Hardware Interfacing with RobotBASIC

- The Fundamentals -

by

John Blankenship and Samuel Mishal

ISBN-13: 978-1438272726   ISBN-10: 1438272723

Contents at a Glance

Table Of Contents

Preface

Although RobotBASIC is best known for its integrated robot simulator, it also has extensive commands for interfacing with external hardware, making it very useful for many control applications in addition to robotics.

Based on the emails we receive, there are many people who want to experiment with controlling a robot and electronic hardware through a personal computer but don't have the necessary skills in electronics or programming. In this book we will show you that RobotBASIC and some readily available hardware, can make the above goal easily attainable.

Modern technology has greatly reduced the complexities of communicating with, and controlling external devices. Companies like Parallax, Lynxmotion, and Pololu now provide motor controllers and sensors that make it easy to accomplish projects that would have required an engineering degree only a short time ago. Even with all these advancements though, a proper understanding of the fundamental principles can help ensure your success.

This book concentrates on how RobotBASIC programs can interface with a wide variety of off-the-shelf hardware. Many options will be explored including both serial and parallel ports and how a separate microcontroller can be used to handle the low-level I/O operations. You will even see how vision, speech synthesis, voice recognition, and Internet communications can be used in your RobotBASIC programs.

The main portion of the book uses straightforward, less intimidating examples to introduce the concepts without being overwhelming. A series of appendices provide additional details for more intricate information.

Visit our web page to download your free copy of RobotBASIC.

www.RobotBASIC.com

Chapter 1

Introduction to I/O Ports

This chapter will introduce you to the various types of ports and some of the terminology associated with them. Later chapters will discuss the details of using these ports.

When you write PC programs that need to obtain sensory data from external devices and control electro-mechanical actuators such as motors, you need to understand what ports are and how to use them.

Input/Output ports can be subdivided into two general categories, serial and parallel.

Parallel Ports

Parallel ports generally make available 8 pins for either input or output or both. That means they can transfer 8 bits of I/O. Other sizes are also possible. The term *bit* originated from the phrase, ***b***inary dig***it***, which simply means that each piece of information can only be one of two values. These values are typically referred to as *zero* and *one*, *on* and *off*, or *low* and *high* where low and high generally correlate to voltage levels in an electronic circuit.

When standard TTL (Transistor-Transistor Logic) circuitry is used, a low usually means zero volts and a high is typically associated with five volts. Since a bit can only take on one of two possible values, there is a threshold voltage (typically around 3 volts, but that varies by the types of electronics being used) that marks the boundary between a logical high and low.

A group of 8 bits is often referred to as a *byte*, and two associated bytes (16 bits) are often called a *word*. There is even a term for a 4-bit grouping. It is called a *nibble*(half a byte).

When a computer program needs to transfer a group of bits to or from some external hardware, a parallel port is required. This term simply means that a group of bits are transferred simultaneously - that is, in parallel.

Serial Ports

Serial ports get their name because the bits of information are transferred between devices in a serial manner - that is one bit at a time. Serial ports come in many different types, as we shall see later.

Comparing Serial and Parallel

Imagine that we have eight ping-pong balls, each labeled with a 1 or 0. Assume that together they make up a byte of information that needs to be transferred from one device to another.

A parallel port is like gathering all eight balls together and throwing them to someone. If you throw the balls one at a time, it is similar to a serial port.

One advantage of a parallel port transfer is speed. It is obviously faster to transfer all the bits at once rather than sending them sequentially. Older serial ports were asynchronous and very slow compared to today's synchronous serial ports.

Synchronous vs. Asynchronous

The most common ***asynchronous*** serial transfer protocol is called RS-232. The actual data being transferred is sandwiched between a *start* and *stop* bit. The start bit simply gives the receiving hardware a wake-up call, indicating that the real data is coming. The start bit is necessary in asynchronous transfers because the data can be sent without warning (thus the term asynchronous). It is the responsibility of the receiving hardware to watch for the arrival of data. Timing is of paramount importance and it is measured in relation to the time of arrival of the start bit. The stop bit does not actually indicate the end of the data, but only serves as a small delay between the transfer of each group of 8 bits to ensure that the next start bit can be distinguished from the last bit of the 8 bits.

