8

CONTROLLING MOTORS WITH MOSFETS

AVRs cannot directly control motors. In order to enable this, we need to use external components: *metal-oxide-semiconductor field-effect transistors (MOSFETs)*, transistors that can switch or amplify voltages in circuits.

In this chapter, you'll learn how to:

- Use PWM with MOSFETs to control DC motors.
- Use MOSFETs to control larger currents.
- Use motor driver ICs to interface larger motors with your AVR microcontrollers.

Along the way, you'll build a temperature-controlled fan and a two-wheel-drive robot vehicle, building on prior knowledge to complete more interesting and complex projects. By the end of the chapter, you'll have the skills to begin using MOSFETs in your own projects both for fun and for more serious applications, such as robotics, automation, or toys.

The MOSFET

We use MOSFETs when we need to control large currents and voltage using a small signal, such as that from a microcontroller's digital output pin. MOSFETs are available in various sizes, such as those shown in Figure 8-1, to match different projects' requirements.

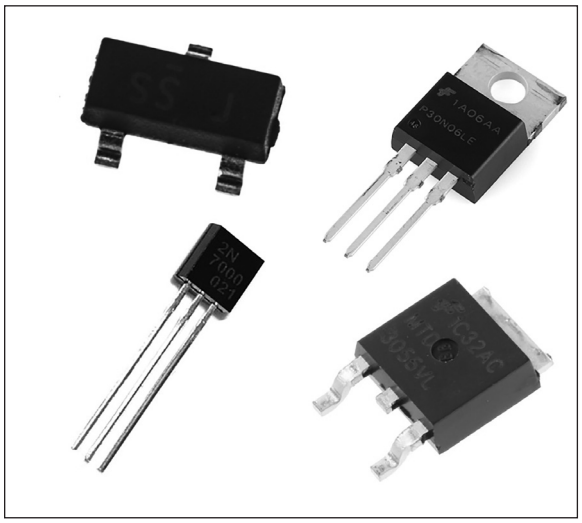

Figure 8-1: Various MOSFETs

We will be using the small 2N7000 version shown at the bottom left in Figure 8-1, which has three leads. When you're looking at the front of the 2N7000—the flat-faced-side—the pins are, from left to right, the source, gate, and drain pins (I'll explain their functions momentarily).

Figure 8-2 shows the schematic symbol for the 2N7000 MOSFET.

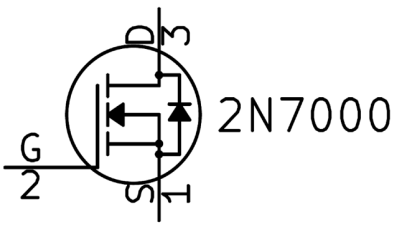

Figure 8-2: Schematic symbol for 2N7000 MOSFET

It's easy to operate a MOSFET. When you apply a small current to the gate pin, a large current can flow in through the drain pin and out through the source pin. You can also connect a PWM signal to the gate pin of a MOSFET, allowing you to control lights, motors, and more in various ways. That's what we'll focus on in this chapter.

Our 2N7000 MOSFET can handle up to 60 V DC at 200 mA continuously, or 500 mA in bursts. When choosing a MOSFET for your projects, be sure to check the voltage and current maximums against the signal you want to switch.

We connect a 10 kΩ resistor between the MOSFET's gate and the source pin every time we use a MOSFET, as you'll see in the following project. This acts to keep the gate switched off when a current is not applied to it, in the same way a resistor pulls down a button, as shown in Chapter 3; it stops the MOSFET from turning slightly on or off at random.

Project 34: DC Motor Control with PWM and MOSFET

This project demonstrates how to control a small DC motor using PWM and a MOSFET. As the microcontroller cannot provide enough current for the motor on its own, we use an external power supply and a MOSFET to handle the motor's requirements.

The Hardware

For this project, you'll need the following hardware:

- USBasp programmer
- Solderless breadboard
- ATmega328P-PU microcontroller
- Jumper wires
- Small DC motor and matching power
- 2N7000 MOSFET
- 10 kΩ resistor

A small DC motor model like the one shown in Figure 8-3 with a maximum of 12 V DC will suffice.

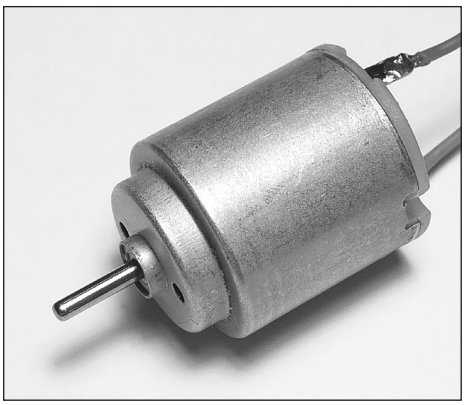

Figure 8-3: Small DC motor

You will also need external power, such as a battery pack that holds several AA cells. A 6 AA cell pack like the one shown in Figure 8-4 will provide up to 9 V DC, enough power to run a 12 V DC motor nicely.

Figure 8-4: AA battery pack

Assemble the circuit as shown in Figure 8-5. Note that the black/negative lead from the battery pack will be connected to GND.

