33

USING THE OOPIC MICROCONTROLLER

While the Basic Stamp described in Chapter 31 is a favorite among robot enthusiasts, it is not the only game in town. Hardware designers who know how to program their own microcontrollers can create a customized robot brain using state-of-the-art devices such as the PIC16CXXX family or the Atmel AVR family of eight-bit RISC-based controllers. The reality, however, is that the average robot hobbyist lacks the programming skill and development time to invest in custom microcontroller design.

Recognizing the large market for PIC alternatives, a number of companies have come out with Basic Stamp work-alikes. Some are pin-for-pin equivalents, and many cost less than the Stamp or offer incremental improvements. And a few have attempted to break the Basic Stamp mold completely by offering new and unique forms of programmable microcontrollers.

One fresh face in the crowd is the OOPic (pronounced "OO-pick"). The OOPic uses *object-oriented* programming rather than the "procedural" PBasic programming found in the Basic Stamp. The OOPic—which is an acronym for *O*bject-*O*riented *P*rogrammable *I*ntegrated *C*ircuit—is said to be the first programmable microcontroller that uses an object-oriented language. The language used by the OOPic is modeled after Microsoft's popular Visual Basic. And, no, you don't need Visual Basic on your computer to use the OOPic; the OOPic programming environment is completely stand-alone and available at no cost.

The OOPic, shown in Fig. 33.1, has built-in support for 31 input/output (I/O) lines. With few exceptions, any of the lines can serve as any kind of hardware interface. What enables them to do this is what the OOPic documentation calls "*hardware objects,*" digital I/O lines

that can be addressed individually or by nibble (4 bits), by byte (8 bits), or by word (16 bits). The OOPic also supports predefined objects that serve as analog-to-digital conversion inputs, serial inputs/outputs, pulse width modulation outputs, timers-counters, radio-controlled (R/C) servo controllers, and 4x4-matrix keypad inputs. The device can even be networked with other OOPics as well as with other components that support the Philips I2C network interface.

The OOPic comes with a 4K EEPROM for storing programs, but memory can be expanded to 32K, which will hold some 32,000 instructions. The EEPROM is "hot swappable," meaning that you can change EEPRPOM chips even while the OOPic is on and running. When a new EEPROM is inserted into the socket, the program stored in it is immediately started.

Additional connectors are provided on the OOPic for add-ins such as floating-point math; precision data acquisition; a combination DTMF, modem, musical-tone generator; a digital thermometer; and even a voice synthesizer (currently under development). The OOPic's hardware interface is an open system. The I2C interface specification, published by Philips, allows any IC that uses the I2C interface to "talk" to the OOPic.

While the hardware capabilities of the OOPic are attractive, its main benefit is what it offers robot hackers: Much of the core functionality required for robot control is already embedded in the chip. This feature will save you time writing and testing your robot con-

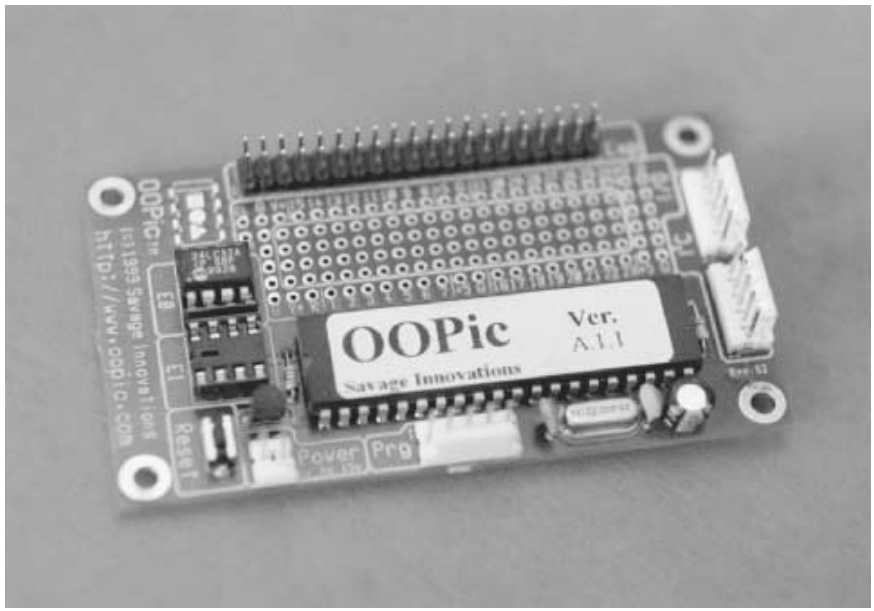

FIGURE 33.1 **The OOPic supports 31 I/O lines and runs on 6–12 vdc power. Connectors are provided for the I/O lines, programming cable, memory sockets, and Philips I2C network.**

trol programs. Instead of needing several dozen lines of code to set up and operate an RC servo, you need only about four lines when programming the OOPic.

