6
CHAPTER

Basic Stamp 2 Robot Programming

Now that you have had a chance to play around with the *TAB Electronics Build Your Own Robot Kit* and have had a chance to understand better the operation of the different behaviors, you can now start developing your own application code to learn more about robot programming. By simply adding a Parallax Basic Stamp 2 (BS2) to the robot, you are no longer limited to the simple operations and behaviors built into the robot and the remote control.

If you have never programmed a computer before, I'm sure that the idea of programming the robot is both exciting and scary. In the previous chapters, I have introduced you to the basic concepts of programming and have given you an introduction to programming behaviors and artificial intelligence. I'm sure you'll think that it looks easy when I do it, but it will probably become very difficult to do yourself when you are just starting out.

I think that you will find that when you try to get your first program working, it will take a long time and will probably cause you to ask a number of questions. It may take you as long as two weeks just to get a simple program that says "Hello World!" on your PC's screen.

When this happens, please do not feel like you're never going to "get it" and give up. After you get your first application running, you will find that it will take probably half the time to get the second working, then maybe a few hours for the third, and as you start doing more, the time required will be reduced.

Remember to start off small and work your way up. In the experiments/applications listed in this chapter, I will demonstrate some very small applications that you can copy.

Before attempting to write your own application, there are four things that I recommend you do. The first is to skim through the Parallax *BASIC Stamp Programming Manual Version 2* that is located on the CD-ROM. This manual is the complete reference for all the different Parallax BASIC Stamps and it should be your primary reference when you are programming and trying to work through problems. This is not to say that I do not introduce you to some of the fundamentals and features of BS2 PBASIC programming, it's just that the manual should be your primary reference.

Once you have gone through the BS2 programming manual, I recommend that you sign up for the Parallax StampList (instructions for doing this are on the CD-ROM's html pages). The StampList is a very active ListServ group in which user's questions, successes, and ideas are presented and discussed. The StampList is provided and monitored by Parallax, and it is an excellent resource for fielding your questions about programming the BS2 in the robot.

You can get more information about the StampList, as well as how to sign up at Parallax's web site at the following:

`http://www.parallaxinc.com`

Another resource that you should use (and definitely look at before starting to program your own applications) is:

`http://www.stampsinclass.com`

as well as the *TAB Electronics Build Your Own Robot Kit* web page. On this site, there are a variety of resources you can access, such as FAQs answering your basic questions, sample applications, a forum for new questions as well as a list of resources that will help you to design your own applications.

For the rest of this chapter, I will describe how you can develop your own custom robot applications using the *TAB Electronics Build Your Own Robot Kit*.

The Parallax Basic Stamp 2 and AppMod

One of the most innovative hobbyist, educational, and professional products of the 1990s was the Parallax BASIC Stamp. The first version of the BASIC Stamp made electronics and computer programming accessible to many different people. In the 10 years since the introduction of the first BASIC Stamp, Parallax has added more powerful versions of the BASIC Stamp to their catalog as well as many different supporting products to make it easier to convert your ideas into reality. Ease of use has always been a hallmark of the BASIC Stamp product family, and this makes them ideal for use as the "central nervous system" of the *TAB Electronics Build Your Own Robot Kit*.

All Parallax Basic Stamps have been designed using the same model of the block diagram shown in Figure 6-1. A microcontroller's program memory is loaded with an interpreter that executes a series of instructions that are downloaded from a PC. When these instructions are downloaded, they are stored in a serial EEPROM ("SEEPROM" in Figure 6-1) and retrieved as the application is executed. The microcontroller's I/O pins are used as the I/O pins of the BASIC Stamp.

There are two important features that you should be aware of in all Basic Stamps. The first is that there is a custom PC programming interface built into the microcontroller. In the first Basic Stamp (known as the "BASIC Stamp 1" or "BS1"), the programming interface is a synchronous serial interface similar to the one implemented for the *TAB Electronics Build Your Own Robot Kit* control interface that connects the robot to a BASIC Stamp 2 ("BS2"). The "BS2", which the *TAB*

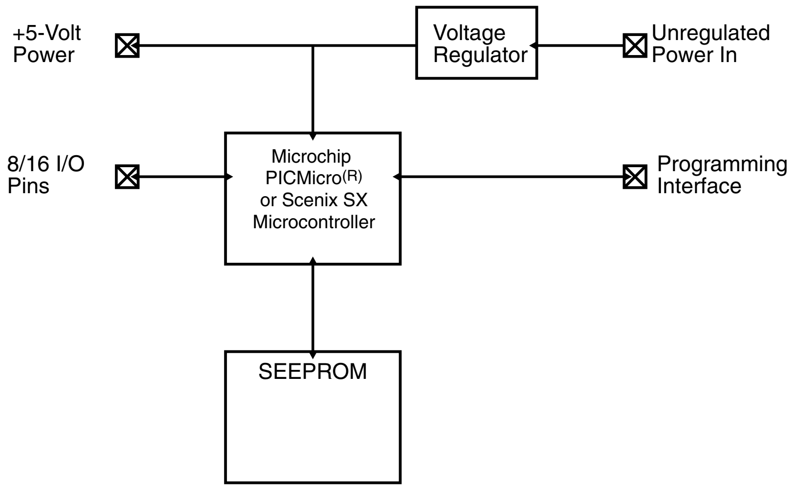


Figure 6-1 Block diagram of Parallax "Basic Stamp."

Electronics Build Your Own Robot Kit is designed to work with, is programmed serially via a PC's RS-232 interface.

The program (or "application") code is developed on a PC using a Windows-compatible editor/development system known as "stampw." BASIC applications are created using this tool and then compiled into the series of instructions that are executed by the BASIC Stamp. I usually refer to these instructions as "tokens" as they represent BASIC functions and statements, but they are not the actual instructions executed by the microcontrollers built into the BASIC Stamp. In the following sections, I will demonstrate how applications are developed using "stampw" on a PC and how they are loaded into the BASIC Stamp 2 plugged into the *TAB Electronics Build Your Own Robot Kit*.

The second feature that you should be aware of is that all BASIC Stamps have a built-in voltage regulator to convert larger voltages into the correct voltages required by the microcontrollers built into the BASIC Stamp. These regulators are generally low-current with low parasitic power requirements that make them ideal for powering the BASIC Stamp plugged into the robot (in fact, the unregulated 9-volt power from the battery is used to drive the BASIC Stamp to avoid loading down the robot's 78L05 voltage regulator). In addition to being able to drive the circuitry on the BASIC Stamp, these regulators generally have enough left over current for a few LEDs or simple logic chips.

These regulators do not, however, have enough current to drive large loads. In recognition of this, and to protect the BASIC Stamp's built-in voltage regulator, the *TAB Electronics Build Your Own Robot Kit* does not pass the BASIC Stamp's regulated power to any other parts. As I will discuss below, for some applications, you will have to provide your own regulator circuit when you are adding circuitry to robot applications.

As I stated above, the first BASIC Stamps ("BS1s") were programmed by a PC's parallel port. This method for the most part works well, but it cannot insure that the connection will work with all PCs and parallel ports. The BASIC Stamp 2 ("BS2") was designed to be programmed by a PC's RS-232 serial port, which is a much more robust interface with very well-defined voltage levels and data rates.

Another deficiency of the BS1 over the BS2 is its limitation of eight input and output (“I/O”) pins. The BS2 has 16 I/O pins, along with the programming port, which can be used as an RS-232 interface. 14 of the pins are available for your use in developing *TAB Electronics Build Your Own Robot Kit* applications. The I/O pins are TTL/CMOS level bidirectional pins, and through the built-in PBASIC functions, can be used for a lot more than just simple digital input and output.

The BS2 has significantly more storage than the BS1. For program space, depending on the model of BS2 that you work with, the difference can be eight to 64 times that of the BS1. Each BS2 has twice the RAM (variable) space of the BS1. This increase will allow for larger and more complex applications. While I have never run into space problems when developing applications for the BS1, the extra storage available in the BS2 virtually guarantees that there will not be any problems with running out of EEPROM space when you are programming applications for the *Tab Electronics Build Your Own Robot Kit.*

The BS2 is significantly faster in executing its instructions than the BS1. Parallax states that the BS1 is capable of running at 2,000 instructions per second and that the BS2 runs at 4,000 to 12,000 instructions per second, depending on the model. Personally, I dislike blanket statements about execution speed simply because different functions and statements execute at different rates; the time required to execute is more dependent on how the applications are programmed, than how fast the BASIC Stamp’s microcontroller executes.

An important concept that I do not discuss enough in these chapters is that I consider function and statement execution speed to be an incorrect measurement when benchmarking an application. You will find that even the 2,000 instructions per second speed of the BS1 will be more than adequate for the robot applications that I present because the robot executes at human speeds, not what I call *machine* or *digital electronics* speeds.

I consider the important measurements for a robot application to be:

1. *Readability*. When somebody else looks at the code, how hard is it for them to understand how it works?
2. *Protection*. Are all the different possible situations and scenarios accounted for? For the *TAB Electronics Build Your Own Robot* there are a limited number of inputs to be considered, but not considering them or reacting inappropriately will result in an application that gets stuck in corners or responds inappropriately.
3. *Responsiveness*. I know I stated that I considered processor speed to be a misleading indicator of application performance, but that does not mean that I like waiting for something to happen. The application must respond to input in a timely manner and not take an unreasonably long time to decide how to act.

The last advantage of the BS2 over the BS1, and probably the most important reason for choosing the BS2 for the *TAB Electronics Build Your Own Robot Kit*, is the enhanced features and built-in functions of its “PBASIC” language. When you have read through the Parallax *BASIC Stamp Programming Manual* (a PDF copy is included on this CD-ROM), then you will discover that there are a number of built-in functions and standard programming statement capability improvements built into the BS2 version of PBASIC over the BS1 version.

As far as the *TAB Electronics Build Your Robot Kit* is concerned, the "SHIFTIN" and "SHIFTOUT" instructions allow very simple bits of code to be used for the BS2 to communicate with and control the robot. Other built-in functions allow the BS2 to communicate with other devices in a variety of different ways and use different standard protocols. Along with this, the BS2 PBASIC language allows for much more complex assignment and "if" statements, giving you more flexibility with your applications.

I think the best way to describe the BS2 is to look at its "pinout" (how the pins are arranged) as well as the function of each of the different pins. The BS2 plugs into a standard 0.600" (600 mil) Dual In-line Package ("DIP") socket with the features and pins shown in Figure 6-2.

When you are working with any kind of electronic chip, it is important to know where "Pin 1" is. This pin indicates how the chip is to be installed in the circuit. In some chips, Pin 1 is indicated with a dot beside it. In the BS2, the semicircular mark at the "top" of the chip indicates the end that Pin 1 is at. When the BS2 is installed in the robot, this semicircular mark should be used to line up with the similar mark that has been placed on the white marking (known as "silkscreen") on the *TAB Electronics Build Your Own Robot Kit* printed circuit board.

The other pins on the board are numbered by incrementing counterclockwise (looking from the top) from Pin 1. Using this convention, Pin 2 ("SIN") is the pin beside Pin 1, Pin 3 ("ATN") is beside Pin 2, and so on to Pin 24 ("VIN"), which is directly across from Pin 1. This numbering convention is used for all DIP chips (and many others of different technologies).

Unregulated power is applied to the BS2 through the "VIN" and "VSS" (often referred to as "Ground" or "Gnd") pins. The regulator built into the BS2 converts the voltage applied at VIN to +5 volts and applies it to the parts on the BS2 as well as external parts via the "VDD" (sometimes referred to as the "Vcc") pin. The BS2 can have +5 volts applied directly through the VDD pin if regulated power is already available in the application. As I said earlier, the BS2 is powered by the unregulated power supplied by the 9-volt battery and the BS2 regulated voltage is not available anywhere else in the robot.