A ***synchronous*** transfer requires two wires instead of just one used for asynchronous transfers.

> Of course, both methods also require a ground wire to complete the electrical circuit.

On asynchronous transfers, the second wire carries a *clock* signal whose pulses indicate exactly when data is available on the first wire. This can make synchronous transfers much faster than asynchronous ones because this additional clocking line helps make the timing much more precise which helps in maintaining the transfer integrity and synchronization even at high clocking rates.

Modern Ports are Serial

Not long ago, all printers came with a parallel port interface operating through a 25-pin connector. Actually, the interface was composed of three separate parallel ports. One port carried the data (the codes for the letters to be printed) and a second port was used to send error information (paper out, printer off line, etc.) back to the computer. The third port was used to control the transfer, telling the printer when another piece of data was ready to be taken. RobotBASIC has commands that allow reading and writing to parallel ports, even though most computers today do not have them anymore.

Nearly all printers today communicate using a USB port, which is a synchronous serial port with well-defined standards. Because of its synchronous nature, USB ports are nearly as fast as parallel ports, but can be implemented less expensively.

Converting Serial to Parallel

When the 8 serial data bits arrive at their destination they must be reconstituted back into a byte so that it can be used in the computer (or microcontroller) as a single byte representing a number or character. Fortunately, most computers and microcontrollers provide either hardware or software modules that make it easy to send and receive serial data relieving you from having to understand all the low-level details.

Virtual Serial Ports

On the PC, the Windows OS allows a device plugged into a USB port to establish itself as a virtual serial port. Since many languages, including RobotBASIC, have commands to read and write to a serial port, this provides an easy way to communicate with many modern external devices.

Most modern PCs no longer have the original RS-232 serial ports just as they no longer have parallel ports. However, many microcontrollers and even some sensors communicate using the RS-232 standard, so robotic projects will often need a standard serial port. In order to provide compatibility, many companies provide a USB-based serial port.

I^{2}C Communication

USB is not the only synchronous interface. Many serial memory modules, electronic compasses, and other devices useful to robotic hobbyists use the I^{2}C synchronous interface which has the advantage of being able to communicate bi-directionally with only two wires (in addition to the ground wire).

As with the USB interface, it is often easy to find hardware or software modules that make it easy to utilize

I^{2}C communication without having to understand all the details of its internal low-level operation.

Virtual Parallel Ports

Just as many companies provide the necessary circuitry to create a virtual RS-232 serial port over a USB port, so there are also companies that provide circuitry to create a virtual parallel port using a USB port. One such company is USBmicro (www.USBmicro.com). Their U4x1 devices are not just parallel port replacements. They also provide many powerful features (see next chapter). RB has a suite of functions to control the U4x1 and utilize all its facilities.

Utilizing Microcontrollers for I/O

Since microcontrollers are designed for low-level control applications, they provide an easy way to add I/O operations to nearly any PC. The micro controller can be programmed to communicate with RobotBASIC using an RS-232 interface. This allows the controller to perform the actual Input/Output operations on its I/O pins as requested through serial commands sent by RB.

You might dislike the idea of having to program a micro controller to act as your I/O interface, especially since one of the major advantages of using RobotBASIC to control a robot is that you get to program in an easy-to-use high-level language. There is an important distinction though. The program in the controller only has to handle very basic operations, so it is easy to write, and, as you will see in later chapters, the low-level program generally only has to be written once. All of the major programming, that is all of the decision-making and intelligence, can be fully implemented in RobotBASIC.

Summary

This chapter provided an overview of various types of ports and some terminology associated with them. The next chapter will provide some practical examples of how to utilize parallel ports in your RobotBASIC programs.

Chapter 2

Parallel Port Examples

In this chapter, we will look at some practical examples of how a PC's parallel I/O ports can be used to send and receive data to and from external circuitry. Let's start with the most generic I/O commands in RobotBASIC, `InPort` and `OutPort`. These commands allow you to read and write to a valid I/O port on your PC. Because of that, these commands should be considered *dangerous*. Many PC devices (such as disk drives, video cards, sound cards, and interrupt controllers, just to name a few) use I/O ports to control their operation. Inadvertently writing improper data to the wrong ports can alter your PC's operation in strange ways and potentially damage it.

