Mini Construction Project:

Improved 24-line I/O card for PCs

Here's an improved design for a flexible input/output card for ISA-bus personal computers. It provides a total of 24 digital lines, each of which can be programmed as either an input or output as required for interfacing to outboard input isolation circuitry or output relay or SCR/Triac drivers. It can be configured easily for any of the I/O base addresses allocated for 'prototyping cards', and the outboard connections are made easier by using three 10-way IDC headers positioned along the rear edge.

by JAMES BARKER

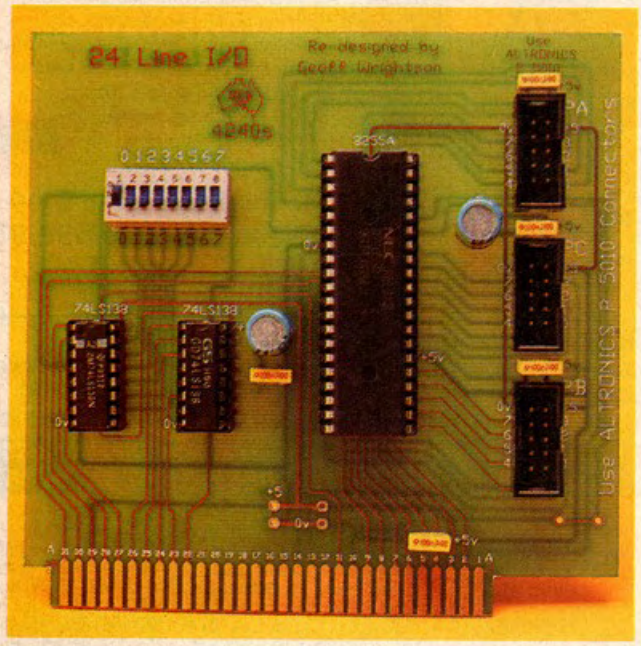
Back in its June 1989 edition, *EA*'s former stablemate *ETI* published the design for a low cost, easy to build 24-line I/O card for PC's, presented by designer Graham Dicker. Christened the ETI 1623, it was a very handy card for anyone who needed to use a PC for monitoring and control of other equipment, and my friend Bob Barnes of Sydney-based PCB maker RCS Radio tells me that his firm alone has sold a large number of boards for it — so it was clearly a very popular project.

Very few designs meet everyone's needs, though, and recently I was made aware that the original design had a couple of small but sometimes frustrating limitations. One was that its off-board connections were all brought out to a single 26-way DIL IDC pin header along the *top* edge, which meant that the ribbon cable to the 'outside world' had to be taken through one of those clumsy 90° turns to get out of the computer — and then split up into separate bits, if you wanted hook it up to separate input and output driver boards.

The other limitation was that the original card was configured for an I/O base address of 279H — that nominally allocated to the PC's games port. By cutting tracks on the PCB and fitting new links it could be moved to various other addresses between 201H and 280H, and no doubt this provided a satisfactory range of choices in 1989. However nowadays with many PCs having sound cards and/or games port cards fitted as standard, it's often more convenient to have an I/O card located in a different part of the PC's 'I/O space' — and hopefully reconfigurable more easily, say with a jumper strip or DIP switch.

A couple of months ago, a friend of mine — Geoff Wrightson, of Cardiff Heights in NSW — decided that the time had come to try coming up with an improved version of the 1989 design, to remedy these limitations. Geoff himself worked out the modified I/O address decoding circuitry to allow the card to be repositioned in the area set aside for 'prototyping cards' (300 - 31FH), with either a jumper header or DIP switch for convenience, and then I was roped in to help produce a new PCB design.

The new card you see in the photo is



The new card is easily configured for any of eight I/O base addresses in the 'prototyping card' range, and provides its 24 I/O lines in three groups of eight along the rear edge.

