Bluetooth Communications For Everyone
Using The BasicStamp 2 & RobotBASIC

The Need

Microcontrollers like the BasicStamp 2, SX, PicMicro, 68HC12 etc. are powerful and very useful in hardware projects and as robot controllers. When programmed with higher-level languages like C and BASIC you are able to do simple math like multiplication and division. However, often you have to take care doing these operations due to over and underflow because many of these processors are limited to 16 bits or even 8 bits math.

Most languages for microcontrollers do not provide floating-point nor even fixed-point math operations. Some languages provide *integer* trigonometric and other math functions, but these are often not sufficient for doing complex calculations.

Another shortcoming of most microcontrollers is the limited memory available for program and data space. This limitation curtails the ability to create programs that can handle complicated tasks like robotics and AI. Even when a microcontroller does not have most of the limitations mentioned above, it is very rare that it would be able to do matrix math, which is often a requirement in many AI projects.

Another major hindrance in programming microcontrollers is the cycle of writing the program on a PC, compiling it, downloading it to the processor, trying to run it and then having to repeat the entire process multiple times before getting a program to work.

In robotic projects there is the added frustration of not being able to test the program while the robot is linked to the PC and thus the inability to debug the program by reporting on certain parameters while the robot is carrying out its tasks. Even if you use an onboard LCD it is often impractical to follow the robot trying to read the LCD.

The Solution

If your robot is big enough you can dispense with the microcontroller and use a PC or laptop as the onboard controller. However, there is a solution that works with any size robot and is a great solution for robotic as well as non-robotic based projects. If you have a PC program that communicates wirelessly with the microcontroller you would be able to dispense with the need of writing complicated programs on the controller.

The controller program becomes a simple program that receives commands from the PC and actuates the required electronics to carry out the commands. Some of the commands may require feedback to the PC to report on the status of sensors and other instruments. Other commands would require starting and stopping motors and other devices. The microcontroller becomes a transceiver that collects data and sends it to the PC. It also receives data and commands to actuate various onboard devices.

Another advantage of this solution is that the project may be a distributed one where multiple microcontrollers are in charge of controlling widely distributed devices. Additionally, even if the devices are close together, if the number of devices is more than

one processor can handle, multiple microcontrollers would be required. With this solution the PC will act as a central hub collecting data from various controllers and sending commands to the ones that need to actuate an action after analyzing the data and arriving at appropriate control parameters.

The PC executes all the AI calculations required to manipulate the data using all the power available through a PC program. The microcontroller's program becomes a simple one that does not require any math or large memory.

Another advantage in using the above setup is the ability to simulate the project on the PC before doing the actual work on a real system. The functionalities of the project can be simulated on the PC by using routines that simulate sensors and actuators on the real project. The AI algorithms can be tested and honed using the simulator and once you are satisfied with the results you can tell the same PC program to carry out the algorithm on the real project by switching from communicating with the simulated routines to communicating with routines on the microcontroller.

The Protocol

To illustrate the above principle we are going to write a program that sends data from a PC to a BasicStamp 2 processor and also to receive data from the BasicStamp 2 back to the PC. The system will use a USB Bluetooth transceiver on the PC and the EB500 Bluetooth transceiver on the BasicStamp 2. Bluetooth will be abbreviated to BT and BasicStamp to BS2.

To be able to communicate between the PC and the BS2 we will need to write programs that carry out the processing as well as the communications. On the BS2, the PBasic language will be used. On the PC side we shall use the freely available programming language and robot simulator RobotBASIC (WWW.RobotBASIC.Com).

The project will not be too complicated. The PC will send two numbers to the BS2 representing a frequency and duration. The BS2 will output the desired frequency for the desired duration on pin P13. If a speaker is connected to P13 then a sound will be heard. If the PC keeps repeating the action each time sending a different frequency and duration a song can be played on the BS2 side.

Also to illustrate communications in the reverse direction, the BS2 will send a number representing the status of pin P14 back to the PC. If a push button or dipswitch is connected to P14 then a 1 or 0 can be sent back to the PC. The PC will then test the received number and if it is a 0 the PC will pause sending data to the BS2 as long as the button is pushed. When the button is released the PC will resume sending data to the BS2. However if the button is held down for longer than a certain time the PC will terminate the program.

