

GROWBOT

Manual Rev. 1.2 for Rev C Circuit Board

The Parallax GrowBot is a mobile robot kit based on the BS2-IC module. The kit includes the parts required to install: two LEDs, a front bumper, two photoresistors, a pushbutton, a speaker, and a dual servo drive train. Two AppMod headers allow AppMods to be connected for functional upgrades and provide additional access to the BS2's I/O pins. This documentation shows you how to assemble, program, and customize your GrowBot. This document will be updated as necessary and the most recent version available in Adobe PDF format on the Parallax web site <http://www.parallaxinc.com>.

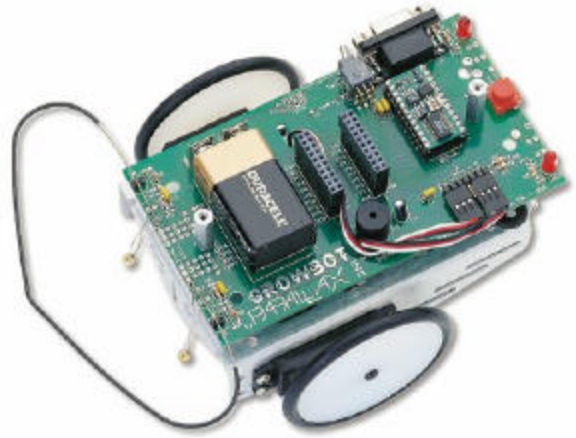


Table of Contents

| | |
|--|----|
| Chapter 1: Packing List..... | 2 |
| Chapter 2: Tools Required..... | 4 |
| Chapter 3: Soldering Tips..... | 5 |
| Chapter 4: GrowBot PCB Assembly..... | 6 |
| Chapter 5: GrowBot PCB Testing..... | 11 |
| Chapter 6: Chassis Assembly..... | 13 |
| Chapter 7: Final Assembly..... | 14 |
| Chapter 8: Program the GrowBot..... | 15 |
| Appendix A: GrowBot Schematics..... | 17 |
| Appendix B: GrowBot Programming Tips and PBASIC Quick Reference Guide..... | 19 |

AppMods are a cool way to expand your GrowBot's abilities. You can add sounds to your GrowBot and/or give it a user interface, or add whatever you want with an AppMod Breadboard.

Currently available for your GrowBot:








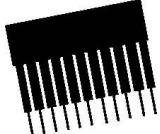




| | |
|-------------------------------|---|
| Sound Module (#27911) | Mini sound studio: manual & stamp controlled record/playback of sounds. |
| LED Display Terminal (#27912) | 4 digit LED terminal with 4 pushbuttons. |
| Breadboard (#29114) | Blank breadboard that fits into an AppMod header for expansion. |









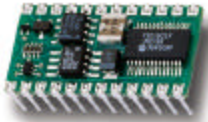

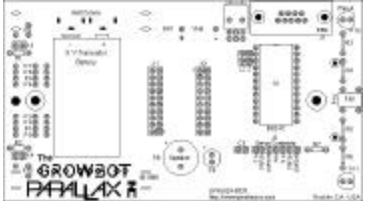


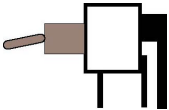



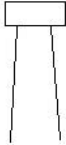
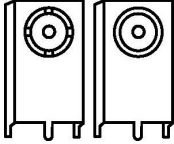




GrowBot starts here!

Chapter 1: Packing List

Before building your GrowBot, verify that you have all the parts. Parallax stock codes are provided in case you need to order a spare or replacement part.

| | | |
|--|--|--|
| <p>Three (3) 10K Ohm 1/8W 5% brown-black-orange #150-01031</p>  | <p>Four (4) 220 Ohm 1/8W 5% red-red-brown #150-02211</p>  | <p>One (1) 24 pin socket #450-02401</p>  |
| <p>One (1) 220K Ohm 1/8W 5% red-red-yellow #150-02240</p>  | <p>One (1) 100K Ohm 1/8W 5% brown-black-yellow #150-01041</p>  | <p>One (1) tact button switch #400-00002</p>  |
| <p>One (1) red snap-on button #400-00005</p>  | <p>Two (2) tall headers #450-02002</p>  | <p>Two (2) 2-pin SIP headers #451-00201</p>  |
| <p>One (1) 7-pin SIP header #451-00701</p>  | <p>Two (2) LED red diffused T 1-3/4 #350-00006</p>  | <p>Two (2) flathead screws 3/8" Phillips 4-40 #700-00016</p>  |

| | | |
|--|--|---|
| <p>One (1) 10uF 16V aluminum electrolytic capacitor #201-01061</p>  | <p>Five (5) 0.1 uF ceramic Capacitors #200-01040</p>  | <p>One (1) DB-9 connector #452-00005</p>  |
| <p>One (1) sound generator #900-00001</p>  | <p>Two (2) Modified Servos #900-00008</p>  | <p>One (1) antenna wire 0.026" 303 stainless #700-00021</p>  |
| <p>Two (2) drive wheels #700-00013</p>  | <p>Two (2) stand offs, 4-40 3/4alum. #700-00008</p>  | <p>BASIC Stamp II #BS2-IC</p>  |
| <p>One (1) 4 AA battery holder 700-00005</p>  | <p>One (1) GrowBot PCB – Rev C #300-15002</p>  | <p>One (1) GrowBot disk #950-00003</p>  |
| <p>One (1) serial cable #800-00003</p>  | <p>One (1) switch #400-00013</p>  | <p>Two (2) O-ring wheels #700-00011</p>  |

| | | |
|---|---|--|
| <p>Two (2) photoresistors #350-00009</p>  | <p>One (1) set of 9-volt battery clips #452-00013 and -14</p>  | <p>One (1) "ball" tail wheel #700-00009</p>  |
| <p>One(1) GrowBot chassis & hardware #29101</p>  | <p>One (1) #4 nylon washer 700-00015</p>  | <p>One(1) all-important cotter pin 700-00023</p>  |

Chapter 2: Tools Required

Building the GrowBot requires the following tools not included in this kit (Radio Shack part numbers are included in case you want to buy these pieces locally):

1. Soldering iron (RS #64-2051)
2. Solder (RS #64-005)
3. Small Phillips screwdriver
4. Small cutters (RS #64-1833)
5. Small needle-nose pliers

Once the GrowBot is built and you are ready to program and run, you'll need an IBM PC compatible computer running DOS or Windows, four (4) AA alkaline or NiCad batteries, and one (1) 9-volt battery.

Chapter 3: Soldering Tips

Attaching components to the GrowBot's PCB assembly requires soldering. If you haven't soldered before, there are some great tips included in Scott Edwards' *Programming and Customizing the BASIC Stamp Computer*¹. Below is a consolidated summary of a few of Scott's soldering tips.