A second important benefit of the OOPic is that its various hardware objects are multitasking, which means they run independently and concurrently of one another. For example, you might command a servo in your robot to go to a particular location. Just give the command in a single statement; your program is then free to activate other functions of your robot—such as move another servo, start the main drive motors, and so forth. Once started by your program, all of these functions are carried out autonomously by the objects embedded within the OOPic. This simplifies the task of programming and makes the OOPic capable of coordinating many hardware connections at the same time.

Fig. 33.2 shows a fire-fighting robot that uses several networked OOPics as its main processor. This two-wheeled robot hunts down small fires and literally snuffs them out with a high-powered propeller fan.

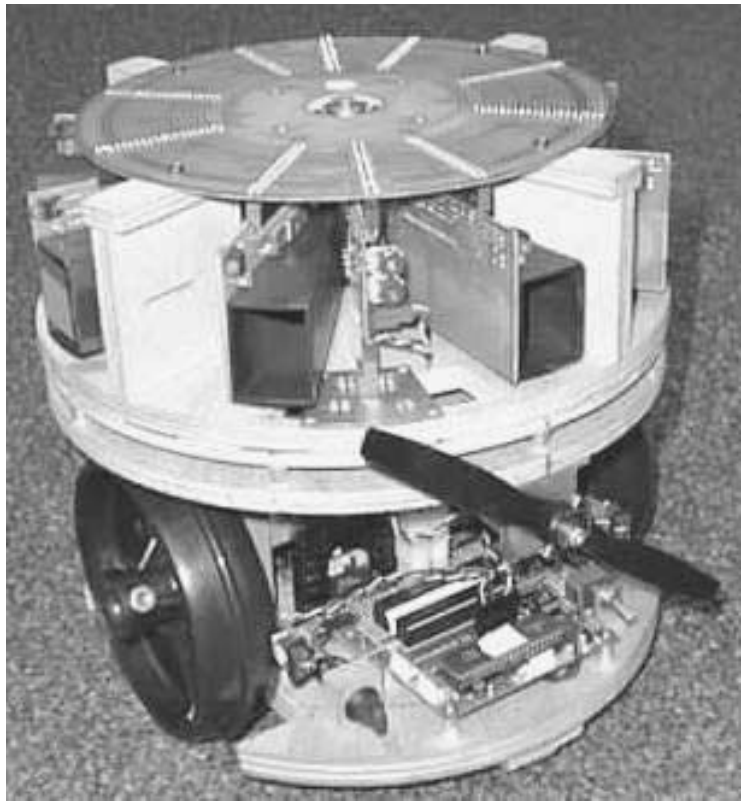

FIGURE 33.2 **This fire-fighting robot, built by OOPic developer Scott Savage, uses three OOPics wired together in a network to control the machine's central command, sensors, and locomotion.**

Objects and the OOPic

Mention the term *object-oriented programming* to most folks and they freeze in terror. Okay, maybe that's an exaggeration, but object-oriented programming seems like a black art to many, full of confusing words and complicated coding. Fortunately, the OOPic avoids the typical pitfalls of object-oriented programming. The OOPic chip supports an easy-to-use programming language modeled directly after Microsoft Visual Basic, so if you already know VB, you'll be right at home with the OOPic. Future versions of the OOPic software development platform will support C and Java syntax for those programmers who prefer these languages.

The OOPic VB-like language offers some 41 programming commands. That's not many commands actually, but it's important to remember that the OOPic doesn't derive its flexibility from the Basic commands. Rather, the bulk of the chip's functionality comes from its built-in 31 objects. Each of these objects has multiple *properties, methods,* and *events.* You manipulate the OOPic's hardware objects by working with these properties, methods, and events. The Basic commands are used for program flow.

Here's a sample OOPic program written in the chip's Basic language. I'll review what each line does after the code sample. This short program flashes a red LED on and off once a second. Fig. 33.3 shows how to connect the LED and a current-limiting resistor to I/O line 1 (pin 7 on the I/O connector) of the OOPic.

```
Dim RedLED As New oDio1

Sub Main()
RedLED.IOLine = 1
RedLED.Direction = cvOutput
Do
    RedLED.Value = OOPic.Hz1
Loop
End Sub
```

These lines comprise a complete, working program. Here's the program broken down:

```
Dim RedLED As New oDio1
```

The *Dim* statement creates a new *instance* of a particular kind of digital I/O object. This I/O object, referred to as *oDio1,* has already been defined within the OOPic. All of the behaviors of this object have been preprogrammed; your job is to select the behavior you want and activate the object. Note that all of the OOPic's object names start with a lowercase letter *O,* such as *oDio1, oServo,*and *oPWM.*

```
Sub Main()
     ...
End Sub
```

