Next Previous Contents

6. Some useful ports

Here is some programming information for common ports that can be directly used for general-purpose TTL (or CMOS) logic I/O.

If you want to use these or other common ports for their intended purpose (e.g., to control a normal printer or modem), you should most likely use existing drivers (which are usually included in the kernel) instead of programming the ports directly as this HOWTO describes. This section is intended for those people who want to connect LCD displays, stepper motors, or other custom electronics to a PC's standard ports.

If you want to control a mass-market device like a scanner (that has been on the market for a while), look for an existing Linux driver for it. The Hardware-HOWTO is a good place to start.

http://www.hut.fi/Misc/Electronics/ is a good source for more information on connecting devices to computers (and on electronics in general).

6.1 The parallel port

The parallel port's base address (called ``BASE'' below) is 0x3bc for `/dev/lp0`, 0x378 for `/dev/lp1`, and 0x278 for `/dev/lp2`. If you only want to control something that acts like a normal printer, see the Printing-HOWTO.

In addition to the standard output-only mode described below, there is an `extended' bidirectional mode in most parallel ports. For information on this and the newer ECP/EPP modes (and the IEEE 1284 standard in general), see http://www.fapo.com/ and http://www.senet.com.au/~cpeacock/parallel.htm. Remember that since you cannot use IRQs or DMA in a user-mode program, you will probably have to write a kernel driver to use ECP/EPP; I think someone is writing such a driver, but I don't know the details.

The port `BASE+0` (Data port) controls the data signals of the port (D0 to D7 for bits 0 to 7, respectively; states: 0 = low (0 V), 1 = high (5 V)). A write to this port latches the data on the pins. A read returns the data last written in standard or extended write mode, or the data in the pins from another device in extended read mode.

The port `BASE+1` (Status port) is read-only, and returns the state of the following input signals:

- Bits 0 and 1 are reserved.
- Bit 2 IRQ status (not a pin, I don't know how this works)
- Bit 3 ERROR (1=high)
- Bit 4 SLCT (1=high)
- Bit 5 PE (1=high)
- Bit 6 ACK (1=high)
- Bit 7 -BUSY (0=high)

The port `BASE+2` (Control port) is write-only (a read returns the data last written), and controls the following

status signals:

- Bit 0 -STROBE (0=high)
- Bit 1 -AUTO_FD_XT (0=high)
- Bit 2 INIT (1=high)
- Bit 3 -SLCT_IN (0=high)
- Bit 4 enables the parallel port IRQ (which occurs on the low-to-high transition of ACK) when set to 1.
- Bit 5 controls the extended mode direction (0 = write, 1 = read), and is completely write-only (a read returns nothing useful for this bit).
- Bits 6 and 7 are reserved.

Pinout (a 25-pin female D-shell connector on the port) (i=input, o=output):

```
1io -STROBE, 2io D0, 3io D1, 4io D2, 5io D3, 6io D4, 7io D5, 8io D6,
9io D7, 10i ACK, 11i -BUSY, 12i PE, 13i SLCT, 14o -AUTO_FD_XT,
15i ERROR, 16o INIT, 17o -SLCT_IN, 18-25 Ground
```

The IBM specifications say that pins 1, 14, 16, and 17 (the control outputs) have open collector drivers pulled to 5 V through 4.7 kiloohm resistors (sink 20 mA, source 0.55 mA, high-level output 5.0 V minus pullup). The rest of the pins sink 24 mA, source 15 mA, and their high-level output is min. 2.4 V. The low state for both is max. 0.5 V. Non-IBM parallel ports probably deviate from this standard. For more information on this, see http://www.hut.fi/Misc/Electronics/circuits/lptpower.html.

Finally, a warning: Be careful with grounding. I've broken several parallel ports by connecting to them while the computer is turned on. It might be a good thing to use a parallel port not integrated on the motherboard for things like this. (You can usually get a second parallel port for your machine with a cheap standard `multi-I/O' card; just disable the ports that you don't need, and set the parallel port I/O address on the card to a free address. You don't need to care about the parallel port IRQ if you don't use it.)

6.2 The game (joystick) port

The game port is located at port addresses 0x200-0x207. If you want to control normal joysticks, you're probably better off using the drivers distributed with the Linux kernel.

Pinout (a 15-pin female D-shell connector on the port):

- 1,8,9,15: +5 V (power)
- 4,5,12: Ground
- 2,7,10,14: Digital inputs BA1, BA2, BB1, and BB2, respectively
- 3,6,11,13: ``Analog'' inputs AX, AY, BX, and BY, respectively

The +5 V pins seem to often be connected directly to the power lines in the motherboard, so they may be able to source quite a lot of power, depending on the motherboard, power supply and game port.

The digital inputs are used for the buttons of the two joysticks (joystick A and joystick B, with two buttons each) that you can connect to the port. They should be normal TTL-level inputs, and you can read their status directly from the status port (see below). A real joystick returns a low (0 V) status when the button is pressed and a high (the 5 V from the power pins through an 1 Kohm resistor) status otherwise.

The so-called analog inputs actually measure resistance. The game port has a quad one-shot multivibrator (a 558 chip) connected to the four inputs. In each input, there is a 2.2 Kohm resistor between the input pin and the multivibrator output, and a 0.01 uF timing capacitor between the multivibrator output and the ground. A real joystick has a potentiometer for each axis (X and Y), wired between +5 V and the appropriate input pin (AX or AY for joystick A, or BX or BY for joystick B).

The multivibrator, when activated, sets its output lines high (5 V) and waits for each timing capacitor to reach 3.3 V before lowering the respective output line. Thus the high period duration of the multivibrator is proportional to the resistance of the potentiometer in the joystick (i.e., the position of the joystick in the appropriate axis), as follows:

R = (t - 24.2) / 0.011,

where R is the resistance (ohms) of the potentiometer and t the high period duration (microseconds).

Thus, to read the analog inputs, you first activate the multivibrator (with a port write; see below), then poll the state of the four axes (with repeated port reads) until they drop from high to low state, measuring their high period duration. This polling uses quite a lot of CPU time, and on a non-realtime multitasking system like (normal user-mode) Linux, the result is not very accurate because you cannot poll the port constantly (unless you use a kernel-level driver and disable interrupts while polling, but this wastes even more CPU time). If you know that the signal is going to take a long time (tens of ms) to go down, you can call usleep() before polling to give CPU time to other processes.

The only I/O port you need to access is port 0x201 (the other ports either behave identically or do nothing). Any write to this port (it doesn't matter what you write) activates the multivibrator. A read from this port returns the state of the input signals:

- Bit 0: AX (status (1=high) of the multivibrator output)
- Bit 1: AY (status (1=high) of the multivibrator output)
- Bit 2: BX (status (1=high) of the multivibrator output)
- Bit 3: BY (status (1=high) of the multivibrator output)
- Bit 4: BA1 (digital input, 1=high)
- Bit 5: BA2 (digital input, 1=high)
- Bit 6: BB1 (digital input, 1=high)
- Bit 7: BB2 (digital input, 1=high)

6.3 The serial port

If the device you're talking to supports something resembling RS-232, you should be able to use the serial port to talk to it. The Linux serial driver should be enough for almost all applications (you shouldn't have to program the serial port directly, and you'd probably have to write a kernel driver to do it); it is quite versatile, so using non-standard bps rates and so on shouldn't be a problem.

See the `termios(3)` manual page, the serial driver source code (`linux/drivers/char/serial.c`), and http://www.easysw.com/~mike/serial/ for more information on programming serial ports on Unix systems.

Next Previous Contents