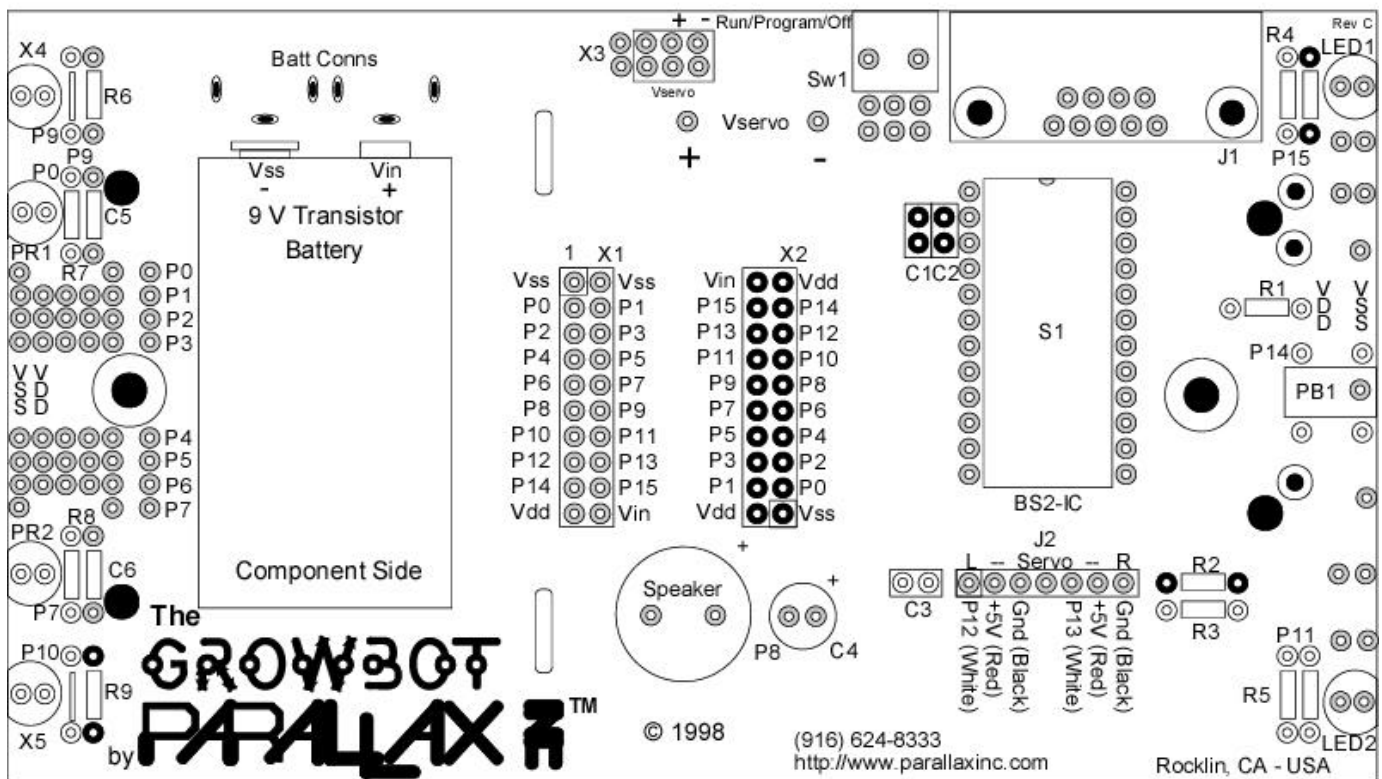
Accidents can and do happen. Soldering irons are very hot and can produce severe burns. Please read the following hints carefully and heed all warnings.

1. Use water soluble rosin core solder. Rosin core solder will provide better quality solder joints and will clean up easily with warm water after the soldering is over.
2. Solder only in a well lit, well ventilated area. Ensure that there are no explosive/combustible/flammable fumes or vapors in the room. Do not solder in the presence of flammable materials. Breathing solder fumes is not healthy. If you experience a headache or nausea, stop soldering, get some fresh air, air out the room, and increase the ventilation in the room that you are soldering in.
3. Always wear goggles. As the solder is heated and the rosin starts fluxing, small pops can eject minuscule blobs of molten solder quite far . You **MUST** protect your eyes from this.
4. Do not eat or drink while soldering. Solder contains lead. Lead is poisonous and it is easily absorbed through the skin. Be sure to wash your hands after soldering.
5. When the iron is hot, take it out of the stand and wipe the tip on the sponge with a match-strike motion. This will remove excess solder and clean the tip a bit.
6. Moisten the sponge with water so it is damp, but not soaking wet. Put the iron in the stand and plug it in. Don't touch the heating element or anything else that is metallic on the soldering iron. To avoid painful burns, touch only the plastic portions of the handle.
7. Apply a little solder to the tip of the soldering iron. If it beads up, wipe the iron on the sponge again and repeat until the tip of the soldering iron has a clean shiny thin coating of solder. This is called "tinning".
8. Bring the freshly tinned tip of the iron into contact with the wires/metals you want to solder. A tinned clean tip will help transfer heat into the target metals. After one second, bring the solder into contact with the metals (not the soldering iron itself). If the metals are hot enough, the flux will flow out of the solder and clean the area, then the solder will melt and flow around the metals.
9. Remove the solder – pause – then remove the iron without jostling the molten solder left behind. If soldered properly, smooth shiny fillets will result. A bulbous or grainy appearance indicates a poor solder joint and should be resoldered.

¹ Order part #27951 from Parallax for \$34.95, less for educational use.

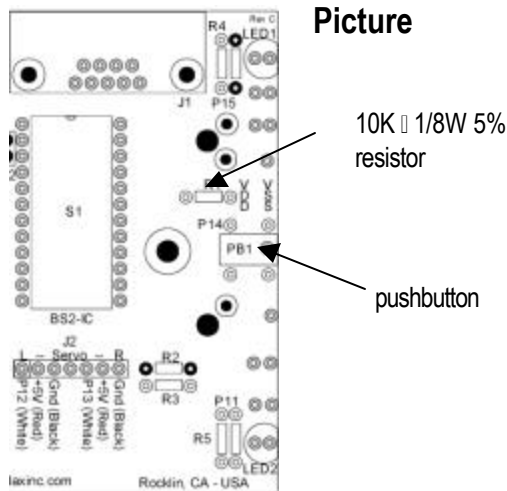
The section shows you how to assemble GrowBot's I/O devices step by step. If you experience difficulty, or find inconsistencies between these instructions and reality, check the downloads section on our web site <http://www.parallaxinc.com> for updated documents, photographs, and pointers. If you are still stuck please call our technical support at (916) 624-8333.

Component locations are labeled on the GrowBot printed circuit board. Soldering is done on the bottom of the GrowBot's printed circuit board (PCB) except where noted. (The top of the pcb is depicted below.) Throughout this documentation references are made to the "front" of the GrowBot. The front is shown below as the left side.

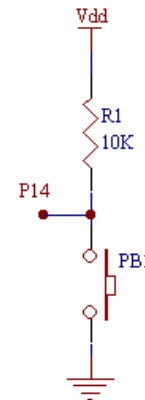


Pushbutton Circuit

1. Build the pushbutton circuit by bending the leads one (1) 10K ohm resistor (brown black orange) and placing it in position R1 onto the component side (top) of the pcb.
2. Insert the pushbutton where designated by position P14 (PB1).
3. Turn the GrowBot PCB over, solder these components and trim the leads. Do not install the red plastic button operator until after the pcb has been washed (end of chapter 4).



Schematic



Battery Monitoring Circuit / Sound Generator

The BS2-IC shares I/O line 8 (P8) between the battery monitoring circuit and the sound generator, so it makes sense to build these circuits together.

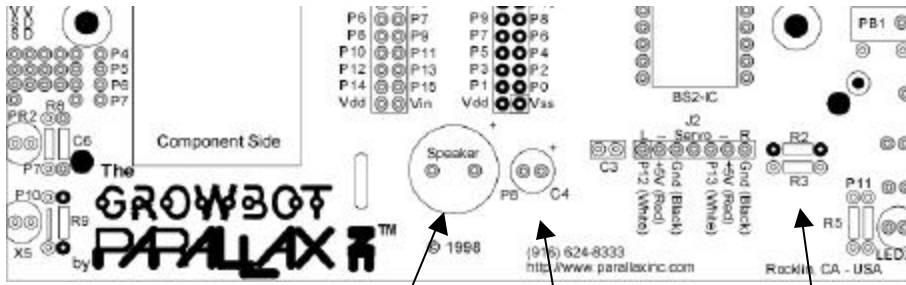
1. Build the battery monitoring circuit by bending the leads on the 220K ohm resistor (Red Red Yellow) and the 100K ohm resistor (Brown Black Yellow) and placing them into positions R2 and R3, respectively.
2. Insert the sound generator and the 10uF 16V aluminum electrolytic capacitor into P8 and C4, respectively. The capacitor's white "-" stripe is away from the PCB's "+" sign. The speaker's "+" lead should be placed accordingly on the board, next to the "+" sign.
3. Turn the GrowBot PCB over, solder these components and trim the leads.