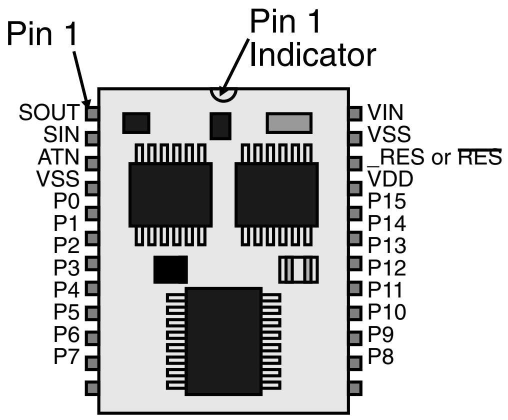


Figure 6-2 Parallax BASIC Stamp 2 pinout.

The “SOUT,” “SIN,” and “ATN” pins are used for programming the BS2 and can be used for an RS-232 interface. The “ATN” pin is connected to the programming PC’s “DTR” pin and is used to reset the BS2 for programming. If ATN is not active when the BS2 powers up, then the application already stored in the BS2 starts executing automatically. This feature is very useful since it allows you to program the robot and then try it out somewhere else.

The remaining 16 pins of the BS2 are the I/O pins. These are bidirectional CMOS input and output pins, each with the ability of sinking and sourcing approximately 20 mA. Total current sunk or sourced by the BS2 should not exceed 75 mA to ensure that the BS2’s voltage regulator is not overloaded.

There are a number of different ways of accessing the BS2’s I/O pins. They can be accessed directly, as if they were variables, by using one of the predefined values listed below:

Function	**Word**	**8 bits**	**4 bits**	**Single bits**
Input bits	INS	INL, INH	INA-IND	IN0-IN15
Output bits	OUTS	OUTL, OUTH	OUTA-OUTD	OUT0-OUT15
Data direction bits	DIRS	DIRL, DIRH	DIRA-DIRD	DIR0-DIR15

The Data Direction Bits are used to enable the CMOS drivers in the BS2. If a “1” is written to the bit, then the corresponding pin is in “output” mode, while a “0” in the bit will put the pin in “input” mode (which means that it cannot be driven).

Another important concept to understand about the I/O pins is the difference between the “IN” and “OUT” values for the bit. “IN” is the actual value at the I/O pin. This value can be the same as the “OUT” value, but if the pin is in input mode or the pin is being “overdriven,” then it will be different. As I’ve shown in Figure 6-3, the “OUT” value is stored in a flip flop and can be driven out to the I/O pin when the “Pin Output Enable” is set to a 1.

For example, to set pin “P0” to output as a low value, you could use the code

```
OUT0 = 0                        ‘  Load the Pin Output F/F with “0”
DIR0 = 1                        ‘  Put the Pin in Output Mode
```

In addition to writing directly to the pins, you can also use a number of built-in functions. For example, the two lines above could be replaced with the single function

```
LOW 0                           ‘  Output “0” from Pin 0
```

While this pin architecture seems to be somewhat confusing, it actually is the best way of implementing the pins. Other microcontrollers implement the same functions in different ways and can have some problems for users trying to correctly output specific values. To simplify how you work with the BS2 I/O pins in your applications, I recommend that you write to the “OUT” labels and read the value at the I/O pin using “IN,” and set the mode using “DIR.”

I felt it was important to explain how the I/O pins work for simple I/O so that you would have some ideas of what is happening when you start working through the applications below. Again, I recommend that you read through the *BASIC Stamp Programming Manual* PDF on the CD-ROM, so that you have an idea of where to look for information in the manual.

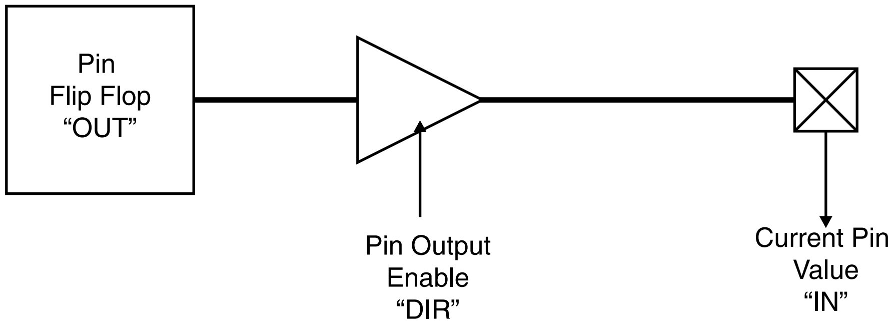


Figure 6-3 Block diagram of Basic Stamp I/O pins.

As I write this, there are four different models of the BS2 to choose from. All of them have the same pinouts and can execute the same basic application code, but they all have differences as I have summarized in the following table:

Model	Price	Speed	EEPROM	RAM	Current required
BS2	$49	4k in/sec	2 kBytes	26 Bytes	7 mA/50 µA Sleep
BS2e	$54	4k in/sec	16 kBytes	26 Bytes 63 Bytes Scratchpad	20 mA/200 µA Sleep
BS2sx	$59	10k in/sec	16 kBytes	26 Bytes 63 Bytes Scratchpad	60 mA/200 µA Sleep
BS2p24	$79	12k in/sec	16 kBytes	26 Bytes	40 mA/50 µA Sleep

The BS2e is an enhanced version of the BASIC Stamp 2 with a more powerful microcontroller and more storage. The BS2e's processor is slowed down to reduce the current required by the device. The BS2sx is identical to the BS2e, but runs the microcontroller at the maximum speed possible (resulting in higher current requirements). The BS2p24 has an enhanced PBASIC function set compared to the other devices and has a sibling device, the BS2p40, which is a 40-pin version of the BS2 with 16 additional I/O pins.

For the *TAB Electronics Build Your Robot Kit*, I recommend that you use the original BS2 for two reasons. The first is that it uses the least amount of power. The second is that when you include the $15 coupon from Parallax, by ordering the BS2 from Parallax (include the ISBN number of the *Tab Electronics Build Your Own Robot Kit Box* as specified on the coupon that came with the kit), it is by far the cheapest way to add the BS2 capability to your robot. As I've indicated above, the slower speed is not an issue, and as you start working through the example experiments below, you will discover that for most applications you will require just a fraction of the 2 kBytes available to the BS2.

To install the BS2 into the *TAB Electronics Build Your Own Robot Kit*, the BS2 is gently pressed into the socket provided in the robot (until it "clicks" in) as shown in Figure 6-4. Note that Pin 1 of the BS2 points toward the rear of the robot and not the front. To remove the BS2, you can use a small, flat-bladed screwdriver to pry it out of the socket or use a plastic "chip puller" to perform the same function. Using either tool, be careful not to damage the BS2 or the surrounding components (most importantly, the IR LED that is just in front of the socket).

I recommend that the BS2 is never removed from the *TAB Electronics Build Your Own Robot Kit*. The reason for this recommendation is the possibility of damaging either the BS2 or the robot during the operation. If you leave the BS2 in, there are some simple Null programs that you can load into the BS2 to render it inert as far as the robot is concerned. I will present these instructions to you later in this chapter.

To make BS2 experimentation easier and more efficient, Parallax has specified what amounts to a simple bus for the BS2 that can be used to connect different interface products to it. This bus is known as AppMod, and to allow you to take advantage of these products, the *TAB Electronics Build Your Own Robot* has an AppMod socket beside the BS2, along with a hole for a standoff in the middle of the robot.

As I write this (August 2001), Parallax has five different AppMod products (known as "Modules") available:

Module	**Price**	**Function**
Sound	$89	Will digitally record and playback up to 60 seconds of sound.
LED Terminal	$89	Four LED Alpha Numeric Digits.
Compass	$79	The compass is designed for use with robots to give them a sense of direction.
Breadboard	$24	Breadboard module for prototyping application circuits.
Protoboard	$19	Solderable application prototype board.

The three AppMods with functions already built in (sound, LED terminal, and compass) communicate with the BS2 using the built-in serial functions. In this chapter, I will be concerned primarily with the Bread Board AppMod (the layout is shown in Figure 6-5), and I will be presenting you with an experiment that you can build on it to enhance your *TAB Electronics Build Your Own Robot Kit*.

There are three things that you should be aware of when you want to use an AppMod with the robot. The first is that you will have to use the socket "Extender"

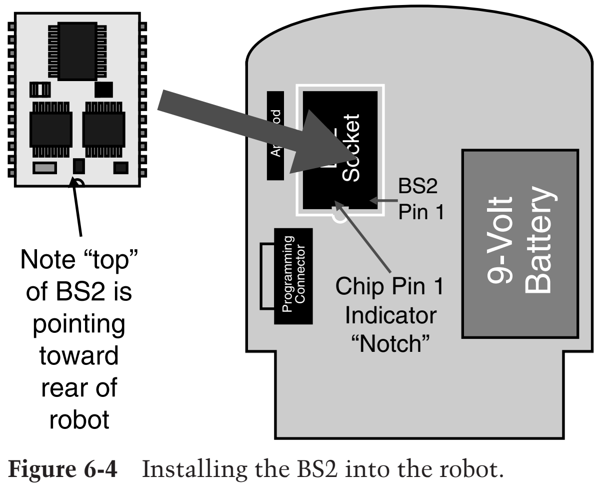

Note "top" of BS2 is pointing toward rear of robot
Socket
BS2 Pin 1
Programming Connector
Chip Pin 1 Indicator "Notch"
9-Volt Battery

Figure 6-4 Installing the BS2 into the robot.

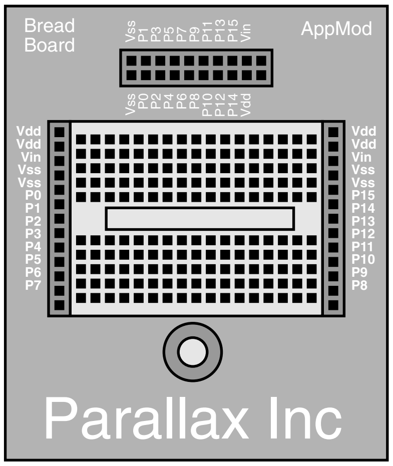

Bread Board
AppMod
Vss P1 P3 P5 P7 P9 P11 P13 P15 Vin
Vss P0 P2 P4 P6 P8 P10 P12 P14 Vdd
Vdd Vdd Vin Vss Vss P0 P1 P2 P3 P4 P5 P6 P7
Vdd Vdd Vin Vss Vss P15 P14 P13 P12 P11 P10 P9 P8
Parallax Inc

Figure 6-5 Parallax breadboard "AppMod."

that comes with the AppMod board to use it with the *TAB Electronics Build Your Own Robot Kit*. Some parts of the robot (like the motors, their wiring, large capacitors, and the IR LEDs) will interfere with it being plugged down directly onto the robot. As an added bonus, using the extender will save a lot of wear and tear on the robot's socket when you plug in and pull out the AppMod.

Secondly, I highly recommend using the stand off that is included in the AppMod kits. You will find that the vibration caused by running the robot across the floor can result in the AppMod loosening and rising up out of the socket if it is not "battened down." To use the included standoff with the *TAB Electronics Build Your Own Robot Kit*, I found that I had to insert three standard metal #6 washers between the standoff and the AppMod to prevent the AppMod's and the robot's connectors from being damaged. Figure 6-6 shows how the AppMod is to be installed on the robot.

The third issue to be concerned with when using the AppMod with the *TAB Electronics Build Your Own Robot Kit* is how power is supplied to the circuitry on the AppMod. In its "standard" configuration, +5 volts ("VDD") is passed to the AppMod circuit, but in the *TAB Electronics Build Your Own Robot Kit*, the +5 volts that is provided for the microcontroller and sensors on the card is not passed to the AppMod socket. The three AppMod applications with functions on them have built-in +5-volt regulators to provide the power for the circuitry. The breadboard and prototype AppMods do not have a regulator built in, which means you have to add your own.

This is not terribly hard to do using the circuit shown in Figure 6-7, which requires just a 78L05 (in a TO-92 package), a 10 µF (electrolytic) capacitor, and a 0.1 µF capacitor (any type). To wire it into the AppMod so that it can be used with the *TAB Electronics Build Your Own Robot Kit*, use the circuit shown in Figure 6-8—it should just take you a few seconds.