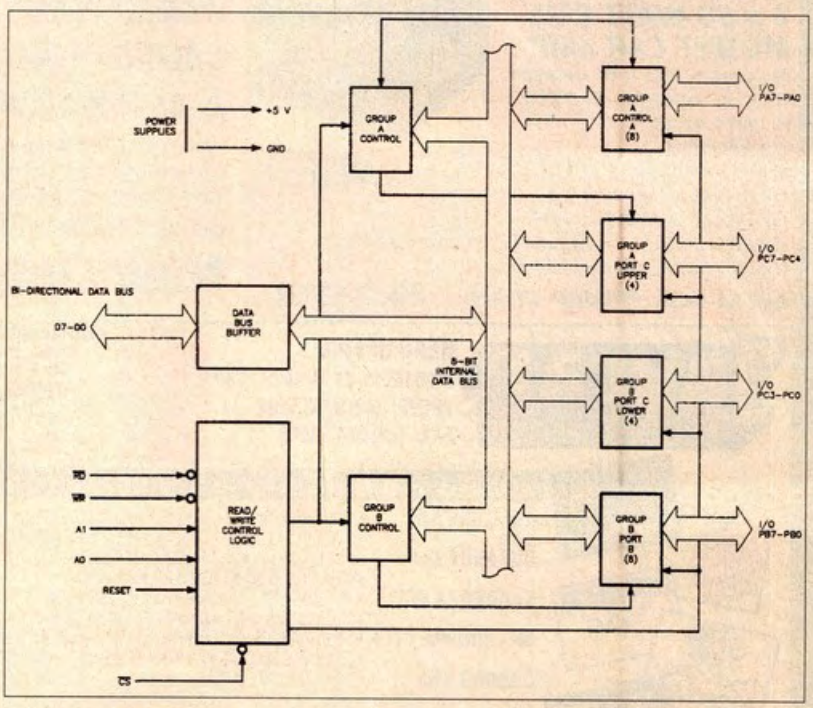
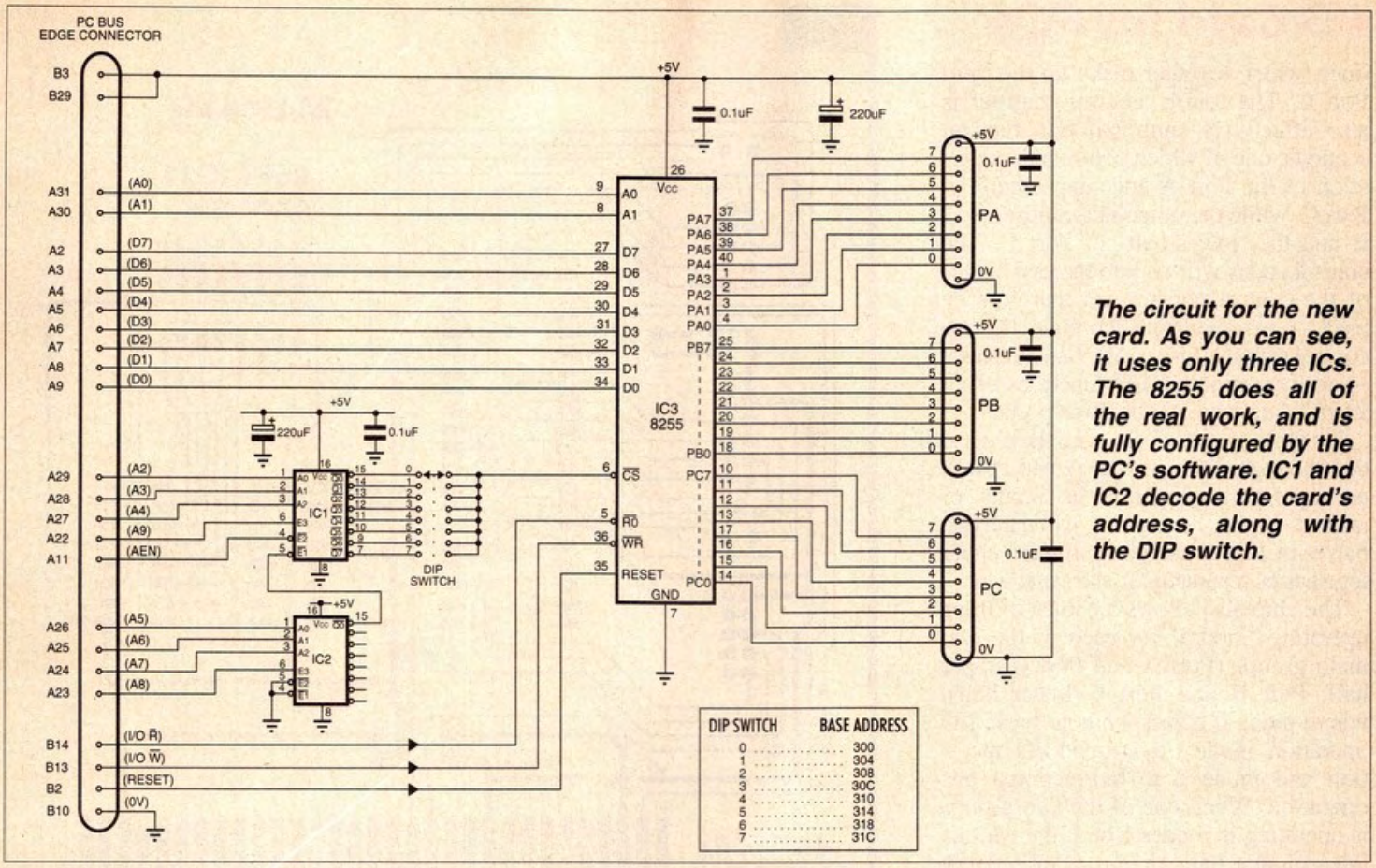