CB Sound Generator Specifications

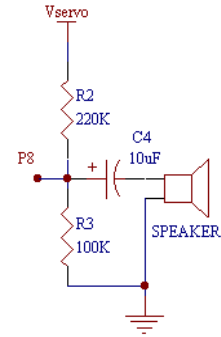
| Rated Voltage (VDC) | Average current consumption (mA) | Sound pressure level (dB) | Resonance frequency |
|---------------------|----------------------------------|---------------------------|---------------------|
| 20K | 8.5 | min 85 typ 89 at 10 cm | 4,096 Hz |

Picture

Schematic



speaker
10uF 16v cap
R2 – 220K
R3 – 100K



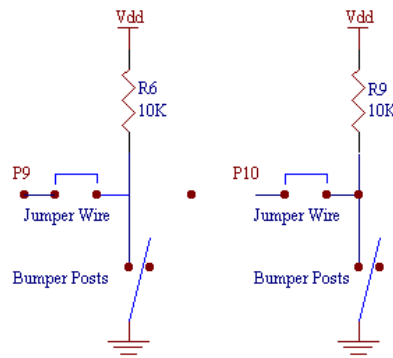
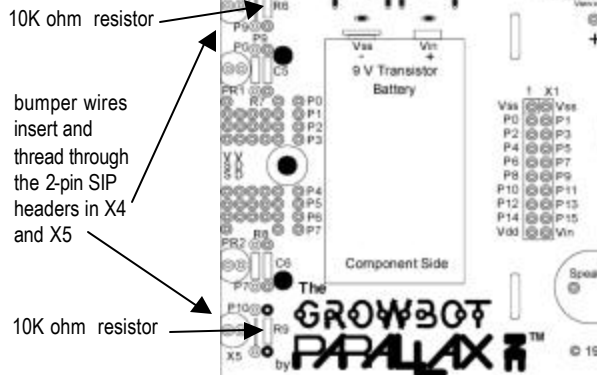
Bumper

The bumper wire is a 0.026 inch diameter stainless steel spring wire covered with insulating shrink tubing. The bumper wire uses the front X4 and X5 I/O locations, with a 2-pin SIP header providing contact closure. To install the bumper:

1. Insert the two 2-pin SIP headers into the front I/O locations X4 and X5. The two locations are denoted by the outermost round circles.
2. Insert two 10K ohm resistors (Brown Black Orange) into positions R6 and R9.
3. Cut off two short leads from a scrap resistor and solder them across P9 and P10.
4. Flip the board over (to the solder side), solder and trim the leads.
5. Insert the ends of the bumper wire into the set of holes next to PR1 and PR2, touching the back portion when the bumper is pushed. Don't solder the end of the bumper wire leads – bend the leads about 45° to keep the bumper in place.

Picture

Schematic



Photoresistors

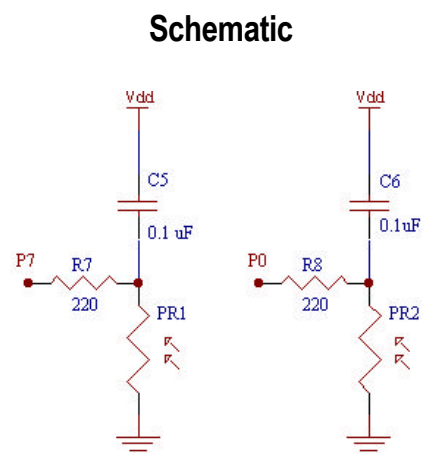
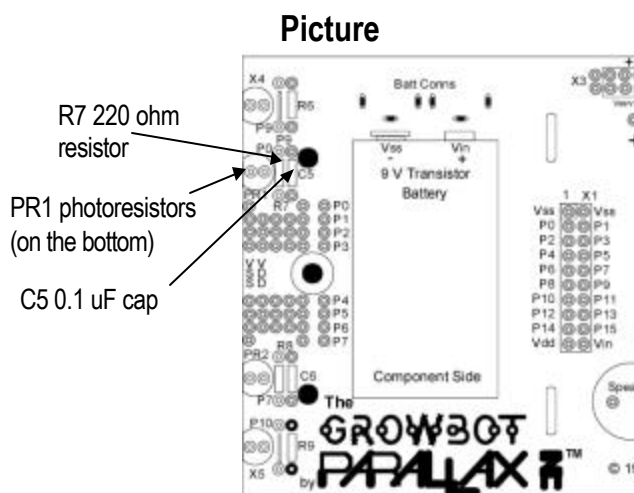
GrowBot includes two photoconductive cells (photoresistors or LDRs) from EG&G Vactec (part #VT935G group B). The cells provide an inexpensive solution for light level sensing applications. These cells are from Allied Electronics as part number 980-0105. Below are the specifications in case you are interested in getting some more (you can also get these from Parallax):

Photoresistor Specifications

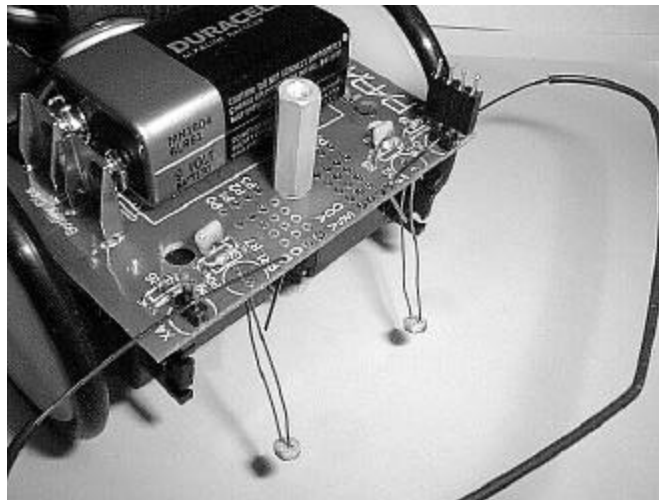
| Resistance (Ohms) | | | | | Peak Spectral Response nm | V_{MAX} | Response Time @ 1 fc (ms. typ.) | |
|-------------------|-------|------|------|------|---------------------------|-----------|---------------------------------|------------|
| 10 Lux 2850K | | | Dark | | | | Rise (1-1/e) | Fall (1/e) |
| Min | Typ. | Max. | Min. | Sec. | | | | |
| 20K | 29.0K | 38K | 1M | 10 | 550 | 100 | 35 | 5 |

The rover.bs2 program does not use the photoresistors, but edge.bs2 does. To install the photoresistors:

1. Flip the GrowBot PCB upside down. The photoresistors are installed on the bottom of the PCB, but the capacitor and the resistor are installed on the top.
2. Insert the two photoresistors into PR1 and PR2, respectively. The photoresistors should face downward as shown in the photograph on the next page.
3. Turn the GrowBot PCB right side up. Solder and trim the leads.
4. Insert the two 220 ohm resistors into R7 and R8.
5. Insert the two 0.1 μ F capacitors into positions C5 and C6. Note: you may have to bend the legs a bit; if so, use care to not bend the legs where they enter the body of the capacitor.
6. Turn the GrowBot PCB over, solder and trim the leads.



Installed Bumper and Photoresistors



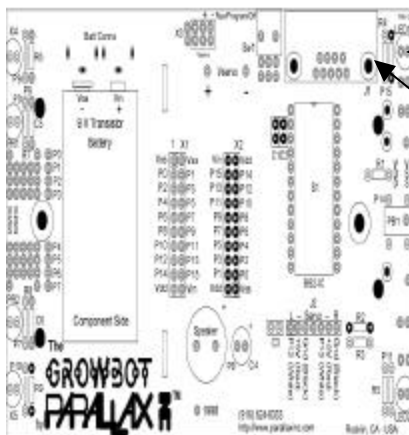
LEDs

The LEDs are installed in two positions: LED1 and LED2. GrowBot uses Liteon LTL-4203 high intensity wide viewing angle LEDs. Any other color or size LED could be substituted provided an appropriately sized current limit resistor is used.

