

Figure 29.20: Interrupt-based Unipolar Stepper Drive Functions

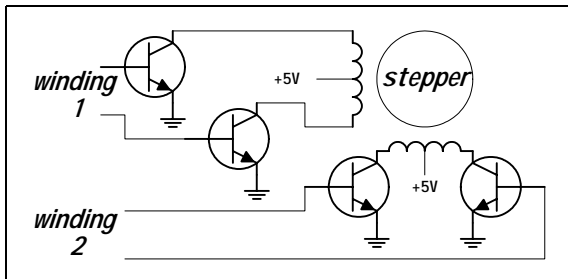


Figure 29.21: Unipolar Stepper Drive

were shown in Figure 29.15 and Figure 29.16. While bridges are even available in pairs in single chips, such drivers are not as readily available as discrete transistors.

Unipolar motors are much easier to interface, as shown in Figure 29.21. The two windings of the bipolar motor each have a center tap; they can be recognized by having 6 leads total, and by tying the center tap to a positive voltage, current can flow from the center tap either forward through one half of the winding or reverse through the other half, depending on which end of the winding is grounded. Since the current is flowing out of the winding in either case, the drive can be as simple as two transistors. A Uni-

polar motor is somewhat less efficient in utilization of copper because the current is going through only half the winding at any one time.

Many applications require more torque than is readily obtained in a small stepper motor. Rather than pursuing a larger stepper with very high current requirements, consider smaller stepper motors with gear reductions. Many applications (stepper or DC motor) do not need *both* high speed *and* high torque. Also, many low-cost motors are available from electronic salvage houses if your project is not intended for volume production.

More Stepper Drive Techniques

Ramping Without going exhaustively into stepper motor drive techniques, I will point out that stepper motors can step faster than the starting rate if the rate is *ramped* up and down to overcome the rotor inertia. For example, a motion could start at a step delay of 10mS and speed up to a step delay of 5mS after a few steps.

Series Resistor You can also get a higher stepping

rate by using a higher-than-rated voltage with current limiting resistors; when the current has not built up, all the voltage appears across the inductance of the winding, but as the current increases, the voltage divides across the resistor. The higher voltage overcomes the winding inductance more quickly.

Dynamic Braking It is possible to get the rotor settled at a new position more quickly by issuing a reverse step some time after the last step and then reissuing a forward step. The reverse field provides dynamic braking. The actual timing of this extra step depends on the dynamics of the system.

Micro-stepping A drive method exists called micro-stepping, in which the windings are driven by varying (analog) currents to produce a net field *anywhere* in between the normal poles. The result is the ability to position the rotor anywhere, not just at one of the 24 or 48 step positions. I have no details on a specific application, but it would probably require precision analog drives, pulse width modulated digital drive, or position feedback.

Stepper Motor Drive Software

It happened that stepper motors were used in the first development software example in Chapter 7 (page 57) and in the modular development examples of Chapter 8 (page 67). Those examples all relied on a time delay and represent a more primitive solution than is possible using interrupts and timers, as will be shown next.

Figure 29.22 and Figure 29.23 show a more sophisticated program to drive a stepper motor, with the flowcharts in Figure 29.20. For something to do, the program produces a back-and-forth motion of a stepper motor. Instead of using a software loop, the delay between steps comes from the 1mS timer2 interrupt function taken from Figure 11.9 on page 118. The details of the stepper winding interface are buried in the driver routines. In this case the full-step driver was enabled, but it would be just as easy to use the half-step function or another function to interface a more sophisticated step/direction driver IC.

Registers & variables Figure 29.22 shows the 16-bit register definitions for timer 2 and the configuration function that initializes the hardware.

Alternate Stepper Drivers The first two functions

```
#include <C8051F020.h> // Register definition file
#pragma NOAREGS
#define uchar unsigned char
#define cw 1
#define ccw 0sfr16 RCAP2=0xca;
sfr16 TREG2=0xcc;
unsigned int msec; // global time counter
uchar dir, speed, speedcnt, steps;
bit stepdone;
void config (void) {
    WDTCN = 0x07;
    WDTCN = 0xDE; // Disable WDT
    WDTCN = 0xAD;
    OSCICN = 0x07; // Internal Osc. 16MHz
    XBR2 = 0x40; // XBAR2
    POMDOUT = 0x0F; // configuration for P0
    // P0.0-3 are stepper drives (Push-Pull Output)
    RCAP2 = -1174; // Timer 2 reload for 1mS
    TREG2 = -1174; // Timer 2 value
    T2CON = 0x04; // auto-reload enabled
    IE = 0xA0; // Enable T2int & EA
}
/* direct drive full steps (24/rev) */
void fullstep (uchar dir) {
    static uchar phase[4] = {5, 9, 0xa, 6};
    static uchar pindex;
    if (dir == ccw) pindex = ++pindex & 3;
    else pindex = --pindex & 3;
    P0 = P0 & 0xf | phase[pindex];
}
/* direct drive for half stepping (48/rev) */
// void halfstep (uchar dir) {
//     static uchar phase[8] = {5, 4, 9, 8, 0xa, 2, 6, 4};
//     static uchar pindex;
//     if (dir == ccw) pindex = ++pindex & 7;
//     else pindex = --pindex & 7;
//     P0 = P0 & 0xf | phase[pindex];
// }
void T2msecint (void) interrupt 5 using 1 {
    TF2 = 0; // clear timer overflow bit
    msec++;
    if (stepdone == 0) { // taking steps
        if (--speedcnt == 0) { // decrement every mS
            speedcnt = speed; // set next step time
            fullstep (dir); // take the step
            if (--steps == 0) stepdone = 1; // finished
        }
    }
}
}
```

Figure 29.22: Unipolar Stepper Drive Functions: 1

of Figure 29.22 work through a table of phase patterns, as in Figure 29.18. Static variables are local but remain until the next use of the function. The