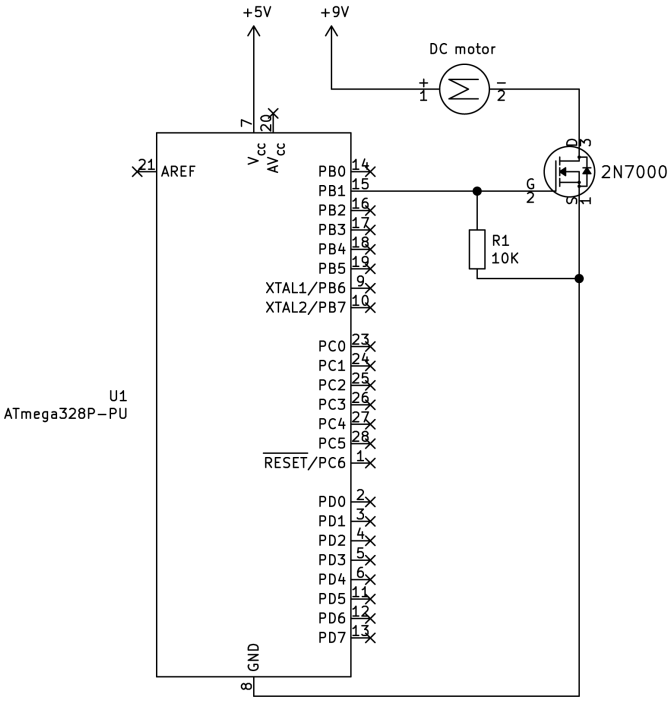

Figure 8-5: Schematic for Project 34

Don't disassemble the circuit once you've finished using it, as you will use it as part of the circuit in the following project.

The Code

Open a terminal window, navigate to the *Project 34* subfolder of this book's *Chapter 8* folder, and enter the command **`make flash`**. The DC motor should start from zero, increase speed to a complete on state, then reduce speed back to a complete off state, then repeat the procedure.

Let's see how this works:

```
// Project 34—DC Motor Control with PWM and MOSFET

#include <avr/io.h>
#include <util/delay.h>

❶ #define wait 10
❷ void initPWM(void)
{
   // Timers 1A and 1B
   TCCR1A |= (1 << WGM10);        // Fast PWM mode
   TCCR1B |= (1 << WGM12);        // Fast PWM mode
   TCCR1B |= (1 << CS11);
}

void motorOn(void)
{

 ❸ TCCR1A &= ~(1 << COM1A1);      // Disconnect PWM from PB1
   PORTB |= (1 << PORTB1);        // Set PB1 on
}

void motorOff(void)
{
 ❹ TCCR1A &= ~(1 << COM1A1);      // Disconnect PWM from PB1
   PORTB &= ~(1 << PORTB1);       // Set PB1 off
}

void motorPWM(uint8_t duty)
{
 ❺ TCCR1A |= (1 << COM1A1);       // Connect PWM to OCR1A—PB1
   OCR1A = duty;
}

int main(void)
{
   DDRB |= (1 << PORTB1);         // Set PORTB pin 1 as output
 ❷ initPWM();
   uint8_t a;

   while(1)
   {
      motorOff();                 // Motor off
      _delay_ms(3000);

      for (a = 1; a <255; a++)    // Slowly increase motor speed
      {
         motorPWM(a);
         _delay_ms(wait);
      }

      motorOn();                  // Motor full on
      _delay_ms(1000);
```

```
      for (a = 254; a > 0;--a)   // Slowly decrease motor speed
      {
         motorPWM(a);
         _delay_ms(wait);
      }
   }
}
```

You should be familiar with the code used in this project by now as you learned about using PWM in Chapter 7, but let's go through it together. First, we set the required registers to initialize PWM operation ❷. To make control easier, we use three functions: `motorOn()`, `motorOff()`, and `motorPWM()`. The `motorOn()` function turns the motor completely on by first disconnecting PORTB1 from PWM ❸ and then setting it to high. This gives 100% power to the motor via the MOSFET at all times.

We use the `motorOff()` function to completely turn the motor off by disconnecting PORTB1 from PWM ❹ and setting it to low. This turns off the MOSFET gate pin, so the motor has no power. Again, this is necessary as you can't send a 0 percent duty cycle to the OCR1A register and expect it to be off 100 percent of the time. Even with a duty cycle of 0 percent, every time the hardware timer resets the output is turned on briefly during the reset.

Finally, the function `motorPWM()`, which accepts the required duty cycle value, is used to set the motor speed with PWM. It connects PORTB1 to PWM ❺ and then loads the OCR1A register with the required value.

Our main code repeatedly turns the motor on and increases the speed to 100 percent, then reduces the speed back to 0, then turns the motor off for 3 seconds. We turn the motor off at the start of the code, to allow the end user a moment's notice before spinning it up. You can change the delay time in the PWM loops by altering the value of `wait` ❶.

Now that you know how to control a DC motor, let's apply this skill to a practical example by building a temperature-controlled fan system.

Project 35: Temperature-Controlled Fan

In this project, you'll combine your existing knowledge of motor control with your newfound MOSFET skills, using temperature sensors to make a temperature-controlled fan.