To install the red LEDs:

1. Insert the two 220 ohm (red red brown) resistors into positions R4 and R5.
2. Insert the two red LEDs with the flat side (cathode) facing the edge of the board, fitting within the PCB's footprint outline.
3. Flip the GrowBot board upside down, solder and trim the leads.

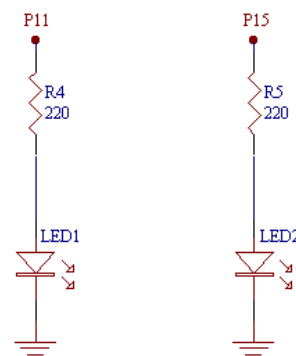
Picture



red LEDs into
'LED1' and 'LED2'

220 ohm resistors
into R4 and R5

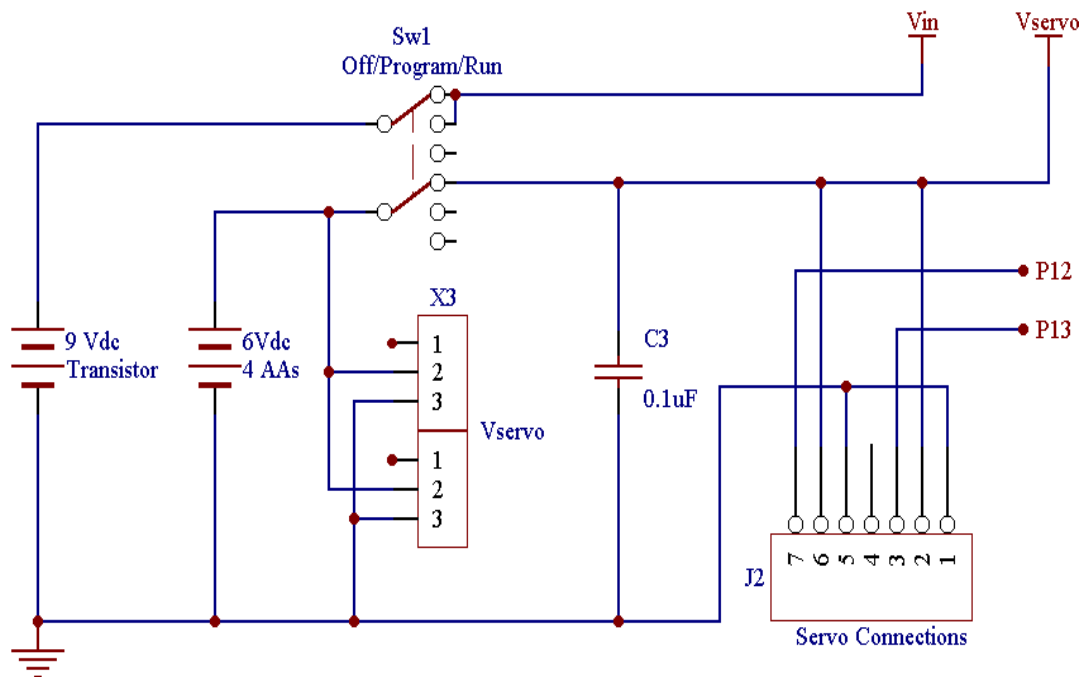
Schematic



Servo and Switch Circuitry

1. Insert two (2) 0.1 μ F capacitors into positions C1 and C2. These are for the serial port.
2. Insert one (1) 0.1 μ F capacitors into position C3. This capacitor filters the servo power leads.
3. Insert the switch into position SW1.
4. Turn the GrowBot PCB over, solder these components and trim the leads.

Schematic



Remaining Hardware

The remaining parts may be soldered onto the PCB once they are inserted in the proper locations.

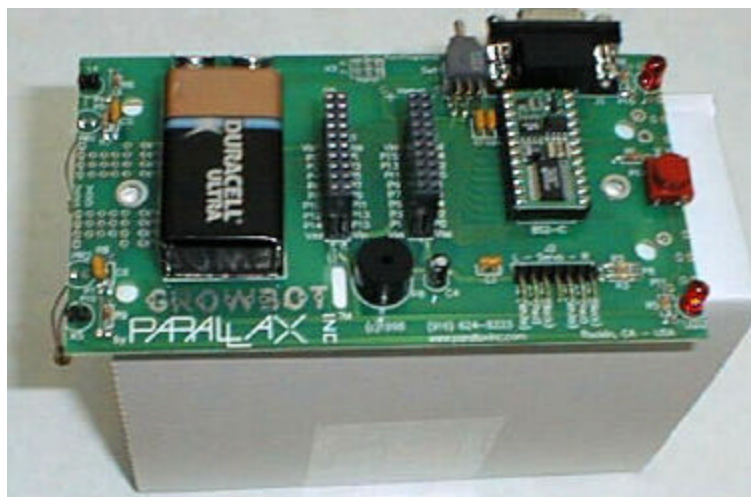
1. Insert the DB-9 connector into position J1.
2. Insert the 9-volt battery clips observing the proper orientation.
3. Plug the 24-pin socket into position S1. The semi-circle goes towards the top of the PCB. This is the BASIC Stamp 2's position indicator as well.
4. Insert the two (2) tall headers into positions X1 and X2. These are for future AppMods in case you want to expand the GrowBot's functionality.
5. Insert one (1) 7-pin right angle SIP header into position J2.
6. Turn the board over. Solder these components and trim the leads.
7. Remove the wire bumper and wash the pcb with warm water if you used water soluble rosin core solder.
8. Once the PCB is dry, snap the red pushbutton cover onto the pushbutton.
9. Replace the wire bumper.

A BS2-IC module was included in this kit. The module has been pre-programmed with a diagnostic program (growtest.bs2) which will show some that some features of the GrowBot's printed circuit board are working at this stage of assembly.

1. Insert the BS2-IC module into socket S1 with Pin 1 by the switch. The biggest chip on the BS2-IC, the PIC 16C57, is closest to the servo connections (if you are using a BS2sx-IC, then the SX28AC/SS is nearest the servo connections).
2. Attach a 9-volt battery to the clips and put the switch (SW1) in center position. After a short duration, the speaker will play the DTMF tones that approximate "Mary Had a Little Lamb", then the LEDs should be blinking. Push and hold the pushbutton. The speaker will click a few times, then the LEDs will extinguish. From then on, the LEDs will reflect what is sensed by the bumper. Try it!

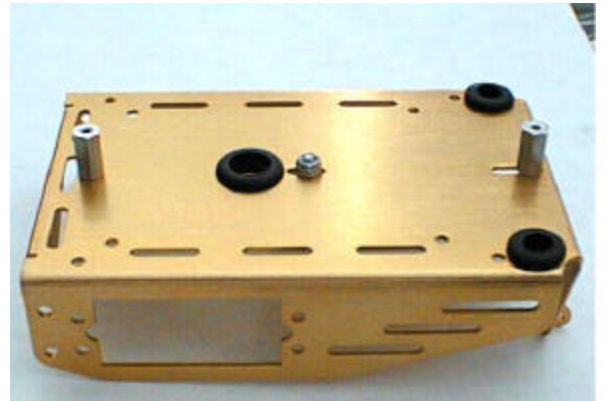
| Still doesn't work? Try these GrowBot PCB test fixes. . . |
|--|
| Are the battery clips installed with the correct polarity? The negative connection should be closest to the edge of the board, and the positive connection is closer to the BS2-IC socket. |
| Is the battery voltage at least 8-volts? Check voltage using a digital multi-meter. |
| Put the BS2-IC in backwards? Verify that the BS2-IC's Pin 1 is by the switch. |
| LED polarity must be installed with flat side near the edge of the board. |
| Speaker and capacitor C4 are installed with correct polarity? |
| Try reloading the program GROWTEST.BS2. |

Your GrowBot thus far...