The above is not very complex but demonstrates all the required actions:

1. The PC sends data to the BS2 representing commands.
2. The BS2 acts according to the data received and actuates the desired actions.

3. The BS2 sends data to the PC representing the status of sensors and instruments.
4. The PC acts on the data received, processes it and decides what to do next.

This protocol can be used in any project and we shall show later how RobotBASIC has an inbuilt protocol tied with the robot simulator to enable control of a real robot using a similar procedure as outlined above.

The Simulation

Before we do any of the above we shall write a program on the PC using RobotBASIC to simulate the actions that are going to be performed in hardware. Once the program on the PC is running and is fully debugged we shall see how to write a program on the BS2 to be able to receive the commands and send back the status of the button. Also we will show what modifications are necessary on the PC side to enable the communication.

The command Sound *Frequency, Duration* in RobotBASIC sends a signal of a certain frequency for a certain duration to the speaker on the PC to make the sound of the note. If we have an array of frequencies and durations we can play a song. Also the command GetKey *K* checks to see if a key is pressed on the keyboard. We shall use this to simulate the pushbutton on the BS2. Figure 1 shows a program written in RobotBASIC that plays the Jingle Bells tune.

Above the *MainProgram* label there are certain parameters that affect the behavior of the program. The main program calls a subroutine to set up the array required for creating 7 scales of the musical notes A-G (including sharps). Also a subroutine is called to set up the array for the song notes. The call to the subroutine *SetUpCommPort* is not required during the simulation but will be required later; the subroutine will not execute any commands if the constant *COMM_PORT_NUMBER* is set to zero. A flag *StopIt* is set to false and then a loop is entered to repeatedly play the song. The loop will terminate if you terminate the program or if the flag *StopIt* ever becomes true. We shall see how this flag is set later.

The subroutine *SetUpNotes* uses the Data statement to fill an array with frequencies for the various musical notes for 7 musical scales. Also constants are set to make it easy to specify a note. The constant P is to set a pause and S is to specify a scale. The other constants specify the note's index in the array. Since the array starts at note C then C is index 0, C# is represented by the constant CS and is next number in the array etc. To specify a scale the constant S is used to specify what scale and thus *C+Scale*12* becomes the actual index into the *Notes[ ]* array for the note C in that scale. The number 12 is used since there are 12 notes in each scale.

The subroutine *SetUp_Jingles* specifies an array *Song[ ]* of notes and durations. The durations are numbers 1,2,4,8,16,32, etc. which when divided into the *Tempo* value give the length in milliseconds of the note (or pause). The variable Tempo sets the tempo of the song (1000 is one second). The default scale will be 4 if not specified. The constant S specifies a scale (0 to 6) and once specified, stays that way until changed by another

occurrence of S. The constant P specifies a pause and duration just as if it were a note but with no sound (frequency 0).

The subroutine *PlaySong* iterates through the array *Song[ ]* looking at a pair of data *Song[i]* and *Song[i+1]* where the first one specifies the note position in the scale. The next value specifies the duration divider. The subroutine sets the variables *Frequency* and *Duration* according to the data in the array and calls the subroutine *PlayNote_PC* or *PlayNote_BS*, depending on the value of the *COMM_PORT_NUMBER* constant at the top of the program. If this constant is set to zero, it indicates that the simulation is required rather than actual communication to the BS2. If the value is other than zero then communications to the BS2 will take place through the comm. port specified and the music will be played on the BS2 not the PC. The *PlayNote_BS* subroutine will be developed later. Since the subroutine does not yet exist if you change the value of the constant to other than zero you will get an error saying the subroutine does not exist.

For now we will continue with the simulation. The subroutine *PlayNote_PC* uses the sound command to make the speaker sound the frequency required for the duration specified. The routine then checks if a keyboard button is pressed. If a key is pressed it enters in a loop after starting a timer. In side the loop a check is made to see if a certain time has elapsed if it has then the flag *StopIt* is set to true to signal a desire to terminate the program. This will percolate all the way up to the main program and cause a program termination. The routine also checks if the key is ever released. The loop is exited if the key is released before timeout, but without setting the flag.

