Communicating RobotBASIC With The Propeller Chip

The Propeller Chip (propeller) is a an advanced microcontroller that can be programmed using Assembly language or a fascinating and pleasure to use proprietary language called Spin. We shall develop programs to achieve serial communications between a PC running a RobotBASIC program and the propeller running a Spin program.

The article will not attempt to explain the nuances of Spin. You are assumed to be familiar with Spin and the propeller. However, the article will explain options for connecting the propeller to a PC so as to be able to communicate with an RB program running on it.

The developed programs will be of sufficient complexity to illustrate the communication most likely to be needed with projects that require communicating a PC and the propeller using RB and Spin.

The Layout:

Figure 1 is taken from page 5 in the Propeller Data Sheet and it shows the connection the Propeller Tool (programming IDE) assumes in order to program the propeller.

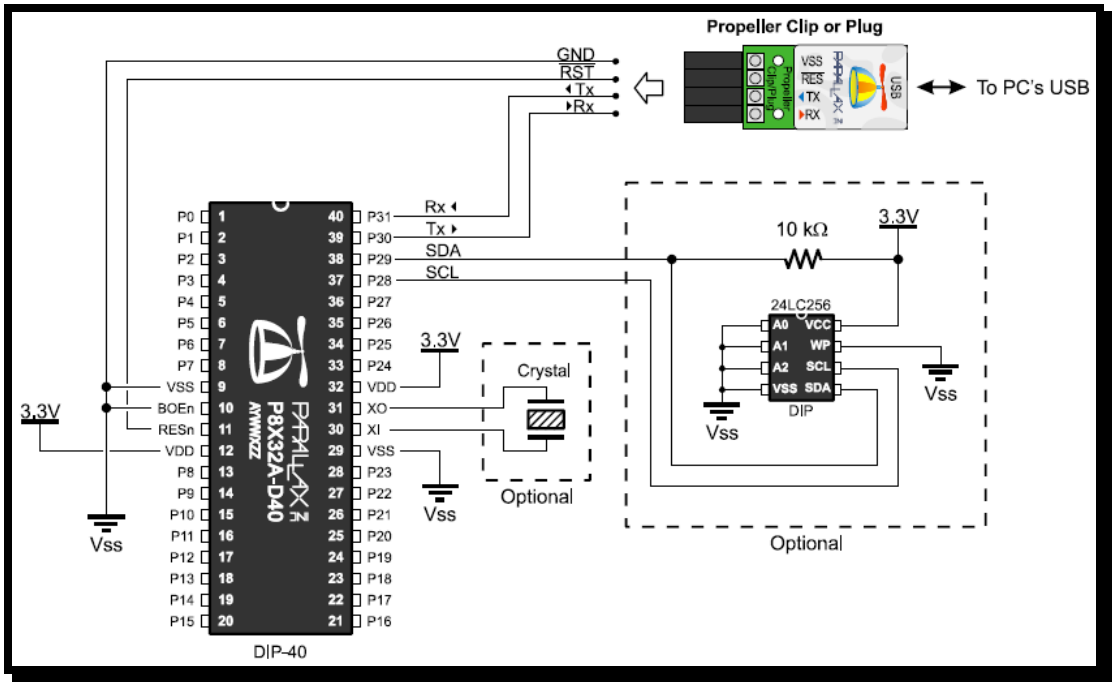

Figure 1: Propeller Programming Setup (Propeller Data Sheet page 5)

The Propeller Plug (or its built in equivalent on certain development boards) provides a Serial-Over-USB connection to the propeller. This is effectively a Com port for all intents and purposes no different from a normal RS232 Com port.

The Propeller Tool (programming IDE) uses this Com port to control the propeller and to program it either writing the program to RAM in the propeller or to an ***external*** EEPROM from which the propeller loads its RAM program every time it resets. This way the propeller does not have to be reprogrammed following a reset.

Notice that this arrangement requires the use of physical Pins 37 to 40. These pins are also known as I/O pins 28 to 31. The propeller plug also makes a connection to the RESn pin through the PC's Serial port DTR pin. We shall see the significance of this later.

Note: ***From this point forward we shall refer to pins by their functional use. So physical pin 11 for example will be called RESn and we will refer to physical pin 40, for example, as I/O pin 31 which is its I/O number***.

Creating A Connection:

You can establish a serial link to the propeller from the PC in one of two ways:

1- Using any two I/O pins other than 31 to 28.
2- Using I/O pins 30 and 31.