The Hardware

For this project, you'll need the following hardware:

- USBasp programmer
- Solderless breadboard
- ATmega328P-PU microcontroller
- Jumper wires
- Small DC motor and matching power

- 2N7000 MOSFET
- TMP36 temperature sensor
- 0.1 µF ceramic capacitor
- 10 kΩ resistor

You can use the small DC motor from the previous project just to see how this works, or you can pick up a DC motor–powered cooling fan from an electrical retailer, such as the unit from PMD Way (part number 59119182) shown in Figure 8-6. Some fans may have four wires, but only two of these are required (power and GND). Once again, we'll need to use external power for the fan.

Figure 8-6: DC cooling fan

Assemble the circuit as shown in Figure 8-7.

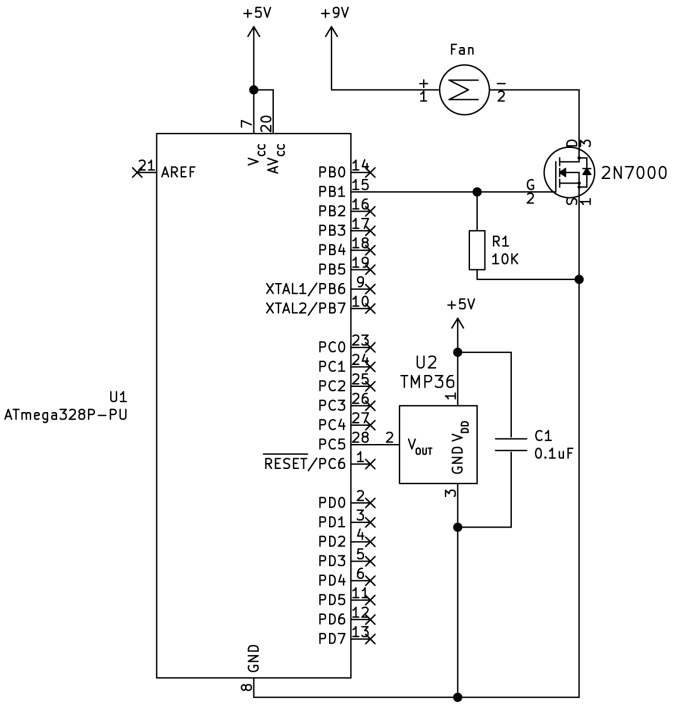

Figure 8-7: Schematic for Project 35

As you assemble the project, note that the black/negative lead from the battery pack or fan power supply will be connected to GND. Also, don't forget to connect AV_{CC} to 5 V.

The Code

Open a terminal window, navigate to the *Project 35* subfolder of this book's *Chapter 8* folder, and enter the command **`make flash`**. Once you've applied power, the project should wait three seconds before taking the temperature and operating the fan, depending on the current temperature.

To see how this works, take a look at the code:

```
// Project 35—Temperature-Controlled Fan

#include <avr/io.h>
#include <util/delay.h>

void startADC()
{
   ADMUX |= (1 << REFS0);                 // Use AVcc pin with ADC
   ADMUX |= (1 << MUX2) | (1 << MUX0);    // Use ADC5 (pin 28)
   ADCSRA |= (1 << ADPS1) | (1 << ADPS0); // Prescaler for 1 MHz (/8)
   ADCSRA |= (1 << ADEN);                 // Enable ADC
}

void initPWM(void)
{
   // Timers 1A and 1B
   TCCR1A |= (1 << WGM10);                // Fast PWM mode
   TCCR1B |= (1 << WGM12);                // Fast PWM mode
   TCCR1B |= (1 << CS11);
}

void motorOff(void)
{
   TCCR1A &= ~(1 << COM1A1);              // Disconnect PWM from PB1
   PORTB &= ~(1 << PORTB1);               // Set PB1 off
}

void motorOn(void)
{
   TCCR1A &= ~(1 << COM1A1);              // Disconnect PWM from PB1
   PORTB |= (1 << PORTB1);                // Set PB1 on
}

void motorPWM(uint8_t duty)
{
   TCCR1A |= (1 << COM1A1);               // Connect PWM to OCR1A—PB1
   OCR1A = duty;
}

int main(void)
{
   DDRB |= (1 << PORTB1);                 // Set PORTB pin 1 as output
```

```
❶ startADC();
  initPWM();
❷ uint8_t ADCvalue;
  float voltage;
  float temperature;

  // Delay motor action for a few moments on start
❸ _delay_ms(3000);

  while(1)
  {
     // Get reading from TMP36 via ADC
   ❹ ADCSRA |= (1 << ADSC);              // Start ADC measurement
     while (ADCSRA & (1 << ADSC) );      // Wait until conversion complete
     _delay_ms(10);

     // Get value from ADC register, convert to 8-bit value
     ADCvalue = ADC >> 2;

     // Convert reading to temperature value (Celsius)
     voltage = (ADCvalue * (5000 / 256));
   ❺ temperature = (voltage-500) / 10;

     // Now you have a temperature value, take action
   ❻ if (temperature<25)
     {
        // Under 25 degrees, turn motor off
        motorOff();
     }
   ❼ else if ((temperature>=25) & (temperature <35))
     {
        // At or above 25 and below 35 degrees, set motor to 50% PWM
        motorPWM(127);
     }
   ❽ else if (temperature>=35)
     {
        // 35 degrees and over, turn motor full on
        motorOn();
     }
   ❾ _delay_ms(500); // Prevent rapid motor speed changes
  }
}
```

