

**Parallax, Inc.
Basic Stamps**

**US First Seminar
Plymouth, MA**

Dear Attendee,

Thank you for coming to the BASIC Stamp Seminar. This seminar will cover stamp basics, including basic programming, and practical electronics as it pertains to stamp applications. The seminar culminates with each student or student pair wiring and programming an autonomous robot - The BoeBot. If time permits, we will delve into code optimizing techniques and write a script based command parser using more advanced commands and programming methods.

We prefer a relaxed environment so feel free to ask questions at any time. We will take several small breaks during the day with an hour-long break for lunch at the usual time. Of course, attendees are welcome to remain in the lab to complete their projects and/or experiment beyond the scope of the course. The instructors will remain available to help out in any way.

The documentation style of this pamphlet was chosen because of its flexibility. We recognize that people learn different concepts at different rates. Those who wish to work ahead can, and those who wish to spend more time on a particular part may do so, then catch up later. Naturally, we recommend that you try to stay in step with the rest of the class.

Quick Start Guide

- 1) If there is not an icon on your desktop for the stamp, insert the stamp floppy diskette into the A drive. Access the A drive, click on and drag the Stampw.exe icon over to the desktop. You may close the A drive window now.
- 2) Start the stamp software by double clicking on the Stampw.exe icon.
- 3) Verify that the BS2 is correctly inserted into the Board of Education (BOE). If not, do so by plugging the BS2 into the 24-pin DIP socket with the largest chip on the BS2 closest to the Parallax, Inc. logo on the BOE.
- 4) Apply power to the BOE; the on-board LED should glow green. Disconnect the power if the LED glows any other color, or not at all, and alert a trainer for assistance.
- 5) Connect the serial cable between the PC's serial port and "J2" on the BOE.
- 6) Using the mouse, click on "Run" and drag down to "Identify", then release the mouse button. The stamp software will attempt to find and communicate with the BS2. If this is successful, a message box will announce that the BS2-IC was found. If not successful, alert one of the trainers for assistance.

You're ready to begin!

Lesson #1 - Notes

If you are familiar with the Basic language, you will find many similarities between it and the PBASIC language used by the BS2. Labels must be located in column 0 (the most left position in the editor). Commands may be located anywhere, but to enhance the readability of the program, they are generally tabbed in. Comments may be added to your program to enhance readability, and need only be preceded by a single quotation mark, ('), so the compiler knows to ignore it while generating the output file. The programs written in these lessons are formatted to show what is considered by most programmers to be good form.

Within the BS2, program execution starts at the top of the program. Each instruction is executed, and when completed, the next instruction is executed. This continues endlessly unless a goto or gosub instruction is encountered. The gosub instruction is explained in a later lesson, the goto instruction is covered below.

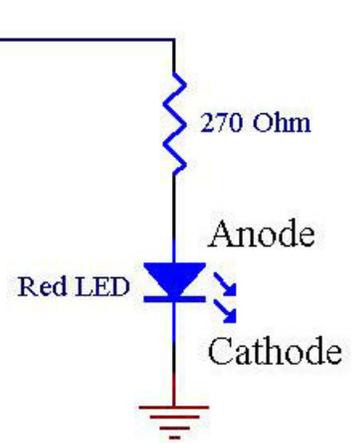
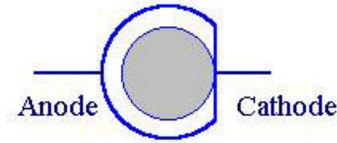
New Commands Used in This Lesson

- High *n*** The high command makes the specified pin *n* an output, and drives that output high (to 5 Vdc). Valid values for *n* are 0 through 15 and correspond to the I/O pin numbers found on the BS2, P0 - P15.
- Low *n*** The high command makes the specified pin *n* an output, and drives that output low (to 0 Vdc). Valid values for *n* are 0 through 15 and correspond to the I/O pin numbers found on the BOE, P0-P15.
- Goto *label*** The normal program execution is interrupted, and redirected to start executing the code following the specified *label*. This instruction is handy for jumping over code that you wish to skip, or when the program execution gets to the bottom of the page, to send it back up to the top of the page.
- Pause *n*** The pause command suspends program execution for the specified amount of time. Outputs are maintained during this idle time. The amount of time *n* is in milliSeconds (mSec) and may range from 1 to 65535, (1 mSec to 65.5 Seconds).