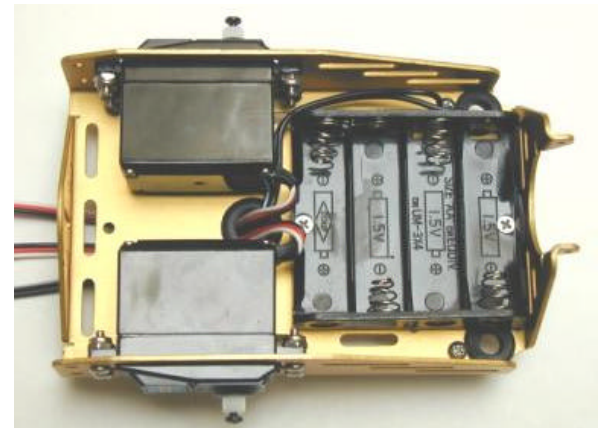


Chapter 6: GrowBot Chassis Assembly

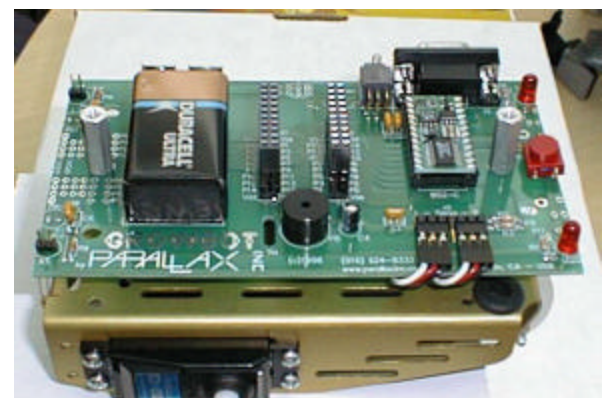
1. Insert the 13/32" rubber grommet into the hole in the center of the chassis, and insert the two 9/32" rubber grommets into the two corner holes as shown.
2. Use the two 1/4" 4-40 flat head screws to attach the battery holder to the underside of the GrowBot chassis. Note that the screw adjacent to the center hole will be secured with a lock nut while the flat head screw at the distal end will be secured with the female end of one of the 1/2inch stand-off. Do not tighten the screws yet as they may need to move a bit when we assemble the pcb to the chassis later.



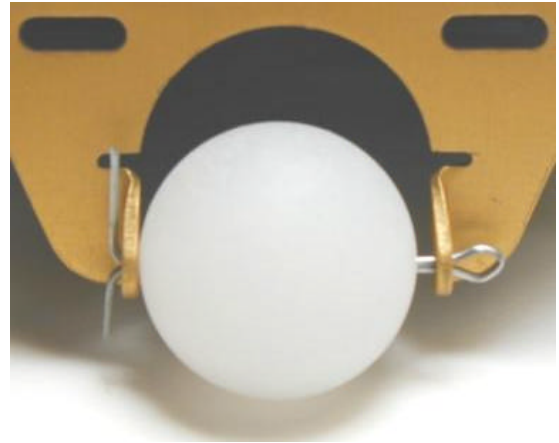
3. Using eight 1/4inch 4-40 machine screws and lock nuts, fasten both servos to the chassis as depicted.
4. Feed the battery holder power connector through the center hole to the topside of the chassis followed by the servo wires.



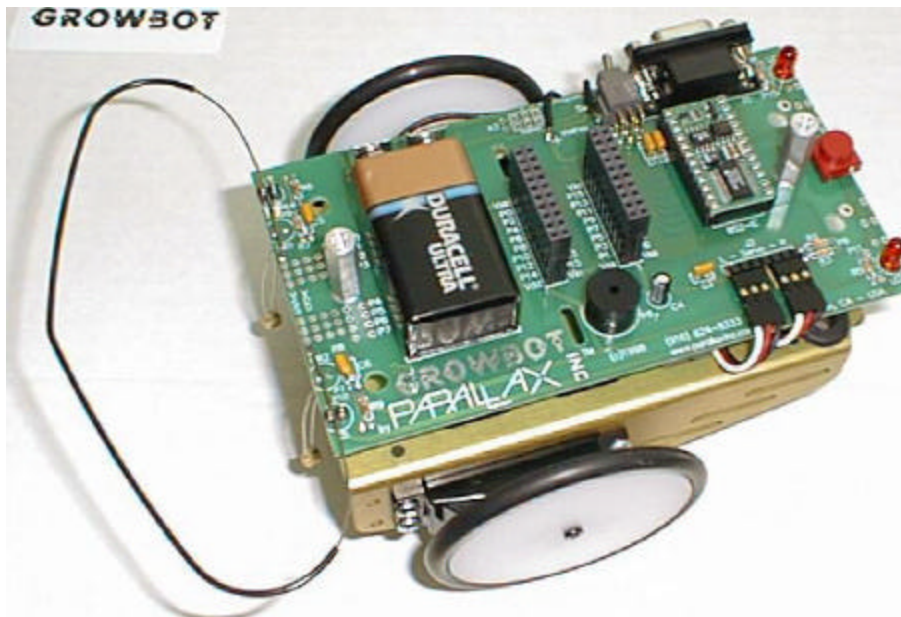
5. Strip and tin the wires from the servo battery pack, then solder them to the "Vservo" pads on the PCB observing the proper polarity (the white striped wire should be the positive lead).
6. Position the battery holder such that the 1/2inch stand-offs line up with the mounting holes on the GrowBot pcb and secure the pcb to the chassis with the 3/4inch male-female stand-offs.
7. Connect the two servo wires to J2 observing both the proper orientation of each plug as well as the ordering (left-right).



1. The plastic ball is used as the GrowBot's rear or tail wheel, and the cotter pin is its axle. Run the cotter pin through the holes in the rear of the chassis so that it holds the one-inch plastic ball in place as shown.
2. Flare the end of the cotter pin to prevent it from working its way out.



1. Push the drive wheels (broached side) onto the servo shafts (the spline) and then secure them with the screws provided with the servos.
2. Stretch the rubber tires onto the drive wheels.
3. Install four AA's into the servo battery pack under the GrowBot.
4. Now you're ready to program your GrowBot.



Chapter 8: Program the GrowBot

Run the Editor

BASIC Stamp programs are created and run from a text editor program. The program you need to run is called stampw.exe. From within stampw.exe, you may open any .bs2 file and run it on your GrowBot. A DOS version of the software, stamp2.exe, is also available on the GrowBot diskette. Follow these steps to activate your GrowBot:

1. Using Windows Explorer create a directory on your hard drive named GrowBot.
2. Copy the entire contents of the GrowBot disk into the GrowBot directory.
3. Double-click on the BASIC Stamp II icon (small BS2-IC) to launch the editor program.
4. Under the FILE menu, OPEN rover.bs2. This is a simple roving program.

The screen should look like this:

The screenshot shows a window titled 'BASIC Stamp - P:\Technical Support\Manuals\GrowBot\Rover.bs2'. The window contains a text editor with the following code:

```

'*****
'*  GrowBot Rover Program          BS2-IC Version  *
'*  November 16, 1998              *
'*                                *
'*  This program puts the GrowBot into 'roving' mode. *
'*  GrowBot will move forward, hitting an obstacle *
'*  and moving into reverse to avoid. The red LED *
'*  lights will be flashing. Photoresistors are not *
'*  enabled in this program.      *
'*****

                                ' DEFINE GROWBOT I/Os
Leye      con    0              ' left photoresistor
Reye      con    2              ' right photoresistor
eyepower  con    4              ' +5V for rctime
spkr      con    8              ' sound generator
Lbumper   var    in9            ' left bumper
Rbumper   var    in10           ' right bumper
Rled      con    11             ' right LED
rightWheel con    12            ' right wheel (servo)
leftWheel  con    13            ' left wheel (servo)
pbutton   var    in14           ' pushbutton
Lled      con    15             ' left LED

                                ' DEFINE RAM USED
temp      var    byte           ' temporary byte
speed     var    word           ' holds desired speed of wheels
steer     var    word           ' holds control word for steering
xsteer    var    word           ' forward steer command modified by photocell
bumpcount var    byte           ' bumper hit accumulator
mcount    var    nib            ' unstick maneuver counter
counter   var    byte           ' @@@

24: 31