This code builds upon the ADC and temperature sensor from Project 19 in Chapter 4 and the PWM motor control used in Project 34. First, we activate the ADC to read the TMP36 temperature sensor and activate PWM for variable-speed motor control ❶ (the `startADC()` and `initPWM()` functions are defined at the beginning of the program). We introduce the variables required to calculate the temperature for the thermostat ❸, and then we introduce a delay at startup so the motor doesn't jump into life straight after a reset or power-up ❹.

In the main loop, we take the value from the ADC ❹ and convert it to degrees Celsius ❺. The code can now use this temperature value to

determine whether to operate the motor. In this project, the motor is switched off if the temperature is below 25 degrees ❻. If the temperature is between 25 and 34 degrees inclusive, the fan runs at half speed ❼. If the temperature is 35 degrees or over, the fan runs at full speed ❽.

Finally, after checking the temperature, there is a short delay ❾ to avoid *hysteresis*—that is, rapid changes in the characteristics of the circuit. For example, if the sensor were in the path of a breeze or a fluttering curtain, the temperature might fluctuate rapidly between 24.99 and 25 degrees, causing the motor to turn continuously on and off. The delay allows us to avoid this.

At this point, I hope you're beginning to see how we can combine basic AVR code and tools to solve new problems. Building on prior knowledge, we've started to move beyond the simpler projects in earlier chapters to more complex, practical applications.

Now that we've experimented with basic motor control using the MOSFET, we'll move on to controlling the direction of rotation as well as the speed of a DC motor. To do this, we'll use the L293D motor driver IC.

The L293D Motor Driver IC

To control the speed and direction of one or two small DC motors, we'll use the L293D motor driver IC from STMicroelectronics, shown in Figure 8-8. This is in the same type of package as a microcontroller, and thus we can easily use it in a solderless breadboard for experimenting.

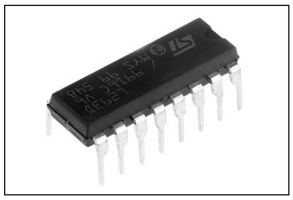

Figure 8-8: L293 motor driver IC

You can use small motor driver ICs like the L293D for robotics or small toys that run from 4.5 to 36 V DC at up to 600 mA, with some restrictions with regard to heat that I'll explain later. The L293D saves you a lot of time, as it takes care of distributing power to the motors and spares you from building a bunch of external circuitry. It is known as an *H-BRIDGE IC* because it has an internal circuit of MOSFETs and other components configured in the shape of the letter H, as shown in Figure 8-9.

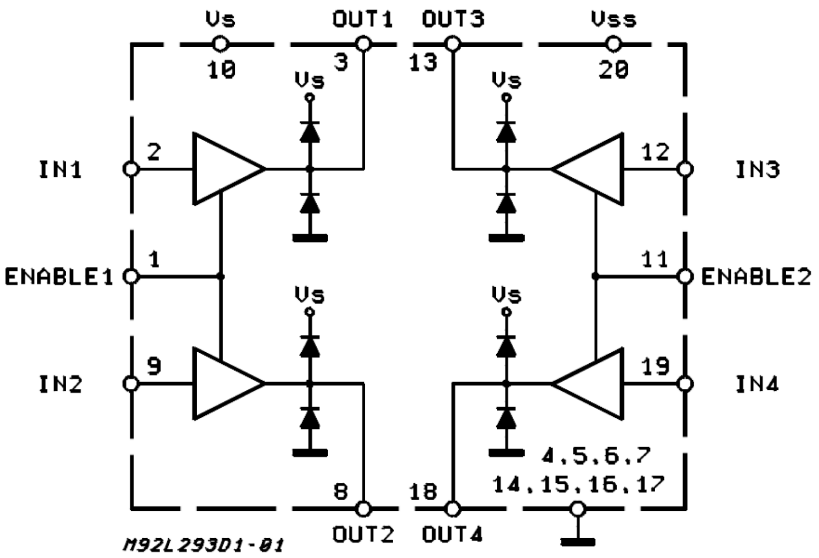

Figure 8-9: L293D IC block diagram

Thankfully, we don't need to build the L293D IC's circuitry ourselves; it's already set up and ready for us to connect the motors, control logic, and power. Instead, we just connect motors, power, GND, and outputs from a microcontroller. To see how to wire up the L293D to one DC motor, take a look at the pinouts in Figure 8-10.

