Interfacing the PC

Part 2

Plumbing the further mysteries of I/O address decoding, it turns out that the world is really controlled by jumper settings.

STEVE RIMMER

In the first installment of this series, we had a look at the basic port addressing scheme of the 8088 microprocessor that drives the IBM PC and all its lineal – and oftentimes illegitimate – descendants. If you're still a bit dazed by it all, the contents of this article probably won't help you very much. We're going to dig a bit further into the whole writhing mess.

Regrettably, without a workable I/O decoding system fluttering about your head, all that comes later will be grass in the wind.

The basic I/O decoding circuitry that we checked out last month was workable, but only in the most rudimentary sense. In common with North American television and most Tory politicians, it ignored the real world to the best of its abilities. Plugged into just about any PC, it would probably have found itself instantly at odds with something.

This month, we're going to make it a bit more likeable – something that's relatively easy to do for a card, as it turns out. All it takes is a soldering iron. This is more that you can say for a Tory politician... although there are certainly those who would be willing to try the same approach on one just to see what happens.

Rabbits and Other Jumpers

If you flip back to the last episode in this drama, you'll recall that we devised an I/O decoder which would raise a line if any port in the range of 300H through 31FH were to be addressed by the processor. The only serious sweat about this is that it was fixed in this range, and no power in the known universe was likely to talk it out of its little hole in the wall.

In the original specification for the IBM PC, this range of port addresses was reserved for designing prototype cards. However, it should be realized that the people who designed this system were genuinely a bit short sighted. They figured that 64 kilobytes of memory, for example, would do for most souls, with 128 kilobytes for the greedy ones. As such, the original port address allocations of the system have become a bit blurred over time as designers realized that most of what they wanted to do with the system hadn't been allowed for.

Some of the original port addresses of the IBM PC *have* remained, of course. Things like disk drives exist in all machines, and, as such, their ports are invariably spoken for. Other devices, like hard drives, are sufficiently common that one would not want to design a card which infringed on their real estate. Of the remaining port ranges, it's largely impossible to realistically carve out any specific range of addresses for a custom peripheral card and blindly assume that it won't already be in use by some other card.

This all being the case, it's pretty well essential that in designing cards for the PC one make the port address range a bit flexible, such that the person using the card can find a free space in the port allocation of her or her particular system and plug in our card. This, of course, is why there are so many jumpers and DIP switches on cards. Ours will be no exception, of course.

To get started with all this, let's have a look at the "official" port allocation strategy of the PC.

000-00FDMA chip
020-021Interrupt controller
040-043 Timer chip
060-063 Parallel port chip
080-083 DMA controller
0A0-0AFNMI mask
0C0-0CFReserved
0E0-0EFReserved
100-1FF Not for rent
200-20FJoystick port
210-217 Expansion box
220-24F Reserved
278-27F Reserved
2F0-2F7Reserved
2F8-2FFCOM2 serial port
300-31FPrototype card
320-32FHard drive
378-37FPrinter
380-38CSDLC communications
3A0-3A9Binary synchronous communications
3B0-3BFMonochrome card
3C0-3CFReserved
3D0-3DFColour card
3E0-3F7Floppy controller
3F8-3FFCOM1 serial port