THE BASICS – Lesson #1

Construct the following circuit:

The 270 Ohm resistor is colored with Red, Violet, Brown, and Gold bands. LEDs are polarity sensitive. The top view of the LED should reveal a flat. The leg nearest the flat is the cathode and should be connected to ground (Vss).



Once the circuit is complete, type the following

```
Start:          high 2          `Turn on the LED
```

Once the code is complete type the Ctrl-R key sequence

or click on the tool button shaped like a triangle to program your BS2.

Notice the following:

The status bar flashes on the screen to indicate the status of the download.

The LED should light.

If you stare at the LED closely, you should notice that the LED flickers at a consistent rate. Specifically, it stays on for 2.3 seconds and flickers off for ~18 mS, (0.018 Seconds).

Why is this so?

Extra for Experts

For those who want to work ahead, make the LED blink at a 2 Hz rate.

Now try to *fade* the LED on and off at a 2 Hz rate.

Lesson #2 - Notes

New Terms Used in This Lesson

Remove the previous circuit. The BS2 uses several predefined names to access the I/O pins in various ways. Using different names to address the input pins allows you to read an individual pin (a bit), a group of four pins (a nibble), a group of 8 pins (a byte), or all of the input pins in one swell foop! (a word).

InX InX is the name of the group of any input pin P0 through P15 whereas X is any number 0 through 15 which correspond to that pin.

InA InA is the name of the group of input pins P0 through P3

InB InB is the name of the group of input pins P4 through P7

InC InC is the name of the group of input pins P8 through P11

InD InD is the name of the group of input pins P12 through P15

InL InL is the name of the group of input pins P0 through P7

InH InH is the name of the group of input pins P8 through P15

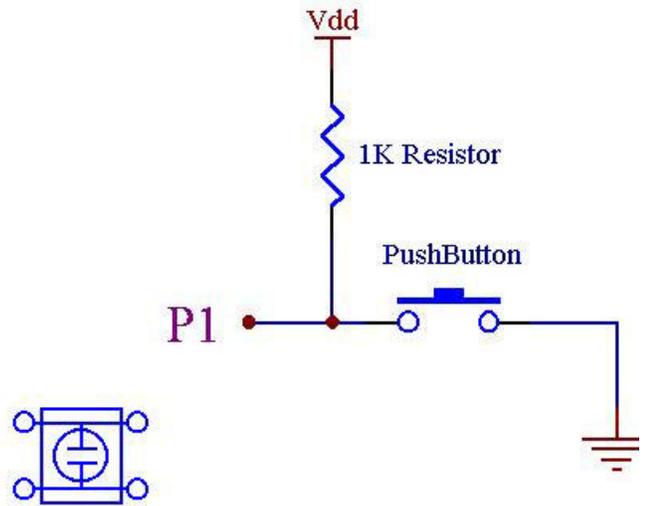
if cond then label

The if – then statement allows you to command the stamp to make a decision in real-time, that is, while the program is running. If the *cond* evaluates to non-zero or a true condition, then the program control jumps to the *label* specified and continues execution thereafter. If *cond* evaluates to a zero or a false condition, program execution is resumed with the next instruction.

THE BASICS – Lesson #2

Construct the following circuit:

The pushbutton is a small button with four legs. Care should be taken to ensure that the pushbutton is inserted the proper way into the breadboard. The switch element is across the two narrow legs of the switch.



Once the circuit is complete, type the following

```
Start: pause 200           `Wait 0.2 seconds
      Debug ?in1          `Show the status of P1 on the debug screen
      goto Start          `Repeat endlessly
```

Once the code is complete, type the Ctrl-R key sequence or click on Run|Run to program your BS2.

Notice the following:

After the program downloads, a debug window pops open to display the default status of P1 (which should be a '1'). Depressing the pushbutton results in the '0s' in the debugging window.

Why the resistor?

The resistor is necessary to assure a clear transition from 5 Vdc to 0 Vdc and back to 5 Vdc as the switch is pressed and released.

Extra for Experts

Use the 'if' statement to read the pushbutton, and display one of two messages to the debug window.

Lesson #3 - Notes

Remove the previous circuit.

New Commands Used in This Lesson

rctime *n,s,r* The RCTIME function makes the specified pin *n* an input and starts a timer. As the flux capacitor charges, the I/O pin drifts from 5 Vdc to 0 Vdc. When the voltage passes the threshold, 1.4 Vdc, the timer stops and places the time value measured in the result register *r*. The *s* parameter configures the polarity (depends on how the circuit is wired up) and should be a '1' for this test.