```

Load the Program into the GrowBot

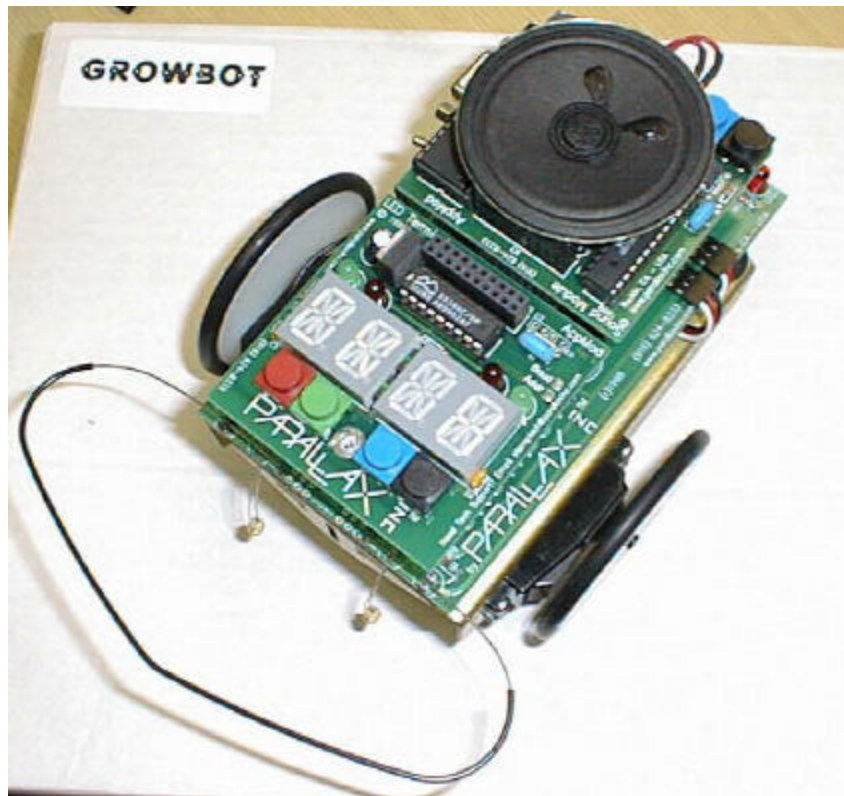
1. Connect the serial cable to an available serial port on your PC.
2. Connect the other end of the serial cable to your GrowBot's DB-9 connector.
3. Put the GrowBot's power switch in the middle position. This is the "on" position without servos (servos are enabled by switching to the run position).
4. Under the RUN menu, select RUN.

The GrowBot is now programmed to operate in it's roving mode! Push the bumper to see the red LEDs and hear the speaker. Disconnect the serial cable, push the switch to the left position and the GrowBot will be autonomous.

GrowBot programs include a simple rover program using the bumper (rover.bs2); a light / dark sensing edge avoidance program (edge.bs2); and a player piano program adapted from the BASIC Stamp Activity Board ((rbopiano.bs2).

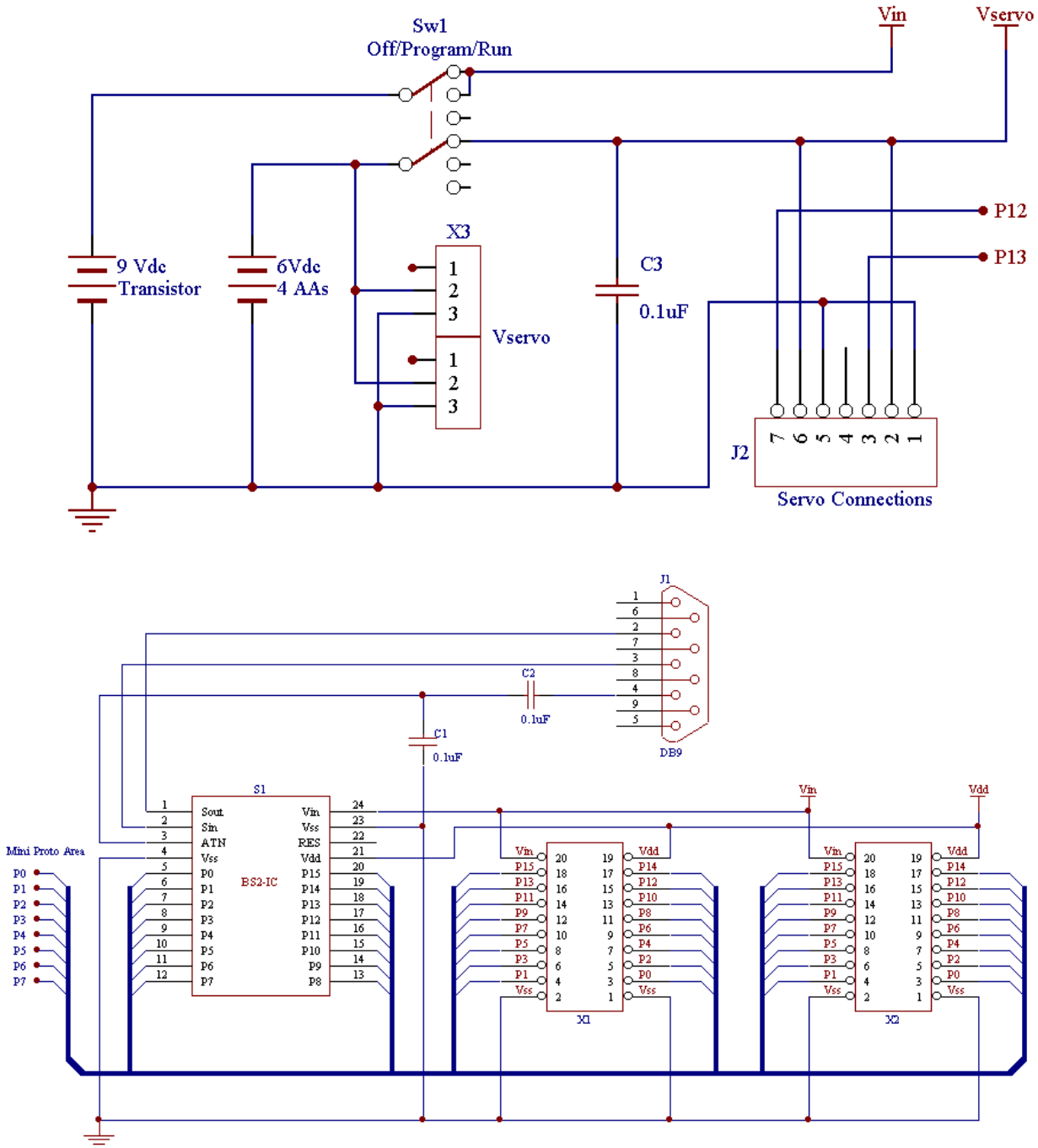
Expand your GrowBot with AppMods

(Alphanumeric LED Terminal and Sound Module AppMods shown)