The outcome of all the above is that the program will play the music Jingle Bells forever until you stop the program. If you press any key on the keyboard the song will pause until you release the button. If you do not release the button within 5 seconds (as specified by the *TIME_OUT_DURATION* value of 5000 milliseconds) the program will terminate.

```
//------Variables and Constants
   TIME_OUT_AMOUNT = 5000
   COMM_PORT_NUMBER = 0
//=====================================================================
//=====================================================================
MainProgram:
   GoSub SetUpNotes
   GoSub SetUp_Jingles
   GoSub SetUpCommPort
   StopIt = false
   while !StopIt
      GoSub PlaySong
   wend
   xystring 1,1,"Terminated....",spaces(70)
End
//=====================================================================
//=====================================================================
SetUpNotes:
   data Notes;32.703,34.648,36.708,38.891
   data Notes;41.203,43.654,46.249,48.999,51.913,55.0,58.27
   data Notes;61.735,65.406,69.296,73.416,77.782,82.407,87.307
```

```
   data Notes;92.499,97.999,103.83,110.0,116.54,123.47,130.81
   data Notes;138.59,146.83,155.56,164.1,174.61,185.0,196.0
   data Notes;207.65,220.0,233.08,246.94,261.63,277.18,293.66
   data Notes;311.13,329.63,349.23,369.99,391.99,415.31,440.0
   data Notes;466.16,493.88,523.25,554.37,587.33,622.25,659.26
   data Notes;698.46,739.99,783.99,830.61,880.0,932.33,987.77
   data Notes;1046.5,1108.7,1174.7,1244.5,1318.5,1396.9,1480.0
   data Notes;1568.0,1661.2,1760.0,1864.7,1975.5,2093.0,2217.5
   data Notes;2349.3,2489.0,2637.0,2793.8,2960.0,3136.0,3322.4
   data Notes;3520.0,3729.3,3951.1

   S=-2 \ P=-1\ C=0 \ CS=1 \ D=2 \ DS=3 \ E=4
   F=5 \ FS=6 \ G=7 \ GS=8 \ A=9 \ AS=10 \ B=11
Return
//=================================================================
SetUp_Jingles:
   Tempo = 1000
   data Song;S,4,E,8,E,8,P,32,E,4,P,32,E,8,E,8,P,32,E,4,P,32
   data Song;E,8,G,8,P,32,C,4,D,16,P,32,E,2,P,16
   data Song;F,8,F,8,P,32,F,8,F,16,P,32,F,8,E,8,P,32
   data Song;E,8,E,16,P,32,G,8,G,8,F,8,D,8,P,32,C,2
Return
//=================================================================
PlaySong:
  Scale = 4
  FOR i = 0 TO MaxDim(Song,1)-1 step 2
     if Song[i] = P
       Frequency = 0
       Duration = Tempo/Song[i+1]
     elseif Song[i] = S
       Scale = Song[i+1]
       continue
     else
       Frequency = Notes[Song[i]+12*Scale]
       Duration = Tempo/Song[i+1]
     endif
     If COMM_PORT_NUMBER = 0
        GoSub PlayNote_PC
     else
       GoSub PlayNote_BS
     endif
     if StopIt then break
  next
Return
//=================================================================
PlayNote_PC:
  xystring 1,1,"Note = ",Frequency;"Duration = ",Duration,spaces(10)
  sound Frequency,Duration
  getkey K
  if K > 0
    xystring 1,1,"Waiting",spaces(70)
    StartTime = Timer()
    repeat
      getkey K
      if Timer()-StartTime > TIME_OUT_AMOUNT
        StopIt = true
```

```
        break
      endif
    until K = 0
  endif
Return
//=================================================================
SetUpCommPort:
   if COMM_PORT_NUMBER < 1 then return
   setcommport COMM_PORT_NUMBER
   clearserbuffer
   settimeout TIME_OUT_AMOUNT
Return
//=================================================================
```

Figure 1: Program for Simulation. Actual Communications will be established later

Communicating Between RobotBASIC and BasicStamp

To make RobotBASIC communicate with the BS2 we would need a serial communication medium and programs on the BS2 and PC to send data back and forth. The communication medium can be a direct wire or a wireless one.

