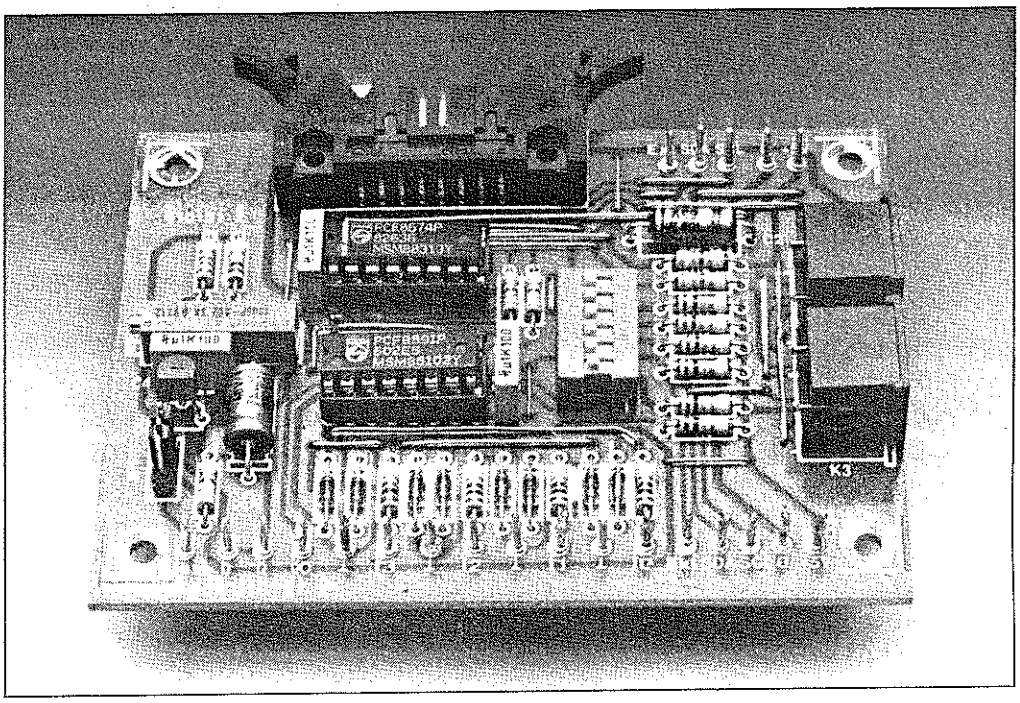
ADC/DAC AND I/O FOR I²C BUS



This article discusses a compact interface that allows PC users to communicate with I²C compatible ICs and circuits. The interface comprises an I/O port and a combined 8-bit analogue-to-digital and digital-to-analogue converter. Also, system software is described that brings life to the I²C PC insertion card described last month.

Design by J. Ruffell

As promised last month, this article tackles the software necessary to control the I²C interface for PCs (Ref. 1). This device driver is basically an extension of the disk operating system (DOS) implemented on the PC, and contains all the routines necessary to write and read I²C codes to and from ICs connected to any I²C bus system. The device driver has been written to comply with the protocols drawn up by Philips for the I²C bus.

Device drivers are used at several levels in a PC. Examples of device drivers include 'software handles' for the screen, the printer, the RAM disk, and the keyboard, to mention but a few. There are basically two types of device driver: block drivers and character drivers. Block drivers are used for media such as disk drives, while character drivers are used for the screen, the keyboard and, in this case, the I²C bus. Any device driver is an extension of the DOS, and is invariably called via the DOS. According to the DOS specification, a device driver can contain up to 17 routines (i.e., not all of these need to be implemented). They are:

0* Driver initialisation
1 Media check
2 Build BIOS parameter block
3 I/O control read
4* Read
5 Non-destructive read
6 Input status
7 Erase input buffers
8* Write
9* Write and verify
10 Output status
11 Erase output buffers
12* I/O control write
13* Open device
14* Close device
15 Removable media
16* Output until busy

The routines marked with an asterisk are implemented in the present I²C device driver, which is written in machine language, and available on a diskette (along with the source file) supplied through our Readers Services. A full description of the operation of the device driver is beyond the scope of this article, and readers interested in the programming aspects are advised to print the source file for close analysis. Further information on device drivers for PCs, and machine code programming, may be found in the many books and other publications that have been written on these subjects.

