PARALLAX SEMICONDUCTOR
www.parallaxsemiconductor.com
sales@parallaxsemiconductor.com
support@parallaxsemiconductor.com
phone: 916-632-4664 • fax:916-624-8003

Application Note AN018

Communication with a PC Application

Abstract: Many embedded applications share information with external devices, and the preferred connection between devices is asynchronous serial communications. The multicore architecture of the P8X32A enables the designer to create and deploy device-to-device communications strategies with no impact on the primary application code. In this example a deployed communications support cog manages data between the main application and the serial I/O firmware, using VB.NET and the P8X32A QuickStart board.

Introduction

Making communication between the Propeller and the PC virtually transparent to the designer greatly reduces the effort required to exchange data. The designer can focus on writing the main code without having to worry about the code handling the communication. The designer need only assign or read global variables in order to exchange data with the PC.

Three things are needed for communication between the Propeller chip and a PC:

1. Hardware; in this case a Propeller P8X32A development platform and a Windows PC.
2. Software; in this case a custom Spin object and a custom PC application.
3. Protocol; a format for sending the data between the Propeller and the PC must be devised.

The recommended hardware connection to the PC should be via the USB or serial interface used to program the Propeller. This connection provides the most reliable data integrity and allows for the possibility to reset the Propeller if the need should arise.

The Spin object used to facilitate communication is designed to run as a separate process in another cog on the Propeller. The activity of this object is transparent to the user program, and all that is needed to start the object is the following line of code:

```
cognew(PC_Comms, @stack)
```

The first parameter (**PC_Comms**) is the name of the **PRI** method we are launching into the new cog. **@stack** refers to the declaration for stack space for this method. This method in turn starts the FullDuplexSerial.spin object which launches into a third cog. The code running in the **PC_Comms** cog handles the exchange of data between the FullDuplexSerial.spin object and two global arrays. The FullDuplexSerial.spin object runs in a separate cog and handles the bit timing, sending, receiving, and buffering of data between the Propeller and the PC.

In addition to the single line of code three declarations are necessary:

```
Byte datain[16]
Byte dataout[16]
Long stack[32]
```

The first line establishes a 16-byte array, **datain**, used to store incoming data from the PC application. The second line establishes a 16-byte array, **dataout**, used to store data we want to send to the PC application. The third line allocates 32 longs of stack space for the Spin **PC_Comms** method launched into another cog. The two 16-byte arrays are global variables, which means that they are accessible from any public or private method within the object, including the **PC_Comms** method launched into another cog.

Communication

The incoming and outgoing arrays hold 16 bytes each; the serial protocol only supports byte-sized values. Word- and long-sized data must be sent one byte at a time. Likewise, bit- and nibble-sized data must be encoded into a byte.

With packets of data going back and forth it is important to know where the beginning of the data array is, therefore a prefix is sent before the packet of data. This header will help delimit the data packet and ensure that the data bytes are placed into the array in the proper position.

Packets going from the Propeller to the PC will be prefixed with "!P1" as well as an ASCII 10 (linefeed), while packets going from the PC to the Propeller will be prefixed with "!VB". The ASCII 10 (linefeed) is required for the PC due to the way the VB.NET ReadLine works. The prefix from the PC application also includes an ASCII 10 (linefeed).

To develop a well-written and useful application, first determine which data to display within the PC application and what control data to send the Propeller object. Once this is done, decide which array elements, and how many, will be needed to create your application. Be sure to document which array elements are for which data in order to properly assign data to/from user variables in the Propeller object as well as controls in the PC application.

Propeller Communication

As mentioned above, the **PC_Comms** method launched into another cog actually uses the FullDuplexSerial.spin object written by Parallax to handle serial communication. This library object launches into a third cog and handles buffering of 16 bytes of data in each direction, bit timing, as well as other functions not used in this application. The sole purpose of the **PC_Comms** method is to receive a data packet from the PC application and respond with a data packet. The Propeller will only send a single packet after receiving one. This ensures that the incoming array is not being overwritten with null data and that the Propeller is not needlessly sending data if the PC application is not listening or running.

The Propeller application includes a **PUB Main** method, which first launches the **PC_Comms** method into another cog. Since this object is intended to be a template no error checking is done here; it assumes that there is an available cog for the **PC_Comms** method. After the **cognew** command the user should feel free to place their program ensuring only that the **PC_Comms** method stays intact.

The four lines in the QuickStartCommunicatorV1.0.spin object demonstrate how to easily activate I/O using the data from the **datain** array, confirm activation by setting the **dataout** array, and place data (such as key presses) into the **dataout** array.

The following line of code gets data from the **datain** array element zero, places it into **dataout** array element zero, and then writes it to P23 through P16 on the Propeller. On

the P8X32A QuickStart board[1] this effectively turns on LEDs on the board for each bit set in the received byte.

```
    outa[LED7..LED0] := dataout[0] := datain[0]
```

Copying the byte back into the **dataout** array provides a method of feedback to the PC application. Use this return data to update visual controls as a confirmation that the Propeller received the sent command.