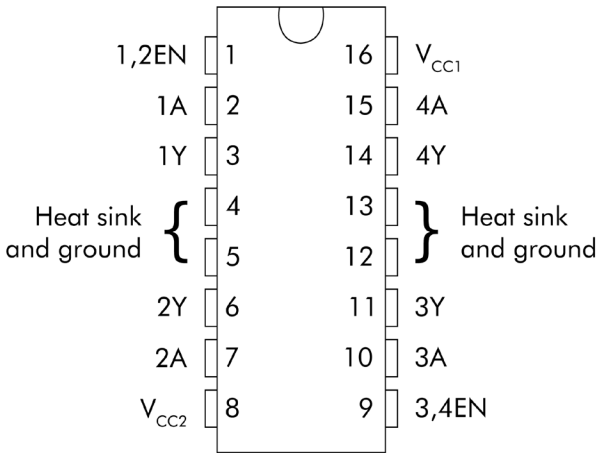

Figure 8-10: L293D IC pinouts

There are four GND pins: 4, 5, 12, and 13. Connect those to GND. Next, locate the two power pins. Connect the first one—$V_{CC}1$, which is the logic (or control) power pin—to the 5 V, as you did with your

microcontroller in our earlier projects. Then connect the second power pin, $V_{CC}2$, to the positive of the motor power supply (up to 36 V DC). Finally, connect the motor: one wire to pin 3 and the other to pin 6.

Controlling the motor requires three signals from digital outputs on our microcontroller. First, we set the ENABLE pin: either to high, so that the driver IC sends power to the motor, or to low, so that the motor stops. The signals from the next two pins, 1A and 2A, control the polarity of the power to the motor, and thus the rotational direction. With ENABLE set high, the motor will rotate in one direction with 1A high and 2A low and rotate in the other direction with 1A low and 2A high. Table 8–1 summarizes all this for easy reference.

Table 8-1: L293D Single Motor Control

ENABLE pin/EN1 (pin 1)	1A pin/out 1 (pin 2)	2A pin/out 2 (pin 7)	Motor action
High	High	Low	Forwards
High	Low	High	Backwards
Low	High or Low	High or Low	Stop

There's no way to tell from the outside whether your motor will run forwards or backwards; you will need to do a test run to determine which of the two 1A/2A combinations is which for your motor. You can alter the speed of the motor by applying a PWM signal to the ENABLE pin.

A FEW WORDS ABOUT HEAT

The L293D can become warm (or hot) when running toward the higher end of its capacity. It shouldn't be used in a solderless breadboard in these situations, as the four GND pins are also used as a heatsink. This means they might melt the plastic around the pins, leaving the L293D stuck in the breadboard. If you're going to control larger motors, build your circuit using your own PCB, use a breakout board for the motor control, or solder the circuit onto a stripboard.

Now that you're familiar with the theory of the L293D, let's put it into practice in the next project.

Project 36: DC Motor Control with L293D

This project demonstrates how you can control a small DC motor using PWM and the L293D motor driver IC, operating the motor in either direction and at various speeds. This will give you the remaining skills you need to build your first moving robot vehicle in the next project.

The Hardware

For this project, you'll need the following hardware:

- USBasp programmer
- Solderless breadboard
- ATmega328P-PU microcontroller
- Jumper wires
- Small DC motor and matching power
- L293D motor driver IC

Use the same DC motor and matching power supply you used for Project 34. Assemble the circuit as shown in Figure 8-11.

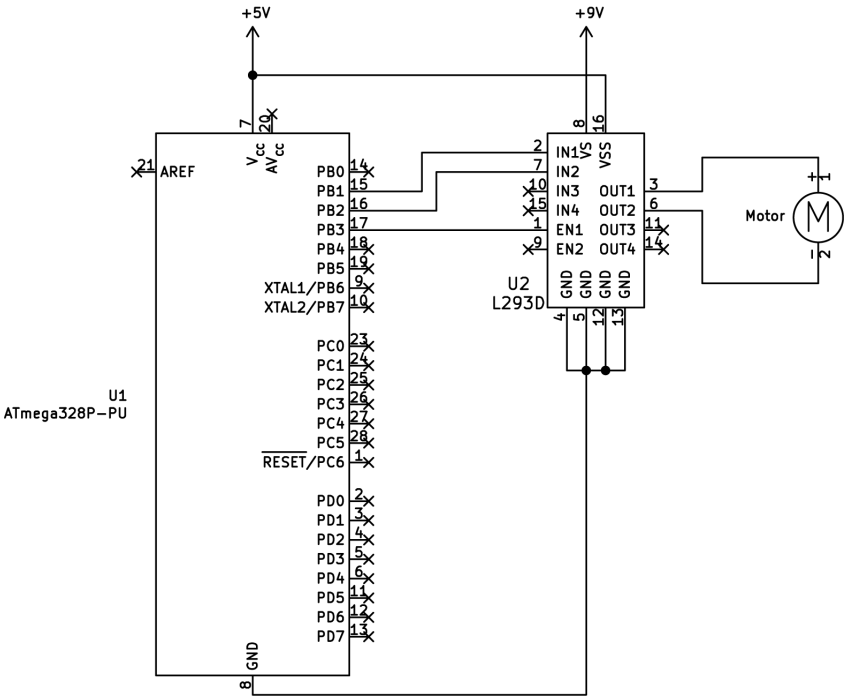

Figure 8-11: Schematic for Project 36

As you assemble the circuit, connect the black/negative lead from the battery pack or external power to GND once again.