Programming PC Setup

Your PC should be running Microsoft Windows/95, Windows/98, Windows/ME, Windows/NT, or Windows/2000. If you are running MS-DOS, there is a version of the

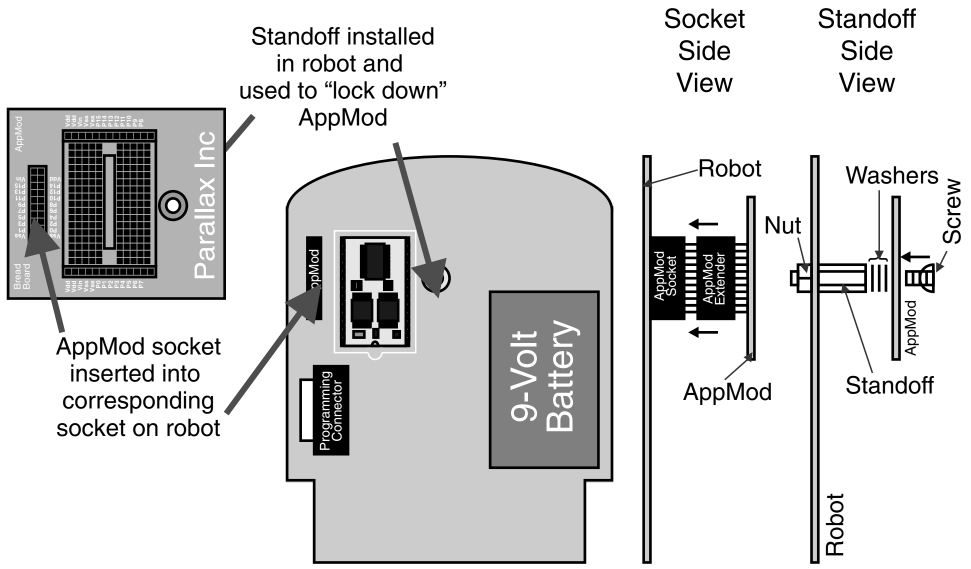


Figure 6-6 Installing an AppMod module onto the robot.

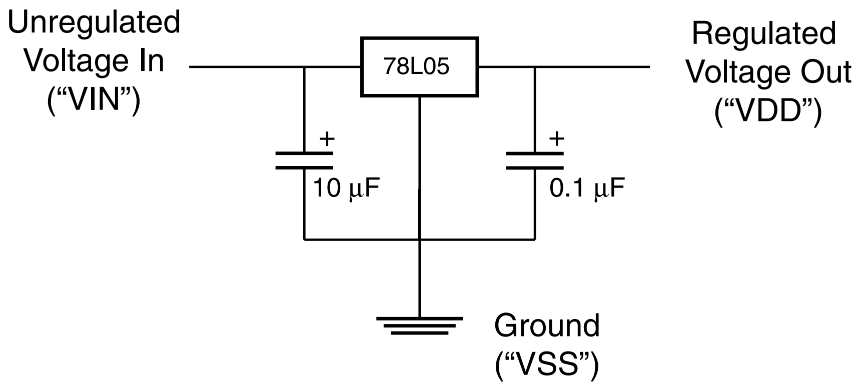


Figure 6-7 Circuit to provide +5 volts to AppMod.

BASIC Stamp 2 programming software (known as "Stamp2.exe"), which you can download from the Parallax website and use with the applications I have presented here. In this chapter, I focus on using the Windows version of the software. If you are using other operating systems, contact Parallax for development system availability for your operating system.

The first thing that you should do to get your PC ready for programming the BS2 in the *TAB Electronics Build Your Own Robot Kit* is to shutdown your PC and connect a 6- to 10-foot "straight-through" 9-pin "D-Shell" cable to your PC. These cables are often labeled "serial extenders" or "serial extensions." The cables must NOT be labeled as being "NULL MODEM" or for use with modems in any way (normally

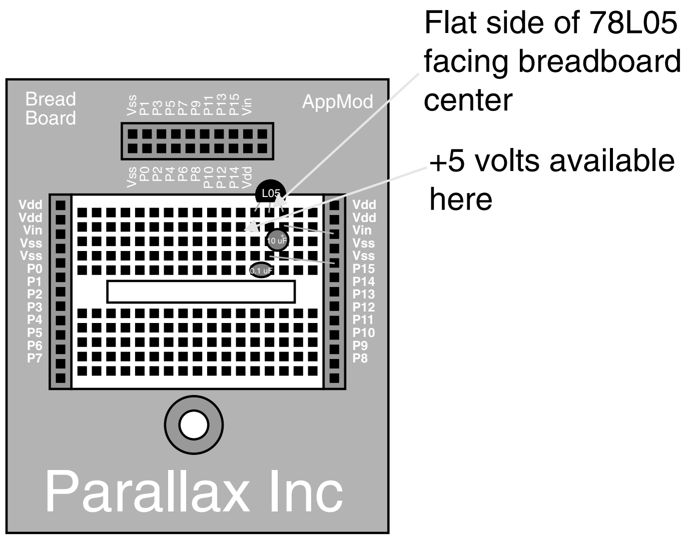

Flat side of 78L05 facing breadboard center
+5 volts available here
Bread Board
AppMod
Vss P1 P3 P5 P7 P9 P11 P13 P15 Vin
Vss P0 P2 P4 P6 P8 P10 P12 P14 Vdd
L05
10 uF
0.1 uF
Vdd Vdd Vin Vss Vss P0 P1 P2 P3 P4 P5 P6 P7
Vdd Vdd Vin Vss Vss P15 P14 P13 P12 P11 P10 P9 P8
Parallax Inc

Figure 6-8 Parallax breadboard AppMod with +5-volt power supply.

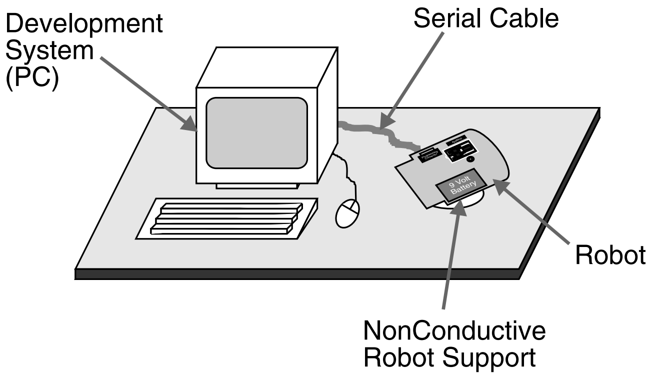


Figure 6-9 Recommended programming workstation.

modems have a 25-pin connector). Each cable should have a “male” and a “female” connector on each end (you should be able to plug the ends into one another to form a loop with the cable). The cable should not cost you more than $5.

Some PCs have a 25-pin RS-232 serial connector; in these situations, you will have to buy a DB-25 to 9-pin conversion connector. These are available for just a few dollars from most computer retailers.

When you have the cable connected to the PC, you are ready to boot the PC and follow the instructions on the CD-ROM that show you how to copy the files from the CD-ROM onto your PC’s hard drive. I have not included the information regarding loading the hard file here as it may change with different versions of the Parallax BASIC Stamp software. The CD-ROM will also point you to the *TAB Electronics Build*

Your Own Robot Kit's web page so that you can ask questions if you have problems loading the PC's hard drive.

Finally, when you have the serial cable connected to the PC and the software loaded, you can set aside a spot to program the robot while you are working on it. I recommend making a space beside your PC so you can test out the application and use the "debug" function to feed back to you what is happening with the application. Along with this, I recommended that you find something that is nonconductive for the robot to sit on that keeps the wheels from contacting anything when you are testing out the software (I use an old plastic wire reel). This stand will prevent the robot from running away from you when you first power it up or when the application is executing.

The "workstation" layout that I am proposing can be seen in Figure 6-9.

Experiments, Parts, Equipment, and Tools

With a BS2 plugged into the robot and the PC ready to start programming, you are ready to jump to the next level with the *TAB Electronics Build Your Own Robot Kit* and start to program the robot yourself. The experiments presented here will take you only a few moments to set up and will help explain different concepts of robot programming.

Along with the PC loaded with the programming tools, source code for the experiments presented above, and a serial cable, you will also require the following parts to work through the experiments:

TAB Electronics Build Your Own Robot Kit

Parallax BASIC Stamp 2

Parallax "Breadboard" AppMod card

Breadboard wire kit

78L05 +5-volt regulator

10-μF electrolytic capacitor

0.1-μF capacitor (any type)

10-LED "bargraph" display

10-pin, 470-ohm "SIP" resistor

470-ohm resistor

Just a few words on the different parts: most of them are available from a variety of different sources, including

Parallax
Radio Shack
Digi-Key
Jameco

Contact information for these sources (along with others) can be found on the CD-ROM HTML files.

The multiple-pin parts (the 78L05, 10-LED "bargraph display," and "10-pin, 470-ohm SIP" resistor) are all polarized parts. This means that you will have to be careful with the orientation of the parts as you wire them into the circuits. In the wiring drawings, I have made sure that the orientation of these parts is obvious. Don't worry if you seem to have miswired the component; I don't think any of these will be damaged if they are put in the wrong way.

Robot Control Application Programming Template

To make the job of developing applications easier, I have created the template that I have listed below. This template is available on the CD-ROM as a text file that can be loaded into the "stampw" application and allow you to add your own code to create an application. When you copied the CD-ROM files onto your PC's hard drive, this file was placed in the

```
C:\tabrobotkit\template
```

subdirectory.

This file should be used as the basis for all the applications that you create for the *TAB Electronics Build Your Own Robot Kit*.

I have included the text of the template below. Don't worry if you do not understand everything that's in it. I will explain the important statements as I work through the applications that follow.

```
`  Programming Template - Put Application Description HERE
`
`  Template originally created by Myke Predko
`  Copyright (C) 2001 McGraw-Hill
`
` { $STAMP BS2 }


`  Mainline
      high SC                       `  Set the I/O Bits As O/P
      high SD                       `   and High


`  #### - Put Application Code Here
`  Robot Interface Code Follows:
`
`  Myke Predko
`
`  Copyright (C) 2001 McGraw-Hill
`
`  Robot Commands
RobotStop     con  0                `  Stop the Robot
Behavior1     con  1                `  Random Movement
Behavior2     con  2                `  Photovore
Behavior3     con  3                `  Photophobe
Behavior4     con  4                `  Wall Hugger/Maze Solver
RobotForward  con  5                `  Move Forward for 200 msecs
RobotReverse  con  6                `  Move Reverse for 200 msecs
RobotLeft     con  7                `  Turn Left for 200 msecs
```

```
RobotRight    con  8                  `  Turn Right for 200 msecs
RobotLEDOn    con  9                  `  Turn on the Robot's LED
RobotLEDOff   con 10                  `  Turn off the Robot's LED
RobotPWM0     con 11                  `  PWM = 0% Duty Cycle
RobotPWM1     con 12                  `  PWM = 1st "Notch"
RobotPWM2     con 13                  `  PWM = 2nd "Notch"
RobotPWM3     con 14                  `  PWM = 3rd "Notch"
RobotPWM4     con 15                  `  PWM = 100% Duty Cycle
RobotPWM      con 16                  `  Return the Current PWM Value
RobotState    con 17                  `  Return the Executing State
RobotWhiskers con 18                  `  Return State of the "Whiskers"
                                      `   Bit 0 - Left "Whisker"
                                      `   Bit 1 - Right "Whisker"
RobotCDSL     con 19                  `  Return Value of Left CDS Cell
RobotCDSR     con 20                  `  Return Value of Right CDS Cell
RobotButton   con 21                  `  Return Last Remote Button
                                      `   Pressed
                                      `   0 - No Buttons Pressed
                                      `   1 - Leftmost Button Pressed
                                      `   2 - Middle Button Pressed
                                      `   3 - Rightmost Button Pressed
                                      `  After "RobotButton" Operation,
                                      `   Button Save in Robot is Cleared


`  Robot Interface Pins
SC con 14                             `  Define the I/O Pins
SD con 15
`  Robot Interface Variables
RobotData var byte                    `  Data Byte to Send to/Receive
                                      `   from Robot


`  Robot Operation Subroutines
RobotSend                             `  Send the Byte in "RobotData"
      low SC                          `  Hold Low for 1 msec before
      pause 1                         `   Shifting in Data
      shiftout SD, SC, LSBFIRST, [RobotData]
      high SC
      return


RobotSendReceive                      `  Send the Byte in "RobotData"
      low SC                          `  Hold Low for 1 msec before
      pause 1                         `  Shifting in Data
      shiftout SD, SC, LSBFIRST, [RobotData]
      pause 1                         `  Wait for Operation to Complete
      shiftin SD, SC, LSBPOST, [RobotData]
      high SC
      return
```

Experiment 1—"Hello World!"

