WEEK 2
Experiment # 4

The activities on this week are about instructing the Boe-Bot where to go and how to get there. You will write programs to make the Boe-Bot perform a variety of maneuvers. You will also learn how to build a low battery indicator first.

Low Battery Indicator

When the voltage supply drops below the level a device needs to function properly, it is called *brownout*. The BASIC Stamp protects itself from *brownout* by making its processor and program memory chips inactive. A circuit called "Low Battery Indicator" will detect this kind of condition.

When the voltage comes back, the BASIC Stamp has been *reset*. On *reset* condition, the BASIC Stamp behaves the same as if you had unplugged the power and then plugged it back in. In either case, the program the BASIC Stamp was running before the *reset* condition starts over again at its beginning.

One way to indicate *reset* is to include a code at the beginning of all the Boe-Bot's program. This code will make the piezoelectric speaker installed emits a tone each time the BASIC Stamp runs from the beginning OR *reset*. Figure 2.1 shows how does the piezoelectric speaker look like.

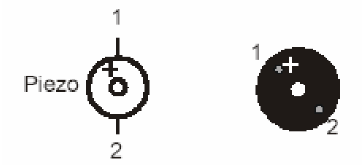

Figure 2.1: Piezoelectric speaker

Activity:

Attach the speaker to the breadboard. Connect the *positive* lead of the speaker to one of the I/O pins on the side of the breadboard, on this case we are using pin P2. You also need to connect the other lead of the speaker to the Vss on the side of the breadboard. Your wiring construction should look like Figure 2.2.

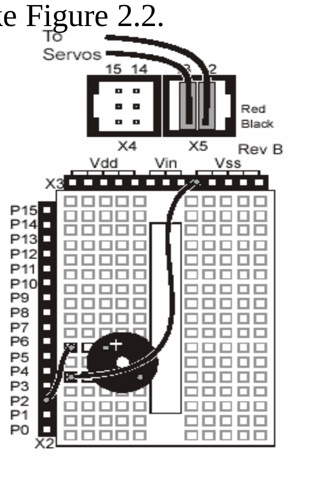

Figure 2.2 Wiring diagram

Type the following codes onto the Stamp Editor.

```
‘ Program Listing 2.1, The Battery Indicator.
‘ {$Stamp bs2}                       ‘ Stamp Directive

debug cls, “Beep!!!”                 ‘ Display while speaker beeps
output 2                             ‘ Set P2 as output pin
freqout 2, 2000, 3000                ‘ Send a 3 kHz signal for 2 seconds

loop:                                ‘ Loop label
   debug “Waiting for reset …”, cr   ‘ Display while the BS2 is waiting
goto loop:                           ‘ Go to the loop label (infinite loop)
```

Program explanation:

- *debug cls, "Beep!!!"* : clears the screen and then displays *Beep!!!* message on the debug terminal
- *output 2* : sets pin P2 as an output pin (ie used to send a signal out)
- *freqout 2, 2000, 3000* : produces 3000 Hz (= 3 kHz) tone through pin P2. One Hertz (Hz) is simply one time-per-second. The tone is played for 2000 milliseconds (= 2 seconds)
- *debug "Waiting for reset …", cr* : displays *Waiting for reset …* message on the debug terminal and then positions the cursor on the next line
- *loop: … goto loop:* : repeats the command(s) in between (…) infinitely.
  It executes the line of *loop:*, then it goes to *debug "Waiting for reset …", cr* command. After that, it executes *goto loop:* command where it will take to *loop:* label again and so on.
  So, it is an infinite loop

Save this program as 'Prog2_1.bs2'. How??
Click on File – Save (or alternatively use the short-cut: Ctrl + S). Type in '*Prog2_1*' as the file name on the box next to the 'File name:' text. Click on '*Save*' button. Don't worry about the '*.bs2*' extension because it only indicates that this file is a Basic Stamp 2 file.
Run this program. How??
Click on Run – Run (or alternatively use the short-cut: Ctrl + R).

Verify that the low battery indicator works by pressing and releasing the reset button on the Board of Education (BOE). When the program first ran, the speaker should play a high pitched tone for 2 seconds. At the same time, the debug terminal should display the "Beep!!!" message. Then, it should go silent while the debug terminal displays line after line of the "Waiting for reset …" message.

Task:
Copy the '*output 2*' and '*freqout 2, 2000, 3000*' commands from Prog2_1.bs2 and insert them into the beginning of the previous program you have. Run that program again.
The Boe-Bot should remain still and beep for two seconds before starting to move.

Hint:
If you need to copy one or more lines from one Program Listing to the other Program Listing you can use the *Copy* and *Paste* command on Stamp Editor. Do the following steps:

- Make sure that the Program that you want to copy the lines from (source Program) is opened along with your current Program.
- Click on the source Program tab.
- Highlight whichever lines you want to copy.
- Click on Edit on the menu bar, then choose Copy.

  Alternatively you can use the short-cut: Ctrl + C.
- Click on the current Program tab.
- Position the cursor on the location where you want to put those lines on.
- Click on Edit on the menu bar, then choose Paste.
  Alternatively you can use the short-cut: Ctrl + V.

Controlling Distance

An infinite loop command doesn’t know when to stop, to get out of the loop. The best way to fix the problem is to replace the infinite loop with another kind of loop called a *for … next* loop. You can use a *for … next* loop to specify how many times the commands inside the loop are executed.