The Code

Open a terminal window, navigate to the *Project 36* subfolder of this book's *Chapter 8* folder, and enter the command `make flash`. Once you've applied power, the project should wait three seconds before operating the motor at two different speeds successively, both forwards and backwards.

Let's see how this works:

```
// Project 36—DC Motor Control with L293D

#include <avr/io.h>
#include <util/delay.h>

❶ void initPWM(void)
{
   TCCR2A |= (1 << WGM20);     // Fast PWM mode
   TCCR2A |= (1 << WGM21);     // Fast PWM mode, part 2
   TCCR2B |= (1 << CS21);      // PWM Freq = F_CPU/8/256
}
❷ void motorForward(uint8_t duty)
{
   // Set direction
 ❸ PORTB |= (1 << PORTB1);     // PB1 HIGH
   PORTB &= ~(1 << PORTB2);    // PB2 LOW

   // Set speed
 ❹ if (duty == 255)
   {
      PORTB |= (1 << PORTB3);  // Set PORTB3 to on
   } else if (duty < 255)
   {
    ❺ TCCR2A |= (1 << COM2A1); // PWM output on OCR2A—PB3
      OCR2A = duty;            // Set PORTB3 to PWM value
   }
}

❻ void motorBackward(uint8_t duty)
{
   // Set direction
   PORTB &= ~(1 << PORTB1);    // PB1 LOW
   PORTB |= (1 << PORTB2);     // PB2 HIGH

   // Set speed
   if (duty == 255)
   {
      PORTB |= (1 << PORTB3);  // Set PORTB3 to on
   } else if (duty < 255)
   {
      TCCR2A |= (1 << COM2A1); // PWM output on OCR2A—PB3
      OCR2A = duty;            // Set PORTB3 to PWM value
   }
}

❼ void motorOff(void)
{
   // Disconnect PWM output from OCR2A—PB3
   TCCR2A &= ~(1 << COM2A1);
   // Set ENABLE to zero for brake
   PORTB &= ~(1 << PORTB3);
}

int main(void)
```

```
{
   // Set PORTB3, 2, and 1 as outputs
   DDRB |= (1 << PORTB3)|(1 << PORTB2)|(1 << PORTB1);
❽ initPWM();
   _delay_ms(3000);            // Wait a moment before starting
   while(1)
   {
    ❾ motorForward(64);
      _delay_ms(2000);
      motorOff();
      _delay_ms(2000);
      motorForward(255);
      _delay_ms(2000);
      motorOff();
      _delay_ms(2000);
      motorBackward(64);
      _delay_ms(2000);
      motorOff();
      _delay_ms(2000);
      motorBackward(255);
      _delay_ms(2000);
      motorOff();
      _delay_ms(2000);
   }
}
```

This code builds on that of previous motor control projects in this chapter, with the required additions for the L293D. We set up PWM at points ❶ and ❽. The first of the motor control functions, `motorForward()` ❷, rotates the motor in one direction and accepts a duty cycle value of between 1 and 255. Per Table 8–1, we set the outputs as high and low for motor directional control ❸. The code then checks if the required duty cycle value is 255 ❹, and if so simply switches the ENABLE pin to high for full-speed motor running instead of using PWM. However, if it's less than 255, then PWM is enabled for the output pin controlling the L293D ENABLE pin ❺ and the required duty cycle value is dropped into OCR2A.

The motor control method used in `motorForward()` is repeated with the function `motorBackward()` ❻, except with the outputs for motor control set to low and high for reverse rotation. Finally, the `motorOff()` function ❼ turns off the motor by first disabling PWM for the output pin controlling the L293D ENABLE pin and then setting it to low. With all this complete, you can now use the motor control functions to control the speed and direction of motor rotation, as demonstrated in the main loop of the code ❾.

Now that you know how to control the speed and direction of a DC motor, let's use two motors to control a small robot vehicle.

Project 37: Controlling a Two-Wheel-Drive Robot Vehicle

In this project, you'll learn to control a small two-wheel-drive robot vehicle. The suggested hardware includes two DC motors and a *castor* (a small,

swiveling wheel fixed to the bottom of your robot vehicle), allowing you to easily control the speed and direction of travel. I hope this inspires you to create your own more complex robotic creations!

The Hardware

For this project, you'll need the following hardware:

- USBasp programmer
- Solderless breadboard
- ATmega328P-PU microcontroller
- Jumper wires
- Two small DC motors and matching power
- 2WD robot vehicle chassis (such as PMD Way part number 72341119)
- Four AA battery cells
- 1N4004 power diode
- L293D motor driver IC

The Chassis

The foundation of any robot vehicle is a solid chassis containing the motors, drivetrain, and power supply. You can choose from many chassis models available on the market. To keep things simple, this project relies on an inexpensive robot chassis with two small DC motors that operate at around 6 V DC and two matching wheels, as shown in Figure 8-12.

