Many hobbyists seem to harbour a desire to control stepper motors from a PC. Unfortunately, complete stepper motor interfaces that can communicate with a PC are pretty rare. In this article we present a compact circuit that can control unipolar stepper motors. The companion software is suitable for Windows, but it also supports DOS.

Design by Z. Otten

I²C stepper motor controller

a unipolar stepper motor under digital control

This design comes originally from the Dutch HCC Model Railroad Automation Users Group. They needed a simple interface to allow the smokestacks of model locomotives to be filled with a few drops of smoke liquid from a dispenser. This liquid produces a realistic plume of smoke as the locomotive thunders over the rails.
The mechanical actuation in this case is provided by an inexpensive four-phase stepper motor, scavenged from an old floppy-disk drive. The interface between the stepper motor and the serial port of the PC is constructed using standard components. A circuit trick is used to allow the serial port to support the I²C interface, which is a simple and efficient solution.
The necessary software has been developed in Delphi 3.0 and supports a number of basic commands, such as rotating left or right for a defined number of steps.

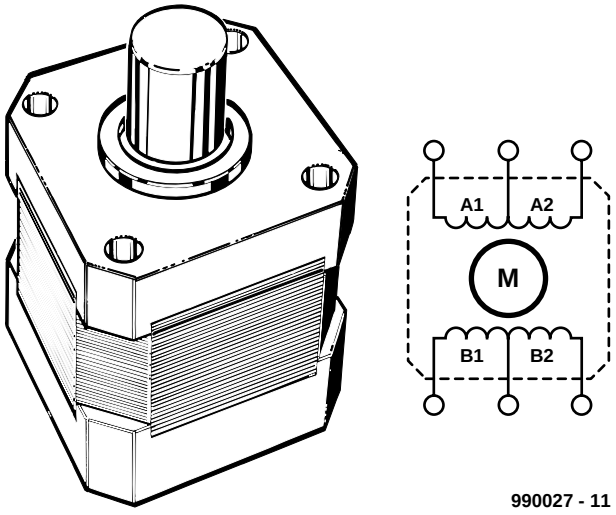


Figure 1. The basic design of a unipolar stepper motor.

Set by step

Stepper motors are DC motors which allow the amount of rotation of the shaft to be precisely controlled. A special interface is necessary for this — a stepper motor controller. As already mentioned, the present circuit is designed to work with four-phase, unipolar stepper motors. Such motors have centre-tapped windings, so that the magnetic field can be reversed without reversing the direction of current flow in the controller. The basic design of such a motor is shown in **Figure 1**. The four windings are activated in turn by a series of pulses, as shown in **Figure 2**. This causes the shaft of the motor to turn. The direction of rotation (left or right) depends on the order in which the pulses are applied to the windings, and the speed of rotation depends on the pulse repetition rate. In other words, the direction and speed of rotation of a stepper motor can be digitally controlled. Since the pulses overlap, the motor is never without power, so a separate brake is not necessary.
The DC resistance of the windings, and the resulting current flow, depends strongly on the particular model of stepper motor. This is equally true of the value of the operating voltage. To allow the controller to be used with a wide variety of stepper motors, the output stage is designed using TIP112 Darlington transistors. These can switch currents

up to 1 ampère. The motor voltage can range from 5 V to 25 V without presenting any problems to the controller, so that all commonly-available stepper motors can be used. This brings us to the circuit diagram of the controller, which is depicted in full in **Figure 3**. As you can see, it is a simple but effective design.

A simple controller

Since the output transistors are driven by a simple, repetitive set of waveforms, the logic circuitry can be made relatively simple. Only three ICs are used. IC2, a 74HC(T)193, is a binary counter that is driven by the clock signal generated by IC1 (PCF8574). A rising edge on pin 5 of IC2 (Count Up) increases the counter state by one, while a rising edge on pin 4 (Count Down) decreases the count by one. The Qa and Qb outputs (pins 3 and 2) thus count from one to four, which is precisely the four states that we need for the four phases of the motor drive. The IC following IC2 (IC3, a 74HC(T)155) is a 2-to-4 decoder. Only one of its four outputs is active at any given time, according to the binary code applied to its inputs. This information now has to be converted to the drive signals for the transistors. **Figure 4** shows how the control signals A, B, C and D are generated with the help of four NAND gates. High-level output signals from the NAND gates turn on the associated output transistors to apply power to the stepper motor.

You can use LS, HC or HCT ICs for IC2 and IC3. However, use only an HC or HCT type for IC4, since an LS type cannot provide an adequate amount of base current.

The I²C circuit is built in an unconventional manner. The heart of this circuit is formed by IC1, a PCF8574. Note that there is an alternative version available, the PCF8574A. These differ in the base address (40_H and 70_H, respectively).

The clock signal alerts the PC via the TxD line. Bidirectional data communication, as used on the SDA line, is implemented with the help of the CTS and DTR signals. Using the serial port for this interface is particularly attractive, since good support is available in software development environments such as Visual Basic and Delphi.

