FEATURE

PC Hardware Interfacing Part 8

This month we're going to look at a paper design problem involving the 8250 serial port chip we looked at last month. Rather than simply seeing how it works, we're going to see how it can be *made* to work.

STEVE RIMMER

A whole book full of circuits isn't nearly as good as knowing how to design something. Looking at circuits which other people have gotten together will show you how the hardware in question operates... assuming you can read schematics and know what all the chips do... but it won't really teach you how to take a pile of loose chips, some specification sheets and a prototype board and make a working circuit out of them.

This is especially true when one is looking at hardware design for computers. Much of the confusion in this area stems from the observation that no two chip manufacturers use the same nomenclature for anything and, in fact, it is not at all uncommon for two chips from the *same* manufacturer to be described in different terms. As such, the "cookbook" electronics of computers really isn't... you can't just assemble the appropriate building blocks and go rock 'n roll. Rather, you must really understand the functionality of all the parts.

The upside of this is that, assuming you do get your head around what all the lines and signals are for, you can create paper designs that usually work. Most of the computer hardware I've prototyped has done what it was supposed to do as soon as it was powered up, barring a few solder bridges and bad parts. This is not because I'm unusually good at this stuff but, rather, because computer hardware design is really an exercise in logic. You can't just wire together black boxes and hope it'll all work, but you can wire together functional elements once you understand their functions.

Out of this obtuse bit of electronic zen, let's get down to a design problem of a manageable level of hugeness. We're going to design an interface between the 8250 serial port chip we looked at last month and the PC's peripheral bus. Now, this is a good design project because it's not very hard, happens to proceed logically without any mysterious secrets and... perhaps most important for this sort of illustration... has already been done. Once we have worked through the logic of the process we can peek into the IBM technical reference manual to see if our design will really work.

Ride That Bus

This month, let's see how the basic interfacing of the chip to the address and data buses takes place. This is a variation of the I/O decoder stuff we've looked at to date. Please note that this month's schematics are not complete... don't attempt to build anything just yet.

The 8250 requires a range of eight ports. By convention, the serial ports on a PC live in the ranges of 3F8H to 3FFH for the primary port and 2F8H to 2FFH for the secondary port. We're going to be designing a primary port here... for one thing because the decoding is so gloriously easy.

We know that we will have to select among the eight ports of the chip, even if we don't know what they actually do. As such, we will leave the three lowest order address lines out of the decoding problem. These can be connected directly to the A0 through A2 address lines of the bus, as, when our decoder says that the port address of the 8250 is being accessed by the processor, these lines will contain the port address of the actual internal register to be dealt with. Ignore this stuff for the moment.

The port range will be addressed by the processor's address lines A3 through A8, for a total of six lines. Because our port sits at the top of the PC's port range, if all these lines are high... and several other signals are as they should be, as you may recall from previous installments in this series... the PC is obviously trying to

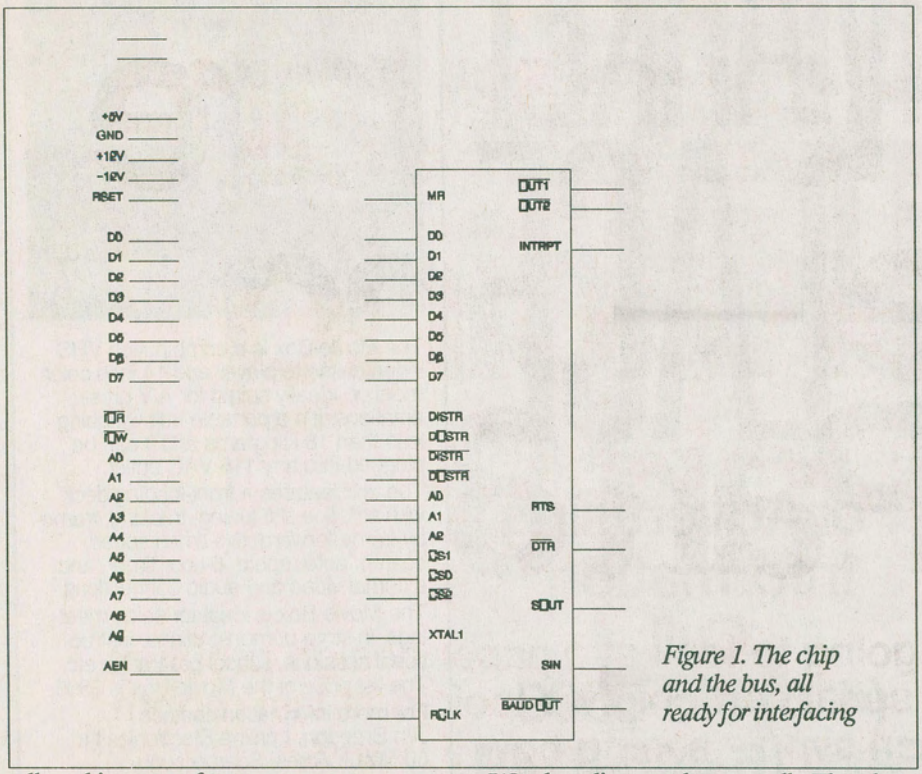

Figure 1. The chip and the bus, all ready for interfacing

talk to this range of ports.