Using any I/O pin other than 31 to 28

Two I/O pins are needed; one for transmitting (Tx) from the propeller to the PC (Rx) and one to receive (Rx) into the propeller from the PC (Tx). However there is a problem. All the propeller's I/O pins are 3.3 V TTL and thus cannot be connected directly to the PC's RS232 port. Figure 2 below shows a circuit diagram that you can use to establish a connection.

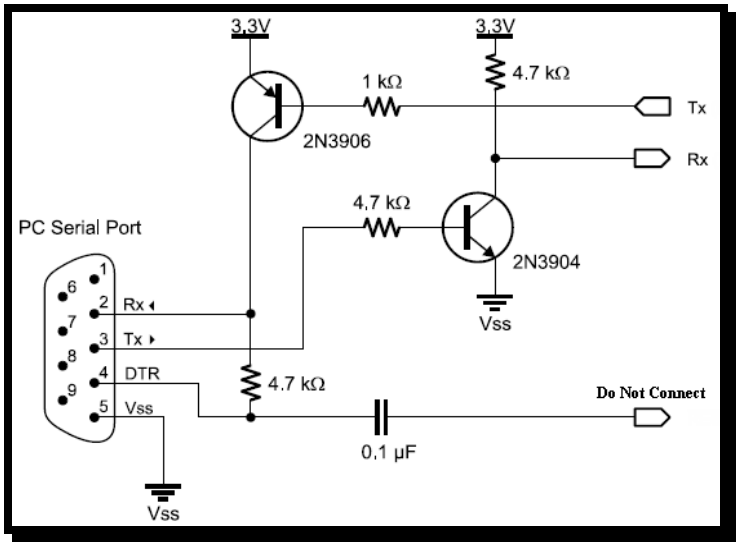

Figure2: A circuit for connecting the I/O pins of the Propeller to an RS232 port on the PC (From the Propeller Data Sheet Page 5)

Using the circuit in Figure 2 you can connect any two I/O pins from the Propeller to the PC's RS232 port. If you do not have an RS232 port you can establish a Serial-Over-USB serial port using the Parallax USB to Serial (RS-232) Adapter Part#28030 from Parallax Inc. (www.Parallax.com). This will give you a 9-pin port just like the one shown in Figure2.

A better alternative to the above is to use another Prop Plug Part # 32201. This will give you the ability to connect directly from the female pin on the plug labeled Tx to the I/O pin on the propeller you want to use as the Rx pin and from the Rx pin on the plug to the I/O pin on the propeller you want to use as the Tx pin. Do not connect the Reset pin. The Vss pin should be connected to the same ground as the propeller.

A third alternative is to use the Propeller Professional Development Board Part #32111. Besides the USB port for programming, this board also has a 9-Pin RS232 output that can be connected directly to I/O pins form the propeller since it has all the necessary circuitry. Thus you can use it as a direct connection to the PC's RS232 or to a Serial-Over-USB mentioned above. This board is a great prototyping tool that can be of great convenience and versatility for developing and prototyping propeller projects, especially for the beginner.