Before starting to create BS2 applications for the *TAB Electronics Build Your Own Robot Kit*, I want to first show you how the BS2 is programmed and how you can use the PBASIC "debug" function to feed back information to you. This application is a very traditional one for first-time programmers or people working with a new programming language or development system. Its purpose is to demonstrate that the programmer can implement a simple application.

While you can find this application in the

```
C:\tabrobotkit\code
```

subdirectory, I would like to walk you through the process of creating the application on your own. I will not repeat these detailed instructions for creating an application in the later experiments, but this should act as a guide for you in creating your own applications.

First, the robot should have the BS2 installed (with Pin 1 pointing toward the rear as I have discussed above), be connected to the PC, and placed on a nonconducting stand (with the wheels free to turn) as I've shown in Figure 6-9.

With this done, turn on the robot. The LED on the robot will go on for a half second and the wheels will start to turn. Press the "Stop" button on the remote control, and leave the robot turned on. The robot is now waiting for a command from either the remote control or the BS2. As I have indicated elsewhere, the robot's battery will power both the robot and the BS2 for a long period of time—you do not have to feel as though you are going to have to "rush" through the process of programming the robot.

With the robot connected to the PC, you can now start up "stampw." This application can be invoked from either

```
C:\tabrobotkit\Hello World
```

or from the desktop (after copying the icon to the desktop as I showed in the software installation instructions).

"Stampw" will come up with the blank dialog box shown in Figure 6-10. Next, "open" the "template.bs2" file in

```
C:\tabrobotkit\Template
```

by either clicking on "File" and then "Open" or by simply pressing the "Ctrl" and "O" keys together. In either case, the "Open" dialog box shown in Figure 6-11 will come up. Click on "template" to highlight it and then click on the "Open" button. This will load the "template.bs2" file into the stampw editor.

Before you change the file, you should rename it. This is done by clicking on "File" and then "Save As" (as shown in Figure 6-12). When you have done this, you will get the dialog box shown in Figure 6-13. For this example, I changed the subdirectory so that the changed file was saved in to

```
C:\tabrobotkit\Hello World
```

and I saved the file as "My Hello World." Now, the "stampw" dialog box that you are working with should look like Figure 6-14. Where I have put in a comment (this is the single quote, '—everything to the right of this is ignored) and four "pound signs" (####), replace this with the text

```
    debug "Hello World!", cr         '  Output Debug Message
    end                              '  Nothing more to do
```

This can be done by deleting the four pound sign line and simply putting in the two new lines of text. The "stampw" dialog box should look like Figure 6-15.

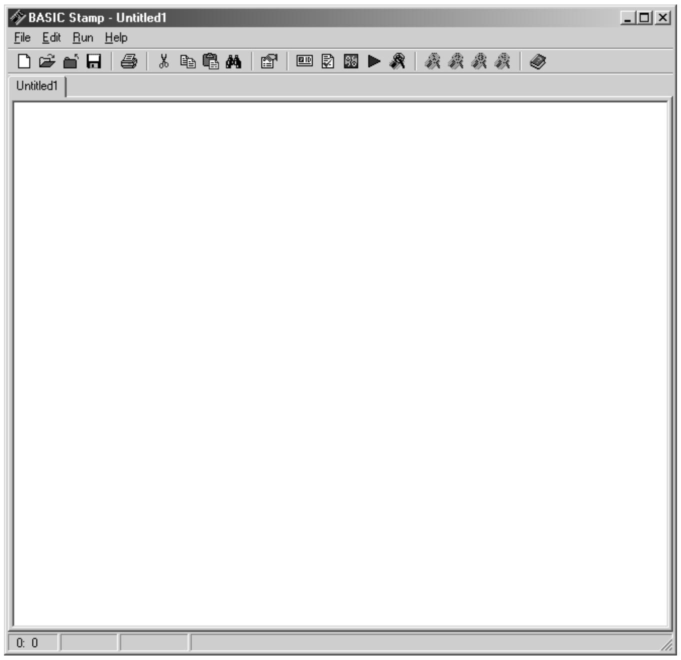


Figure 6-10 Parallax "Stampw" startup dialog box.

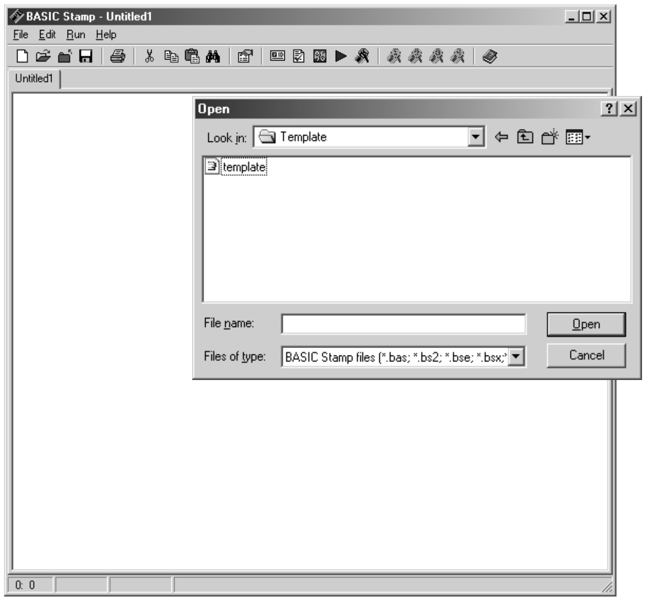
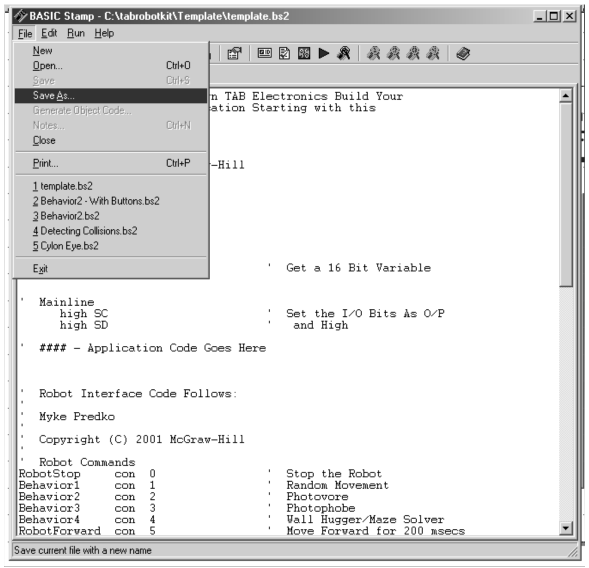
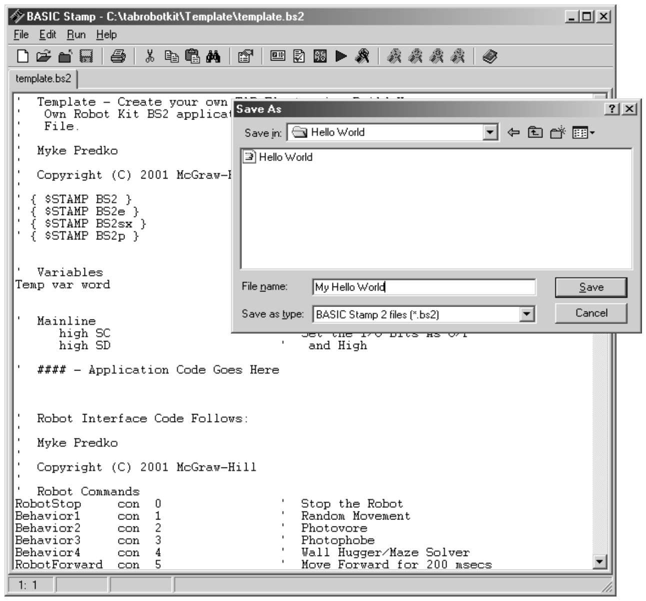


Figure 6-11 Loading the BS2 "template" file.



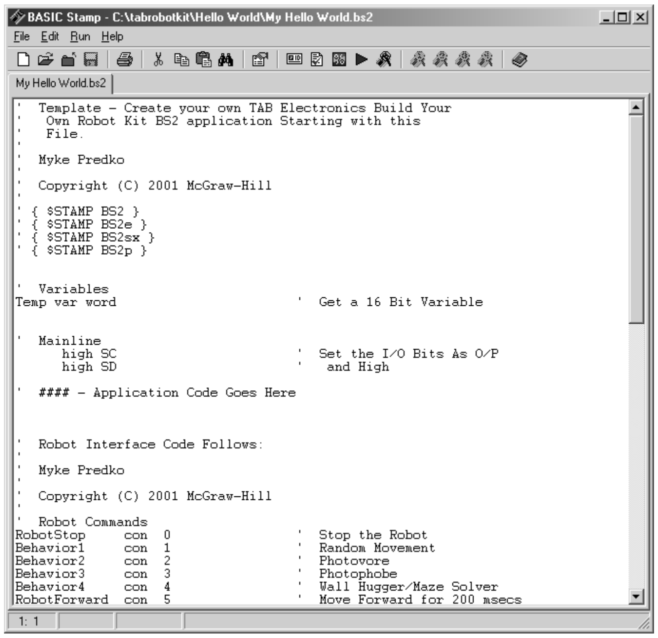
```
n TAB Electronics Build Your
ation Starting with this

-Hill

                              '  Get a 16 Bit Variable

'  Mainline
     high SC                  '  Set the I/O Bits As O/P
     high SD                  '   and High

'  #### - Application Code Goes Here


'  Robot Interface Code Follows:
'
'  Myke Predko
'
'  Copyright (C) 2001 McGraw-Hill
'
'  Robot Commands
RobotStop     con  0          '  Stop the Robot
Behavior1     con  1          '  Random Movement
Behavior2     con  2          '  Photovore
Behavior3     con  3          '  Photophobe
Behavior4     con  4          '  Wall Hugger/Maze Solver
RobotForward  con  5          '  Move Forward for 200 msecs
```

Figure 6-12 Saving a new copy of the BS2 template.