OutPort

The `OutPort` command sends data to some external circuitry and requires two parameters as shown below.

```
OutPort PortNumber, Data
```

The **`PortNumber`** is an address that specifies which of 65,536 ports to use (addresses 0-65,535). Most port addresses are not used, and of those that are, many should only be used by the operating system or other programs that know how to use them properly. Let's see how to use `OutPort` by looking at a specific example.

Older PC's had a special 25-pin connector for interfacing with printers. Nowadays, most computers use a USB port for connecting to a printer, so spending a lot of time on the older interface would not be fruitful. It does provide a simple introductory example though, and you may have an older computer that has a parallel printer interface. This example is also valid, because the techniques shown here can be used to exchange data with *any* parallel port, not just the printer port.

RobotBASIC can run on any version of the Windows OS from 95 to Vista and 7. If you have an older computer you can use it to do your experimentation. RB needs no installation and is easily usable from a thumb-drive or CD or even through the Internet. Using this possibility means you can make use of old PCs and you do not have to worry about damaging it.

As mentioned earlier, the old parallel port printer interface is actually composed of three different I/O ports; a data port, a control port, and a status port. The data port was an 8-bit output port used to send data to the printer. The control port was a 4-bit output port used to initialize the printer and to tell it when new data was available on the data port. The status port was a 5-bit input port that allowed the PC to receive information from the printer (indicating, for example, that the printer is busy or out of paper).

You can specify a hexadecimal number in RobotBASIC by using 0x. For example 0xA4B3 would mean the hexadecimal number A4B3, which is 42163 in decimal. To specify a binary number use 0%. So 0%1101 is the binary number 1101, which is 13 in decimal.

On most desktop PC's the location for these ports started at a base address of 0x378 (888 in decimal), although some PC's and laptops use a different address. The data port was located at the base address followed by the status port and then the control port. For this example we will only use the data port. Let's assume we have three LED's connected to the three least significant bits (LSB) of the port, as shown in Figure 2.1. The figure also shows the pin assignments for the 25-pin connectors used for parallel printer interfaces.

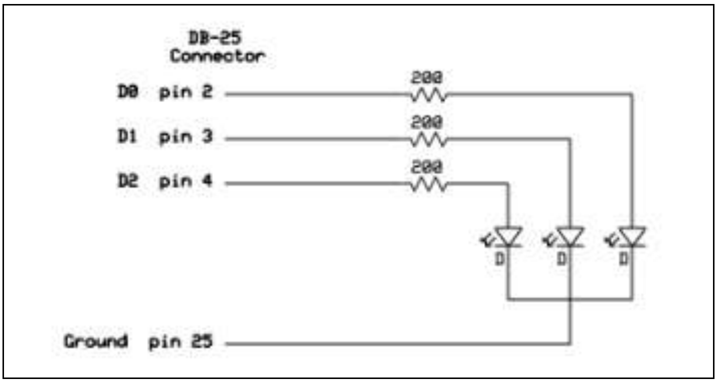

Figure 2.1: Three LEDs connected as shown allow us to see the data written to the port.

When a l is written to one of the port pins, that LED will light because it will have a voltage applied to its anode terminal (typically 3-4 volts) and ground to its cathode terminal. We can light all three LEDs by writing a 7 (binary 111) to the port. Let's see how to light the LED's from RobotBASIC. Look at the one-line program below.

```
OutPort 0x378, 1
```

If you run this program with the LEDs connected, the D0 LED will light. If you send a 2 (0%010) to the port, the D1 LED will light, and a 4 (0%100) will light D2. Remember, this program assumes the parallel port is at address 378. If your printer port is at some other address, or if you are using a different type of parallel port entirely, you must use the appropriate address.

The short program in Figure 2.2 will cause the D0 LED to blink at a one-hertz rate.

```
while true            // loop forever
  OutPort 0x378,1     // turn the LSB on
  delay 500           // wait ½ second
  OutPort 0x378,0     // turn all bits off
  delay 500           // wait ½ second
wend
```

Figure 2.2: This program blinks the LED on the LSBit.