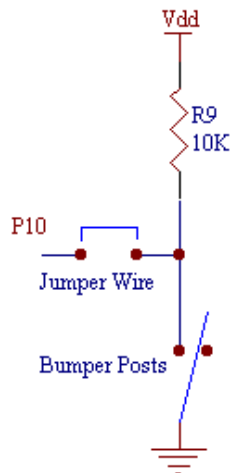
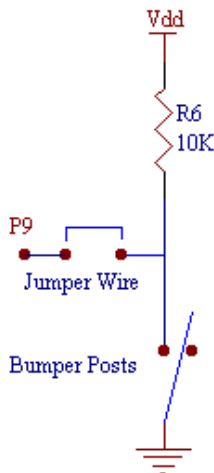
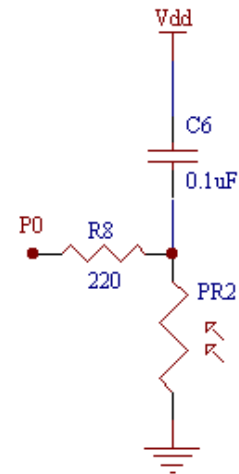
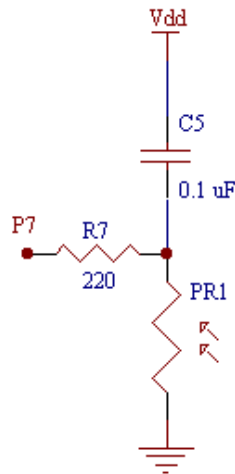
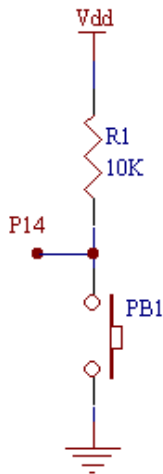
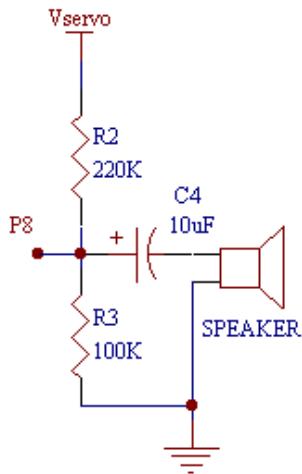


Appendix A: GrowBot Schematic

Each BASIC Stamp 2 I/O pin is accessible by the AppMod headers and at solder points on the front of the GrowBot. When using I/Os, watch for conflict with existing on-board devices. These schematics are for the Rev. C GrowBot, December 18, 1998.



PARALLAX 7



GROWBOT BS2 I/O PIN ASSIGNMENTS

| I/O Pin | Use |
|---------|---------------------------|
| 0 | Right photoresistor |
| 7 | Left photoresistor |
| 8 | Speaker / battery monitor |
| 9 | Right bumper |
| 10 | Left bumper |
| 11 | Left LED |
| 12 | Left wheel |
| 13 | Right wheel |
| 14 | Pushbutton |
| 15 | Right LED |

Appendix B: GrowBot Programming Tips and PBASIC Quick Reference Guide

The Parallax BASIC Stamp Manual Version 1.9 consists of approximately 450 pages of PBASIC command descriptions, application notes, and schematics. This manual is not included in this GrowBot kit, but it's available for download from <http://www.parallaxinc.com> in Adobe's PDF format.

Although several programs are already provided for the GrowBot, don't be afraid to venture out and write your own. Here's a brief version of the manual that covers most of the commands that you will need to use while exploring your GrowBot world.

| |
|--|
| BRANCHING |
| IF...THEN IF <i>condition</i> THEN <i>addressLabel</i> Evaluate condition and, if true, go to the point in the program marked by <i>addressLabel</i> . <ul style="list-style-type: none"> • <i>Condition</i> is a statement, such as "x=7" that can be evaluated as true or false. • <i>AddressLabel</i> is a label that specifies where to go in the event that the condition is true. |
| BRANCH BRANCH <i>offset</i> , [<i>address0</i> , <i>address1</i> , ... <i>address N</i>] Go to the address specified by <i>offset</i> (if in range). <ul style="list-style-type: none"> • <i>Offset</i> is a variable / constant that specifies which of the listed address to go to (0-N). • <i>Addresses</i> are labels that specify where to go. |
| GOTO GOTO <i>addressLabel</i> Go to the point in the program specified by <i>addressLabel</i> . <ul style="list-style-type: none"> • <i>AddressLabel</i> is a label that specifies where to go. |
| GOSUB GOSUB <i>addressLabel</i> Store the address of the next instruction after GOSUB, then go to the point in the program specified by <i>addressLabel</i> . <ul style="list-style-type: none"> • <i>AddressLabel</i> is a label that specifies where to go. |
| RETURN Return from subroutine – sends the program back to the address (instruction) immediately following the most recent GOSUB. |

LOOPING**FOR...NEXT**

FOR *variable* = *start to end* {*stepVal*} ...NEXT

Create a repeating loop that executes the program lines between FOR and NEXT, incrementing or decrementing *variable* according to *stepVal* until the value of the variable passes the *end* value.

- *Variable* is a bit, nib, byte, or word variable used as a counter.
- *Start* is a variable or constant that specifies the initial value of the variable.
- *End* is a variable or constant that specifies the end value of the variable. When the value of the variable passes *end*, the FOR . . . NEXT loop stops executing and the program goes on to the instruction after NEXT.
- *StepVal* is an optional variable or constant by which the variable increases or decreases with each step through the FOR / NEXT loop. If *start* is larger than *end*, PBASIC2 understands *stepVal* to be negative, even though no minus sign is used.

NUMERICS**LOOKUP**

LOOKUP *index*, [*value0*, *value1*,... *valueN*], *variable*

Look up the value specified by the index and store it in a variable. If the index exceeds the highest index value of the items in the list, *variable* is unaffected. A maximum of 256 values can be included in the list.

- *index* is a constant, expression or a bit, nibble, byte or word variable.
- *value0*, *value1*, etc. are constants, expressions or bit, nibble, byte or word variables.
- *variable* is a bit, nibble, byte or word variable.

LOOKDOWN

LOOKDOWN *value*, {*??*,} [*value0*, *value1*,... *valueN*], *variable*

- *value* is a constant, expression or a bit, nibble, byte or word variable.
- *??* is =, <>, >, <, <=, =>. (= is the default).
- *value0*, *value1*, etc. are constants, expressions or bit, nibble, byte or word variables.
- *variable* is a bit, nibble, byte or word variable.

RANDOM

RANDOM *variable*

Generate a pseudo-random number.

- *VARIABLE* is a byte or word variable in the range 0..65535.

DIGITAL I/O**INPUT**

INPUT *pin*

Make the specified pin an input (write a 0 to the corresponding bit of DIRS).

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

OUTPUT

OUTPUT *pin*

Make the specified pin an output (write a 1 to the corresponding bit of DIRS).

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

REVERSEREVERSE *pin*If *pin* is an output, make it an input. If *pin* is an input, make it an output.

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

LOWLOW *pin*Make *pin* output low (write 1 to the corresponding bit of DIRS and 0 to the corresponding bit of OUTS).

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

HIGHHIGH *pin*Make the specified *pin* output high (write 1s to the corresponding bits of both DIRS and OUTS).

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

TOGGLETOGGLE *pin*Invert the state of a *pin*.

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

PULSINPULSIN *pin, state, variable*Measure an input pulse (resolution of 2 μ s).

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *state* is a constant, expression or a bit, nibble, byte or word variable in the range 0..1.
- *variable* is a bit, nibble, byte or word variable.
- Measurements are in 2uS intervals and the instruction will time out in 0.13107 seconds.

PULSOUTPULSOUT *pin, period*Output a timed pulse by inverting a *pin* for some time (resolution of 2 μ s).

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *period* is a constant, expression or a bit, nibble, byte or word
- *variable* in the range 0..65535 representing the pulse width in 2uS units.

BUTTONBUTTON *pin, downstate, delay, rate, workspace, targetstate, label*

Debounce button, perform auto-repeat, and branch to address if button is in target state.

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *downstate* is a constant, expression or a bit, nibble, byte or word variable in the range 0..1.
- *delay* is a constant, expression or a bit, nibble, byte or word variable in the range 0..255.
- *rate* is a constant, expression or a bit, nibble, byte or word variable in the range 0..255.
- *workspace* is a byte or word variable.
- *targetstate* is a constant, expression or a bit, nibble, byte or word variable in the range 0..1.
- *label* is a valid label to jump to in the event of a button press.

SHIFTIN

SHIFTIN *dpin, cpin, mode, [result{bits} { ,result{bits}...]*

Shift bits in from parallel-to-serial shift register.