Fig.2: Taken from Intel's handbook, a block diagram showing the internal structure of the 8255 I/O controller.



The circuit for the new card. As you can see, it uses only three ICs. The 8255 does all of the real work, and is fully configured by the PC's software. IC1 and IC2 decode the card's address, along with the DIP switch.

what we've produced, and we believe it overcomes the original limitations fairly neatly. It's a little larger than the original, but still quite small enough to fit inside pretty well any standard PC. And since the off-board connections are brought out to three 10-way 'protected' header strips along the board's rear edge, the ribbon cables can be run straight out through the usual rear PC slot without any 90° bends, and connected easily to different driver modules.

Each of the 10-way headers makes available a set of eight I/O lines, together with connections to the PC's +5V and ground rails — which can be used to power low-drain input conditioning or output driving circuitry.

Circuit description

The circuitry on the card itself is very similar to the original design, with only three ICs: a pair of 74LS138 decoders and an 8255A programmable peripheral interface (PPI) IC, as made by Intel, NEC and other firms. However Geoff Wrightson's modified address decoding now allows the card to be located at any of the eight possible I/O base addresses in the range 300 - 31CH, simply by turning on one of eight DIP switches in SW1.

As you can see from the schematic, IC2 decodes PC bus address lines A5 - A7, and is also gated on when A8 is high. The Q0-bar output of IC2 is then used to gate on IC1, along with A9 and the address enable (AEN) line. When IC1 is enabled, it then decodes address lines A2 - A4, and as a result each of its eight outputs goes low only for a particular group of four I/O addresses, each with its lowest or 'base' address as shown in the table. Closing the appropriate DIP switch in SW1 therefore allows the 8255 PPI chip IC3 to be enabled only when the CPU specifies that group of addresses.

Base Address + 3 →	CONTROL REGISTER
Base Address + 2 →	DATA PORT C
Base Address + 1 →	DATA PORT B
Base Address →	DATA PORT A

Fig.1: The registers inside the 8255 chip occupy the base address and the three addresses above it.

This decoding scheme is very convenient as the 8255 has four internal eight-bit registers — three data registers, and a control/status register. And the chip's internal addressing allows any one of these registers to be 'connected' to its eight data I/O lines (D0 - D7), simply by manipulating the logic levels at its pins 9 and 8. So by connecting the PC bus address lines A0 and A1 to these pins, the chip's internal registers are automatically located at the four I/O addresses starting at the base address selected by IC2, IC1 and SW1 (Fig.1).

As you can see, PC control lines I/O Read-bar, I/O Write-bar and Reset are also connected to the appropriate control pins of U1, allowing the CPU to exercise full control over data flow to and from the chip, when it is addressed. It's a very simple and straightforward arrangement as far as hardware is concerned, but still allows the software to manipulate IC3 as needed.