We can make the light appear to move back and forth from one side to the other by sending the proper data to the port. In particular, the data we need to send to the ports is shown below. If these numbers are sent over and over, the lit LED will appear to move back and forth.

Decimal	Binary
1	001
2	010
4	100
2	010

The program below shows how to accomplish this in RB. You can adjust the delays to make the light move faster or slower.

```
while true
  OutPort 0x378, 0%001
  delay 200
  OutPort 0x378, 0%010
  delay 200
  OutPort 0x378, 0%100
  delay 200
  OutPort 0x378, 2
  delay 200
wend
```

Figure 2.3: This program causes a light to move back and forth on three LEDs.

InPort

Let’s see now how to input data from any PC port. The `InPort` function is similar to `OutPort` except that it returns the data from the designated port address. For our next example, we will assume the circuitry in Figure 2.4 is connected to the status port for the parallel printer interface.

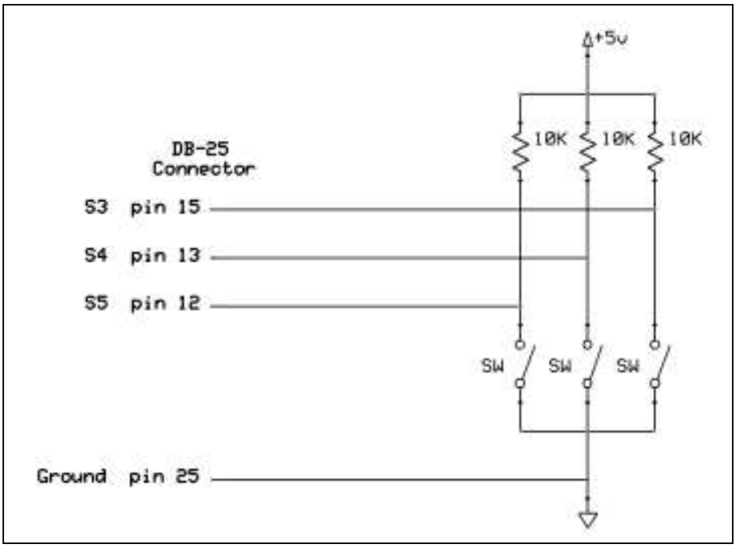

Figure 2.4: The status port can be used for input.

Each input pin is connected to a switch and a resistor. When the switch is closed, the input line is pulled to ground causing the input to read a zero. When the switch is open, the pull-up resistor will pull the line towards five volts, causing the input to read a logical one. This means that a normally-open switch will input a zero when pressed. If you prefer the opposite, you can use normally-closed switches or invert the data either with hardware or in software. It is worth mentioning, that since the printer port connector does not have the 5-volt supply available, that a separate 5-fvolt supply must be connected. Note also that the input data lines do not enter on the least significant bits of the port (S0 would be the LSBit). This means that the data must be shifted to the correct position after it has been obtained.

The program in Figure 2.5 obtains the data from the switches and displays it in the raw form as well as inverted and shifted. The program prevents flicker by updating the display only when a new number is obtained from the port by comparing the number obtained to the old number. Also notice how the bit-wise AND (`&` or `bAND`) is used to *clear* (or *mask out*) all the unwanted bits. This action only keeps bits marked by 1's in the number &ed with the data. Individual bits in data can be *inverted* by bit-wise exclusive-ORing them with a one. In this case, we want to invert the three least-significant bits, so we `bXor` (or you can use the @ operator) them with 7. If you are not familiar with bit-wise operations, refer to Appendix A.

```
d=0 \ oldData=d \Gosub DisplayData
while true
  InPort 0x379, d    // get data from port 0x379
                     // and place in d
  d = d & 0%00111000 // zero all unwanted bits
  if d<>oldData
    Gosub DisplayData
    oldData=d
  endif
wend
end
DisplayData:
  clearScr
  print "           Raw Data = ",d
  print "               binary ",bin(d,3)
  print
  print "       Shifted Data = ",d>>3
  print "               binary ",bin(d>>3,3)
  print
  print " Shifted & Inverted = ",(d>>3)bXor(7)
  print "               binary ",bin((d>>3)@7,3)
return
```

Figure 2.5: This program displays data from an input port in several forms.