We can simply run all of the little devils into a big NAND gate. If the output of the gate goes low, our 8250 is being addressed. This is, to be sure, I/O decoding with no pain at all.

The 8250 provides three chip select lines. We'll tie CS0 and CS1 high, as they're not going to be used. Remember, the NAND gate will go low when our chip is being addressed, so we want a select line that will be active when the voltage on it *drops*. In other words, we want $\overline{CS2}$, the inverse chip select.

When the 8250 is selected, it will communicate with the data bus, reading and writing data as is appropriate. When it is not selected, it appears as if it doesn't exist as far as the processor is concerned.

In fact, convention dictates that the data bus of the PC be separated from that of the 8250 by a 74LS245 bidirectional buffer. As such, we have not tied the 8250's data lines right to those of the PC, but have splashed one of these glue chips in between. We'll see more about this next month.

If the PC wanted to access port 3F8H of the 8250... that's the serial data input register, as it happens... it would put the number 3F8H on its data bus and jiggle the right lines for a port access. We can look at this as follows

Hex address 3F8

Binary address 001111111000

It's easy to see here why our simple I/O decoding works so well. The three zeros under the eight represent the three lowest order address lines, the ones which actually select our port. The six ones under the three and the F are the six lines which must go high if our port range is to be selected. The two zeros under the three are ignored lines.

If we wanted to make our card a secondary serial port, down at 2F8H, we would do so by changing one line. If this is the decoding for the secondary port range,

Hex address 2F8

Binary address 001011111000

it's clear that all we have to do to modify our I/O decoding for this range is to put an inverter between data line eight of the PC's bus and the big NAND gate. In a real world design, we could make this feature jumper selectable.

There are a few other lines to consider in this basic level of decoding, but we'll leave them until next month.

Byte Sized Chunks

In theory, the I/O decoding of the 8250 should work very much like that of the basic PC card we've looked at in this series. In practice, there are a few spaniels in the works. The port range is different, for one thing, although this has turned out to actually make the I/O decoding a bit easier. The more important aspect of this real world application is that it's no longer obvious which lines of the thing we'll be interfacing to connect to the lines of the PC's peripheral bus. Even in the little bit of interfacing we've seen here, it has been necessary to understand a bit of what is going on in the workings of the 8250.

Next month we'll finish off the basic address and data bus interface and touch on some of the really odd bits of the 8250. ■

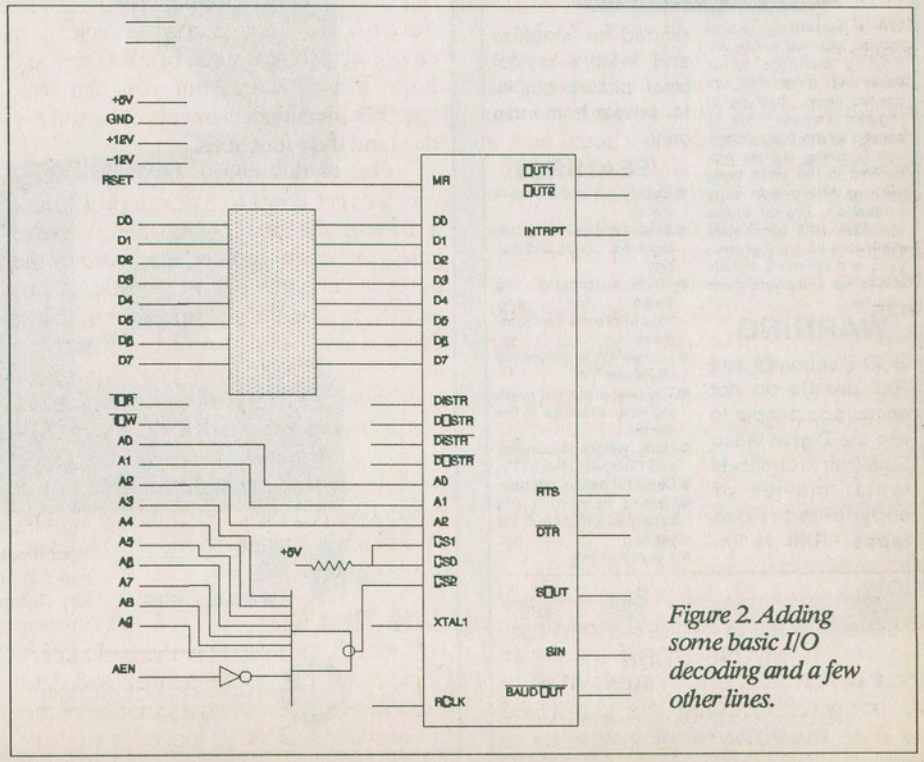

Figure 2. Adding some basic I/O decoding and a few other lines.