The various setups for carrying out the project should be:

1. A PC connected to the BS2 to program it and debug it using the PBasic IDE. The connection to the BS2 can be either:
    a. A serial port with the standard 9 pin connector on the PC and on the BS2 carrier board, such as the Education Board or the Professional Development Board.
    b. A USB port connection on the PC side with drivers to set up a virtual comm. port and a converter on the BS2 side to connect to the BS2 carrier board's standard 9 pin connector.
    c. A USB connection on both the PC and BS2 carrier board with the required drivers provided by Parallax Inc. The carrier board provided with the BoeBot kit has this setup. Also the Professional Development Board has this setup.

2. A PC with RobotBASIC running on it that will communicate with the BS2 via either:
    a. A serial wire which can be either:
        i. A USB connector on the PC side with virtual serial comm. port drivers. The other side has a converter to be able to connect to a pin on the BS2. Conversion might be required to allow for inverted voltage levels etc. This conversion is described in the BS2's manual, which can be downloaded as a PDF file from the www.Parallax.Com web site.
        ii. A standard 9 pin serial port on the PC connected to the BS2 as described in option 2.a.i above.
    b. A wireless serial connection which can be either:

i. A Bluetooth device that sets up a virtual comm. port on the PC which will connect with the EB500 Bluetooth transceiver connected to the BS2.
ii. A 940 MHz transceiver with a board to power it and to enable communication to a serial port on the PC, and a compatible transceiver connected to the BS2.

3. The PC in step 2 above can be the same as the PC in step 1 above. But the PC will need a second 9 pin serial port or USB port if you are not using option 2.b.i.

If you happen to own Parallax's Professional Development Board, your best option would be 2.a.ii (can be one PC). This board has a USB connector as well as a 9 pin serial connector for programming the BS2 and also it has a 9 pin connector which can be used to communicate with the onboard BS2 via any pin. Additionally this board has pushbuttons, dipswitches, LEDs and many other devices with which you can carry out many experiments without the drudgery of wiring. Also there is a Piezo speaker that can be used for this project with ease. Furthermore, with this option you do not need to buy the Bluetooth devices for the BS2 and PC. Of course if you will be using this methodology to control a robot some time later you would still need to have some kind of wireless connection for more effective projects.

We will use Option 2.b.i with the two PCs being one PC. Also the Education Carrier board will be used for the BS2. The EB500 BT transceiver is available from Parallax Inc (WWW.Parallax.Com). This device communicates with any other BT transceiver and makes the data received available to the BS2 via a serial link. Also data from the BS2 can be sent via a serial link to the BT transceiver to be sent to any BT transceiver ready for receiving it.

Figure 2: The Board Of Education carrier board with the EB500 connected.

The Education Carrier board accepts the EB500 as shown in Figure 2. The connector on the education board provides the necessary power connections and also connects the various pins on the EB500 to pins P0, P1, P2, P3, P5 and P6 on the BS2. P0 will be the receive pin and P1 will the transmit pin. P2 and P3 will connect to the RTS and CTS

hardware handshaking pins on the EB500, but will not be used in this project. Also P5 and P6 will connect to the Status and Mode pins on the EB500 and will not be used in this project. There is a warning in the EB500 manual that pin P5 on the BS2 must not be an output pin. So before you connect the EB500 to the BS2 make sure that you have programmed the BS2 previously ensuring that P5 is set up as an input. This can be achieved by programming the BS2 with nothing more than with the command END as a program. This makes sure that any previous programs in the BS2's memory are erased and that the BS2 is reset to all pins being inputs. We will develop programs on the BS2 later on.

The EB500 defaults to serial communications mode of 9600 Baud, 8 Data bits, no parity, 1 stop-bit and no flow control. This mode is very suitable for communicating with the BS2. You can change these settings; refer to the manual provided in PDF format on the Parallax site. Also the manual provides many configurations with which to connect to the BS2.

On the PC side we will use the D-Link BT120 USB connector (you can use any equivalent device you can obtain). This BT device connects to any available USB port and comes with software that enables the device to search and connect to any active BT device within range. Once you have established a connection to the EB500 on the BS2 the PC will have a virtual port that acts for all intents and purposes as if it were a real RS-232 port connected directly to the BS2.