The following line of code places the current state of the switches into the **dataout** array element one.

```
    dataout[1] := buttons.State
```

Note: if using active-low pushbuttons, such as those on the Propeller Professional Development Board[2], use the following line of code instead:

```
    dataout[1] := !ina[BTN7..BTN0]
```

PC Communication

PC programmers have many choices for PC development software including C/C++/C#, Delphi, Real BASIC and Visual BASIC to name a few. Visual Studio 2008 Express Edition and VB.NET were used to create this demo application. The 2010 Express edition of Visual BASIC and the QuickStart Communicator application are available at the links provided in the References section on page 5. The QuickStart Communicator application zip file contains a source project folder which would typically be placed in your 'My Documents' folder within the Visual Studio 2010 folder.

To customize the application only two sections need to be modified. First and foremost the main form was set up with an I/O Data group which contains all the controls specific to the demo application. To create your own application, remove these controls leaving the I/O Data group box empty, then place the controls you need in your application, resizing the form and I/O Data group box as necessary.

The only section in the code you should need to modify is the TextOut sub. You can see how to assign control values and data to the outgoing array as well as how to assign incoming data from the input array to various controls. This will be very similar for other form controls such as text boxes and TrackBar controls. Just remember that the values to and from the array are byte values and therefore are limited to 0-255 or a single ASCII character. Always select 38.4 kbps for this application since the timeout and delay values have been optimized for this data transfer rate. Other baud rates are listed for the sole purpose of making that section of the application flexible for other uses.

QuickStart Communicator Spin Template

```
CON

  _CLKMODE = XTAL1 + PLL16X                     ' Use crystal x 16
  _XINFREQ = 5_000_000                          ' 5 MHz crystal (system clock = 80 MHz)


CON

  RX_PIN  = 31                                  ' Propeller RX pin
  TX_PIN  = 30                                  ' Propeller TX pin
```

```
  SDA_PIN = 29                                  ' I2C SDA pin
  SCL_PIN = 28                                  ' I2C SCL pin

  LED7    = 23                                  ' Highest button on QuickStart Board
  LED0    = 16                                  ' Lowest button on QuickStart Board


CON

  PC_BAUD = 38_400                              ' Optimum Baud Rate for PC application
  MODE    = %0000                               ' Serial Port mode (See FullDuplexSerial.spin)


OBJ

  serial  : "FullDuplexSerial"                  ' Handles serial communication
  buttons : "Touch Buttons"                     ' Reads Touch Pads On QuickStart Board


VAR

  Byte datain[16]                               ' 16 byte array for incoming data
  Byte dataout[16]                              ' 16 byte array for outgoing data
  Long stack[32]                                ' Stack space reserved for Communicate method


PUB Main

  cognew(PC_Comms, @stack)                      ' Launch Communicate method into a new cog
  buttons.start(CLKFREQ / 100)                  ' Launch new cog for button detection
                                                ' (only used by QuickStart Board for Demo)

'' User code begins here

  dira[LED7..LED0] := %1111_1111                ' Set LED 0-7 I/O lines to output
                                                ' (required for QuickStart Communicator Demo)

  repeat                                        ' Loop indefinitely (req. for QuickStart Demo)
    outa[LED7..LED0] := dataout[0] := datain[0]     ' Activate LEDs and return confirmation
                                                    '  to VB app. (req. for QuickStart Demo)
    dataout[1] := buttons.State                     ' Encode touch switches into second byte
                                                    ' of array

PRI PC_Comms | iobyte, index                        ' Declare private method and 2 local vars

  serial.start(RX_PIN, TX_PIN, MODE, PC_BAUD)       ' Start UART (FullDuplexSerial) cog
  serial.rxflush                                    ' Flush receive buffer

  repeat                                        ' Loop indefinitely
    index := 0
    repeat
      iobyte := serial.rxtime(10)               ' Get character
      if iobyte == byte[@pcheader][index]       ' Header match?
        index += 1                              ' Yes, try next
      else
        index := 0                              ' No, restart
    until index == 4                            ' Header complete

    repeat index from 0 to 15                   ' Get 16-byte packet from PC
      iobyte := serial.rxtime(10)
      if iobyte => 0
        datain[index] := iobyte
      else
        quit                                    ' Abort on timeout
```

```
    serial.str(@myheader)                       ' Send header to PC
    repeat index from 0 to 15                   ' Send 16-byte packet to PC
      serial.tx(dataout[index])


DAT

pcheader                Byte    "!VB", 10
myheader                Byte    "!P1", 10, 0
```

Resources

The following zip archives are available for download from this application note's web page at www.parallaxsemiconductor.com/an018.

AN018 QuickStart Custom Visual BASIC Project folder
AN018 QuickStart Communicator Spin code archive

References

1. P8X32A QuickStart Board; part #40000; www.parallaxsemiconductor.com
2. Propeller Professional Development Board; Parallax part #32111, www.parallax.com
3. Microsoft Visual Studio 2010 Express: http://www.microsoft.com/express/Windows/

Revision History

Version 1.0: original document.