BASIC Stamp - C:\tabrobotkit\Template\template.bs2
File Edit Run Help
template.bs2
Save As
Save in: Hello World
Hello World
File name: My Hello World
Save as type: BASIC Stamp 2 files (*.bs2)
Save
Cancel
1: 1

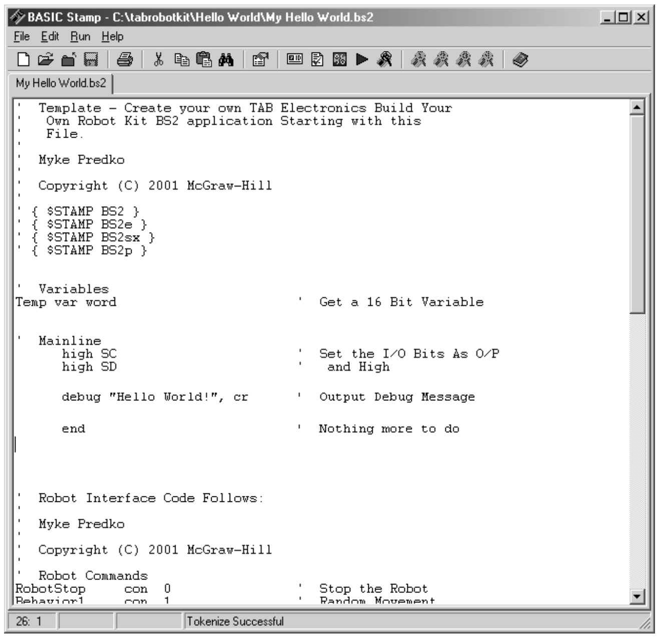
```
'  Template - Create your own
'   Own Robot Kit BS2 applica
'   File.
'
'  Myke Predko
'
'  Copyright (C) 2001 McGraw-H
'
' { $STAMP BS2 }
' { $STAMP BS2e }
' { $STAMP BS2sx }
' { $STAMP BS2p }


'  Variables
Temp var word


'  Mainline
     high SC
     high SD                  '   and High

'  #### - Application Code Goes Here


'  Robot Interface Code Follows:
'
'  Myke Predko
'
'  Copyright (C) 2001 McGraw-Hill
'
'  Robot Commands
RobotStop     con  0          '  Stop the Robot
Behavior1     con  1          '  Random Movement
Behavior2     con  2          '  Photovore
Behavior3     con  3          '  Photophobe
Behavior4     con  4          '  Wall Hugger/Maze Solver
RobotForward  con  5          '  Move Forward for 200 msecs
```

Figure 6-13 Renaming the template file.



```
'  Template - Create your own TAB Electronics Build Your
'   Own Robot Kit BS2 application Starting with this
'   File.
'
'  Myke Predko
'
'  Copyright (C) 2001 McGraw-Hill
'
' { $STAMP BS2 }
' { $STAMP BS2e }
' { $STAMP BS2sx }
' { $STAMP BS2p }


'  Variables
Temp var word                       '  Get a 16 Bit Variable


'  Mainline
     high SC                        '  Set the I/O Bits As O/P
     high SD                        '   and High

'  #### - Application Code Goes Here



'  Robot Interface Code Follows:
'
'  Myke Predko
'
'  Copyright (C) 2001 McGraw-Hill
'
'  Robot Commands
RobotStop     con  0                '  Stop the Robot
Behavior1     con  1                '  Random Movement
Behavior2     con  2                '  Photovore
Behavior3     con  3                '  Photophobe
Behavior4     con  4                '  Wall Hugger/Maze Solver
RobotForward  con  5                '  Move Forward for 200 msecs
```

Figure 6-14 Editing the template file.



```
'  Template - Create your own TAB Electronics Build Your
'   Own Robot Kit BS2 application Starting with this
'   File.
'
'  Myke Predko
'
'  Copyright (C) 2001 McGraw-Hill
'
' { $STAMP BS2 }
' { $STAMP BS2e }
' { $STAMP BS2sx }
' { $STAMP BS2p }


'  Variables
Temp var word                       '  Get a 16 Bit Variable


'  Mainline
     high SC                        '  Set the I/O Bits As O/P
     high SD                        '   and High

     debug "Hello World!", cr       '  Output Debug Message

     end                            '  Nothing more to do




'  Robot Interface Code Follows:
'
'  Myke Predko
'
'  Copyright (C) 2001 McGraw-Hill
'
'  Robot Commands
RobotStop     con  0                '  Stop the Robot
Behavior1     con  1                '  Random Movement
```

Figure 6-15 Application code added to template.

The first line of the new text ('debug "Hello World!", cr...') prints out the message "Hello World!" and starts a new line. The "debug" function will print the quoted text, numeric values, or control strings after it. Each new piece of data is separated by a comma (,). The "cr" control string is a short form for "Carriage Return" and starts a new line.

If you are going to print out the contents of a variable, you will probably want to modify the variable with something like the "dec(variable)" function, which will print out a numeric value for the contents of the variable instead of an ASCII representation of this value. You can find out more about "debug" and other built-in functions of PBASIC in the *Parallax Programming Manual* available on the CD-ROM.

After the "debug" statement, I have put in an "end" statement. This instruction stops the BS2 from further executing and puts the microcontroller built into the BS2 into "sleep" mode. "Sleep" is a very low power mode that essentially turns off the microcontroller while leaving its outputs unchanged. As a rule, you should always have an "end" statement at the end of your BS2 application and before the robot interface API subroutines at the bottom of the "template.bs2" file.

After you make changes to the application code, I recommend that you save it. This is done by either clicking on "File" and then "Save" or by pressing "Ctrl" and "S" together. If you are working on a complex application you may wish to save the application under a different file name, using the "Save As..." process I outlined above.

I have not included the source file for this application in this section because the actual application code is only two lines long and I think that you should attempt to enter it in yourself. I have included *my* version of this application in the

```
C:\tabrobotkit\Hello World
```

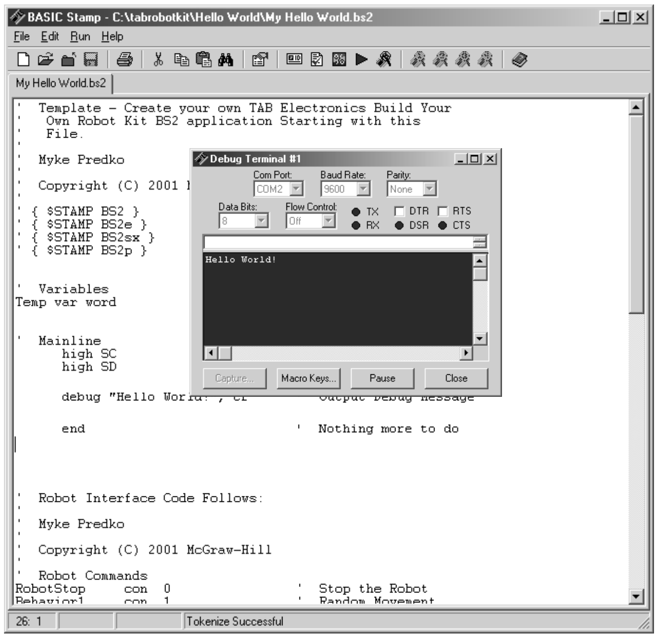
subdirectory in case you run into any problems or you aren't sure how the code should look after it has been entered into the "template.bs" file.

Once you make the changes you can run the application. This is done by either clicking on "Run" and then "Run" or by pressing "Ctrl" and "R" together. If everything goes well, a dialog box indicating that the code is being downloaded into the BS2 will appear and the "Debug Terminal" dialog box (shown in Figure 6-16) will pop up.

The "Debug Terminal" is used to display the information printed by the "debug" statements in the running BS2 application. For your first few applications, I recommend that you put in "debug" statements to help you understand where the BS2 application is and how it is executing.

There is a good chance that you will get the "Hardware Not Found" programming failure dialog box shown in Figure 6-17. This failure indicates that the PC was unable to communicate with the BS2. The first time you attempt to program the robot, the "stampw" application searches each of the serial ports built into the PC to see if there are any BS2s connected to the port.

The *TAB Electronics Build Your Own Robot Kit* has circuitry built into it that should allow "stampw" to find the BS2 connected into the robot automatic. If it can't find the port, then the error dialog box shown in Figure 6-17 will be displayed.



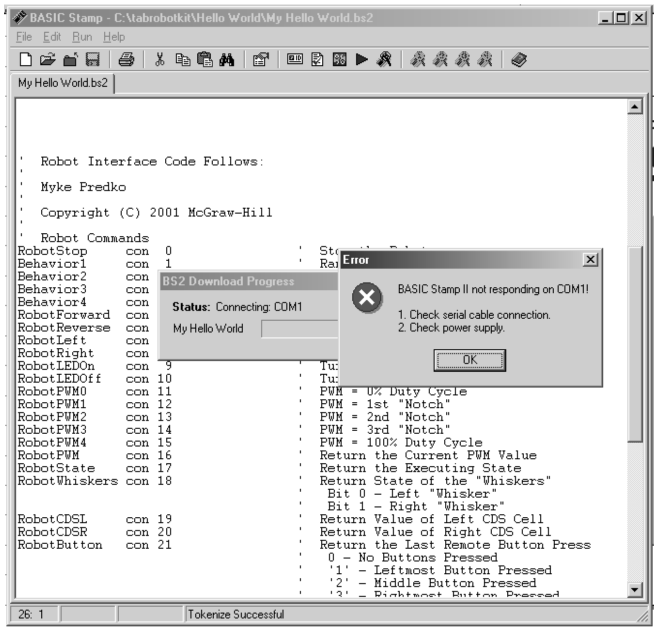
```
'  Template - Create your own TAB Electronics Build Your
'   Own Robot Kit BS2 application Starting with this
'   File.
'
'  Myke Predko
'
'  Copyright (C) 2001 
'
' { $STAMP BS2 }
' { $STAMP BS2e }
' { $STAMP BS2sx }
' { $STAMP BS2p }


'  Variables
Temp var word


'  Mainline
     high SC
     high SD

     debug "Hello World!", cr          Output Debug Message

     end                          '  Nothing more to do



'  Robot Interface Code Follows:
'
'  Myke Predko
'
'  Copyright (C) 2001 McGraw-Hill
'
'  Robot Commands
RobotStop     con  0               '  Stop the Robot
Behavior1     con  1               '  Random Movement
```

Com Port: COM2 Baud Rate: 9600 Parity: None
Data Bits: 8 Flow Control: Off ● TX ☐ DTR ☐ RTS ● RX ● DSR ● CTS

```
Hello World!
```

26: 1 Tokenize Successful

Figure 6-16 Application running properly.



```
'  Robot Interface Code Follows:
'
'  Myke Predko
'
'  Copyright (C) 2001 McGraw-Hill
'
'  Robot Commands
RobotStop     con  0               '  Sto
Behavior1     con  1               '  Ra
Behavior2     con
Behavior3     con
Behavior4     con
RobotForward  con
RobotReverse  con
RobotLeft     con
RobotRight    con
RobotLEDOn    con  9               '  Tu
RobotLEDOff   con 10               '  Tu
RobotPWM0     con 11               '  PWM = 0% Duty Cycle
RobotPWM1     con 12               '  PWM = 1st "Notch"
RobotPWM2     con 13               '  PWM = 2nd "Notch"
RobotPWM3     con 14               '  PWM = 3rd "Notch"
RobotPWM4     con 15               '  PWM = 100% Duty Cycle
RobotPWM      con 16               '  Return the Current PWM Value
RobotState    con 17               '  Return the Executing State
RobotWhiskers con 18               '  Return State of the "Whiskers"
                                   '   Bit 0 - Left "Whisker"
                                   '   Bit 1 - Right "Whisker"
RobotCDSL     con 19               '  Return Value of Left CDS Cell
RobotCDSR     con 20               '  Return Value of Right CDS Cell
RobotButton   con 21               '  Return the Last Remote Button Press
                                   '   0 - No Buttons Pressed
                                   '   '1' - Leftmost Button Pressed
                                   '   '2' - Middle Button Pressed
                                   '   '3' - Rightmost Button Pressed
```

Status: Connecting: COM1
My Hello World

BASIC Stamp II not responding on COM1!

1. Check serial cable connection.
2. Check power supply.

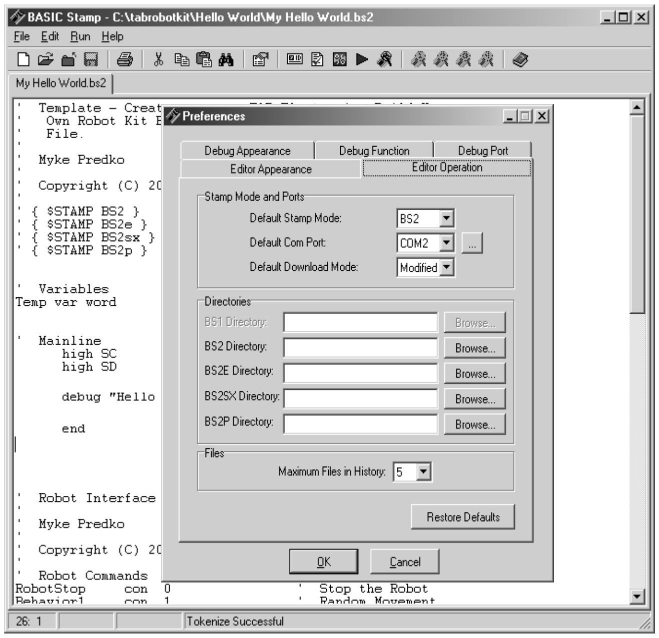
26: 1 Tokenize Successful

Figure 6-17 Failure programming robot.