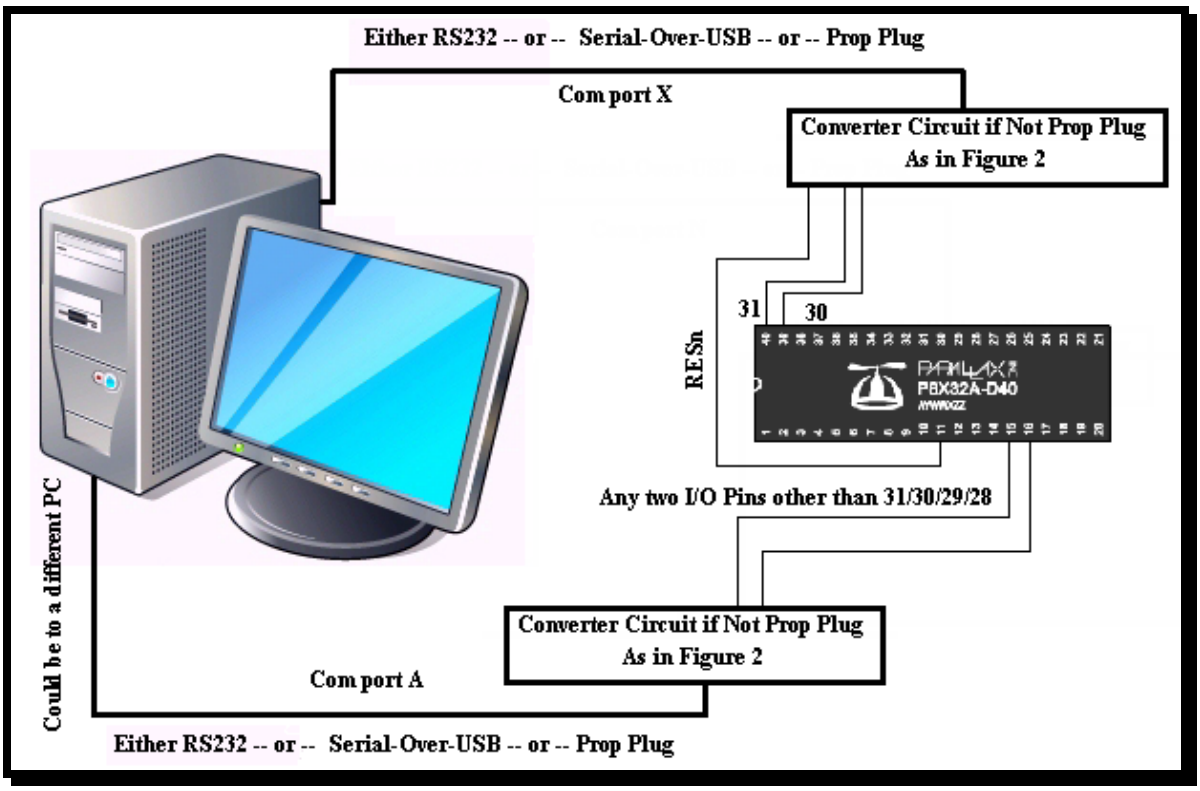

Figure 3: Effective setup for programming the propeller and at the same time communicating to it from a PC running a RobotBASIC program that be the same as the programming PC or a different PC.

Regardless of which method you use the final outcome is that you have the setup shown in Figure 3. In this setup you are communicating from the RobotBASIC program to the propeller through two I/O pins that are not the ones used by the programming connection. The link could be to a different PC than the programming PC or it could be the same. Moreover, the link could be wired or even wireless.

The final outcome is that RobotBASIC will communicate to the propeller over a Com port that is different from the Com port being used by the Propeller Tool to program it. There advantages to this and disadvantages.

The advantages are:

1- You can use the programming com port to also communicate with a program called ParallaxSerialTerminal that acts as a debug tool for debugging your propeller. So you can be using the debug tool at the same time you are communicating to the propeller from RB.

2- When the Propeller Tool starts programming the propeller it will need control of the com port. If you are also using this com port with RB then you must stop the RB program before you can program the propeller. So you will be endlessly starting and stopping the RB program if you use the same port for communicating to the propeller as for programming it. Using two separate ports avoids this trouble.

3- Using I/O pins other than the programming pins avoids conflicts and is what you would normally be doing if you wanted to communicate the propeller to devices other than the PC. So it is a good practice to also consider the PC with the RB program as a device even though it might be the same PC as the programming PC. You can think of the RB side as a device and thus treat it like a device and using I/O pins other than 30 and 31 is what you normally would do with other devices.

4- The link can be wireless so you would use something like the Easy Bluetooth Module Part#3085 from Parallax and would have to use I/O pins other than 30 and 31 anyway.

5- While developing the program for the propeller you can write to RAM directly not to EEPROM. If you use the same com port for communicating from RB to the propeller as for programming it you would have to write the program to the EEPROM since the com port's DTR pin is connected to the reset pin and when RB takes over the port it drives the DTR low. This will reset the propeller chip. If the program is in RAM it will be lost. If the program is in EEPROM the propeller will read it from there and start running it from the beginning again.
This could be a problem or an advantage, nevertheless you do not have the choice, while with using pins other than the programming pins you have the choice. You can either use the EEPROM or RAM.

The disadvantages:

1- You would need to establish a second Com port using another Prop Plug or a Serial-Over-USB (or RS232) but with the extra circuitry in Figure 2.

Using I/O pins 30 and 31

With this setup you are using the same Com port with RB to communicate to the propeller as the one used by the Propeller Tool to program it. So you would be using the Prop Plug used to program the propeller to also communicate with it from RB.

The disdvantages:

1- You cannot use the port within RB at the same time as the Propeller Tool. So you must stop the RB program or write it in such a way as to disconnect from the com port when it is needed to program the propeller.

2- RB will reset the propeller when it establishes control over the com port. This occurs because as you can see from Figure 1, the Reset button of the propeller is connected to the Prop Plug reset pin. This pin is actually the DTR pin on the Com port from the PC. This pin will be driven low momentarily by RB when it establishes control over the port. However, this will cause the propeller to reset too.
This could be a nice feature to have RB reset the propeller when it starts running, or it could be a problem. In any case, you therefore cannot have a program only in RAM. If the program is only in RAM and the propeller is reset there will be no program to run.
So if you are going to use the programming port to also communicate from RB to the propeller then you must always write your programs to the EEPROM.
This is a disadvantage since writing the EEPORM is a lot slower than writing to the RAM. Also the EEPROM has a limited write cycles. So if you are developing the program and are going to write many times you will be a lot better off to write the program to RAM before you finally write the working program to EEPROM. But with this option you cannot.