Activity:
Type the following codes onto the Stamp Editor.

```
‘ Program Listing 2.2, Controlling Distance.
‘ {$Stamp bs2}                      ‘ Stamp Directive

‘ ------Declarations -----
pulse_count   var word              ‘ Declare a variable for counting

‘ ----- Initialization -----
output 2                            ‘ Set P2 as output pin
freqout 2, 2000, 3000               ‘ Signal program is (re)starting
low 12                              ‘ Set P12 and P13 to output-low
low 13

‘ ---- Main routine -----
main:
   forward:                         ‘ Forward routine
      for pulse_count = 1 to 100    ‘  Loop  that  sends  100  forward-move
pulses
         pulsout 12, 500            ‘ 1.0 ms pulse to right servo
         pulsout 13, 1000           ‘ 2.0 ms pulse to left servo
         pause 20                   ‘ pause for 20 millisecond (20 ms)
      next
   stop                             ‘ Stop until the reset button is pressed
```

Program Explanation:

The line of *forward:* is just used to indicate that the following lines belong to *forward* routine.

This program is using a finite loop, specifically in the *forward* routine. It is using a word type of variable called *pulse_count*, which is used to specify that the commands in between *for … next* will be executed only 100 times. It executes *for pulse_count = 1 to 100* command then it executes *pulsout 12, 500*, *pulsout 13, 1000*, and *pause 20* commands consecutively. After that, it executes *next* command which will bring to *for pulse_count = 1 to 100* command again. So, the three command lines:

pulsout 12, 500
pulsout 13, 1000
pause 20

will be repeated 100 times.

A word variable can store numbers between 0 and 65535. Look at Table 2.1 for the other options of variable declaration

Size Declaration	Number of Bits	Can Store Number Ranging from-to
bit	1	0 to 1
Nib	4	0 to 15
Byte	8	0 to 255
Word	16	0 to 65535 (or -32768 to +32767)

Table 2.1: Variable Declaration Sizes

The *stop* command is executed after the *for … next* command executed completely (100 times). The '*stop*' command will stop the program.

Save this program as 'Prog2_2.bs2'. Run this program.

The Boe-Bot should remain still and beep for two seconds before starting to move. After some time, it should stop completely.

If it stops and you want to run it again, you can press the Reset (Rst) button on the BOE.

Notes:

Remember to unplug the battery pack from the BOE when you are not using it.

If the tone plays for no apparent reason when the Boe-Bot is in the middle of a maneuver, it indicates a brownout condition caused by low batteries.

Task:

Insert the following codes onto your Program Listing 2.2. Make sure you insert it before the *stop* command.

```
      pause 500                        ‘ Pause for 0.5 second (0.5 s)

   backward:
      for pulse_count = 1 to 100       ‘ Send 100 backward-move pulses
         pulsout 12, 1000              ‘ 2.0 ms pulse to right servo
         pulsout 13, 500               ‘ 1.0 ms pulse to left servo
         pause 20                      ‘ Pause for 20 ms
      next

      pause 500                        ‘ Pause for 0.5 s
```

Save this program with the same file name (Prog2_2.bs2). Run this program.
Your Boe-Bot should remain still and plays the tone for 2 seconds. Then it starts moving forward. Stops for about 0.5 second and then it should move backward and stops on its initial position.

Distance Calculation:

When programming the Boe-Bot, the goal is often to make it moves a specific distance or to execute a particular turn. It is helpful to know how to calculate how far the Boe-Bot will travel or turn. Here are the detail calculation for that.
Circumference of the Boe-Bot wheel :

π * wheel diameter = 3.14159 * 6.67 cm $\approx$ 21 cm.

So, now we know that with one complete turn of the wheels, the Boe-Bot travels about 21 cm.
If we send pulses to the servo for the correct amount of time, the Boe-Bot can be made to travel a specific distance. This is because we can measure the speed that the Boe-Bot wheels turn. Once the speed is known, speed multiplied by time gives you distance. For example, with a *pulsout* command of 1000 delivered every 20 ms, the servo will turn at about 37.5 revolutions per minute (RPM), or 0.625 revolutions/sec. So the speed will be about:

21 cm/revolution * 0.625 revolutions/sec = 13.125 cm/s.

The time it takes to make the Boe-Bot travel 50 cm is ;

t_{travel} = 50 cm $\div$ 13.125 cm/s $\approx$ 3.81 seconds

Since the pulse and pause periods are known, the time it takes for a single loop can be calculated:

t_{loop} = 1.0 ms + 2.0 ms + 20 ms = 23 ms

Calculating the number of loops is simply by dividing t_{loop} with t_{travel} :

Number of loops = 3.81 sec $\div$ 23 ms/loop = 3.81 sec $\div$ 0.023 s/loop $\approx$ 166 loops.

Task:

Modify 'Prog2_2.bs2' for 166 forward loops (about 50 cm distance) and 166 backward loops. Run the program.
Your Boe-Bot should remain still and plays the tone for 2 seconds. Then it starts moving forward for about 50 cm. Stops for about 0.5 second and then it should move backward and stops on its initial position.

Notes:

Several factors can effect how far it moves, including differences between servos and battery voltage. However, the estimate of 166 is a good initial value to try. The exact amount for the loops can then be fine tuned.