By far the easiest procedure to establish communications is:

1. Turn on the power to the carrier board with the BS2 and EB500 connected as shown in Figure 2.
2. Go to the PC and start the Bluetooth Settings program. This can be done by double clicking on the Bluetooth icon in the system tray or through the Start menu.
3. You only need to do this step one time only for the first time. Later you can use the icon that will be established to carry out the rest of the procedure skipping to step 4.
    a. Let the BT manager search for the EB500 using the "New Connection" button.
    b. Once found accept all the options as provided.
    c. You will need to establish trust by giving the Passkey (or PIN) "0000". That is four zeros.
    d. Note the virtual comm. port number assigned to this device. It may be 40 or 41. Accept the suggested default.
    e. Continue the process accepting all defaults.
    f. Finally you will be back to the original Bluetooth Settings Screen.
4. Double click on the EB500 icon to connect to the EB500. If you do not remember what port it is connected to, right click the mouse on the icon and choose the Detail menu option from the dropdown menu. This will open up a window that shows the comm. port number.

The Programs and Hardware

Once you have completed the above procedure you are ready to communicate with the BS2, but you still need a program on the BS2 to do the job. We are going to develop the program on the BS2 as well as on RobotBASIC to carry out the project simulated in the previous section.

Figure 3 shows the program that should be entered using the PBasic IDE and downloaded to the BS2. Figure 4 shows the new subroutine that will be used to carry out the actual communication between the BS2 and RobotBASIC. You will also need to change the constant at the top of the program in Figure 1 to say *COMM_PORT_NUMBER = nn* where *nn* is the number that you noted in step 3.d above. Also add this new subroutine to the end of the program in Figure 1.

For now we will not connect a speaker. But you do need to connect pin P14 to a High (1) (i.e. Vdd) using a wire between the pin and the Vdd slot provided on the carrier board. You now can run the program on the PC and will see it running without sound. This indicates that it is sending commands to the BS2. However the BS2 does not have any speaker connected to P13 and P14 is always high as if the pushbutton is not pressed.

The *DEBUG* statements in the BS2 program will display information on the programming PC screen in the debug window and will show that the procedure is working even though there is no speaker or push button. If you connect a speaker later you can comment this Debug statement to speed up the procedure. There is need to waste time sending out the data to the debug screen when you can hear the speaker.

For a speaker you can use the arrangement in Figure 6 and for the pushbutton the one in Figure 5. The pushbutton is configured to be active low. This means that when the button is not pushed a high (1) will be on P14 and when the button is pushed a low (0) is reflected on P14. Figure 7 shows the actual connection on the board of education.

```
' {$STAMP BS2}
' {$PBASIC 2.5}
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
'=============  Variables  =================
    ReceivePin PIN 0
    SendPin PIN 1
    Note VAR Word
    Duration VAR Word

'============= MainProgram ==================
Main:
  PAUSE 1000 'WAIT FOR the eb500 tO be ready
  DO
    SERIN ReceivePin,84,[DEC Note]
    SERIN ReceivePin,84,[DEC Duration]
    DEBUG DEC Duration,"  ",DEC Note,CR
    FREQOUT 13,Duration,Note
    SEROUT SendPin,84, [IN14]
    IF IN14 = 0 THEN
        GOSUB WaitForit
```

```
    ENDIF
  LOOP
END
'======== Subroutines  ======================
WaitForit:
  DO
     IF IN14 = 1 THEN
        SEROUT SendPin,84,[IN14]
        EXIT
     ENDIF
  LOOP
RETURN
```

Figure 3: The program on the BasicStamp side.

```
PlayNote_BS:
   xystring 1,1,"Note = ",Frequency;"Duration =
",Duration,spaces(10)
   serout round(Frequency),","
   delay 10
   serout round(Duration),","
   delay Duration*1.1
   serbytesin 1,m,c
   if c = 0
     StopIt = true
   elseif ascii(m) = 0
     xystring 1,1,"Waiting",spaces(70)
     while true
       serbytesin 1,m,c
       if c > 0
         if ascii(m) = 1 then break
       else
         StopIt = true
         break
       endif
     wend
   endif
Return
```

Figure 4: New subroutine to be added to the base program of Figure 1. It carries out the actual communication from the PC to the Basic Stamp.