3- The ParallaxSerialTerminal program is great for debugging the propeller while you are developing the program. If your program involves lots of complicated things as well as communicating with RB then it would be nice to also have the ability to see messages from the propeller besides the data it is going to send to the RB program. You won't have this ability with this option.

The advantages:

1- You do not need extra circuitry or Prop Plug. You just use the same one as the programming one. Also you would not need to tie up two I/O pins.

The Programs:

Note: The programs developed below will assume that you are going to use RB over the same port as the programming port. This is because it is the quickest way to test the programs and since they are already written and debugged you won't need to do multiple writes to EEPROM.

Note: The Spin programs below should be compiled and written to the EEPROM (F11).

Note: Once the propeller is programmed, run the RobotBASIC program. The RB program will reset the propeller and start it and thus ensures correct synchronization.

Note: If you are going to use a different com port than the programming port as described above, you must ensure that the RB program is running then reprogram the propeller which will also start it running. If the program is in EEPROM then just reset the propeller using the reset switch.

The RobotBASIC programs below use some advanced buffer manipulation techniques. This is described in detail in Section 3 of the document RobotBASIC_Networking.PDF. You may need to refer to this document for details. You also should refer to the RobotBASIC Help File. Also refer to the document RobotBASIC_Serial_IO.pdf for information on the serial IO system in RobotBASIC.

The following two demos perform the exact same action. They allow you to enter two numbers on the PC using RB and then, at the push of a button, these two numbers are sent to the propeller where they are added and the result is sent back to the PC where it is displayed. You can repeat the actions as often as desired. Initially the RB program will wait for you to start the propeller if you are using a separate com port than the programming port. If you are using the same port as the programming port then RB will reset the propeller automatically (ensure the program is in EEPROM). Once the propeller starts its program will send the string "Rdy" to indicate that it is ready to receive the numbers and that it is up and running. This also serves as a way of synchronization.

The programs are in corresponding pairs. Each pair has a RobotBASIC (.Bas) program and a corresponding Spin (.Spin) program. The first pair constitute the first demo that sends and receives the numbers as Text. The second pair sends and receives the numbers as 32 bit binary numbers with the Little-endian order (see section 3 in the documentRobotBASIC_Networking.PDF) .

Note: ***The demos assume that the propeller will receive data through its I/O pin 31 and will send data through its I/O pin 30. That means that RB will be using the same port as the programming port used by the Propeller Tool. This port is assumed to be Com 5, so the RobotBASIC program will use this port. If you have a different setup all you need to do is change the appropriate assignments in the programs.***

The program pairs are:

RB2Prop_IO_Text.Bas	and	**RB2Prop_IO_Text.Spin**
RB2Prop_IO_Binary.Bas	and	**RB2Prop_IO_Binary.Spin**

RB2Prop_IO_Text.BAS (Send/Receive Text numbers):

```
//----------RB2Prop_IO_Text.BAS-----------------------
CommPort = 5  //change this to reflect your setup
setcommport CommPort,br115200,8,pNone,sbOne,fcNone
ClearSerBuffer
xystring 10,5,"Start The Propeller Program now...."
while true
   repeat  //wait for the "Rdy"
     checkserbuffer n
```

```
   until n == 3
   SerIn s \ if s == "Rdy" then break
wend //repeat until the "Rdy" string is received
//add edit boxes and buttons and instructions
addedit "v1",50,50,100,0,70
addedit "v2",170,50,100,0,48
addedit "re",310,50,100,0,"0"  \ readonlyedit "re"
addbutton "Calc",80,100,100
xystring 10,5,"OK the Propeller is ready...."+spaces(10)
xystring 10,25,"Enter numbers in the edit boxes then press the 'Calc' button"
xytext 155,45,"+","",15
xytext 290,45,"=","",15
while true  //repeat for ever
  repeat  //wait for the calc button to be pressed
  until LastButton() =="Calc"
  //get the numbers from the edit boxes
  v1 = tonumber(getedit("v1"),0) \ v2 = tonumber(getedit("v2"),0)
  //send the two numbers as text one after the other
  //the delay is to give the Propeller time to be ready
  //the char(13) is to indicate the end of the text for the propeller
  serout v1,char(13) \  serout v2,char(13)
  repeat //wait until there is atleast one byte in the buffer
    checkserbuffer n
  until n>=1
  delay 10 //give the the Propeller time to finish
           //sending the text number there is no way to know
           //how many characters but should not take longer
           //than 10 ms to send all of them at the 115200 baud
           //may need longer if slower baud is used
  serin s   //read the buffer
  //assign the number after converting from text to the result edit box
  setedit "re",tonumber(s)
wend
```

