Bit-level Programming Techniques for I/O Port Access

April 2001, ver. 1.10 **Application Note 102**

Introduction

This application note contains basic programming techniques which can be used in a programming project involving I/O ports. When a digital I/O card is used to control multiple devices or interpret multiple inputs, it is often advantageous to deal with data on the bit level rather than the byte level. However, digital I/O cards almost always deal with data on the byte level, requiring an entire 8 bits of data to be read or written at once. This application note will demonstrate how to conveniently and efficiently deal with your digital I/O card's data on the bit level, while still using the byte level interface your card provides.

Concept

The key to interpreting individual bits of a digital input or output port is the bitwise 'AND' and 'OR' operators of your programming language. These operators are represented differently by various programming languages, with Basic using "And" and "Or", and C/C++ using the symbols "&"(ampersand) and "|" (pipe), respectively. The concept of these operators is actually quite simple. The bitwise AND works by comparing every bit of two numeric values and producing a resulting value based on those bit comparisons. Bit 0 of the first number is compared to Bit 0 of the second number, Bit 1 of the first number is then compared to Bit 1 of the second number, and so on through all the individual bits. If two compared bits both have a value of 1, then that bit in the resulting value will be given the value of 1. However, if either of the compared bits have a value of zero, that bit in the resulting value will be given the value of 0. In this way, if both the first number's bit is 1 AND the second number's bit is 1, the resulting bit will be 1. The OR operator works in a similar fashion, with the only difference being that if **either** of the compared bits are 1, the resulting bit will be 1. See the tables below for an example of each operator.

Bit:	7	6	5	4	3	2	1	0
1st number:	0	1	1	0	0	0	1	1
2nd number:	1	1	0	1	0	1	1	1
Result:	0	1	0	0	0	0	1	1

Table 1: AND operator

Bit:	7	6	5	4	3	2	1	0
1st number:	0	1	1	0	0	0	1	1
2nd number:	1	1	0	1	0	1	1	1
Result:	1	1	1	1	0	1	1	1

Table 2: OR operator

Decoding Inputs

Now let's use our knowledge of the 'AND' and 'OR' operators to decode an input value from our I/O card. Let's assume that we have a switch connected to the fourth pin of an input port on the I/O card. This would mean that the switch would control bit number 3 on our input value (bits are numbered 0,1,2,3...). Let's also assume that the switch is wired so that if the switch is pressed, bit 3 of the input value will have a value of 1, while if the switch is not pressed, bit 3 will be 0 (see Appnote 101 for examples of switch circuits). There will probably be several other devices controlling other bits of the input port, but since we are simply interested in detecting whether or not our switch is pressed, we need a way to "mask out" the other bits. We can do this by ANDing the input value that we read with the binary value of 00001000 (notice that bit 3 is the only bit that has a value of 1). Because all bits but bit 3 of this value are zero, bit 3 of the input value is the only bit that can possibly make our final result nonzero. The result of the AND operation will indicate whether or not the switch was pressed. If the result is zero, the switch was not pressed (bit 3 was zero), but if the result is **not** zero, the switch was pressed (bit 3 was 1). See the table representation below.

Input Value:	1	1	0	0	**1**	0	1	0
Mask Value:	0	0	0	0	**1**	0	0	0
Result:	0	0	0	0	**1**	0	0	0

Table 3: Switch ON (nonzero result)

Input Value:	1	0	1	0	**0**	1	1	0
Mask Value:	0	0	0	0	**1**	0	0	0
Result:	0	0	0	0	**0**	0	0	0

Table 4: Switch OFF (zero result)

Constructing Output Values

Often a program which makes use of an I/O card will be controlling multiple devices connected to different bits of the I/O card's output port. Different portions of the program often need to turn a particular device on or off, without knowing whether the other devices need to be on or off. By using the AND and OR operators, it becomes easy to switch the state of one device while leaving all others unchanged. In order to turn a device on, the current state of the output port (the value last written to the port) should be ORed with the value of the bit you would like to turn on. If you would like to turn on bit 3, the last value written to the output port should be ORed with the binary value of 00001000 (note that this is 2^3 (2 raised to the 3rd power). 2 raised to the number of the bit results in the proper value to use for the OR operation). The result of this OR operation should then be written to the output port. Similarly, if you want to turn bit 3 off, the last output value should be ANDed with 11110111. The result of the AND operation should then be written to the output port. As you can see, every time a device is turned on or off, the last value written to the output port is needed. There are basically two methods for obtaining this value. The most intuitive is to store the value in a variable (often works best to be global) every time a value is written to the port. However, a cleaner and more failproof method can be used if your I/O card supports port readback (the CRD155B supports the readback function). Readback simply allows you to retrieve from the I/O card the last value that was written to a particular output port.

Examples

A few examples of these techniques will be given using the BASIC syntax. Assume that any initialization of the card has already been performed.

Outputs:

```
Port = 400  'assume the output port is port 400
LastValue = 0  'turn all devices off initially

' now let's turn on the device connected to bit 5
LastValue = LastValue OR 2^5
OUT Port, LastValue

'now let's turn on bit 0
LastValue = LastValue OR 2^0
OUT Port, LastValue

'finally we'll turn bit 5 back off
LastValue = LastValue AND (255 - 2^5) 'assume we're using an 8-bit port, where 255
OUT Port, LastValue     '                                          is all bits on.
```

Inputs:

```
Port = 400 ' assume the input port is port 400
Value = INP(Port) ' read in the value from the port

'now test to see if bit 2 was on when the value was read
IF (Value AND 2^5) = 0 THEN
      PRINT "Bit 5 was OFF."
ELSE
      PRINT "Bit 5 was ON."
END IF

'now test to see if bit 7 was on when the value was read
IF (Value AND 2^7) THEN     'Note that the result of this test is opposite of the test
      PRINT "Bit 7 was ON."             '  above because I decided to leave out "= 0"
ELSE
```

```
      PRINT "Bit 7 was OFF."
END IF
```

Winford Engineering
4169 Four Mile Road
Bay City, MI 48706

Ph: 1-877-634-2673
FAX: 1-989-671-2941

E-mail: sales@winfordeng.com
Web: **www.winfordeng.com**