The main body of every OOPic program resides within a subroutine called *Main.* OOPic Basic permits you to add additional subroutines to your program, but every program must have a *Main* subroutine. As with Microsoft's Visual Basic, you refer to subroutines by name.

```
RedLED.IOLine = 1
RedLED.Direction = cvOutput
```

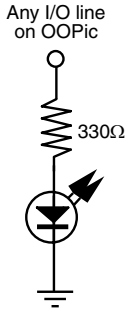

FIGURE 33.3 **The OOPic can source or sink up to 25 mA per I/O line. This sample circuit drives an LED directly. Transistors or bridges are needed when driving a large relay or a motor.**

These two lines set up the I/O line connected to the *RedLED* object. In this case, we've defined that the *RedLED* object is connected to I/O line 1 and that this object will serve as an output (*cvOutput* is a predefined constant; you don't need to define its value ahead of time). All digital I/O lines can be defined as either input or output. The OOPic does not reserve certain lines as outputs and others as inputs.

```
Do
    RedLED.Value = OOPic.Hz1
Loop
```

The statement *RedLED.Value* < *OOPic.Hz1* makes the LED flash once a second. The *Do* loop is used to keep the program running, so the LED continues to flash. Note the *OOPic.Hz1* value that is assigned to the *RedLED* object: *OOPic* is a built-in "system object" that is always available to your programs. One property of the *OOPic* object is *Hz1,* which is a one-bit value that can be used, for example, to change the state of an I/O line (goes from HIGH to LOW) once a second. The following table describes other properties of the OOPic system object you may find useful.

OOPIC PROPERTY	WHAT IT DOES
ExtVRef	Specifies the source of the voltage reference for the analog-toldigital module.
Hz1	1-bit value that cycles every 1 Hz.
Hz60	1-bit value that cycles every 60 Hz.
Node	Used when two or more OOPics "talk" to each other via the I2C network. A Node value of more than 0 is the OOPic's I2C network address.
Operate	Specifies the power mode of the OOPic.
Pause	Specifies if the program flow is suspended.
PullUp	Specifies the state of the internal pull-up resisters on I/O lines 8 –15.
Reset	Resets the OOPic.
StartStat	Indicates the cause of the last OOPic reset.

Using and Programming the OOPic

Other than a 6–12 *VDC* power source, you don't need any other components to begin using the OOPic. For adequate current handling when operating under battery power, I suggest that you use a set of eight alkaline AA batteries in a suitable holder. The OOPic Starter Package comes with a nine-volt transistor battery clip; you can use this clip with Radio Shack's part number 270-387 eight-cell AA battery holder. The holder has connectors for the transistor battery clip.

You can develop programs for the OOPic using a proprietary but free development software (see Fig. 33.4). The development software works under Windows 9x and NT, and it self-installs all the necessary system files.

To program the OOPic you connect a cable between the parallel port of your PC and the programming port of the OOPic. The programming cable is provided as part of the OOPic Starter Package or you can make your own by following the instructions provided on the OOPic home page (*www.oopic.com/*). Once you've written a program in the development software, it is compiled and downloaded through the programming cable. The OOPic is then ready to begin executing your program. Because the OOPic stores the downloaded program in nonvolatile EEPROM, the program will remain in the OOPic's memory until you erase it and replace it with another.

OOPic Objects That Are Ideal for Use in Robotics

Though the OOPic is meant as a general-purpose microcontroller, many of its objects are ideally suited for use with robotics. Of the built-in objects of the OOPic, the oA2D, oDiox, oKeypad, oPWM, oSerial, and oServo objects are probably the most useful for robotics work. In the following descriptions, the term *property* refers to the behavior of an object, such as reading or setting the current value of an I/O line.

ANALOG-TO-DIGITAL CONVERSION

The oA2D object converts a voltage that is present on an I/O line and compares it to a reference voltage. It then generates a digital value that represents the percentage of the volt-

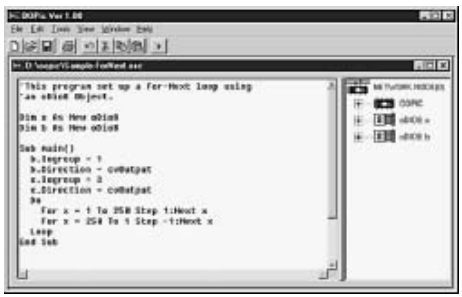

FIGURE 33.4 **Programs are written for the OOPic using a Windows-based software development platform. You open, save, debug, and compile your OOPic programs using pull-down menu commands.**

age in relation to the reference voltage. The *Operate* property of the oA2D object initiates the conversion, and the *Value* property is updated with the result of the conversion. When the Operate value of the oA2D object is 1, the analog-to-digital conversion, along with the *Value* update, occurs repeatedly. Conversion ceases when the *Operate* property is changed to 0.