In the official IBM view of the universe, port addresses above 3FFH can't

New improved I/O decoder... now with the miracle ingredient PortSwitch

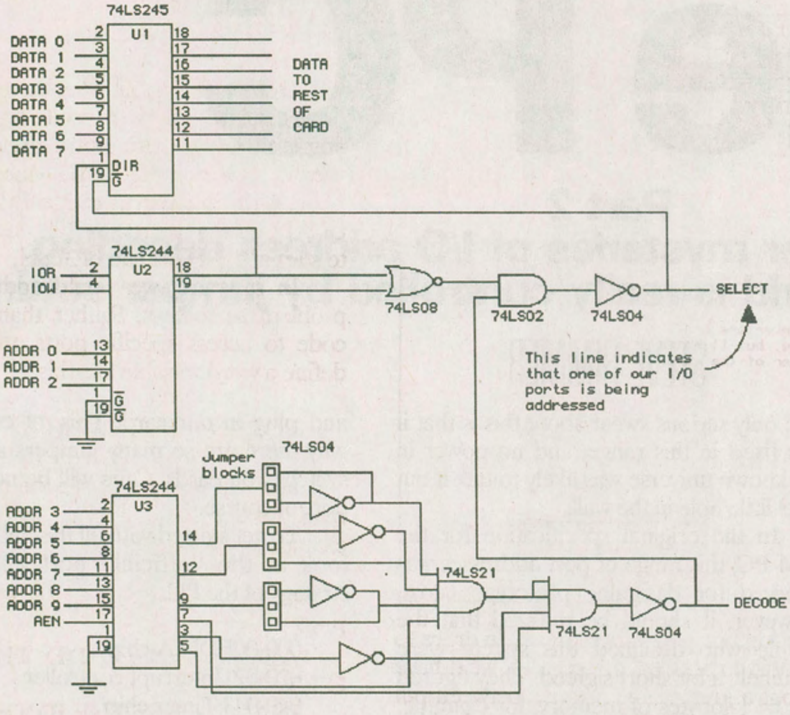


Figure one: The address decoder with modifications to allow for jumper selectable addresses.

be decoded because only address lines zero through nine are supposed to matter to the system when it's doing port I/O. In fact, as we've seen, one could build hardware to handle these extra addresses, although there are several good reasons why not to. The most notable of these is that most real IBM hardware... and clones thereof... do not look, for example, at address line ten. As such, if one attempts to address port 0400H, having designed a card which *can* decode this address, it will look like port 0200H to all the other cards in the system, that is, the ones which ignore line ten.

The low order port addresses, the ones below 200H, are better off left alone unless you specifically want to design a card that creates hardware interrupts. We will do this – but not just now. As such, the area we have to consider in creating this card is the one which ranges from 200H through 3FFH.

There are a number of devices in this list which may seem a bit unusual. For example, you may not have encountered an SDLC communications card. That's okay – they were never a real driving force in the PC universe. Likewise, if you were designing a local area network card, for example, you could probably assume that the sorts of machines that it would come to reside in would not have joystick ports in them.

If this seems a little funky... ya, well, it is. This is why it's important to make the port addressing of any cards we design for general distribution user selectable to some extent. It's impossible to know which bits of the unspoken for real estate other card designers will have snaffled, so you have to let the heads who use your cards decide for themselves.

You also have to find ways to explain to said heads how to ascertain what addresses *are* currently in use. Best of cosmic luck in this.

Even if you're just hacking copper for yourself, it's not a great deal of extra work to get this together, and it can save you some hassles later on, should you subsequently add more fiberglass to your system and find that some of it covets the address space you've set up for your home made interfaces.

Joy Sticks

In order to decode the address space from 300H through 31FH, as we got into last month, we have to see that address lines eight and nine are high and that lines five through seven are low. The lines zero through four constitute the low order part of the address, and don't matter at this stage of the decoding.

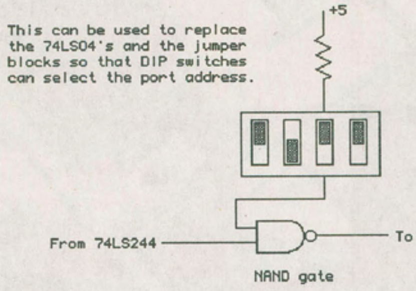
You can see the circuitry which does this in the schematic we used last month, or the one for this month, as nothing has really changed in this respect. The 74LS04 at the bottom of the diagram takes care of the lines five through seven. Lines eight and nine go directly to the second 74LS21 and gate. Thus, if all of lines five through seven are low and line nine is high, all of the inputs to the first quad AND gate will be high because of the 74LS04 inverters, and the output of the gate will be high. If line eight is high as well, all the inputs to the second quad AND gate will be high and its output will be high, indicating that we have decoded an address in the range we're interested in.

Clear as a Toronto sky in smog season, I know.