Assuming you use normally open switches, then if all the switches are off (open) then each input pin will read a high or 1. Since these bits are coming in three bit positions to the left, if all bits are high, the number displayed in the raw mode would be 56 (0%000111000) and the output from the program would look like Figure 2.6.

Once the raw data is obtained from the port, it can be converted to a more desirable form by shifting it right three positions using the shift operator (>>) or inverted the desired bits by binary exclusive ORing them (`@`) with 1's.

```
            Raw Data = 56
                binary 111000

        Shifted Data = 7
                binary 111

Shifted and Inverted = 0
                binary 0
```

Figure 2.6: This is the display from the program in Figure 2.5 when all the switches are open.

The programs examined so far demonstrate the basic principles for dealing with I/O ports utilizing the legacy parallel printer port. If you are using an older PC, or if you have added a third party parallel port board to your PC, then these principles apply. Most modern personal computers though, communicate with external hardware with USB ports. RobotBASIC supports USB ports in multiple ways, one of which is supplying special commands for controlling products from USBmicro.

USBmicro I/O Boards

USBmicro (www.USBmicro.com) produces parallel port I/O boards that can be connected to a PC through the USB ports. The U401, shown in Figure 2.7, is fully supported by RobotBASIC with commands that communicate with a special DLL file supplied by USBmicro. We will explore some of the basic commands here, but refer to

RobotBASIC’s HELP file and to USBmicro’s web page documentation for complete information.

The U401 has 16 I/O lines that can be individually configured as inputs or outputs. The 16 lines are further configured as two 8-bit ports, A and B. The required USBmicro DLL (USBm.DLL) is provided when you download the RobotBASIC zip file for a full install.

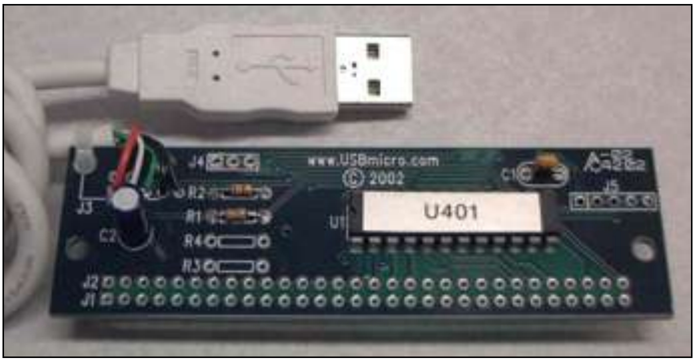

Figure 2.7: This U401 I/O board from USBmicro is fully supported by RobotBASIC.

The output pins on a U401 board have VERY limited current capabilities, so they should be buffered with some form of buffer/driver as shown in Figure 2.8. When a logical one is applied to the inverting buffer, it provides a low output that turns on the associated LED. The rest of the circuitry is similar to what we have seen before.

There are many special RobotBASIC commands for controlling the U401. The program in Figure 2.9 demonstrates a few commands that read the switch data from Port A and send it to the LEDs on Port B. When this program is running, anytime you change the switches, you will see the results on the LEDs. Notice how commands are used to verify that both the USBmicro DLL and I/O board have been found.