There are four physical analog-to-digital circuits implemented within the OOPic. They are available on I/O lines 1 through 4.

DIGITAL I/O

Several digital I/O objects are provided in 1-bit, 4-bit, 8-bit, or 16-bit blocks. In the case of the 1-bit I/O object (named oDio1), the *Value* property of the object represents the electrical state of a single I/O line. In the case of the remaining digital I/O objects, the *Value* property presents the binary value of all the lines of the group (4, 8, or 16, depending on the object used).

There are 31 physical 1-bit I/O lines implemented within the OOPic. The OOPic offers six physical 4-bit I/O groups, three 8-bit groups, and one 16-bit group.

R/C SERVO CONTROL

The oServo object outputs a servo control pulse on any IO line. The servo control pulse is tailored to control a standard radio-controlled (R/C) servo and is capable of generating a logical high-going pulse from 0 to 3 ms in duration in 1/36 ms increments.

A typical servo requires a five-volt pulse in the range of 1–2 ms in duration. This allows for a rotational range of 180°. The duration of the control pulse is determined by setting the *Value, Center,* and *InvertOut* properties of the oServer object. The *Value* property controls the position of the servo while the *Center* property adjusts the control pulse time to compensate for mechanical alignment. An *InvertOut* property is used to reverse the direction that the servo turns in response to the *Value* and *Center* properties. We will say more about servo control in a bit.

KEYPAD INPUT

The oKeypad object splits two sets of four I/O lines in order to read a standard 4x4-keypad matrix. The four row lines are individually and sequentially set low (0 volts) while the four column lines are used to read which switch within that row is pressed.

If any switch is pressed, the *Value* property of the oKeypad object is updated with the value of the switch. A *Received* property is used to indicate that at least one button of the keypad is pressed. When all the keys are released, the *Received* property is cleared to 0.

PULSE WIDTH MODULATION

The oPWM object provides a convenient pulse width modulated (PWM) output that is suitable for driving motors (through an appropriate external transistor output stage, of course). The oPWM object lets you specify the I/O line to use—up to two at a time for PWM output, the cycle frequency, and the pulse width.

ASYNCHRONOUS SERIAL PORT

The oSerial object transmits and receives data at a baud rate specified by the *Baud* property. The baud rate can be either 1200, 2400, or 9600 baud. The oSerial object is used to communicate with other serial devices, such as a PC or a serial LCD display.

Using the OOPic to Control a Servo Motor

Though R/C servo motors are intended to be used in model airplanes, boats, and cars, they are equally useful for robotics applications. Servo motors are inexpensive—basic models cost under $15 each—and they combine in one handy package a DC motor, a gearbox, and control electronics. The typical servo motor is designed to rotate 180° (or slightly more) in order to control the steering wheel on a model car or the flight control surfaces on an R/C airplane. For robotics, a servo can be connected to an armature to operate a gripper, to an arm or leg, and to just about anything else you can imagine.

SERVO MOTORS: IN REVIEW

Let's review the way servos operate so we can better understand how you can interface them to the OOPic. An R/C servo consists of a reversible DC motor. The high-speed output of the motor is geared down by a series of cascading reduction gears that can be made out of plastic, nylon, or metal (usually brass, but sometimes aluminum). The output shaft of the servo is connected to a potentiometer, which serves as the closed-loop feedback mechanism. A control circuit in the servo uses the potentiometer to accurately position the output shaft.

Servos use a single pulse width modulated (PWM) input signal that provides all the information needed to control the positioning of the output shaft. The pulse width varies from a nominal 1.25 milliseconds (ms) to roughly 1.75 ms, with 1.5 milliseconds representing the "center" (or neutral) position of the servo output shaft (note that servo specs vary; these are typical). Lengthening the pulse width causes the servo to rotate in one direction; shortening the pulse width causes the servo to rotate in the other direction. The position of the potentiometer acts to "null out" the input pulses, so when the output shaft reaches the correct location the motor stops.

R/C servos are engineered to accept a standard TTL-level signal, which typically comes from a receiver mounted inside a model car or plane. The OOPic can interface directly to an R/C servo and requires no external components such as power transistors.

CONTROLLING SERVOS VIA OOPIC CODE

You can theoretically control up to 31 servos with one OOPic—one servo per IO line. However, the more practical maximum is no more than 8 to 10 servos. The reason: Servos require a constant stream of pulses, or else they cannot accurately hold their position. The ideal pulse stream is at 30 to 60 Hz, which means that to operate properly each servo

connected to the OOPic must be "updated" 30 to 60 times per second. The OOPic is engineered to provide pulses at 30-Hz intervals; with more than about eight servos the refresh rate is reduced to 15 Hz. While most servos will still function with this slow refresh rate, a kind of "throbbing" can occur if the motor is under load.