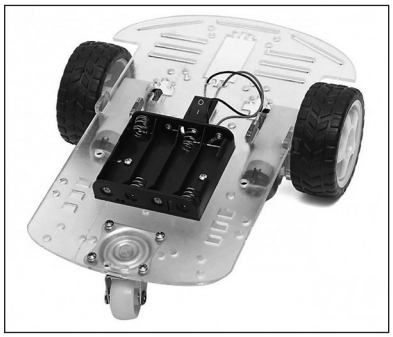

Figure 8-12: Two-wheel-drive robot vehicle chassis (PMD Way part number 72341119)

The task of physically assembling the robot chassis varies between models, but most require a few additional tools beyond those included in the kit, such as screwdrivers. If you haven't settled on a final design and wish to get your robot moving in a temporary configuration, you can attach the electronics to the chassis with a reusable putty adhesive like Blu-Tack.

The Power Supply

The motors included with the robot chassis typically operate at around 6 V DC, so we'll use the 4 AA cell battery holder included with the example chassis in Figure 8-12. We can't use 6 V to power the microcontroller circuit, so we place a 1N4004 diode between the power supply positive and the 5 V connection on the microcontroller. The diode will cause a 0.7 V drop in voltage, bringing the microcontroller supply to around 5.3 V DC. The voltage will again drop as the battery life decreases.

Assemble the circuit as shown in Figure 8-13.

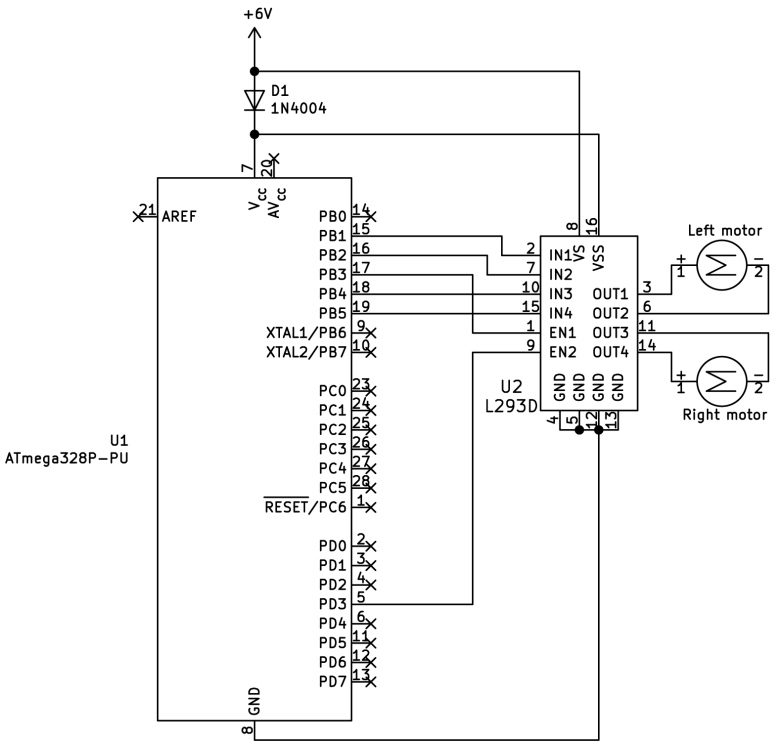

Figure 8-13: Schematic for Project 37

Again, the black/negative lead from the battery pack or external power connects to GND, and the red/positive lead runs to both the L293D $V_{CC}2$ pin and the 1N4004 diode.

The Code

Open a terminal window, navigate to the *Project 37* subfolder of this book's *Chapter 8* folder, and enter the command **`make flash`**. Once you remove the AVR programmer and the vehicle starts up, it should wait three seconds before moving off forward and then turning left, right, and so on as directed by the sequence of functions in the main loop of the code.