There are a number of reasons Figure 6-17 will be displayed. If you are unable to program the robot, the first thing you should do is to explicitly define the serial port that is being used to program the robot. This is done by clicking on “Edit” and then “Preferences” of the “stampw” application dialog box. This will bring up a new dialog box as shown in Figure 6-18. Click on “Editor Operation” and make sure the serial port that “stampw” has chosen is the one you think should be used for the robot programming. If it isn’t, change the “Default COM Port:” to the one you think that should be used.

If this does not fix the problem, then you should go through the following checks:

1. Try other RS-232 ports in the PC. You may be trying to access the wrong port.
2. Check for applications that are already executing on the serial port. For example, if you have a Palm Pilot, its download software may be interfering with the operation of the stampw during the programming operation.
3. Check the power on the robot. Wave your hand in front of the IR Collision sensors, to see if the LED will light. Also check to see if the robot’s power switch is on.
4. Check the orientation of the BS2. It is very easy to put it in incorrectly. The pin 1 of the BS2 (and the semicircular indicator) should be pointing toward the rear (toward the wheels) of the robot.
5. Check the serial cable. You must have a “straight-through” 9-pin cable—you cannot use a “Null Modem” cable.


BASIC Stamp - C:\tabrobotkit\Hello World\My Hello World.bs2

```
'  Template - Creat
'   Own Robot Kit E
'   File.
'
'  Myke Predko
'
'  Copyright (C) 20
'
' { $STAMP BS2 }
' { $STAMP BS2e }
' { $STAMP BS2sx }
' { $STAMP BS2p }


'  Variables
Temp var word


'  Mainline
      high SC
      high SD

      debug "Hello

      end



'  Robot Interface
'
'  Myke Predko
'
'  Copyright (C) 20
'
'  Robot Commands
RobotStop     con  0                  '  Stop the Robot
Behavior1     con  1                  '  Random Movement
```

Preferences — Editor Operation

Stamp Mode and Ports
- Default Stamp Mode: BS2
- Default Com Port: COM2
- Default Download Mode: Modified

Directories
- BS1 Directory:
- BS2 Directory:
- BS2E Directory:
- BS2SX Directory:
- BS2P Directory:

Files
- Maximum Files in History: 5

26: 1 | Tokenize Successful

Figure 6-18 Checking serial interface ports.

If you have checked over the robot and cannot find the problem (or if you don't understand all the points that I am making), then you should check the *TAB Electronics Build Your Own Robot Kit*'s web page at:

`http://www.tabrobotkit.com`

for additional things to check as well as ask for suggestions from people for help.

Experiment 2—Detecting Collisions

After building the robot, you should have "calibrated" the IR proximity detectors by pressing the "Stop" button of the IR remote control and then waving your hands in front of the robot. When the proximity detectors detected your hand, the LED should have lit. Depending on where your hand was when the LED lit, you might want to adjust the potentiometer to "tune" it to the best location. The second experiment carries out much the same function but utilizes the BS2 for the logic instead of just the microcontroller built into the *TAB Electronics Build Your Own Robot Kit*.

In explaining how this application works, I will start by explaining how I would like the code to work and then work through how I implemented the actual BS2 PBASIC code. The function of the PBASIC code is quite simple, and you will be able to master it quite quickly especially in regard to interfacing to the *TAB Electronics Build Your Own Robot Kit*.

The application function could be written out in PBASIC "pseudo-code" as

```
      pause 100                       `  Stop the Robot from Running
      Robot(Stop)                     `  (1)


OuchlessLoop:
      Flag = 0                        `  (2)


Loop:                                 `  Repeat Here Forever  (3)
      pause 500                       `  Poll Once Every 1/2 Second


      if (No Collision) then OuchlessLoop                   `  (4)
                                          `  Collision
                Robot(LEDOn)              `  Flash LED - Turn on
                Pause 500                 `  (5)
                Robot(LEDOff)             ` LED off


                if (Flag = 1) then Loop    `  Don't Print "Ouch" Twice (6)
                Flag = 1                   `  Indicate "Ouch" Output (7)
                      debug "Ouch!  Something Hit Me", cr


      goto Loop                            `  (8)
```

To help explain what is happening in the application code, I have numbered eight sections of code. I will go through each of them below. I have also marked the application source code listed below with the same indicators so that you can see how the code is "actually" implemented. In several of these sections, I have defined some basic rules that you should follow when you develop your own BS2 code for the *TAB Electronics Build Your Own Robot Kit*.

1. After power is applied to the robot, a delay of 100 msecs is made to make sure that the robot's on board microcontroller has had an opportunity to power up and that the regulated power has stabilized. You should always wait a minimum of 100 µsecs before acessing the microcontroller on the robot.
2. A "Flag" is a variable that is used to save a previous state or indicate what else has been happening. In this case, the bit variable "Flag" is used to indicate whether the "ouch" message was printed during the previous time through the loop. This statement is the first one after the "OuchlessLoop" label. If neither collision sensor was active after being polled, then the Flag is reset so that the program can test to see whether or not either of the collision sensors was active during the previous poll. If they weren't, then the "Ouch" message can be printed. The variable "Flag" is declared at indicator (0) of the application source code listed below.
3. The "Loop" label is used to indicate the start of an "inner" loop to "OuchlessLoop" where execution returns if the collision sensors indicate that something is in the robot's way. After this label, a delay of 500 msec ("pause 500") was inserted so that the microcontroller in the robot could execute without being continually "commanded" by the BS2. In all your applications, you should wait a minimum of 20 msec before sending a new command or poll a state of the robot.
4. The IR proximity detectors ("collision sensors") are polled, and if there is no collision execution jumps to "OuchlessLoop" and "Flag" is cleared. If there is a collision, then execution continues to the following statements.
5. The IR proximity detectors have indicated that there is something within collision range of the robot. The BS2 code turns on and off the LED with half-second (500-msec) delays.
6. The "Flag" variable is tested to see if it is currently set. If it is, execution returns to loop where the LED will be polled again after a 500-msec delay. If "Flag" is not set, execution continues to the following statements.
7. The "Flag" variable is set to indicate that the code that prints the "Ouch" message has executed for this collision. After the flag is set, the "Ouch" message is sent to the controlling PC via the PBASIC "debug" function.
8. After the "Ouch" message is printed, execution jumps back to the "Loop" variable. Notice in this program that there is no way for the execution of the application to continue past this statement except to invoke the subroutines that command or poll the robot. This "goto Loop" statement is what makes this application an "endless loop"—it will never stop executing but instead will continually loop around, polling the IR proximity detectors.

To command and poll the *TAB Electronics Build Your Own Robot Kit* from the BS2, I have created two simple subroutines that will perform the communications for you. To make things even simpler for you, I have included them along with a series of constant defines that can be a reference for you and help you understand how the interface works.

To send a command to the robot, the code

```
RobotData = RobotCommand         `  Specify the Command
gosub RobotSend                  `  Send Command to the Robot
```

is used where "RobotCommand" is defined below as the commands from "Robot-Stop" to "RobotPWM4." These commands do not return any value to the BS2 application and the robot will start executing them immediately. The only case where there could be a problem is if you initiate behaviors 2 or 3, and there is an object in front of the robot that stops the behavior before it begins.

Along with the ability to send commands directly to the *TAB Electronics Build Your Own Robot Kit*, you can also request information of ("poll") the robot using the robot commands from "RobotPWM" to "RobotButton." These commands all return a value using code that is almost identical to the simple command send above:

```
RobotData = RobotCommand         `  Specify the Command
gosub RobotSendReceive           `  Send Command to the Robot
```

After the "RobotSendReceive" subroutine returns execution to the mainline, "RobotData" will have been changed to the return value for the function.

When the code

```
RobotData = RobotWhiskers        `  Read the Robot Whiskers
gosub RobotSendReceive
```

is executed, "RobotData" is initially made equal to "RobotWhiskers" or decimal 18. After "RobotSendReceive" has executed and returned, "RobotData" will be the value returned by the robot. If you look at the description for the function below, you will see that I have just defined the bits that are updated by the *TAB Electronics Build Your Own Robot Kit*.

To understand how these bit values work, remember that the least significant bit has a value of "1" while the next bit has a value of "2." So for Bit 0 to be the Left Whisker, a "1" is returned if there is a "collision" at that proximity detector and a "2" is returned if there is a "collision" at the right proximity sensor. The return values can be expressed in the table below:

Left	**Right**	**Bit 0**	**Bit 1**	**Value Returned**
Free	Free	0	0	0—No Collisions
Collide	Free	1	0	1—Left Collision
Free	Collide	0	1	2—Right Collision
Collide	Collide	1	1	3—Collision on Both Sensors

Defining the return values of a function by using the bits of a returned value is often used in hardware programming descriptions. You should not be fazed when you encounter this; instead go back to "Introduction to Programming" to understand what the decimal value is for each bit and what the actual returned byte value will be.

The full application code is listed below and can be found in the subdirectory:

```
C:\tabrobotkit\Detecting Collisions
`  Detecting Collisions - Print Debug
`   Message Indicating that Something has
`   Collided with the Robot
`
`  Myke Predko
`
```

```
'  Copyright (C) 2001 McGraw-Hill
'
' { $STAMP BS2 }


'  Variables
      Flag var bit                  '  One "Ouch" Per Collision (0)
'  Mainline
      high SC                       '  Set the I/O Bits As O/P
      high SD                       '   and High


      pause 100                     '  Stop the Robot from Running
      RobotData = RobotStop         '  (1)
      gosub RobotSend


OuchlessLoop:                       '  (2)
      Flag = 0


Loop:                               '  Repeat Here Forever  (3)
      pause 500                     '  Poll Once Every 1/2 Second
      RobotData = RobotWhiskers     '  Read the Robot Whiskers
      gosub RobotSendReceive


      if (RobotData = 0) then OuchlessLoop  '  (4)
                                    '  Collision
            RobotData = RobotLEDOn  '  Flash LED  (5)
            gosub RobotSend
            Pause 500
            RobotData = RobotLEDOff
            gosub RobotSend


            if (Flag = 1) then Loop '  Don't Print "Ouch" Twice  (6)


                  Flag = 1          '  Indicate "Ouch" Output  (7)
                  debug "Ouch!  Something Hit Me", cr
      goto Loop                     '  (8)
'  Robot Interface Code Follows:
'
'  Myke Predko
'
'  Copyright (C) 2001 McGraw-Hill
'
'  Robot Commands
RobotStop     con  0                '  Stop the Robot
Behavior1     con  1                '  Random Movement
Behavior2     con  2                '  Photovore
Behavior3     con  3                '  Photophobe
Behavior4     con  4                '  Wall Hugger/Maze Solver
RobotForward  con  5                '  Move Forward for 200 msec
RobotReverse  con  6                '  Move Reverse for 200 msec
RobotLeft     con  7                '  Turn Left for 200 msec
RobotRight    con  8                '  Turn Right for 200 msec
```

```
RobotLEDOn    con  9            `  Turn on the Robot’s LED
RobotLEDOff   con 10            `  Turn off the Robot’s LED
RobotPWM0     con 11            `  PWM = 0% Duty Cycle
RobotPWM1     con 12            `  PWM = 1st “Notch”
RobotPWM2     con 13            `  PWM = 2nd “Notch”
RobotPWM3     con 14            `  PWM = 3rd “Notch”
RobotPWM4     con 15            `  PWM = 100% Duty Cycle
RobotPWM      con 16            `  Return the Current PWM Value
RobotState    con 17            `  Return the Robot
RobotWhiskers con 18            `  Return State of the “Whiskers”
                                `   Bit 0 - Left “Whisker”
                                `   Bit 1 - Right “Whisker”
RobotCDSL     con 19            `  Return Value of Left CDS Cell
RobotCDSR     con 20            `  Return Value of Right CDS Cell
RobotButton   con 21            `  Return Last Remote Button
                                `   Pressed
                                `   0 - No Buttons Pressed
                                `   1 - Leftmost Button Pressed
                                `   2 - Middle Button Pressed
                                `   3 - Rightmost Button Pressed
                                `  After “RobotButton” Operation,
                                `   Button Save in Robot is Cleared

`  Robot Interface Pins
SC con 14                       `  Define the I/O Pins
SD con 15


`  Robot Interface Variables
RobotData var byte              `  Data Byte to Send to/Receive
                                `   from Robot


`  Robot Operation Subroutines
RobotSend                       `  Send the Byte in “RobotData”
      low SC                    `  Hold Low for 1 msec before
      pause 1                   `   Shifting in Data
      shiftout SD, SC, LSBFIRST , [RobotData]
      high SC
      return



RobotSendReceive                `  Send the Byte in “RobotData”
      low SC                    `  Hold Low for 1 msec before
      pause 1                   `  Shifting in Data
      shiftout SD, SC, LSBFIRST , [RobotData]
      pause 1                   `  Wait for Operation to Complete
      shiftin SD, SC, LSBPOST, [RobotData]
      high SC
      return
```

When you run this application, the robot can either be connected to the PC or not. If it is connected to the PC, then the “debug” function “Ouch” message will appear on the PC’s screen.