Some robotic projects call for controlling a half-dozen or more servos, such as the six-legged Hexapod II from Lynxmotion (which requires 12 servos working in tandem). However, the typical experimental robot uses only two or four servos. The OOPic is ideally suited for this task, and programming is easy. To operate a servo, you need only provide a few lines of setup code, then indicate the position of the servo using a positioning value from 0 to 63. This value corresponds to the 0–180° movement of the servo output shaft.

With 64 steps the OOPic is able to position a servo with 2.8° of accuracy. This assumes a maximum rotation of 180°, which not all servos are capable of. Note that if you need greater resolution than this you can make use of the OOPic's built-in pulse width modulation object, which can be programmed to provide your servos with far greater positional accuracy. However, for most applications, the OOPic's servo object provides adequate resolution and is easier to use.

Listing 33.1 shows a program written in the OOPic's native Basic syntax and demonstrates how to control an R/C servo using the oServo object. Fig. 33.5 shows how to connect the servo to the OOPic.

LISTING 33.1.

```
' OOPic servo demonstrator
' Uses a standard R/C servo

' This program cycles a servo, connected to IOLine 31,
'  for full rotation (0 to 180 degrees)

' Dimension needed objects
Dim S1 As New oServo
```

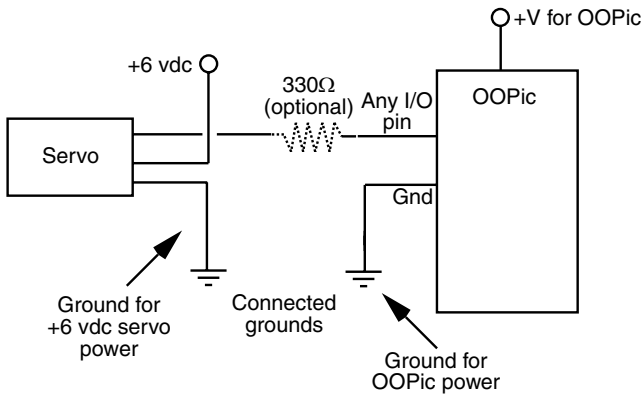

FIGURE 33.5 **Follow this basic wiring diagram to connect a standard R/C servo to the OOPic. Most servos use consistent color coding for their wiring: black for ground, red for V , and yellow or white for input (signal).**

```
Dim x As New oByte
Dim i As New oNibble
'_____________________________
'First routine called when power is turned on
Sub Main()
Call Setup ' set up servo properties
For i = 1 to 5   ' repeat motions five times
     S1 = 0      ' set servo to 0 degrees, and wait a while
     Call longdelay
     S1 = 63      ' set servo to 180 degrees, and wait a while
     Call longdelay
Next i
End Sub
'_____________________________
' Delay loop routine
Sub longdelay()
For x = 1 To 200:Next x
End Sub
'_____________________________
' Setup routine
Sub Setup()
S1.Ioline = 31             ' Set servo to I/O line 31 (pin 26)
S1.Center = 31             ' Set center to 31 (experiment for best
                             results)
S1.Operate = cvTrue        ' Turn servo on
End Sub
```

POWERING THE SERVOS

Note that separate battery power supplies were used for the OOPic and the servo. Most hobby R/C servos are designed to be operated with 4.5 to 7.2 vdc. Connecting both OOPic and servo to a single 6-volt supply can cause the OOPic to reset itself. Most servos draw considerable current when turned on, and this current can cause the supply voltage of a 6-volt battery pack to sag below the 4.5-volt level required by the OOPic. When the voltage drops below 4.5 volts, the OOPic's built-in brownout circuit kicks in, which resets the processor. This repeats continuously, and the net effect is a nonfunctioning circuit.

One alternative is to power the whole shebang from a single 9- or 12-volt supply, but with higher voltage comes overpowered servos. Not all servos are built to handle the extra speed and heat caused by the higher voltage, and an early death for your servos could result. Therefore, it's best to use two different batteries. The OOPic is fine operating from a single 9-volt transistor battery. The servo runs from a set of four AA batteries.

HOW THE OOPIC SERVO CODE WORKS

The first three lines in Listing 33.1 "dimension" (create in memory) the objects used in the OOPic program. *S1* is the servo object; *x* and *I* are simple data objects that hold eight and four bits, respectively. The program itself begins with the *Main* subroutine, which is automatically run when the OOPic is first turned on or when it is reset. The first order of business is to call the *Setup* subroutine, located at the end of the program. In *Setup,* the program establishes that IO line 31 (pin 26 of the OOPic chip) is connected to the control input of the servo.

The servo is then centered using a value of 31 (half of 64, considering 0 as the first valid digit). You need to experiment to find the mechanical center of the servo you are using.