This code is the culmination of our experiments with DC motor control using the L293D motor controller IC and PWM. Let's see how it works:

```
// Project 37—Controlling a Two-Wheel-Drive Robot Vehicle

#include <avr/io.h>
#include <util/delay.h>

void initPWM(void) ❶
{
   TCCR2A |= (1 << WGM20);                 // Fast PWM mode
   TCCR2A |= (1 << WGM21); );                 // Fast PWM mode, part 2
   TCCR2B |= (1 << CS21); );                 // PWM Freq = F_CPU/8/256
}

void moveForward(uint8_t duty)
{
   // Set direction
   PORTB |= (1 << PORTB4)|(1 << PORTB1); ❷ // PB4,1 HIGH
   PORTB &= ~(1 << PORTB5)&~(1 << PORTB2); // PB5,2 LOW

   // Set speed
   if (duty == 255) ❸
   {
      PORTB |= (1 << PORTB3);              // Set PORTB3 to on
      PORTD |= (1 << PORTD3);              // Set PORTD3 to on
   } else if (duty < 255)
   {
      TCCR2A |= (1 << COM2A1); ❹           // PWM output on OCR2A—PB3
      TCCR2A |= (1 << COM2B1);             // PWM to OCR2B—PD3
      OCR2A = duty;                        // Set PORTB3 to PWM value
      OCR2B = duty;                        // Set PORTD3 to PWM value
   }
}

void moveBackward(uint8_t duty)
{
   // Set direction
   PORTB &= ~(1 << PORTB4)&~(1 << PORTB1); // PB4,1 LOW
   PORTB |= (1 << PORTB5)|(1 << PORTB2);   // PB5,2 HIGH

   // Set speed
   if (duty == 255)
   {
      PORTB |= (1 << PORTB3);              // Set PORTB3 to on
      PORTD |= (1 << PORTD3);              // Set PORTD3 to on
   } else if (duty < 255)
   {
      TCCR2A |= (1 << COM2A1);             // PWM output on OCR2A—PB3
      TCCR2A |= (1 << COM2B1);             // PWM to OCR2B—PD3
      OCR2A = duty;                        // Set PORTB3 to PWM value
      OCR2B = duty;                        // Set PORTD3 to PWM value
   }
}

void moveLeft(uint8_t duty)
{
```

```
    // Set direction
    PORTB |= (1 << PORTB4)|(1 << PORTB2); // PB4,2 HIGH
    PORTB &= ~(1 << PORTB5)&~(1 << PORTB1); // PB5,1 LOW

    // Set speed
    if (duty == 255)
    {
        PORTB |= (1 << PORTB3);              // Set PORTB3 to on
        PORTD |= (1 << PORTD3);              // Set PORTD3 to on
    } else if (duty < 255)
    {
        TCCR2A |= (1 << COM2A1);             // PWM output on OCR2A—PB3
        TCCR2A |= (1 << COM2B1);             // PWM to OCR2B—PD3
        OCR2A = duty;                        // Set PORTB3 to PWM value
        OCR2B = duty;                        // Set PORTD3 to PWM value
    }
}

void moveRight(uint8_t duty)
{
    // Set direction
    PORTB |= (1 << PORTB5)|(1 << PORTB1);   // PB5,1 HIGH
    PORTB &= ~(1 << PORTB4)&~(1 << PORTB2); // PB4,2 LOW

    // Set speed
    if (duty == 255)
    {
        PORTB |= (1 << PORTB3);              // Set PORTB3 to on
        PORTD |= (1 << PORTD3);              // Set PORTD3 to on
    } else if (duty < 255)
    {
        TCCR2A |= (1 << COM2A1);             // PWM output on OCR2A—PB3
        TCCR2A |= (1 << COM2B1);             // PWM to OCR2B—PD3
        OCR2A = duty;                        // Set PORTB3 to PWM value
        OCR2B = duty;                        // Set PORTD3 to PWM value
    }
}

void motorsOff(void) ❺
{
TCCR2A &= ~(1 << COM2A1); // Disconnect PWM from OCR2A—PB3
TCCR2A &= ~(1 << COM2B1); // Disconnect PWM from OCR2B—PD3
PORTB &= ~(1 << PORTB3); // Set ENABLE pins to zero for brake
PORTD &= ~(1 << PORTD3);
}

int main(void)
{
    // Set PORTB5, 4, 3, 2, and 1 as outputs
    DDRB |= (1 << PORTB5)|(1 << PORTB4)|(1 << PORTB3)|(1 << PORTB2)|(1 << PORTB1); ❻
    DDRD |= (1 << PORTD3); ❼              // Set PORTD3 as output
    initPWM(); ❽
    _delay_ms(3000);                      // Wait a moment before starting
    while(1)
    {
```

```
moveForward(128);
_delay_ms(2000);
moveLeft(128);
_delay_ms(2000);
moveRight(128);
_delay_ms(2000);
motorsOff();
moveBackward(128);
_delay_ms(2000);
}
        }
```

At ❶ and ❽, the code initiates PWM for two digital outputs so it can control two motors. After PWM initiation comes `moveForward()`, the first of five functions to control the motors. You might need to switch the wires on each motor if they appear to work opposite to the code. Four of these functions—`moveForward()`, `moveBackward()`, `moveLeft()`, and `moveRight()`—are identical, except in the order of motor rotation. They all accept a value for the duty cycle to control the speed of the motors. The function `motorsOff()` cuts the power off to both motors.

In this case, we set the direction of the motors forward by making digital outputs high or low, depending on required rotation type ❷. Refer to Table 8–1 for the requisite output configurations. The motor movement functions check if the user requires full speed (a duty cycle of 100 percent, represented by 255) ❸. If so, it simply sets the ENABLE pins of the L293D to on. However, if you pass a lower value for the duty cycle through a motor movement function, the program activates the PWM output to the ENABLE pins ❹ and fills the PWM registers OCR2A and B with the required duty cycle.

The other three movement functions operate similarly, except that the motor rotations are set up to allow for turning left or right or moving backwards. The `motorsOff()` function stops movement by turning off PWM and setting both L293D ENABLE pins to low ❺. Finally, the program sets the six required pins to outputs to control the L293D ❻.

You can use the functions used in the main loop of the code to change the direction of movement, the speed via the duty cycles, and the duration with the delay functions, and stop the motors when required.

We have used a single timer with two PWM outputs for both motors (OCR2A and OCR2B) so that they share the same PWM generation and will thus synchronize with each other. If you use two different timers for two motors that need to operate together, the PWM signals will differ slightly and the two motors will operate slightly differently from one other.

Now that we have experimented with DC motors, in the next chapter we'll examine another useful tool of the AVR system: the internal EEPROM.