Debug *r* The debug command is not really a command at all but a powerful debugging tool. You may use it to echo the value of any and all RAM bytes to the debugging window. The many variations of 'debug' allow you to format the data being sent to the debugging window in a multitude of ways. We will be exploring several of those ways throughout this course.

/ * + - = The basic mathematical operators are used to make the stamp perform 16-bit integer math. Of course, many more mathematical operators are available but for now these are the only ones we'll mention here.

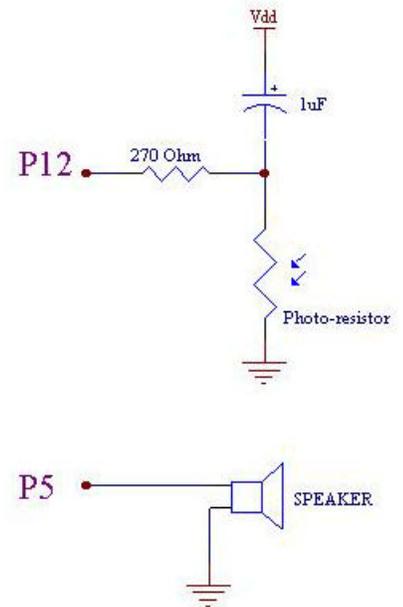
var Var is a BS2 directive, not a command, that directs the compiler to allocate some RAM memory and to assign an alias by which to reference it. RAM comes in various sizes as shown below with examples of how to declare the various types.

Awake	var	bit	'1-bit	range = 0-1
MyIQ	var	nibble	'4-bits	range = 0-15
MyWeight	var	byte	'8-bits	range = 0-255
MyAge	var	word	'16-bits	range = 0-65535

THE BASICS – Lesson #3

Construct the following circuit:

The photoresistor is the small ceramic white disk with two rather long legs on one side. The speaker is a relatively large black cylinder with two short legs on one side. The speaker is polarity sensitive; connect the positive leg (marked “+” on the top of the speaker) to P5 and the other leg to ground. The 1uF capacitor is a small, yellow ceramic capacitor that is polarity sensitive. There is a “+” sign near the leg that goes to Vdd (+5VDC). If you cannot see the “+” sign, the dark line points to the positive leg.



Once the circuit is complete, type the following

```
result var    word           'Declare a 16-bit variable called result

Start: high 12                'Discharge the capacitor
      rctime 12,1,result      'Measure the rc charge time and put that value in 'result'
      debug ?result          'Show the results on the screen
      pause 100              'Wait 0.1 seconds
      goto Start             'Repeat endlessly
```

Once the code is complete type the Ctrl-R key sequence

or click on Run|Run to program your BS2.

Notice the following:

After the program downloads, a debug window pops open and displays the result of the rctime function. Notice the value change as you move your hand over the photoresistor.

Extra for Experts

For those who want to work ahead, enter a math expression that will change the range values received to a range of frequencies suitable for the speaker, then output that value to the speaker.

Display the frequency that you are sending to the speaker in as: “Freq: xxx.x Hz” using the debug command.

Lesson #4 - Notes

Remove the previous circuit. Lesson #4 begins the second phase of this seminar – building the BoeBot. The BoeBot circuitry is more extensive than what we have had to deal with thus far, so it is essential to be considerate of where the components are located on the breadboard. So, from now on, we will suggest locations to place the components we will be adding.

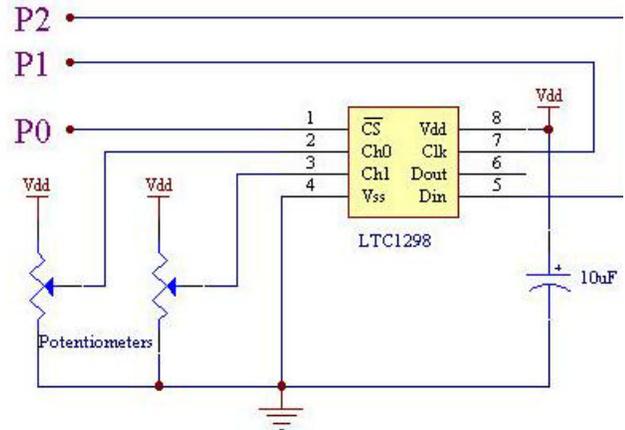
New Commands Used in This Lesson

- gosub *label*** Program control is redirected to *label* just as the goto instruction does, but the address of the instruction following the gosub is memorized also. This is done so that, when the subroutine has completed and a return instruction is executed, program control is redirected to the instruction following the gosub.
- return** The return command retrieves the last address memorized (by a gosub command) and redirects the program to the instruction immediately following the gosub that called the subroutine containing this return statement.
- dec, hex** Basic Stamps internally maintain numbers in binary, but we may view them in either decimal or hexadecimal form when we use the debug command just by placing a modifier before the variable.
- debug** You may format the data Debug displays by mixing literal strings with variable data. Anything within a set of double quotation marks will simply be printed on the screen.