All the real work is done inside the 8255 chip, of course. This is a very flexible chip, which was specifically designed by Intel to implement flexible parallel interfaces for microprocessor systems. The internal structure of the chip is quite complex, as shown in Fig.2 — taken from the Intel publication *Microcomputer Components Handbook, Microprocessors and Peripherals*, Volume 2. I won't go into all the details of its operation here, and if you want more details I suggest you either refer to the Intel book or else the original article in the June 1989 issue of *ETI*.

Expressed simply, though, two of the three internal data registers are each allocated to handling one of two groups of eight I/O lines, labelled Port A and Port B. The remaining data register is effectively split into two groups of four I/O

lines, which together make up the third Port C. The fourth 'control' register is also effectively split into two four-bit sections, one of which controls the operation of the Port A and 'upper half' of Port C, while the other half controls Port B and the 'lower half' of Port C. The control codes written into the two halves of the control register can therefore be used — together with the main PC bus control signals Rd-bar, Wr-bar and Reset — to determine the I/O functions of all 24 I/O data pins, on the three ports.

It's fairly simple to program the chip so that the eight lines of Port A and Port B are configured together as inputs or latched outputs, as required, while the two halves of Port C can be configured either together or separately, in the same way.

The chip also allows a choice of three operating 'modes' for each of the two main groups (Port A and Port C upper half, Port B and Port C lower half), where mode 0 corresponds to basic I/O operation, mode 1 to strobed I/O operation and mode 2 to bidirectional bus operation. When one of the two groups is operating in modes 1 or 2, the various lines of that half of Port C effectively become control lines for the eight data lines of Port A or Port B.

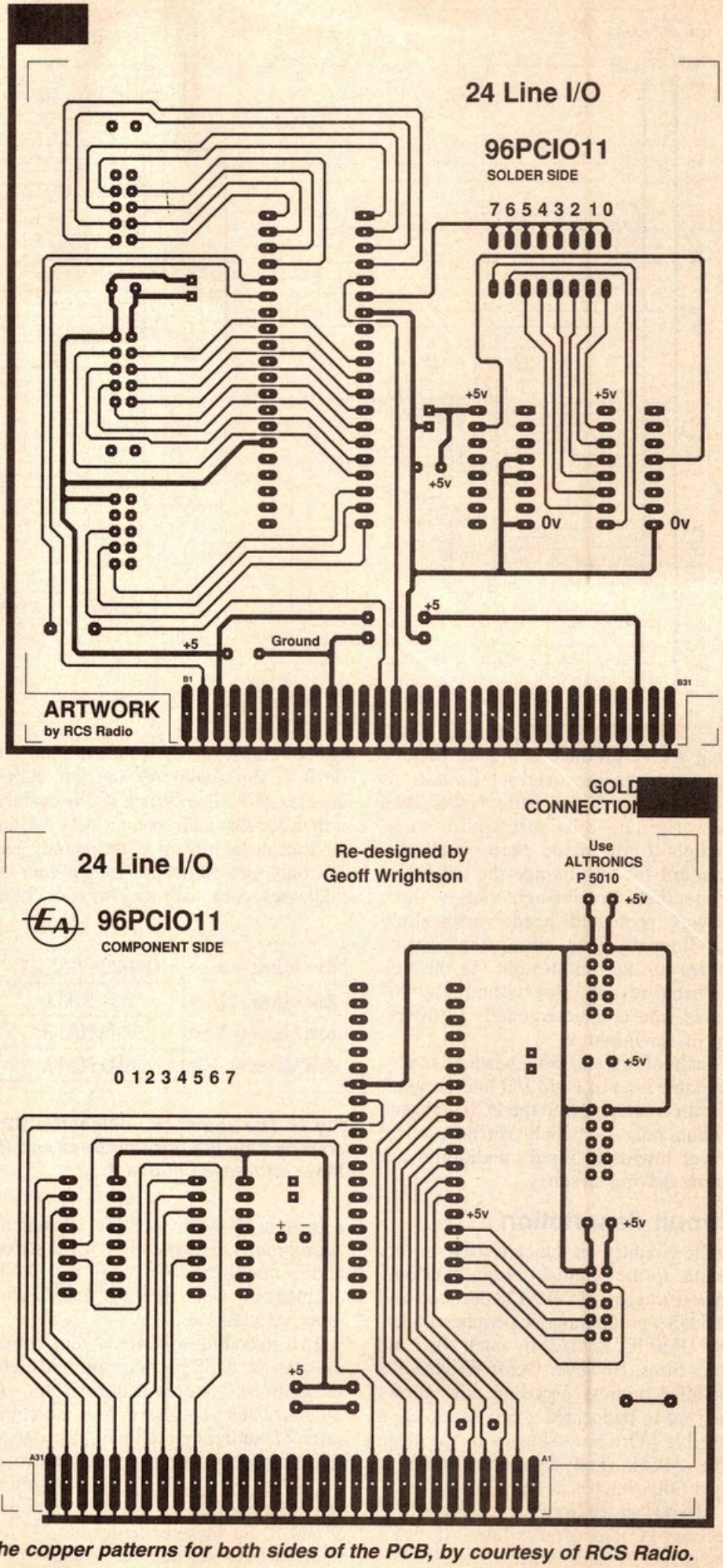
It all sounds rather complicated, but the bottom line is that the operation of the card's 24 I/O lines can actually be controlled quite easily by the CPU and software, simply by writing data to and reading it from the four registers in the 8255. This can be done using OUT and INP commands in BASIC, for example, or the equivalent commands in lower level languages like C or assembly language. We'll look briefly at this shortly.