Now, in order to shift the address range around a little bit, let's consider jumpering out one of the inverters that connects the address lines five through seven to the first quad AND gate. Let's lose the one that handles line seven. This line decides whether the address range in question starts at 300H or 380H. As things are, it must be low for the address to fall in the area we're up for decoding. However, with the inverter removed, it would have to go high. As such, with this inverter gone our decoded address range for this card is suddenly 380H through 39FH.

Ah hah!

This is a particularly handy little port address, as it's where the aforementioned and seethingly mythical SDLC communications adapter is supposed to go. It's prime hunting ground for custom cards.



This can be used to replace the 74LS04's and the jumper blocks so that DIP switches can select the port address.

Figure two: using a NAND gate instead of inverters and jumpers. This takes a bit of extra hardware, but it makes the addresses easier to set for the end user of the card. The NAND gates replace the inverters

If we jumper the inverter in line six, we can switch the range up to 340H through 35FH, which the chart doesn't have anything much to tell us about.

How about jumpering both of them, you cry. Well, yes, this would give us address decoding in the range of 3C0H through 3DFH. The lower half of this range is marked as being reserved, whatever that means, and I'd certainly be willing to trample on it. The upper half, however, overlaps the port addresses for the 6845 CRTC chip that makes the colour card go. This is harmless if you have a Hercules card in your computer, but it could be a tad nasty otherwise.

Now, this range might be useful if we were designing a card which really only wanted to be able to use a maximum of sixteen ports. This is actually a fairly probably occurrence, in which case putting the base of the address range at 3C0H would be quite acceptable.

Jumpering out the three inverters on the card does, of course, only allow us to shift the addresses around in the range of 300H through 3E0H. It ignores the juicy looking space down at 0200H, where the aforementioned game controller and expansion box port ranges dwell. In order to be able to address these areas, we'd have to put a jumperable inverter between pin seven of the 74LS244 that buffers the high order address lines and the second quad AND gate. This would allow us to switch the base of the address range being decoded between 300H and 200H.

In designing actual hardware to allow for the selective jumpering of these inverters, we can set the whole ugly mess up with DIP switches and NAND gates, the preferable way, or we can use three pin header blocks, such that jumper caps fit over two of the three pins to decide how the connection is to be made. I've used jumper blocks in the schematic here to keep things reasonably easy to understand, but you can see how the NAND gate approach works, too.

It may seem that having a variable port address system like this would make writing the software that ultimately speaks to the ports in our card a bit of a nightmare. It doesn't. The 8088 allows for floating port ranges very nicely. It should be kept in mind that no matter where in the absolute address range of the machine the ports actually wind up, all the ports on our card will bear the same relationship to each other. For example, the 8250 serial chip that drives the COM ports on a PC has the following register port addresses when it lives in the address range for the primary serial port.

3F8HData buffer and LSB divisor
3F9HInterrupt enable MSB divisor
3FAHInterrupt identification
3FBHLine control
3FCHModem control
3FDHLine status
3FEHModem status

Don't worry about what these actually do. However, this is a pretty typical example of register use, and therefore port use, in a peripheral card. More to the point, if this serial port suddenly becomes COM2, all the addresses shift downwards to start at 2F8H instead of 3F8H.

In software, we would address this problem as follows. Rather than writing code to access specific ports, we would define a word variable like this

```
PORTBASEDW03F8H
```

and use it to deal with the ports. To read the line status port of this card, for example, we would

```
MOVDX,PORTBASE;GET THE PORT BASE
ADDDX,0005H;ADD THE OFFSET
INAL,DX;GET THE BYTE
```

The number five that we add to the base port address is the difference between the port base and the line status register, that is, five locations. No matter where the card turns up, this will remain constant. As such, if we move the card we need only adjust the value of PORTBASE accordingly.

Ports of Call

I think we've decoded the high order part of the port address question into groveling, helpless submission. It will not rise again. In the next part of this series, we'll look at those five lines that we keep saying don't actually matter. They do, in fact, just not all that much.

By the end of the next part of this engrossing narrative, you'll be able to narrow things down to a single port. It makes one stop and consider the incredible wonder of the universe in all its cosmic order, I know...

'Til then... ■