THE BOE-BOT – Lesson #4

Construct the following circuit:

The LTC1298 is a two channel A/D converter. It is important to orient the chip properly and insert it into the breadboard close to the middle, straddling the trough. Please find the notch at one end of the IC. If you orient the IC such that the notch is at the top, pin one will be located immediately to the left. The pins are numbered counter-clockwise from there. Straddle each potentiometer across the trough just as the LTC1298, and locate one at the top and the other at the bottom of the trough.



Once the circuit is complete, type the following and download it to the stamp.

```

CS          con    0          ' A/D Chip Select; 0 = active
CLK         con    1          ' A/D Clock; out on rising, in on falling edge
DAT         con    2          ' A/D Data I/O pin

temp        var    word       ' Temporary register
ch1         var    word       ' A/D channel 1
ch2         var    word       ' A/D channel 2
config      var    nib        ' Configuration nibble for A/D
ch          var    config.bit2 ' Channel selection bit for A/D

Init:       high CS          ' Deactivate ADC to begin.
            high DAT         ' Set data pin for first start bit.
            config = %1011   ' Cfg A/D: Start Bit, Sgl/Dif, Ch#, msbf
Start:      gosub ReadAD     ' Get raw speed command inputs.
            debug "Ch1: ",dec ch1," Ch2: ",dec ch2,"
            goto Start      ' iteration. Not much room for other stuff!

ReadAD:     ch = 0           ' Set AD channel = 0
            gosub Convert    ' Get ch0 AD 12 bit value
            ch1 = temp
            ch = 1           ' Set AD channel = 1
            gosub Convert    ' Get ch1 AD 12 bit value
            ch2 = temp
            return

Convert:    low CS           ' Activate the ADC.
            shiftout DAT,CLK,lsbfirst,[config\4] ' Send config bits.
            shiftin  DAT,CLK,msbpost,[temp\12]  ' Get data bits.
            high CS        ' Deactivate the ADC.
            return         ' Return to program.

```

Notice the following:

If everything went well, you should be able to adjust potentiometer and see the values change in the debug window. Since this is a 12-bit A/D converter, the expected range is 0 – 4095.

Extra for Experts

For those who want to work ahead, change the code as necessary to convert the 12/bit A/D output to an 8-bit value. What is the range of output now?

Lesson #5 - Notes

Lesson #5 differs from the previous ones in that we do not remove the previous circuit but add the new circuit to the previous one.

New Commands Used in This Lesson

<<, >>

Logical Operators to shift data to the left and right. Usefull for dividing by or multiplying by a power of 2. Ex: divide ch1 by 4:

```
ch1 = ch1>>2
```

THE BOE-BOT – Lesson #5

No circuit to construct.

Now that we have programmed the stamp to report data regarding the potentiometers, its time to massage the data into something usefull.

Amend your stamp program to resemble the following, then download it.

```

CS          con      0          ' A/D Chip Select; 0 = active
CLK         con      1          ' A/D Clock; out on rising, in on falling edge
DAT         con      2          ' A/D Data I/O pin

temp        var      word       ' Temporary registers
ch1         var      word       ' A/D channel 1
ch2         var      word       ' A/D channel 2
config      var      nib        ' Configuration nibble for A/D
ch          var      config.bit2 ' Channel selection bit for A/D

Init:       high CS           ' Deactivate ADC to begin.
            high DAT          ' Set data pin for first start bit.
            config = $B       ' Cfg A/D: Start Bit, Sgl/Dif, Ch#, msbf
Start:      gosub ReadAD      ' Get raw speed command inputs.
            gosub CalcDemand  ' Calculate the speed and direction demands
            debug "Fwd/Rev: ",sdec ch1, " Left/Right: ",sdec ch2,cr
            goto Start

ReadAD:     ch = 0            ' Set AD channel = 0
            gosub Convert     ' Get ch0 AD 12 bit value
            ch1 = temp >> 4   ' Convert to 8 bit value
            ch = 1           ' Set AD channel = 1
            gosub Convert     ' Get ch1 AD 12 bit value
            ch2 = temp >> 4   ' Convert to 8 bit value
            return

Convert:    low CS            ' Activate the ADC.
            shiftout DAT,CLK,lsbfirst,[config\4] ' Send config bits.
            shiftin  DAT,CLK,msbpost,[temp\12]  ' Get data bits.
            high CS          ' Deactivate the ADC.
            return           ' Return to program.

CalcDemand: ch1 = ch1 - 127    ' Establish the center position of each
            ch2 = ch2 - 127    ' potentiometer to be = to 0
            return