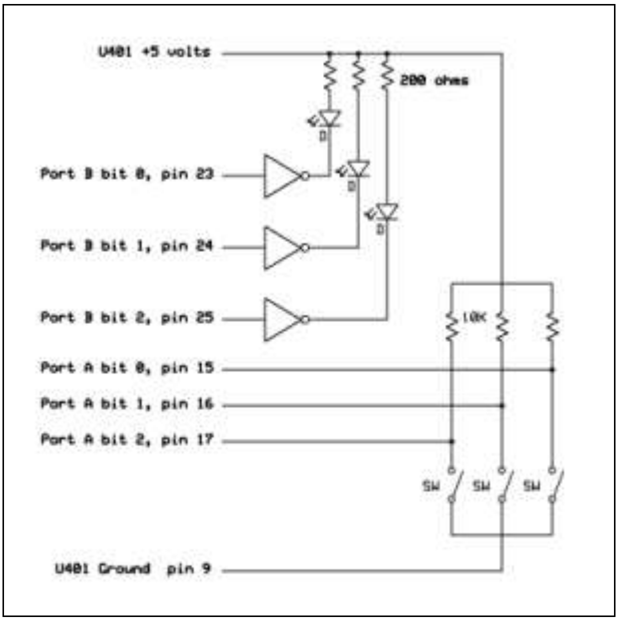

Figure 2.8: External devices are easily connected to the U401 from USBmicro.

```
if usbm_DllSpecs() != ""
  if usbm_FindDevices()
    //---there is a device and it will be device 0
    usbm_DirectionA(0,0,0) //set port A0 to A7
                           // as inputs
    usbm_DirectionB(0,0xFF,0xFF) //set port B0 to B7
                                 //as outputs
    while true
      n = usbm_ReadA(0) // assume device 0 for
                        //both read and write
      usbm_WriteB(0,n)
    wend
  else
    print "There are no Devices"
  endif
else
  print "The USBmicro DLL is not installed"
endif
```

Figure 2.9: This program reads switches and mirrors their status on LEDs.

RobotBASIC has many advantages, when compared to a micro controller, for handling control applications. Microcontrollers, no matter how powerful they are, usually have limitations when it comes to memory size, variable types, multi-dimensional arrays, and high-level language support. The program in Figure 2.10 demonstrates how the simple task of reading and writing port data can be enhanced by using a graphical interface. When the program is run, the LEDs of Figure 2.8 will reflect the state of the checkboxes on the screen and the state of the switches will show in binary on the screen. Some of the lines in our program listing are longer than the book’s page width. When this happens, we use the \ character to allow the code to continue on the next line. You may enter the line into RobotBASIC using the \ character, or you may enter it all on the same line. Either way is acceptable to RobotBASIC.

These examples show only a small portion of the power of the USBmicro I/O boards. Refer to the RobotBASIC HELP file and www.USBmicro.com for more information and additional examples.

```
if usbm_DllSpecs() != ""
  if usbm_FindDevices()
    //---there is a device and we will use device 0
    usbm_DirectionA(0,0,0)  // set port A0 to A7
                             //as inputs
    usbm_DirectionB(0,0xFF,0xFF) //set port B0 to B7
                             //as outputs
    xyText 10,10,"Set Output Port Status:" \
                  ,,20,fs_Bold|fs_Underlined
    for i=0 to 2
      addcheckbox ""+i,430+20*(2-i),20," "
    next
    xyText 90,70,"Input Port Status:" \
                 ,,20,fs_Bold|fs_Underlined
    rectangle 425,70,493,108
    while true
      for i=0 to 2
        if getcheckbox(""+i) // see if each box is
                             // checked
          n = usbm_SetBit(0,8+i) //add 8 for
                             //Port B bits
        else
          n= usbm_ResetBit(0,8+i)
        endif
      next
      // the following is done individually to
      // clarify each task
      n= usbm_ReadA(0)    // read the port
      n= n bXor 7         // invert lower three bits
      n= bin(n,3)         // convert to a binary
                          //     string
      n=right(n,3)        // keep only right
                          //     three bits
      // note: all the above can be done in one line
      //n = right(bin(usbm_ReadA(0)@ 7,3),3)
      xyText 437,75,n,,20,fs_Bold
    wend
  else
    print "There are no Devices"
  endif
else
  print "The USBmicro DLL is not installed"
endif
```

Figure 2.10: A Graphical Interface can enhance interaction with ports.

USB Alternatives

The USBmicro boards are one of the easiest ways of connecting to the real world though a USB interface. Moreover, these boards provide much more than just normal on/off I/O. The boards allow you to control stepper motors with ease and to carry out I^2C, SPI and 1-Wire synchronous serial communications and to control an LCD. The U451 also allows you to switch high-voltage-high-current relays (240V 10A). These properties can be quite useful in many projects.

In the next chapter we will see another method for interfacing with a USB port.