RB2Prop_IO_Text.Spin (Send/Receive Text numbers):

```
''----------RB2Prop_IO_Text.Spin-----------------------
CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000
  RxPin         = 31     'can use a different pin...see note above
  TxPin         = 30     'can use a different pin...see note above
  BaudRate      = 115200 'this is a good baud for 80MHz

OBJ
  SerIO : "FullDuplexSerialPlus" 'object that implements serial I/O
                                 'for the Propeller. It is part of
                                 'the Propeller Education Kit library
PUB RB_IO_Demo_Text |V1,V2
  SerIO.Start(RxPin, TxPin, %0000, BaudRate)
  SerIO.Str(String("Rdy"))      'send the Rdy to synchronize
  repeat
     V1 := SerIO.GetDec         'get dec number i.e. text numeric
     V2 := SerIO.GetDec         'get next one
     V1 := V1+V2                'add them
     SerIO.Dec(V1)              'send back the result as text
```

RB2Prop_IO_Binary.BAS (Send/Receive 32-bit binary numbers):

```
//----------RB2Prop_IO_Binary.BAS-----------------------
CommPort = 5  //change this to reflect your setup
setcommport CommPort,br115200,8,pNone,sbOne,fcNone
ClearSerBuffer
xystring 10,5,"Start The Propeller Program now...."
while true
   repeat  //wait for the "Rdy"
     checkserbuffer n
   until n == 3
   SerIn s \ if s == "Rdy" then break
wend //repeat until the "Rdy" string is received
//add edit boxes and buttons and instructions
addedit "v1",50,50,100,0,70
addedit "v2",170,50,100,0,48
addedit "re",310,50,100,0,"0"  \ readonlyedit "re"
addbutton "Calc",80,100,100
xystring 10,5,"OK the Propeller is ready...."+spaces(10)
xystring 10,25,"Enter numbers in the edit boxes then press the 'Calc' button"
xytext 155,45,"+","",15
xytext 290,45,"=","",15
while true  //repeat for ever
  repeat  //wait for the calc button to be pressed
  until LastButton() =="Calc"
  //get the numbers from the edit boxes
  v1 = tonumber(getedit("v1"),0) \ v2 = tonumber(getedit("v2"),0)
  //send the two numbers as binary buffers that contain the Little-endian
  //format of the numbers....8 bytes total
  serout Buffwrite("",0,v1),buffwrite("",0,v2)
  repeat //wait until there 4 bytes (one 32 bit binary number)
    checkserbuffer n
  until n>=4
  serin s   //read the buffer
  //assign the number after reading it from the binary buffer to the result edit box
  setedit "re",BuffReadI(s,0)  //convert the buffer to Integer value)
wend
```

RB2Prop_IO_Binary.Spin (Send/Receive 32-bit binary numbers):

```
''----------RB2Prop_IO_Binary.Spin-----------------------
CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000
  RxPin         = 31     'can use a different pin...see note above
  TxPin         = 30     'can use a different pin...see note above
  BaudRate      = 115200 'this is a good baud for 80MHz

OBJ
  SerIO : "FullDuplexSerialPlus" 'object that implements serial I/O
                                 'for the Propeller. It is part of
                                 'the Propeller Education Kit library
Var
  Long Vals[2]
  byte buff_index

PUB RB_IO_Demo_Binary

  SerIO.Start(RxPin, TxPin, %0000, BaudRate)
```

```
  SerIO.Str(String("Rdy")) 'send the Rdy to synchronize
  repeat
     GetBinaries                  'get the binary 8 bytes
     Vals[0] := Vals[0]+Vals[1]   'add the values
     SendBinary                   'send the binary 4 bytes

Pri GetBinaries
  'reads in 8 bytes and puts them in the memory area starting
  'at the Vals[] array....that effectively writes in the two
  'numbers into the arrays
  repeat buff_index from 0 to 7
    byte[@Vals + buff_index] := SerIO.Rx


Pri SendBinary
  'sends 4 bytes starting at the address of Vals[]
  'that effectively sends the first element in the array.
  repeat buff_index from 0 to 3
    SerIO.Tx(byte[@Vals + buff_index])
```