```

Notice the following:

Adjusting the potentiometers now yeild signed numbers signifying speed and direction. Note: the numbers within the stamp are still unsigned binary, but it helps us understand the outputs if we have the stamp display those numbers as signed decimal numbers.

Extra for Experts

For those who want to work ahead, add a subroutine to merge the outputs and compensate for an offset to produce two complementary differential outputs that range from 623 to 879. (750+-128).

Lesson #6 - Notes

Servos are DC motors that have been geared down for high torque, and have a feedback system such that they can position their servo horn (on the shaft) based on a proportional signal.

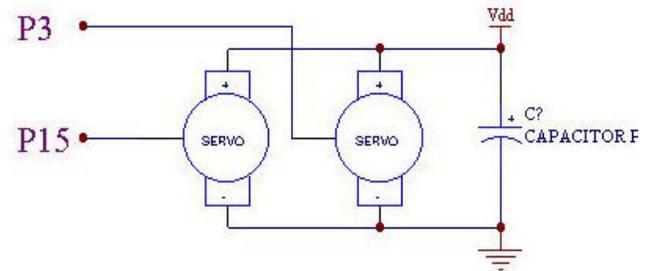
New Commands Used in This Lesson

pulsout *n,d* Pulsout generates a pulse on pin *n* for the duration *d*. Pin number should be between 0 and 15. Duration is specified in 2uS increments. The range of the time parameter is 1 – 65535, which corresponds to 2uS to 131 mSec.

THE BOE-BOT – Lesson #6

Add the following to your circuit:

The servos that provide the motion for the BoeBot are already installed on the BoeBot chassis. There are three wires used connect each servo to the BoeBot board. The Black wire should be connect to Vss (ground), the Red wire should be connected to Vdd (+5Vdc), and the white wire should be connected to the stamp I/O pin specified in the schematic. The capacitor is large and has a strip near the leg that goes to Vss.



Amend your stamp program to resemble the following, then download it.

```

CS          con      0          ' A/D Chip Select; 0 = active
CLK         con      1          ' A/D Clock; out on rising, in on falling edge
DAT         con      2          ' A/D Data I/O pin
Mid         con      750

temp        var      word       ' Temporary registers
ch1         var      word       ' A/D channel 1
ch2         var      word       ' A/D channel 2
config      var      nib       ' Configuration nibble for A/D
ch          var      config.bit2 ' Channel selection bit for A/D

Init:       high CS           ' Deactivate ADC to begin.
            high DAT          ' Set data pin for first start bit.
            config = $B       ' Cfg A/D: Start Bit, Sgl/Dif, Ch#, msbf

Start:      gosub ReadAD      ' Get raw speed command inputs.
            gosub CalcDemand  ' Calculate the speed and direction demands
            gosub CalcDirect  ' Correlate the demands into a vector
            debug "Left Servo: ",dec Rspeed," Right Servo: ",dec Lspeed,cr
            goto Start

ReadAD:     ch = 0            ' Set AD channel = 0
            gosub Convert     ' Get ch0 AD 12 bit value
            ch1 = temp >> 4  ' Convert to 8 bit value
            ch = 1           ' Set AD channel = 1
            gosub Convert     ' Get ch1 AD 12 bit value
            ch2 = temp >> 4  ' Convert to 8 bit value
            return

Convert:    low CS           ' Activate the ADC.
            shiftout DAT,CLK,lsbfirst,[config\4] ' Send config bits.
            shiftin DAT,CLK,msbpost,[temp\12] ' Get data bits.
            high CS          ' Deactivate the ADC.
            return          ' Return to program.

CalcDemand: ch1 = ch1 - 127   ' Establish the center position of each
            ch2 = ch2 - 127   ' potentiometer to be = to 0
            return

CalcDirect: Rspeed = Mid+ch1-ch2 ' Figure the direction command into
            Lspeed = Mid-ch1-ch2 ' the speed command for each servo.
            return

```

Notice the following:

When the potentiometers are in their respective center positions, the Rspeed and Lspeed values debugged should be 750.