Construction

There's very little involved in building the new card, because it uses such a small number of components. The main thing is to use a good quality double-sided PC board, with gold plated edge connectors for good reliability. The prototype boards were made for us by Bob Barnes of RCS Radio, who can now supply them from stock as Cat. No. 4240S, for $41.80 plus postage of $4 within Australia.

Wiring up the PCB should be very straightforward, as there are only the three ICs, an eight-way DIP switch, the three guarded 10-way IDC sockets and a few bypass capacitors. It should all be quite clear from the photo and overlay diagram.

Note that while the two smaller and lower-cost ICs are best soldered directly to the board, I recommend that you use a socket for the more expensive 8255.



The copper patterns for both sides of the PCB, by courtesy of RCS Radio.

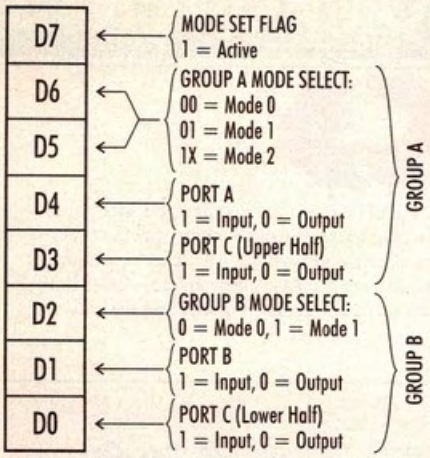
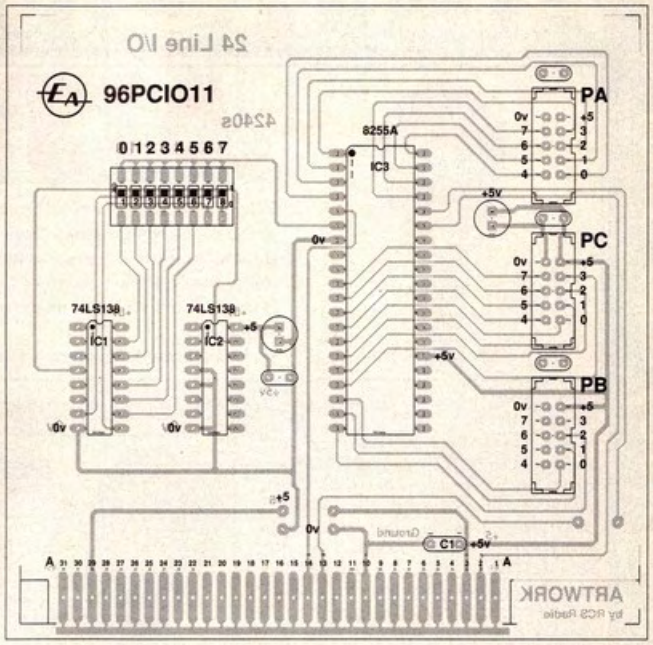


Fig.3 (left): The 8255 is configured by software, using the bits loaded into its control register.

At right is the PCB overlay diagram, to help you in placing the components. Note the orientation of the ICs and I/O headers.



