 15
	111	IRQ 5
2:0	<i>Selects or Displays Status of the DMA Channel the Printer Card Uses</i>	
	000	Uses a Jumpered 8 Bit DMA Channel
	001	DMA Channel 1
	010	DMA Channel 2
	011	DMA Channel 3
	100	Uses a Jumpered 16 Bit DMA Channel
	101	DMA Channel 5
	110	DMA Channel 6
	111	DMA Channel 7

Table 5 - Configuration B Register

The Configuration Register B (cnfgB) can be a combination of read/write access. Some ports may be software configurable, where you can set the IRQ and DMA resources from the register. Others may be set via BIOS or by using jumpers on the Card, thus are read only.

Bit 7 of the cnfgB Register selects whether to compress outgoing data using RLE (Run Length Encoding.) When Set, the host will compress the data before sending. When reset, data will be sent to the peripheral raw (Uncompressed). Bit 6 returns the status of the IRQ pin. This can be used to diagnose conflicts as it will not only reflect the status of the Parallel Ports IRQ, but and other device using this IRQ.

Bits 5 to 3 give status of about the Port's IRQ assignment. Likewise for bits 2 to 0 which give status of DMA Channel assignment. As mentioned above these fields may be read/write. The disappearing species of Parallel Cards which have Jumpers may simply show it's resources as "Jumpered" or it may show the correct Line Numbers. However these of course will be read only.