Each servo, particularly those that have different sizes and come from different manufacturers, can have a different mechanical center. Therefore, adjust this value up or down accordingly. Finally, the servo object is activated using the statement

```
S1.Operate = cvTrue
```

Notice the use of *properties* when working with the OOPic's objects. Properties are defined by specifying the name of the object, such as *S1* for servo 1, a period (known as the *member operator* in programming parlance), then the property name. So, *S1.Ioline* sets (or reads) the IO line property for the *S1* object. Similarly, *S1.Center* sets the center property, and *S1.Operate* turns the *S1* object on or off. Most OOPic properties are read and write, meaning that you can both set and read their value. A few are read-only or write-only.

Once you have set the servo up, you can manipulate it using the *S1.Value* property. In the demonstration program, the *Value* property is inferred because it is the so-called default property for servo objects. Therefore, it is only necessary to specify the name of the object and the value you want for it:

```
S1 = 0
```

This sets the servo all the way in one direction, and the following expression,

```
S1 = 63
```

sets the servo all the way in the other direction. Because the *Value* property is the default for the oServo object, the statement *S1* < 63 is the same as writing *S1.Value* < 63.

Note:

Exercise care when playing around with servos. Not all servos can travel a full 90° from center, especially if you have not properly set the mechanical center using the *S1.Center* property. For initial testing, use values slightly higher than 0 and slightly lower than 63 to represent the minimum and maximum servo movements, respectively. Otherwise, the OOPic may command the servo to move past an internal stop position, which can cause the gears to slip and grind. Left in this state the servo can be permanently damaged.

Operating Modified Servos

As designed, R/C servos are meant to travel in limited rotation, up to 90° to either side of some center point. But by modifying the internal construction of the servo, it's possible to make it turn freely in both directions and operate like a regular-geared DC motor. This modification is handy when you want to use servo motors for powering your robot across the floor.

The steps for modifying servos vary, but the general process is about the same:

1. Remove the case of the servo to expose the gear train, motor, and potentiometer. This is accomplished by removing the four screws on the back of the servo case and separating the top and bottom.

2. File or cut off the nub on the output gear that prevents full rotation. This typically requires removing one or more gears, so you should be careful not to misplace any parts. If necessary, make a drawing of the gear layout so you can replace things in their proper location!
3. Remove the potentiometer, and replace it with two 2.7K-ohm 1 percent (precision) resistors, wired as detailed in Chapter 20. This fools the servo into thinking it's always in the "center" position. Or relocate the potentiometer to the outside of the servo case, so you can make fine-tune adjustments of the center position. If needed, you can attach a new 5K- or 10K-ohm potentiometer to the circuit board outside the servo.
4. Reassemble the case.

See Chapter 20, "Working with Servo Motors," for a step-by-step tutorial on modifying commonly available servos for continuous rotation.

OOPIC CODE FOR MODIFIED SERVOS

Once modified, you can connect the servo to the OOPic just as you would an unmodified servo (see Fig. 33.5). Listing 33.2 shows how to use the OOPic with two modified servos acting as the drive motors for a two-wheeled robot. You can easily construct a demonstrator robot using LEGO parts, like the prototype shown in Fig. 33.6. I cemented two light-

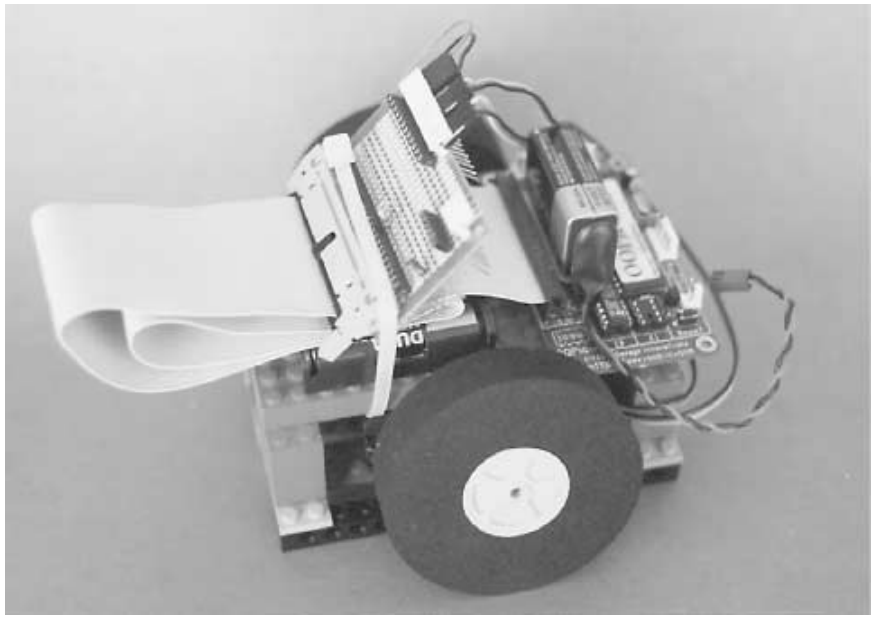

FIGURE 33.6 You can construct a demonstrator for the OOPic two-wheel robot using LEGO bricks. The servos are glued to small LEGO parts to aid in mounting.

weight R/C airplane wheels to control horns (these come with the servos). I also cemented a 2x8 flat LEGO plate to the side of each servo to make it easier to snap the motors to the LEGO-made frame of the robot.