- *dpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the data pin.
- *cpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the clock pin.
- *mode* is a constant, symbol, expression or a bit, nibble, byte or word variable in the range 0..4 specifying the bit order and clock mode. 0 or MSBPRES = msb first, pre-clock, 1 or LSBPRE = lsb first, pre-clock, 2 or MSBPOST = msb first, post-clock, 3 or LSBPOST = lsb first, post-clock.
- *result* is a bit, nibble, byte or word variable where the received data is stored.
- *bits* is a constant, expression or a bit, nibble, byte or word variable in the range 1..16 specifying the number of bits to receive in *result*. The default is 8.

SHIFTOUT

SHIFTOUT *dpin, cpin, mode, [data{bits} { ,data{bits}...]*

Shift bits out to serial-to-parallel shift register.

- *dpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the data pin.
- *cpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the clock pin.
- *mode* is a constant, symbol, expression or a bit, nibble, byte or word variable in the range 0..1 specifying the bit order. 0 or LSBFIRST = lsb first, 1 or MSBFIRST = msb first.
- *data* is a constant, expression or a bit, nibble, byte or word variable containing the data to send out.
- *bits* is a constant, expression or a bit, nibble, byte or word variable in the range 1..16 specifying the number of bits of *data* to send. The default is 8.

COUNT

COUNT *pin, period, result*

Count cycles on a pin for a given amount of time (0 - 125 kHz, assuming a 50/50 duty cycle).

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *period* is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535.
- *result* is a bit, nibble, byte or word variable.

ANALOG I/O**PWM**

PWM *pin, duty, cycles*

Output PWM, then return pin to input. This can be used to output analog voltages (0-5V) using a capacitor and resistor.

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *duty* is a constant, expression or a bit, nibble, byte or word variable in the range 0..255.
- *cycles* is a constant, expression or a bit, nibble, byte or word variable in the range 0..255 representing the number of 1ms cycles to output.

RCTIME

RCTIME *pin, state, variable*

Measure an RC charge/discharge time. Can be used to measure potentiometers.

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *state* is a constant, expression or a bit, nibble, byte or word variable in the range 0..1.
- *variable* is a bit, nibble, byte or word variable.

SOUND**FREQOUT**

FREQOUT *pin, milliseconds, freq1 {,freq2}*

Generate one or two sine waves of specified frequencies (each from 0 - 32767 hz.).

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range of 0..15.
- *milliseconds* is a constant, expression or a bit, nibble, byte or word variable.
- *freq1* and *freq2* are constant, expression or bit, nibble, byte or word variables in the range 0..32767 representing the corresponding frequencies.

XOUT

XOUT *mpin, zpin, [house\keyorcommand{cycles} {, house\keyorcommand{cycles}... }]*

Generate X-10 Powerline control codes. For use with TW523 or TW513 Powerline interface module.

- *mpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the modulation pin.
- *zpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the zero-crossing pin.
- *house* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the house code A..P respectively.
- *keycommand* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying keys 1..16 respectively or is one of the commands in the following table:
- See the Parallax manual for a list of X-10 commands for lights on, off, dim and bright.
- *cycles* is a constant, expression or a bit, nibble, byte or word variable in the range 2..65535 specifying the number of cycles to send. (Default is 2).

SERIAL I/O**SERIN**

SERIN *rpin*{*fpin*}, *baudmode*, {*plabel*,} {*timeout*, *tlabel*,} [*inputdata*]

Serial input with optional qualifiers, time-out, and flow control. If qualifiers are given, then the instruction will wait until they are received before filling variables or continuing to the next instruction. If a time-out value is given, then the instruction will abort after receiving nothing for a given amount of time. Baud rates of 300 - 50,000 are possible (0 - 19,200 with flow control). Data received must be N81 (no parity, 8 data bits, 1 stop bit) or E71 (even parity, 7 data bits, 1 stop bit).

- *rpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..16.
- *fpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *baudmode* is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535.
- *plabel* is a label to jump to in case of a parity error.
- *timeout* is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535 representing the number of milliseconds to wait for an incoming message.
- *tlabel* is a label to jump to in case of a timeout.
- *inputdata* is a set of constants, expressions and variable names separated by commas and optionally proceeded by the formatters available in the DEBUG command, except the ASC and REP formatters. Additionally, the following formatters are available:
 1. STR bytearrayL{E} input a string into byte array of length L with optional end-character of E.
 2. SKIP L input and ignore L bytes.
 3. WAITSTR bytearray{L} Wait for bytearray string of L length, or terminated by 0 (6 byte maximum).
 4. WAIT (value {,value...}) Wait for up to a six-byte sequence.

SEROUT

SEROUT *tpin*{*fpin*}, *baudmode*, {*pace*,} {*timeout*, *tlabel*,} [*outputdata*]

Send data serially with optional byte pacing and flow control. If a pace value is given, then the instruction will insert a specified delay between each byte sent (pacing is not available with flow control). Baud rates of 300 - 50,000 are possible (0 - 19,200 with flow control). Data is sent as N81 (no parity, 8 data bits, 1 stop bit) or E71 (even parity, 7 data bits, 1 stop bit).

- *tpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..16.
- *fpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *baudmode* is a constant, expression or a bit, nibble, byte or word variable in the range 0..60657.
- *pace* is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535 specifying a time (in milliseconds) to delay between transmitted bytes. This value can only be specified if the *fpin* is not specified.
- *timeout* is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535 representing the number of milliseconds to wait for the signal to transmit the message. This value can only be specified if the *fpin* is specified.
- *tlabel* is a label to jump to in case of a timeout. This can only be specified if the *fpin* is specified.
- *outputdata* is a set of constants, expressions and variable names separated by commas and optionally proceeded by the formatters available in the DEBUG command.

DTMFOUT

DTMFOUT *pin, {ontime, offtime,}{key{key...}}*

Generate DTMF telephone tones.

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *ontime* and *offtime* are constants, expressions or bit, nibble, byte or word variables in the range 0..65535.
- *key* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

EEPROM ACCESS**DATA**

DATA {pointer} DATA {@location,} {WORD} {data}{(size)} {, { WORD} {data}{(size)}...}

Store data in EEPROM before downloading PBASIC program.

- *pointer* is an optional undefined constant name or a bit, nibble, byte or word variable which is assigned the value of the first memory location in which data is written.
- *location* is an optional constant, expression or a bit, nibble, byte or word variable which designates the first memory location in which data is to be written.
- *word* is an optional switch which causes DATA to be stored as two separate bytes in memory.
- *data* is an optional constant or expression to be written to memory.
- *size* is an optional constant or expression which designates the number of bytes of defined or undefined data to write/reserve in memory. If DATA is not specified then undefined data space is reserved and if DATA is specified then SIZE bytes of data equal to DATA are written to memory.

READ

READ *location, variable*

Read EEPROM byte into variable.

- *location* is a constant, expression or a bit, nibble, byte or word variable in the range 0..2047.
- *variable* is a bit, nibble, byte or word variable.

WRITE

WRITE *location, data*

Write byte into EEPROM.

- *location* is a constant, expression or a bit, nibble, byte or word variable in the range 0..2047.
- *data* is a constant, expression or a bit, nibble, byte or word variable.

TIME**PAUSE**

PAUSE *milliseconds*

Pause execution for 0–65535 milliseconds.

- *milliseconds* is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535.

POWER CONTROL**NAP**

NAP *period*

Nap for a short period. Power consumption is reduced.

- *period* is a constant, expression or a bit, nibble, byte or word variable in the range 0..7 representing 18ms intervals.
- Current is reduced to 50uA (assuming no loads).

SLEEP

SLEEP *seconds*

Sleep for 1-65535 seconds. Power consumption is reduced to approximately 50 μ A.

- *seconds* is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535 specifying the number of seconds to sleep.

END

END

Sleep until the power cycles or the PC connects. Power consumption is reduced to approximately 50 μ A.

- 50uA reduced current (no loads).

PROGRAM DEBUGGING**DEBUG**

DEBUG *outputdata{,outputdata...}*

Send variables to PC for viewing.

- *outputdata* is a text string, constant or a bit, nibble, byte or word variable. If no formatters are specified DEBUG defaults to ASCII character display without spaces or carriage returns following the value.