

Re: OT: DOS programming EPP

Source: <http://sci.tech-archive.net/Archive/sci.electronics.design/2006-01/msg04178.html>

- *From:* Robert Baer <robertbaer@xxxxxxxxxxxxxx>
 - *Date:* Tue, 24 Jan 2006 09:10:50 GMT
-

PaulCsouls wrote:

On Mon, 23 Jan 2006 02:33:12 GMT, Robert Baer
<robertbaer@xxxxxxxxxxxxxx> wrote:

PaulCsouls wrote:

On Sun, 22 Jan 2006 03:14:06 GMT, Robert Baer
<robertbaer@xxxxxxxxxxxxxx> wrote:

I have done a websearch, and there is nothing available that explicitly shows how to program a parallel port in the EPP mode, and how to *safely* interface with it. The best (incomplete) source i found was on the beyondlogic.org site. However, no matter what i do, the nominally input printer pins (pin 1 = strobe, pin 13 = select and pin 15 = error bar) act like outputs. Data lines when low safely sink 2mA (did not try more as i did not want to zap the MB)

Re: OT: DOS programming EPP

R=45 ohms, and when high safely source 1mA (did not try more R=2.2K).

"Strobe" line pin 1 was always high and safely sink 1.5mA R=730 ohms.

"Select" line pin 13 was always high and safely sink 1.0mA R=2.2K.

There seems to be *no* specifications or equivalent circuits for the parallel port as implemented on the ASICs used in modern PCs.

Therefore, it is completely unknown as to the maximum safe sink current to a logic low pin or the maximum source current from a logic high pin. It is not wise to force a pin that is acting as an output, into the opposite state; so the info is necessary for safety.

I want and need to program this in DOS.

** as an aside, it was interesting to see that when Windoz booted after my fiddling, that i saw "detecting new hardware" etc.

Go back to the beyond logic page and read the parallel port FAQ from the beginning. The EPP page assumes you did. You need to set up the extended control register (BASE + 0x400). In EPP mode Strobe is the WRITE/NOT READ bit and an output. I think you need to do some C or BASIC code to do any of this. I don't think you can just use DOS shell commands.

Re: OT: DOS programming EPP

Paul C

I said *NOTHING* about DOS shell commands!
What follows is a listing of a BASIC program that i have used for preliminary investigation (use monospacing font width 90):

```
' Attempt to read data from parallel port using EPP protocol
DEFINT A-Z
PRNT = &H378
DATAS = PRNT + 0: STATUS = PRNT + 1: CONTROL = PRNT + 2 'SPP
' ^--r/w          ^--read only          ^--r/w (from IBM)
ADDRESSrw = PRNT + 3: DATArw = PRNT + 4 'EPP
' ^--pin 17 pulse*          ^--pin 14 pulse* *=if only one used
BIT5 = &H20: PIN17 = &H8
' The following pins (as output) must be high:
' Address Strobe=pin 17, Data Strobe=pin 14, Write=pin 1, Reset=pin 16.
' NOTE: After system boot, all pins default high except data pins and pin 17.
'          CONTROL bit 5 defaults low.

CLS
LOCATE 1, 1
PRINT " First read STATUS bits (cannot write them)          meas OK?  init"
SEN = INP(STATUS)
PRINT "D0:"; (SEN AND &H1)/&H1, "ghost Pin 01 /STROBE (reads inverted)  1    N    0
PRINT "D1:"; (SEN AND &H2)/&H2, "ghost Pin 14 /AUTO FD (reads inverted)  1    N    1
PRINT "D2:"; (SEN AND &H4)/&H4, "ghost Pin 16 /INIT                    1    Y    1
PRINT "D3:"; (SEN AND &H8)/&H8, "Pin 15 /ERROR                        1    Y    1
PRINT "D4:"; (SEN AND &H10)/&H10, "Pin 13 SLCT (from printer)          1    Y
PRINT "D5:"; (SEN AND &H20)/&H20, "Pin 12 PE (reads inverted)          1    Y
PRINT "D6:"; (SEN AND &H40)/&H40, "Pin 10 /ACK (reads inverted)        1    Y
PRINT "D7:"; (SEN AND &H80)/&H80, "Pin 11 BUSY (reads inverted)       1    Y
PRINT " Then read CONTROL bits"
CTL = INP(CONTROL)
PRINT "D0:"; (CTL AND &H1)/&H1, "Pin 01 /STROBE (reads inverted)      1    Y    0
PRINT "D1:"; (CTL AND &H2)/&H2, "Pin 14 /AUTO FD (reads inverted)      1    Y    0
PRINT "D2:"; (CTL AND &H4)/&H4, "Pin 16 /INIT                            1    Y    1
PRINT "D3:"; (CTL AND &H8)/&H8, "Pin 17 /SLCT IN (to printer)          0    N    1
PRINT "D4:"; (CTL AND &H10)/&H10, "no pin IRQ EN
PRINT "D5:"; (CTL AND &H20)/&H20, "no pin bidirectional (added for EPP)
PRINT "D6:"; (CTL AND &H40)/&H40, "ghost Pin 10 /ACK (reads inverted)    1    Y
PRINT "D7:"; (CTL AND &H80)/&H80, "ghost Pin 11 BUSY (reads inverted)   1    Y
PRINT " Read SPP data:; HEX$(INP(DATAS)); ";";
PRINT " Read EPP address:; HEX$(INP(ADDRESSrw)); ";";
PRINT " Read EPP data:; HEX$(INP(DATArw))
PRINT "One can pulse pin 14 high by writing anything to EPP DATA port."
PRINT "Setting bit 3 in CONTROL low makes pin 17 high.
NCTL = CTL OR BIT5 'sets I/O bit
  WHILE INKEY$ = ""
  SOUND 2000, 1
  FOR I = -32767 TO 32766
  OUT CONTROL, NCTL 'output 1uSec pulses seen during pin 1 high pulse:
  OUT DATArw, &H0 ' high 2.3V@xxxx load from 0.08V low
  OUT DATArw, &HFF ' low 2.4V@xxxx load from 3.5V high
  NEXT I
  WEND
```