LISTING 33.2.

```
' OOPic two-motor (servo) robot demonstrator
' Requires the use of modified R/C servos (see text)

' This program cycles the robot through various movements,
'  including forward, backward, right spin, left spin,
'  and turns.

'  Dimension objects
Dim S1 As New oServo
Dim S2 As New oServo
Dim CenterPos as New oByte
Dim Button As New oDio1
Dim x as New oByte
Dim y as New oWord

'_____________________________
Sub Main()
CenterPos = 31               ' Set centering of servos
Call Setup
Do
      If Button = cvPressed Then
          ' Special program to calibrate servos
            S1 = CenterPos
            S2 = CenterPos
      Else
          ' Main program (IO line is held low)
            Call GoForward
            y = 200              ' Same as LongDelay
            Call Delay           ' Alternative to LongDelay

            Call HardRight
            Call LongDelay

            Call HardLeft
            Call LongDelay

            Call SoftRightForward
            Call ShortDelay

            Call SoftLeftForward
            Call ShortDelay

            Call GoReverse
            Call LongDelay
       End If
Loop
End Sub

'_____________________________
' Set up IO lines and servos
Sub Setup()
Button.Ioline = 7                ' Set IO Line 7 for function input
Button.Direction = cvInput       ' Make IO Line 7 input
S1.Ioline = 30                   ' Servo 1 on IO line 30
```

```
S1.Center = CenterPos            ' Set center of Servo 1
S1.Operate = cvTrue              ' Turn on Servo 1
S2.Ioline = 31                   ' Servo 2 on IO line 31
S2.Center = CenterPos            ' Set center of Servo 2
S2.Operate = cvTrue              ' Turn on Servo 2
S2.InvertOut = cvTrue            ' Reverse direction of Servo 2
End Sub

'_____________________________
' Short delay routine
Sub ShortDelay()
       For x = 1 To 80:Next x
End Sub

'_____________________________
' Long delay routine
Sub LongDelay()
       For x = 1 To 200:Next x
End Sub

'_____________________________
' Selectable delay routine
Sub Delay()
      For x = 1 To y:Next x
End Sub

'_____________________________
' Motion routines (forward, back, etc.)
' "Hard" turns spin robot in place
' "Soft" turns turn robot right or left in forward
' (or backward) motion
'_____________________________

Sub GoForward()
S1 = 0
S2 = 0
End Sub

Sub GoReverse()
S1 = 63
S2 = 63
End Sub

Sub HardRight()
S1 = 0
S2 = 63
End Sub

Sub HardLeft()
S1 = 63
S2 = 0
End Sub

Sub SoftRightBack()
S1 = CenterPos
S2 = 63
End Sub

Sub SoftRightForward()
S1 = 0
S2 = CenterPos
End Sub
```

```
Sub SoftLeftBack()
S1 = 63
S2 = CenterPos
End Sub

Sub SoftLeftForward()
S1 = CenterPos
S2 = 0
End Sub
```

We attached batteries and OOPic to the top of the robot using double-sided tape. Power to the OOPic is provided by a 9-volt battery; power to both servos is provided by a 6-volt pack of AAs. Note that I used a wire-wrap board as a terminal bus, and standard .100"-center connectors instead of hard-soldering any wiring to the various components. This makes it easier to test the robot and possibly add to it at a later date.

REVIEWING THE PROGRAM CODE

The program in Listing 33.2 is a modified version of the program in Listing 33.1. Its main difference, other than employing two oServo objects instead of one, is that the "center" position is used to turn the motor off. Values greater than this center position cause the servos to rotate in one direction; values less than the center position cause the servos to rotate in the opposite direction. The servos are made to turn one direction when their *Value* property is 0 and the other direction when their *Value* property is 63.

Note that in Listing 33.2 the "normal" direction of travel for servo 2 (S2) is reversed from S1, with the following statement:

```
S2.InvertOut = cvTrue
```

This is handy because in the two-wheeled robot the servos are mounted on opposite sides, and therefore one motor must operate in mirror image to the other. That is, one must turn clockwise while the other turns counterclockwise to move the robot forward or backward. Without the *InvertOut* property, you'd have to set the *Value* property of one servo to 0 and the other to 63 to maintain proper forward or backward motion.