Parts List

1	Double sided PCB, 106 x 110mm (RCS Cat. No. 4240s)
2	74LS138 decoder ICs
1	8255 PPI chip
1	40pin DIL socket
3	10-way guarded IDC socket (Altronics P-5010 or similar)
1	8-way DIP switch
5	0.1uF MKT capacitor
2	220uF 10VW RB electrolytic

You'll need a 40-pin DIL socket, and it's a good idea to get a good quality socket to prevent problems down the track.

By the way if you make your own PCB, don't forget to remove the copper strip along the bottom shorting the edge connector pads, before you try plugging it into the PC. As you're probably aware, the strip is only used when the pads are being gold plated.

When your board is wired up, the only other thing to do as far as the hardware is concerned is to set the DIP switch for the I/O base address you want. Only one of the eight switches should be switched on, and in most cases it will probably be OK to set SW0 on — giving a base address of 300H. This shouldn't clash with any of the other cards in most PCs, and in most cases the only time you'd want to set it for another address is if you already have another similar card, already set to 300H.

Programming it

As mentioned, virtually every aspect of the card's operation is controlled by the CPU and its software. I'm not going to describe this in detail, just give you a sample or two to get you started.

The 8255 chip's operating mode 0 is the simplest, and is sufficient for many basic parallel I/O jobs. There are actually 16 different possible combinations of input and output functions for the three main ports, determined by the values of bits D0 and D1 (Group B) and D3 and D4 (Group A) in the byte written to the 8255's control register (Fig.3).

As an example, say you want to configure the 8255 to operate in mode 0, and so that it provides 12 input and 12 output lines — say with the eight lines of Port A all used as outputs, those of Port B all used as inputs, and those of Port C split with the lower four as inputs and the upper four as outputs. To do this you'd need to set the various bits of the control register as follows:

D0 = 1 (Port C lower = inputs)
D1 = 1 (Port B = inputs)
D2 = 0 (Group B = mode 0)
D3 = 0 (Port C upper = outputs)
D4 = 0 (Port A = outputs)
D5 = 0 (Group A = mode 0)
D6 = 0 (Group A = mode 0)
D7 = 1 (Mode set flag active)

And this combination of bits corresponds to a byte of 83H, so all you'd need to do at the start of your program is send this value to the 8255's control register — located at the card's base address plus three. You could do this in Quick BASIC or Visual BASIC by the statement:

OUT BaseAddress + 3, &H83

where 'BaseAddress' would be your variable representing the card's actual base address — say 300H, or whatever you've set the DIP switch for. You'd have already initialised the variable at the start of your program, with a statement like:

BaseAddress = &H300

Once you've set up the 8255 for the input/output configuration you want, and also set up the BaseAddress variable, it's then an equally simple matter to dump data to the various output lines, and read it back from the input lines. For example in Quick BASIC or Visual BASIC, you just use OUT and INP statements. So to set all of the odd-numbered output lines of Port A, say, you'd simply send a byte of value &HAA (10101010) to the 8255 Port A register, at the card's base address:

OUT BaseAddress, &HAA

Similarly you can read in the values on the input lines of Port B, by grabbing the data from its data register:

InputVar = INP(BaseAddress + 1)

where the variable 'InputVar' will end up with a (decimal) value equal to the data byte read back from the Port B register.

Starting to get the idea? With the same configuration, you'd set all four upper bits of Port C to a 1 by sending F0H to the Port C data register, thus:

OUT BaseAddress + 2, &HF0

And if you wanted to read the four lower input lines on Port C, you'd use:

InputVar = INP(BaseAddress + 2) AND 15

where the 'AND 15' masks off the four high-order bits, to leave the lower four.

Of course you can set up the 8255 for any other configuration of input and output lines, and any other operating mode, simply by sending a different control byte to the control register at (BaseAddress + 3). For all of the various possibilities you'll have to refer to the 8255 data sheets.

Needless to say, you can hook up the card's I/O lines to a wide range of input conditioning and output driver circuits. Some basic circuits were given in the original *ETI* article, while others were given in the *EA* article of August 1991.

I hope you find this improved I/O card as handy as I've done. My thanks to Geoff Wrightson for his help in developing it, and also to Bob Barnes of RCS Radio. ❖