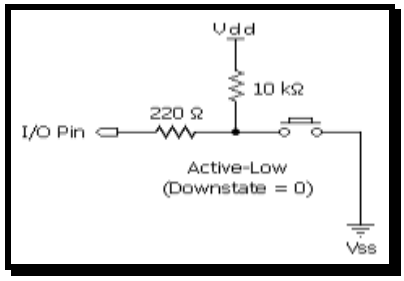

Figure 5: Connecting an active-low pushbutton. We will use P14 on the BS2 to connect to the pushbutton

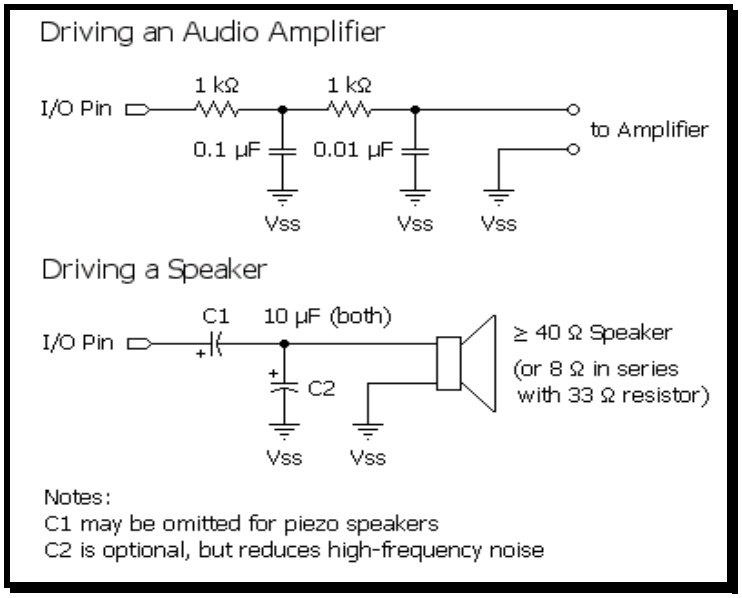

Figure 6: Connecting a Speaker. The speaker can be a Piezo speaker using the lower connection diagram as indicated. We will use P13 on the BS2 to connect to the speaker

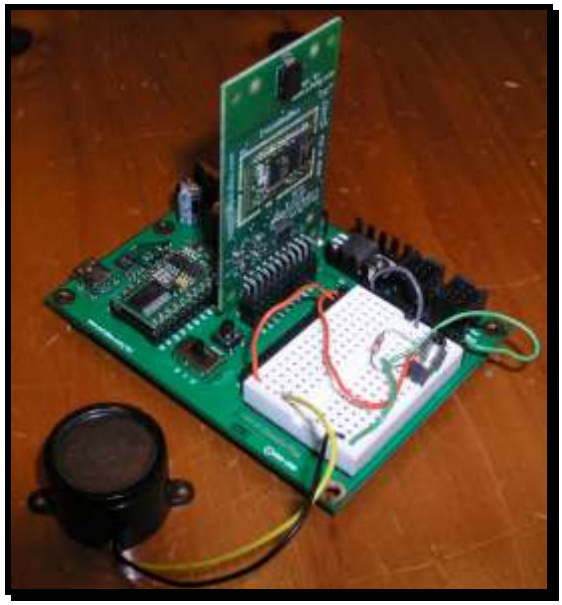

Figure 7: Connections to P14 (pushbutton) and P13 (piezo speaker) on the board of education.

Program Analysis

The subroutine in Figure 4 uses the commands *SerOut* and *SerBytesIn* in RobotBASIC that enable serial communications. The subroutine *SetUpCommPort* in Figure 1, will setup the port as specified in the parameter at the top of the program in Figure 1. The command allows for setting the baud rate etc. but if they are not specified the command assumes 9600,8,N,1,N which is the default for the EB500 and also the BS2 program will

use this mode as indicated by the parameter 84 in the statement *SerIn 0,84,[…]* in Figure 3.