Not shown in Listing 33.2 is a useful feature you may want to implement: values very close to the center position (/- about five steps) will cause the servos to slow down by a proportional amount. For example, if the center position is 31, then a value of 32 for *S1* or *S2* may cause that servo to rotate clockwise very slowly. Higher values will modestly increase the speed in the same direction of travel. Conversely, a value of 30 for the *S1* or *S2* object may cause the servo to rotate counterclockwise very slowly. A value of 29 would make the motor go a little faster, and so on.

Listing 33.2 takes the robot through a series of patterned moves, including forward and backward movement, right and left spins, and turns. Delay routines allow you to specify how long each movement is to last. Vary the delay up or down to experiment with different motions. In the prototype for this book, the program in Listing 33.2 moves the robot back and forth about two feet. The program repeats itself until you reset the OOPic or disconnect the power.

The modified servos use an externally accessible trimmer potentiometer. The trimmer pot, which is attached to the case of the servo with a small piece of double-sided foam tape, serves

to provide an accurate voltage divider by which the servos can be set to center, or neutral, position. The trimmer pots are set by temporarily taking IO line 7 high. This causes the program in Listing 33.2 to run an alternative routine in its *Main* loop, so you can set the *Center* property of both servos to a value of 31. The pots are then adjusted so that the motors just stop—this represents the center position. Using the potentiometer makes it *much* easier to calibrate the servos so they can be used with the program. Once calibrated, you can tie IO line 7 low again.

Using the OOPic to Control Stepper Motors

The OOPic is full of pleasant surprises, including the innate ability to control a standard four-phase unipolar stepper motor. Unlike R/C servos, however, the OOPic is not able to directly drive a stepper motor. For that you'll need an interface with a current and voltage rating for the stepper motor you are using. Chapter 19 provides additional information on using stepper motors.

Listing 33.3 shows a simple stepper motor driving program that uses a feature unique to the OOPic: virtual circuits. Instead of programming each of the four phases of a stepper with on/off values in code, this program uses two *processing* objects, oConverter and oCounter. Processing objects are used to construct virtual circuits, which are like real electronic circuits, only they are created solely using programming statements.

LISTING 33.3.

```
' OOPic stepper motor demonstrator
' Uses a standard four-phase unipolar stepper motor
' Operates motor in half-stepping mode

' Dimension objects
Dim Stepper as New oDio4    ' 4-bit IO for controlling stepper
Dim Driver as New oConverter
Dim Position as New oWord   ' 32-bit value for current position
Dim Mover as New oCounter

'_____________________________
Sub Main()
Call Setup
' The rest of your code here

' To reverse motor, use Mover.Direction = cvNegative
' or Mover.Direction = cvPositive
' To stop-and-hold motor, use Mover.Operate = cvFalse
' To restart motor, use Mover.Operate = cvTrue
' To stop and de-energize motor, use Driver.Blank = 1
End Sub

'_____________________________
' Set up stepper motor
Sub Setup()
Stepper.IOGroup = 1       ' Set stepper to use IO group 1 (pins 8-11)
Stepper.Nibble = 0        ' Picks lower 4 lines from IO group
Stepper.Direction = cvOutput      ' Make lines outputs
Driver.Output.Link(Stepper.Value) ' Set up virtual circuit
```

```
Driver.Input.Link(Position.Value)
Driver.Mode = cvPhase
Driver.Operate = cvTrue
Mover.ClockIn1.Link(OOPic.Hz60) ' Use OOPic 60 Hz object for stepping
Mover.Output.Link(Position.Value)
Mover.Operate = cvTrue      ' Enable counter
End Sub
```

The stepper motor program in Listing 33.3 demonstrates one of the uses for the oConverter numeric-conversion object. This program has the built-in "behavior" of being able to construct the proper phasing to control the forward and backward rotation of a four-phase unipolar stepper motor. The program also uses a counter object, which allows you to define the number of steps you wish to apply to the motor. Keep in mind that the oConverter object specifies an eight-phase cycle, which has the effect of moving the motor in half-step increments (this serves to improve the accuracy and torque of the motor). So, for example, if the motor is rated at 200 steps per revolution, it will require 400 pulses from the OOPic to turn it a full 360° degrees.

Experiment with the OOPic and you'll find it's a capable performer in the field of robotics. By using its objects judiciously, coupled with a liberal sprinkling of virtual circuits, you should be able to construct most any kind of robotic creature using a minimum number of external components.

From Here

To learn more about…	*Read*
Stepper motors	Chapter 19, "Working with Stepper Motors"
How servo motors work	Chapter 20, "Working with Servo Motors"
Different approaches for adding brains to your robot	Chapter 28, "An Overview of Robot 'Brains'"
Connecting the OOPic microcontroller to sensors and other electronics	Chapter 29, "Interfacing with Computers and Microcontrollers"