Lesson #7 - Notes

Now that we have the rough form of the BoeBot done, its time to polish it a little. You have probably noticed that it is difficult to dial in the pots to get the servo motors to stop. A deadband can make this easier, and thus more practical.

New Commands Used in This Lesson

- <, > Logical Operators to compare the magnitude of two numbers .
- () Parenthesis can change the order of operation in an expression. Typically, stamp math expressions are evaluated from left to right, regardless of hierarchical operations.

THE BOE-BOT – Lesson #7

No circuit to construct.

In practical applications, it is often necessary to add in deadbands and offsets. We've already handled the offset for the servos, now we will add the deadband logic.

Add the following code to your current program, then download it.

In the declarations area add:

```
Deadband      con      10          ' Midpoint + - deadband = no movement
```

In the main code, after the calcdemand subroutine but before the servodrive subroutine add:

```
      gosub CheckDband      ' Do deadband logic
```

Add the following subroutine to the end of your code:

```
CheckDband:   if Rspeed > (Mid+Deadband) then RdbOK ' If inside deadband range
              if Rspeed < (Mid-Deadband) then RdbOK ' limit the value to 'Null'
              Rspeed = Mid
RdbOK:       if Lspeed > (Mid+Deadband) then LdbOK ' If inside deadband range
              if Lspeed < (Mid-Deadband) then LdbOK ' limit the value to 'Null'
              Lspeed = Mid
LdbOK:       return
```

Notice the following:

Adjusting the potentiometers now yields a more generous mid point in which the servos are still. Ideally, you should adjust the 'Deadband' constant to suit your liking.

Extra for Experts

This code is written such that it is easy to read, but is it efficient? Click on Run|Memory Map and note the usage. Try to combine subroutines and optimize the code, then note the difference in memory usage. How did you do?

THE BOE-BOT – Lesson #8

No circuit to construct.

Fully optimized? We may never know. There are many ways to skin this cat, and there is always someone more clever than yourself ready to prove it! Here's what we came up with.

```

-----
'Boe-Bot Joystick Simulator          12mS          js9.bs2
-----
'Define Variables and Constants      5%         104 bytes      3.25 words
-----
CS          con      0          ' A/D Chip Select; 0 = active
CLK         con      1          ' A/D Clock; out on rising, in on falling edge
DAT         con      2          ' A/D Data I/O pin
Lservo     con      15         ' Left servo on P15
Rservo     con      3          ' Right servo on P3
Mid         con      755       ' Null midpoint of servos
Deadband   con      10         ' Midpoint + - deadband = no movement
Range      con      Deadband*2 ' Range = twice the deadband
LowPoint   con      Mid-Deadband ' LowPoint = Midpoint - deadband

Rspeed     var      word       ' Reg for Rservo & Ch1
Lspeed     var      word       ' Reg for Lservo & Ch2
ch1        var      byte       ' A/D channel 1
ch2        var      byte       ' A/D channel 2
config     var      nib        ' Configuration nibble for A/D
-----
' Main Code
-----
Init:      high CS          ' Deactivate ADC to begin.
           high DAT        ' Set data pin for first start bit.
ReadAD:    gosub Convert    ' Get ch1 AD 12 bit value
           ch1 = ch2        ' Convert to 8 bit value
           gosub Convert    ' Get ch2 AD 12 bit value
CalcVector: Rspeed = Mid+ch1-ch2 ' Figure the direction command into
           Lspeed = Mid-ch1-ch2+255 ' the speed command of each servo.
CheckDband: if (Rspeed-LowPoint>Range) then ChkL
           Rspeed = Mid
ChkL:     if (Lspeed-LowPoint>Range) then Drive
           Lspeed = Mid
Drive:    pulsout Lservo,Lspeed ' Generate pulses for servos
           pulsout Rservo,Rspeed
           goto ReadAD
-----
' Subroutine
-----
Convert:   config = config ^ 4 ' Sel other channel
           low CS             ' Activate the ADC.
           shiftout DAT,CLK,lsbfirst,[~config\4] ' Send config bits.
           shiftin  DAT,CLK,msbpost,[ch2\8] ' Get data bits.
           high CS           ' Deactivate the ADC.
           return            ' Return to program.
-----

```