Resistors R4 and R5, in combination with diodes D1 and D2, serve to make the voltage at the serial port TTL-compatible. The combined CTS and DTR lines are connected to the SDA input of IC1, and TxD is connected to SCL. The base

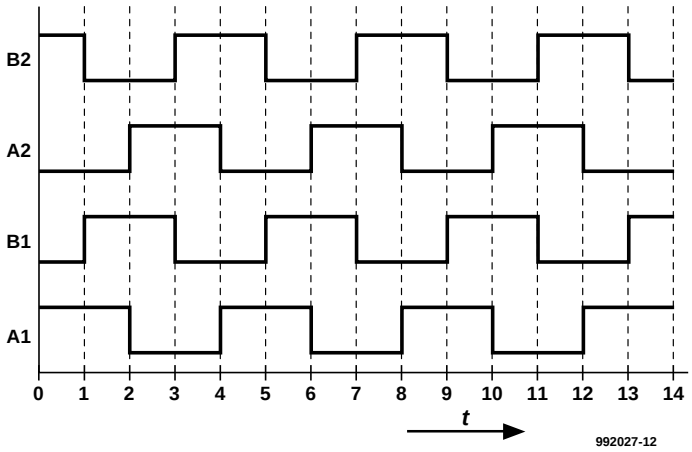
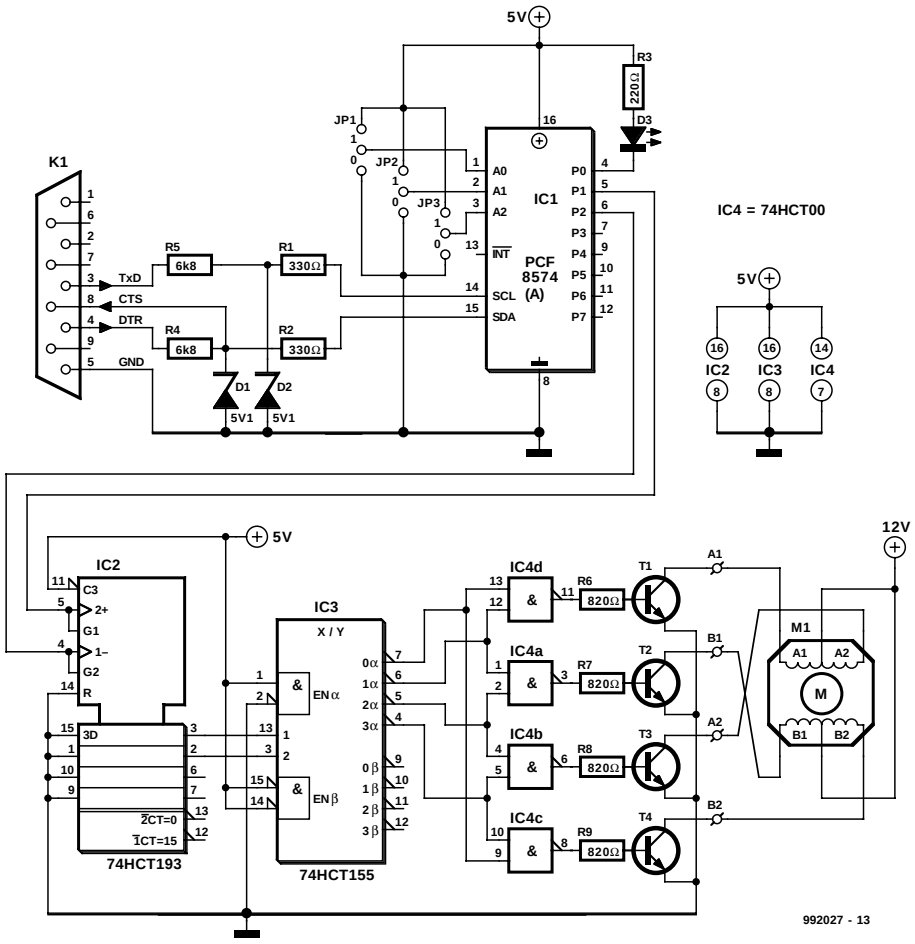


Figure 2. This timing diagram shows how a stepper motor must be driven to cause the shaft to rotate.

address of IC1 can be selected using jumpers JP1 through JP3. This allows up to eight such controllers to be connected in parallel to a single I²C bus. In theory, there are thus a maximum of (2 x 8 x 8) = 128 I/O lines (parallel connections) available. For this application, a single controller is (for the time being) adequate, and only three of its parallel connections are used. Output P0 of IC1 is used to drive a LED, while outputs P1 and P2 are used to supply the

Figure 3. The schematic diagram of the stepper motor controller that is driven via the I²C port.