Re: OT: DOS programming EPP

```
PRINT "CONTROL PORT INIT: "; HEX$(CTL), "NOW: "; HEX$(INP(CONTROL))  
OUT CONTROL, &HCC 'Does not completely reset...
```

SYSTEM

It is obvious that the data port (pins 2-9) "wants" to be an input, but only when pin 1 is high.

Also, *NO* pin seems to act like an input at *any* time.

I am afraid of using a heavier load, as at all times all pins are active drivers (outputs) except perhaps during the "magic" time.

I certainly do not want to damage my motherboard!

Okay, sorry for confusion. This helps alot. The beyond logic page says

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.

Re: OT: DOS programming EPP

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 Timeout 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.

I don't see you doing this. You don't write xxxx0100 to the control port and you're setting bit 5 when you should be clearing it. Bit 5 sets the direction of the port. EPP wants it forward to start. During operation, you don't need to set the direction of the port. Just read/write to the address(BASE+3) and data (BASE+4) and the EPP takes care of the handshaking and direction. I have a schematic somewhere. I believe I just used the address strobe for the ALE (address latch enable) and the Data strobe for the output enable of my registers. Then NAND the data strobe and address strobe to create the WAIT signal with maybe a couple of more gates to delay it. Then just hang what you want to control on the registers.

The inputs and outputs of IEEE-1284 are defined.
<http://www.fapo.com/1284elec.htm>

I hope this helps.

Paul C

If you look at the documentation in the program, you will note that bit 5 **is** clear by default. For the time being, let us ignore all of the folderol; DC source and DC sink measurements show me that any pin acting as an output has the characteristic of a low power TTL totempole driver.

Re: OT: DOS programming EPP

Refer to the "infinite" loop in the program.
Using a scope triggered from pin 1 and looking at any data pin, if i load that pin with 2.2K, i see that the data pin is not in output mode *only* when CONTROL Bit5 is high (lasts only one microsecond).
Furthermore, and most disturbing, the data pin is not in an input mode that is realistic - not even if one assumes it is emulating a *high power TTL* input! It is worse!

Seems to me that my MB is moused and so is useless for EPP.
What i am going to do is go back to an old computer, dig out a spare parallel card, re-wire the CONTROL decoder so that Bit5 drives the OE pin of the LS374 data latch.
That will make the card electrically and programmatically compatible with the EPP protocol; one just has to do the work in software instead of having it done "automatically".
.