The subroutine *PlayNote_BS* uses delays of 10 milliseconds between transmissions of each number due to the fact that the BS2 does not have a receive buffer and thus will loose the data if it arrives before the BS2 is ready to receive it. Also notice that the last delay is set to the value of the duration of the note to be played plus a 10% to allow the BS2 time to play the note and to have time to read the status of the pushbutton before the PC starts waiting for 5000 milliseconds to receive a number from the BS2. If no number is received from the BS2 the subroutine will cause the program to terminate by setting the *StopIt* flag to true before returning. If a number is received it is checked for being zero. If it is not zero the routine returns. If it is zero it will enter into a loop waiting for input from the BS2 and if the number does arrive that is not zero the routine will return. If the pushbutton stays down for 5000 milliseconds, which is the time out on the serial input command the routine will set the *StopIt* flag and return. This works due to the fact that BS2 will send the number again after it became zero only if it becomes one. Thus since there will be no data received (unless the pushbutton is released) the *SerBytesIn* command on the PC will time out and will set the number of bytes received to zero, which the program checks to establish that no data has been received.

The BS2 program enters an endless loop where it waits to receive two numbers one after the other. It then executes the *FreqOut* command at the received frequency and duration and then sends the status of P14. The pins used for receiving and sending are specified at the top of the program as P0 and P1. This is according to the connection as in Figure 2. But you can change these if you are using a different setup.

If P14 becomes zero the subroutine *WaitForIt* is called. This subroutine enters a loop waiting for P14 to become one again. If the button is ever released, the routine sends the pin status again and returns to waiting for further notes. If the button is not released till after the PC program has already timed out the BS2 program will just keep waiting for data until it receives some or you turn it off.

Final Thoughts

This project is fun and achieves an interesting function. However, the aim is to be able to control a robot using the same ideas established here. RobotBASIC has a very powerful, realistic and effective robot simulator that allows for simulating a robot and trying out many robotic activities. There is a book that will be published soon showing many projects and algorithms for doing interesting and challenging tasks.

In the book there is a chapter that describes how RobotBASIC can be made to use the same program that you write to simulate a robot on a real robot using the principles discussed in this article. All the simulator commands and functions can be made to communicate with any microcontroller via wireless (or wired if you wish) serial communication protocol.

The commands do not have to be changed as we have shown here. RobotBASIC has an inbuilt command (rSimulated) that allows you to specify a serial comm. port. Once you have issued this command all the other simulator commands and functions will send and receive data via the comm. port instead of simulating the robot on the screen.

No work needs to be done to make the program do the communications. In this article we had to write a subroutine to replace the Sound command. However the robot simulator commands like rForward, rTurn, rLook() etc. do not need to be replaced in this manner. The language will do the serial communication according to a preset protocol automatically once the *rSimulated* command is issued with a valid comm. port.

All you have to do is write the correct receive and transmit program on the microcontroller to communicate correctly with the RobotBASIC protocol and to carry out the commands received. The program on the microcontroller side would be very similar to the one in Figure 3. The program enters a loop waiting for a command and its parameters. One from a collection of subroutines would be called depending on what command has been received from RobotBASIC. Then a set of data is sent back to the PC to establish that the command has been carried out and also the status of any parameters that are expected.

There would have to be a routine to move forward, to turn, to interrogate the line sensors, bumpers, etc. These subroutines will reflect the ones made available on the RobotBASIC side. If you do not wish to implement all the functionalities in RobotBASIC you can limit your simulated programs to use only the commands and functions that *are* implemented in the actual robot. This way when you switch to using the real robot only the implemented commands would be sent to the robot.

Another solution is to have a catchall subroutine on the BS2 that would be called if any unrecognized, or un-implemented command is received. This catchall routine would allow for graceful error handling without failing the robot.

If you do not wish to use the standard protocol provided within RobotBASIC, or you require more data to be sent back and forth, you can establish your own protocol as we have shown in this article. You will need to do a similar action of making the commands execute the simulator commands if a simulation is required or a replacement subroutine if a communication is to be done; just as we have done here.

Checkout our web site www.RobotBASIC.com for information on the book and to download RobotBASIC and all the programs in this article.