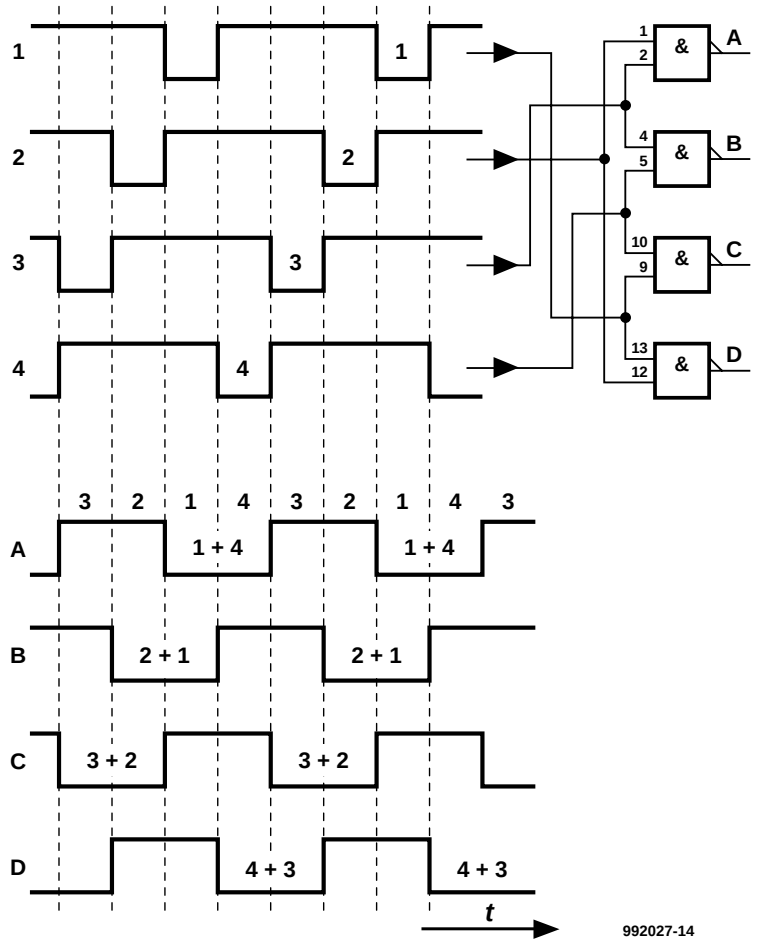
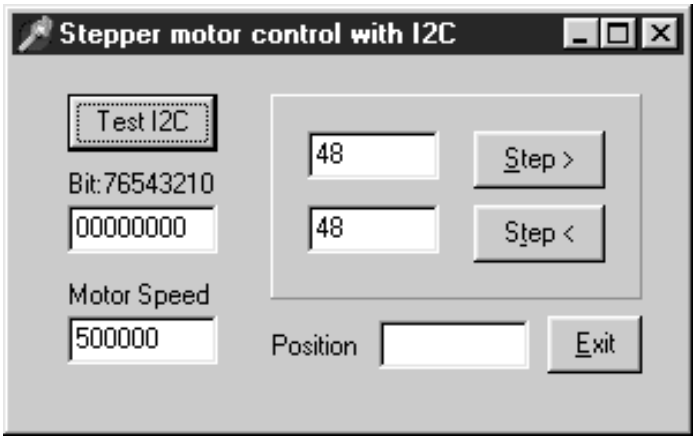


Figure 4. Here you can clearly see how the signals from the 2-to-4 converter are processed by the NAND gates to generate signals suitable for driving the output transistors.

left/right rotation control information for the motor.
The entire circuit operates from a 5 V supply voltage, with a current consumption of around 90 mA. Such a supply voltage can easily be arranged in a PC environment. The operating voltage for the motor depends on the type of motor used. Almost any type of DC power supply can be used to power the motor. The current consumption of the motor depends on the DC winding resistance, which can easily be measured with a multimeter.

Figure 5. Windows interface. The source code file is available on disk no. EPS 996014-1.



Bits and bytes

The software for driving the interface has been developed in Delphi 3.0 and can be used in combination with Windows 95/98. The driver can be readily used with other (possibly home-brew) software. It includes the routines needed to generate signals that comply with the I²C protocol, and to read data from and write data to IC1. Outputs P0–P7 of IC 12 can be directly loaded with an 8-bit value. If output P1 is set low and then high in turn, the motor rotates one step to the left. Similarly, output P2 can be used to make the motor rotate one step to the right.
The rate at which the control pulses are generated determines the rotational speed of the motor. The processing speed of the PC is a factor here; the faster the PC, the higher the maximum possible speed of the motor. To avoid problems due to this effect, a 'motor constant' is defined in the software. The value of this constant must be chosen so that enough pulses are generated in each time interval to allow the motor to reach the maximum desired speed.
The screen dump in **Figure 5** shows the user interface for the driver software (VPLS1.EXE). The left-hand portion of the window relates to the communication with the I²C interface, while the right-hand portion relates to the motor. Activating the button TestI2C should cause the LED on the controller to switch on. The status of the outputs P0 through P7 is displayed in a box; this represents the content of the binary status byte in IC1. The previously-mentioned motor constant can be read and modified via the box at the bottom. The remainder of the interface provides two pushbuttons that allow the motor to be stepped to the left or right by the indicated number of steps, plus a box that displays the current position of the motor (based on the number and direction of steps that it has executed).
Since stepper motors are most commonly used as part of a larger system, it is important that the driver software can be incorporated in other applications. The necessary software, including the source code, is thus available on a diskette from Readers Services (order number **996014-1**). This provides the tools you need to control stepper motors from your own applications.

(992027-1)