In the application, you should note that the first thing I do is command the robot to “Stop.” Despite this, the normal operations of the robot can still be commanded by the remote control. I am just pointing this out because I want to make sure that you understand that the remote control is the highest priority control for the robot.

There is one thing that you should remember when developing your own *TAB Electronics Build Your Own Robot Kit* code and that is that one of the 26 bytes available to applications in the BS2 is used by the interface code/APIs. This means that you have a maximum of 25 bytes available for your application.

This bit "Flag" uses one of the remaining variables for storage. Up to seven more bits can be defined without requiring more bytes from the total available. This means that in the "Detecting Collisions" application there are 24 bytes that are unused.

Experiment 3—Implementing the "Photovore" in the BS2

When I described the behaviors of the *TAB Electronics Build Your Own Robot Kit*, I used PBASIC "pseudo-code" to explain how the behaviors worked in a format that you will be more familiar with as you work with the robot. To demonstrate how accurate this representation of Behavior 2 was, I decided to convert the pseudocode to actual BS2 PBASIC code.

The result of this experiment is listed below and can be found in the

```
C:\tabrobotkit\Photovore
```

subdirectory of your PC.

```
`  Behavior2 - Implement Behavior 2 (the Photovore)
`   In PBASIC
`
`  Myke Predko
`
`  Copyright (C) 2001 McGraw-Hill
`
` { $STAMP BS2 }


`  Variables
i   var byte
j   var byte


`  Mainline
      high SC                       `  Set the I/O Bits As O/P
      high SD                       `   and High
      pause 100                     `  Wait for Robot to Get Setup
      RobotData = RobotStop         `  Stop Random Movement
      gosub RobotSend

      pause 100                     `  Wait for Robot to Get Setup
      RobotData = RobotLEDOn        `  Flash LED Twice
      gosub RobotSend
      pause 500
      RobotData = RobotLEDOff
      gosub RobotSend
      pause 500
      RobotData = RobotLEDOn
      gosub RobotSend
      pause 500
      RobotData = RobotLEDOff
      gosub RobotSend

Behavior2Loop:                        `  Read in the Current
      RobotData = RobotCDSL           `   Light Levels at Both
      gosub RobotSendReceive          `   CDS Cells
      i = RobotData                   `**    i = LeftCDS
      RobotData = RobotCDSR           `**    j = RightCDS
      gosub RobotSendReceive          `**    j = RightCDS
      j = RobotData                   `**    j = RightCDS


      if (i = j) then Behavior2Forward   `  Straight if Equal
      if (i > j) then Behavior2Right     `  Turn towards smaller
                                         `   CDS Cell Value
Behavior2Left:                        `  Left CDS Cell
      RobotData = RobotLeft           `**    Robot(TurnLeft, 80ms)
      gosub RobotSend                 `**    Robot(TurnLeft, 80ms)
      pause 75                        `**    Robot(TurnLeft, 80ms)
      RobotData = RobotStop           `**    Robot(TurnLeft, 80ms)
      gosub RobotSend                 `**    Robot(TurnLeft, 80ms)
      pause 75                        `**    Robot(TurnLeft, 80ms)
      goto Behavior2Forward


Behavior2Right:                       `  Right CDS Cell
      RobotData = RobotRight          `**    Robot(TurnRight, 80ms)
      gosub RobotSend                 `**    Robot(TurnRight, 80ms)
      pause 75                        `**    Robot(TurnRight, 80ms)
      RobotData = RobotStop           `**    Robot(TurnRight, 80ms)
      gosub RobotSend                 `**    Robot(TurnRight, 80ms)
      pause 75                        `**    Robot(TurnLeft, 80ms)


Behavior2Forward:                     `  Finished Turn, go Forward
      RobotData = RobotWhiskers       `  Look for Collisions
      gosub RobotSendReceive
      if (RobotData <> 0) then Behavior2End
      RobotData = RobotForward        `**    Robot(Forward, 80msecs)
      gosub RobotSend                 `**    Robot(Forward, 80msecs)
      pause 75                        `**    Robot(Forward, 80msecs)
      RobotData = RobotStop           `**    Robot(Forward, 80msecs)
      gosub RobotSend                 `**    Robot(Forward, 80msecs)
      pause 75                        `**    Robot(TurnLeft, 80ms)
      RobotData = RobotWhiskers       `  Look for Collisions
      gosub RobotSendReceive
      if (RobotData = 0) then Behavior2Loop


Behavior2End:                         `  At Light Source, Stop
      RobotData = RobotLEDOn          `  Turn on LED
      gosub RobotSend
      end
```

```
'  Robot Interface Code Follows:
   .
   :
```

Note that in listing this application, I have left out the robot definitions and subroutines because they are exactly the same in the previous application and the "template.bs2" file from which they were taken. For the remaining application listings, I will also leave out this code as it is redundant and doesn't add to your understanding of the application. Please remember that this code is present and is called by the different applications.

In the actual BS2 implementation of Behavior 2, you will see that I marked the instances with two asterisks (**) where I deviated from the pseudocode to put in actual BS2 PBASIC statements. In all the cases where the deviations were made, it was done to change a simple statement indicating a command to the robot to the more complex actual statements required to call the subroutines that communicate to the robot.

When working with the CDS cell light sensors on the robot and polling their current values using the "RobotCDSL" and "RobotCDSR" functions, remember that the lower the value returned, the brighter the light the CDS cell is exposed to. This can be confusing and lead to unpredictable operation of your application. If you do find that the CDS cells do not work as you might expect, look at the code and check to see whether or not you expect a larger value for a brighter light. If you did, then by reversing your logic you should find that the application will now work as you designed.

To test out the application, program it into the robot, turn off the robot, and go to a darkened room with a flashlight on at one end. You will find that the robot will move toward the flashlight as with the preprogrammed "Behavior 2" in the robot's microcontroller.

There is one problem with this application, and that is that it's not very convenient to execute. To try out the BS2 application code again you will have to turn off the robot, move it, and then turn it back on in order to restart the BS2 application. I found this to be quite annoying and not all that "user friendly." In the next experiment, I tried to rectify this problem by giving the application code the ability to start and stop the behavior.

At the end of the application, you should notice that I put in an "end" BS2 PBASIC statement. This program does not execute in an endless loop, so I made sure that it "ended" and did not attempt to continue executing where it would attempt to invoke the robot interfacing subroutines.

Experiment 4—Using Buttons to Control the Photovore

When I first implemented "Behavior 2" in the BS2, I found it annoying to have to turn on and off the robot to start the BS2 application. Another problem with the "Experiment 3" code that I didn't mention was that if I wanted to stop the robot or move it while the BS2 application was executing, I would get into a “Duel” between the remote control and the BS2 application. This further reduced the “user friendliness” of the application.

The solution to this problem was to have the remote control turn the application on and off using the three buttons (marked with a single bar, two bars, and three bars) on the remote control. These buttons are designed to allow the remote control to send commands to the *TAB Electronics Build Your Own Robot Kit* while it executes a BS2 application. “RobotButton” returns either the number of which button was last received by the robot of a “0” (indicating that no new button has been pressed). After a button number has been passed to the BS2, the return value is set to “0” indicating that no new button has been pressed as I indicated above.

The updated application code is listed below and can be found in the

```
C:\tabrobotkit\Protovore with Buttons
```

subdirectory.

```
`  Behavior2 - With Buttons
`  Implement Behavior 2 (the Photovore) in PBASIC.
`  Button Control Added to make the application
`   more controllable.
`  Button 1 (¦) = Start Behavior (does this Automatically)
`  Button 2 (¦¦) = Stop Behavior and Run from Remote
`
`
`  Myke Predko
`
`  Copyright (C) 2001 McGraw-Hill
`
` { $STAMP BS2 }


`  Variables
i   var byte
j   var byte


`  Mainline
      high SC                         `  Set the I/O Bits As O/P
      high SD                         `   and High
      pause 100                       `  Wait for Robot to Get Setup
      RobotData = RobotStop           `  Stop Random Movement
      gosub RobotSend


      pause 100                       `  Wait for Robot to Get Setup
      RobotData = RobotLEDOn          `  Flash LED Twice
      gosub RobotSend
      pause 500
      RobotData = RobotLEDOff
      gosub RobotSend
      pause 500
      RobotData = RobotLEDOn
      gosub RobotSend
      pause 500
      RobotData = RobotLEDOff
      gosub RobotSend


Behavior2Loop:                        `  Read in the Current
      RobotData = RobotButton         `  Any Buttons There?
      gosub RobotSendReceive
      if (RobotData = 1) then Behavior2Start
      if (RobotData = 2) then Behavior2End
                                      `  If nothing or Button3
Behavior2Start                        `   Just Carry On
      RobotData = RobotLEDOff         `  Turn off LED
      gosub RobotSend
      pause 75                        `  Delay before Starting Again
      RobotData = RobotCDSL           `   Light Levels at Both
      gosub RobotSendReceive          `   CDS Cells
      i = RobotData                   `**    i = LeftCDS
      pause 75
      RobotData = RobotCDSR           `**    j = RightCDS
      gosub RobotSendReceive          `**    j = RightCDS
      j = RobotData                   `**    j = RightCDS
      pause 75

      if (i = j) then Behavior2Forward `  Straight if Equal
      if (i > j) then Behavior2Right   `  Turn towards smaller
                                      `   CDS Cell Value
Behavior2Left:                        `  Left CDS Cell
      RobotData = RobotLeft           `**    Robot(TurnLeft, 80ms)
      gosub RobotSend                 `**    Robot(TurnLeft, 80ms)
      pause 75                        `**    Robot(TurnLeft, 80ms)

      RobotData = RobotStop           `**    Robot(TurnLeft, 80ms)
      gosub RobotSend                 `**    Robot(TurnLeft, 80ms)
      goto Behavior2Forward


Behavior2Right:                       `  Right CDS Cell
      RobotData = RobotRight          `**    Robot(TurnRight, 80ms)
      gosub RobotSend                 `**    Robot(TurnRight, 80ms)
      pause 75                        `**    Robot(TurnRight, 80ms)
      RobotData = RobotStop           `**    Robot(TurnRight, 80ms)
      gosub RobotSend                 `**    Robot(TurnRight, 80ms)


Behavior2Forward:                     `  Finished Turn, go Forward
      pause 75
      RobotData = RobotWhiskers       `  Look for Collisions
      gosub RobotSendReceive
      if (RobotData <> 0) then Behavior2End
      pause 75
      RobotData = RobotForward        `**    Robot(Forward, 80msecs)
      gosub RobotSend                 `**    Robot(Forward, 80msecs)
      pause 75                        `**    Robot(Forward, 80msecs)
      RobotData = RobotStop           `**    Robot(Forward, 80msecs)
      gosub RobotSend                 `**    Robot(Forward, 80msecs)
      pause 75                        `**    Robot(TurnLeft, 80ms)
      RobotData = RobotWhiskers       `  Look for Collisions
      gosub RobotSendReceive
      if (RobotData = 0) then Behavior2Loop


Behavior2End:                         `  At Light Source, Stop
      pause 75                        `  Delay before Starting Again
      RobotData = RobotLEDOn          `  Turn on LED
      gosub RobotSend
      pause 75                        `  Delay before Starting Again
      RobotData = RobotButton         `  Any Buttons There?
      gosub RobotSendReceive
      if (RobotData = 1) then Behavior2Start
      goto Behavior2End               `  No Start, Just Loop Around
`  Robot Interface Code Follows:
   .
   :
```

This application now executes in an endless loop and the "Photovore" behavior can be specified when you test the application. I found that relatively simple changes such as adding code like:

```
      RobotData = RobotButton         `  Any Buttons Pressed?
      gosub RobotSendReceive
      if (RobotData = 1) then Behavior2Start
```

made the application much easier to control with much less opportunity for "contention" between the BS2 and the remote control for the robot's actions.

I would recommend this type of control for all your applications to allow you to positively stop the robot in case you see that it gets into trouble.

Experiment 5—Cylon Eye

One of the features I am looking forward to taking advantage of in the *TAB Electronics Build Your Own Robot Kit* is the Parallax "AppMod" connector that is built into the robot. As I have said, this connector will allow you to add either predefined hardware features for the robot or ones that you can build yourself using the prototyping AppMods available from Parallax.

The example application that I have chosen is the scanning "eye" of the big clanking robots (see Figure 6-19) that were the humans' enemy in "Battlestar Galactica." While the show itself was quite forgettable, the scanning eye of the evil robots has made itself a fixture for many hardware companies as a quick visual indication that a piece of hardware is running properly. Using the "Breadboard" AppMod and six components, you too can turn your *TAB Electronics Build Your Own Robot Kit* into a monster that Commander Adama would have stopped at nothing to destroy.

The AppMod circuit that I came up with is shown in Figure 6-20 and requires the following parts:

78L05 +5-volt regulator

10 LED "bargraph" display

470-ohm 9-resistor "SIP"

Figure 6-19 Cylon Centurion from "Battlestar Galactica."

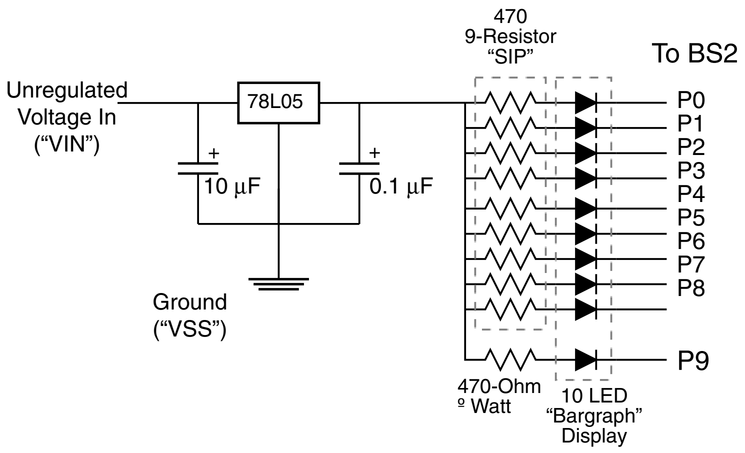


Figure 6-20 Circuit built on AppMod to implement "Cylon Eye."

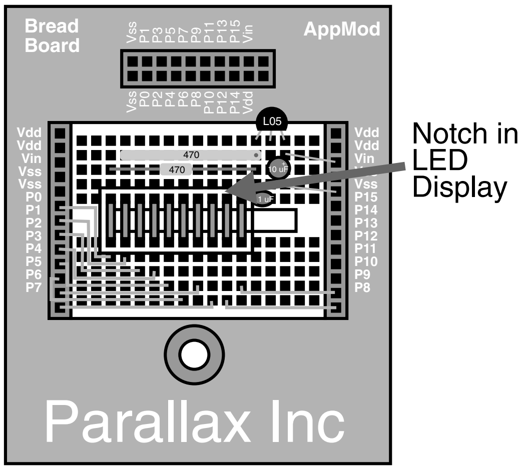

Bread Board
AppMod
Vss P1 P3 P5 P7 P9 P11 P13 P15 Vin
Vss P0 P2 P4 P6 P8 P10 P12 P14 Vdd
L05
470
470
10 uF
1 uF
Vdd Vdd Vin Vss Vss P0 P1 P2 P3 P4 P5 P6 P7
Vdd Vdd Vin Vss P15 P14 P13 P12 P11 P10 P9 P8
Notch in LED Display
Parallax Inc

Figure 6-21 Parallax breadboard AppMod with "Cylon Eye" application.

470-ohm, 1/4-watt resistor

10 μF electrolytic capacitor

0.1 μF capacitor (any type)

Parallax "Breadboard" AppMod board

22-gauge connecting wire

Once you have the parts, you can wire them into the Breadboard AppMod using the diagram shown in Figure 6-21. While it is not very hard to wire, I found that it took me a bit of time to plan it out correctly. I suggest that when you are wiring the 10-LED bargraph display to the BS2 you start with bit "P7" and work your way back to "P0." This should give you a bit more space to put in the wiring.

To test out the application you can use the code below. It is found in the

```
C:\tabrobotkit\Cylon Eye
```

subdirectory of your PC's hard drive. When you first program the robot, you might want to take out the commented code that "Stops the Robot from Running." This way you will be able to test out the application code without having to use the remote control to stop the robot when it is on your workbench.

```
'  Cylon Eye - Run "Scanning" Eye Across LEDs
'   Connected to LEDS connected to P0 to P9
'   of the BS2
'
'  Myke Predko
'
'  Copyright (C) 2001 McGraw-Hill
'
' { $STAMP BS2 }
'  Variables
Temp var word                         '  Get a 16 Bit Variable


'  Mainline
      high SC                         '  Set the I/O Bits As O/P

      high SD                         '   and High
'     pause 100                       '  Stop the Robot from Running
'     RobotData = RobotStop
'     gosub RobotSend

      OUTS = 65534                    '  Make All the I/O Ports High
                                      '  Except P0,
      DIRS = 65535                    '  And Outputs


UpLoop:                               '  Move the LEDs Up
      pause 100                       '  Delay before LED Update
      Temp = (OUTS * 2) + 1           '  Calculate New LED Value
'     debug dec(Temp), cr             '  #### - Output Here for Debug
      if (Temp = 65023) then DownDo   '  At the Last LED?

UPDo:
      OUTS = Temp                     '  Save LED Value
      goto UpLoop                     '  Repeat


DownLoop:                             '  Now, Move Back Down
      pause 100                       '  Delay before LED Update
      Temp = (OUTS / 2) + 32768       '  Divide by 2 to come down
'     debug dec(Temp), cr             '  #### - Output Here for Debug
      if (Temp = 65534) then UpDo     '  Go the Other Way


DownDo:
      OUTS = Temp                     '  Save the New Down Value
      goto DownLoop

AppStop:                              '  #### - Stop for Data Examine
      end


'  Robot Interface Code Follows:
   .
   :
```

Of the example applications that I have presented in this chapter, this one is probably the most confusing in terms of how it is implemented. When I developed the application, I wanted to come up with something that was very efficient. However, it is not easy to understand unless you are familiar with a few concepts.

The most important concept to understand is that when you are wiring an LED to a microcontroller, the cathode (negative terminal) is normally connected to the microcontroller and the anode (positive terminal) is connected to the power supply. The reason for wiring the LED this way is because of the operation of many initial microcontroller and NMOS (a precursor to CMOS) circuits. These circuits could "sink" (pass to ground) a lot more current than they could "source" (pass from V_{cc} or V_{dd}). To follow this convention, I have wired the LEDs with their cathodes wired to the BS2 through the AppMod.

The confusing aspect of wiring the LEDs in this way is that you must drive out a high voltage (a "1") to turn off the LED and use a low voltage (a "0") to turn it on. In the "Cylon Eye" application, I first put all the BS2's I/O ports into "output mode" and then I write a "1" to each of the bits except for the least significant one. This will turn off all the LEDs except for the one connected to P0.

I could have implemented the change in the active LED a number of different ways. I elected to shift the active bit up and down by multiplication and division. To move the LED from a lower bit to a higher bit, I multiplied the value in the 16 I/O pins by 2 and added 1 to the result to ensure that the least significant bit would be a "1." Remember that the value in the I/O port is an odd number, and when you multiply an odd number by two, you get an even number.

To go "down" the LEDs, I reversed the process by dividing the value in the I/O port by two and adding 32,768, which sets the high bit of the I/O port: 2 to the power 15 equals 32,768.

While I've taken into account that the port is sixteen bits in size, note that I kept the bits that I wrote to within the range of the ten LEDs that I wired to the AppMod board. I am pointing this out because while there are sixteen possible bits available to the AppMod board from the BS2, the most significant two bits (bits 14 and 15) are used for communications with the *TAB Electronics Build Your Own Robot Kit*. This means that if you want to "extend" the application with more LEDs you would be limited to 14 bits and not the full 16 available to the BS2.

Null Applications

When I was creating my initial applications for the BS2, I found that I often wanted to stop the robot without having to reach over for the remote control. The following application, found in the

```
C:\tabrobotkit\Null
```

subdirectory of your PC's hard file will stop the robot from running Behavior 1 when it first starts executing. It will allow you to leave the robot on your work bench powered on without worrying about it trying to run away from you.

```
'  Robot Null - Stop the robot's wheels and
'   the BS2 from controlling the robot
'
'  Myke Predko
'
'  Copyright (C) 2001 McGraw-Hill
'
' { $STAMP BS2 }

'  Mainline
      high SC                         '  Set the I/O Bits As O/P
      high SD                         '   and High

      pause 100                       '  Stop the Robot from Running
      RobotData = RobotStop
      gosub RobotSend


      end                              `  Nothing more to do



`  Robot Interface Code Follows:
   .
   :
```

The second application that I found to be quite simple is just an "end" statement executed as a BS2 application. This application is found the same subdirectory as the previous one.

```
`  Robot Null 2 - Stop BS2 Executing
`
`  Myke Predko
`
`  Copyright (C) 2001 McGraw-Hill
`
` { $STAMP BS2 }

`  Mainline
      high SC                          `  Set the I/O Bits As O/P
      high SD                          `   and High


      end                              `  Nothing more to do


`  Robot Interface Code Follows:
   .
   :
```

This second application is useful in cases where the robot's BS2 has been programmed with an application but you do not want to use it and you want to avoid having to pull out the BS2 (which could damage it). By programming this application into the BS2, the robot will run as if there isn't a BS2 in the socket.

Ideas for Experiments

In this chapter and the five previous ones, I have gone over a lot of information about the *TAB Electronics Build Your Own Robot Kit* and presented you with the basic knowledge needed to develop your own BASIC Stamp 2 applications. The sample applications presented in this chapter are designed to be a beginning point for you and to whet your appetite for what you can do on your own with the robot.

I hope that you will become an active participant in the forum provided on the *TAB Electronics Build Your Own Robot Kit* (on http://www.tabrobotkit.com) as well as the StampList, and that you will share your ideas and help out others.

To finish off the chapters on this CD-ROM, I want to leave you with a few ideas for applications that you can implement yourself using the information I have given you. I look forward to seeing what you come up with, and I hope that you are excited to discover how far you can push the *TAB Electronics Build Your Own Robot Kit*.

- *Line following robot*. Add some CDS Cells to the bottom of the robot to follow a lighter or darker line on the floor.
- *Firebot*. Add a pyrometer (heat sensor) and a fan to your robot and see if you can put out a candle burning in a room.
- *Robot shadow*. Come up with a way the robot will follow you and only you.
- *Sound-activated robot*. Add a microphone with an amplifier to control the motion of the robot. A simple example of this would be starting and stopping each time a loud noise was "heard." Could you expand this to respond differently to different sounds?
- *Robot animal*. Using the model of behaviors, can you create a program for your robot so that it acts like an animal? Think about how you would model concepts such as "food," "shelter," "danger," and "prey." These are questions and problems that continue to confound roboticists. Maybe you have the answers to "true" artificial intelligence.