MAIN SPECIFICATIONS

- **Controlled via I²C bus**
- **4 analogue inputs (256 steps)**
- **1 analogue output (256 steps)**
- **8 I/O lines (bidirectional)**
- **Up to 8 boards on one I²C bus**
- **Adjustable ADC/DAC reference voltage**
- **Complete with MSDOS compatible device driver**
- **Source code available in assembler, Pascal and C**

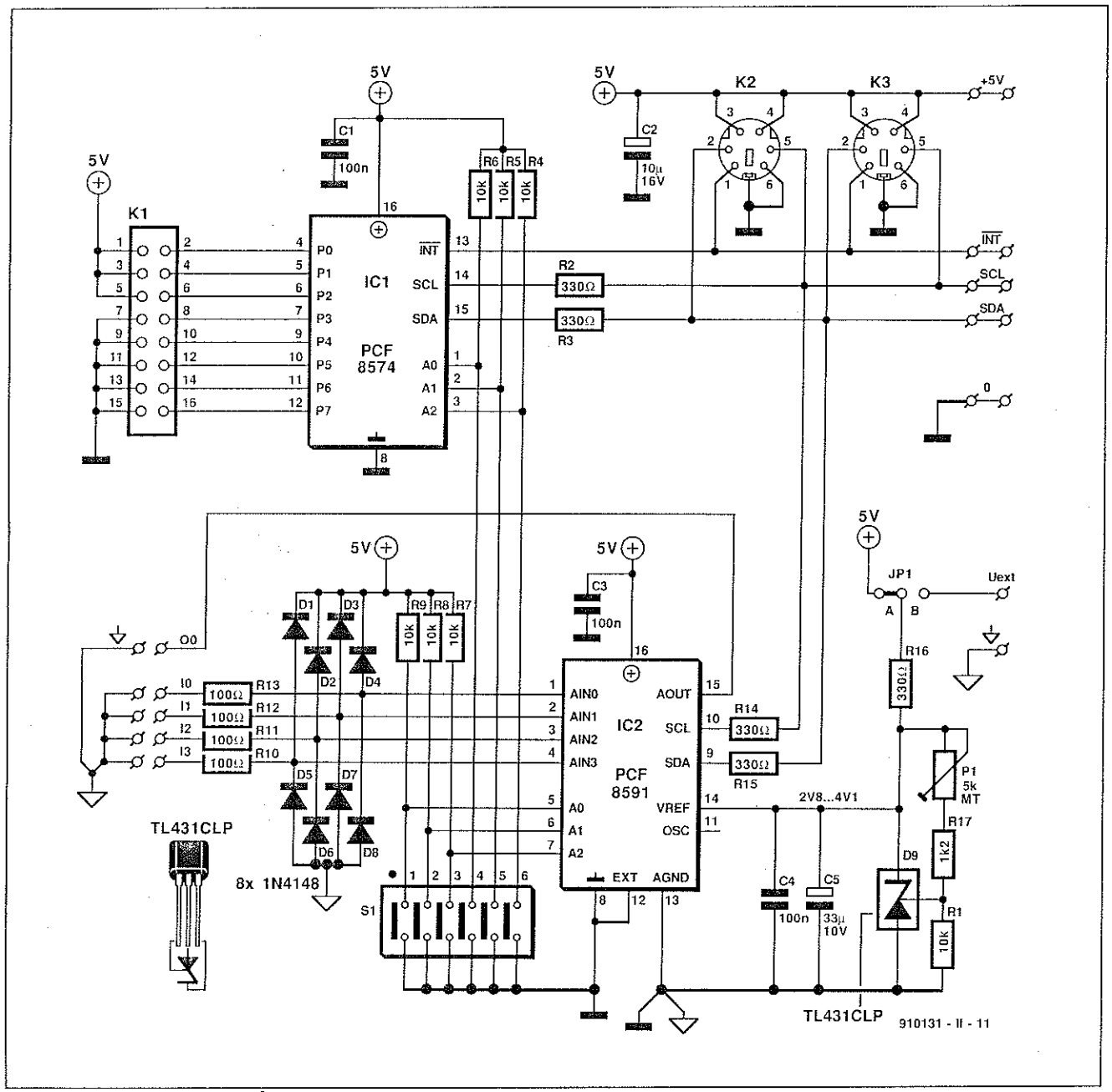


Fig. 1. Circuit diagram of the I²C extension card, which contains an I/O port and an 8-bit ADC/DAC.

Installation

The device driver disk contains the assembled file I2CDRIV.SYS, which may be placed in the root directory of the PC. Next, the CONFIG.SYS file has to modified by adding the line

device = I2CDRIV.SYS

You may type two parameters after 'I2CDRIV.SYS': B:xxxx and/or C:y, where

xxx is the base address of the I²C insertion card. This address is set to a value between 300_H and $3FE_H$ with the aid of DIP switches.

y is a code that selects the clock frequency, SCL, used on the I²C bus. The available clock frequencies deviate slightly from the those mentioned in the datasheets, because a clock of 7.16 MHz is used instead of the more usual 8 MHz. Parameter y can take the following values:

0: f_{SCL} = 81 kHz
1: f_{SCL} = 40 kHz
2: f_{SCL} = 9.8 kHz
3: f_{SCL} = 1.3 kHz

The default values for parameters B and C are 300_H and 9.8 kHz respectively.

When the PC is switched on or reset, it reads the new CONFIG.SYS file, and from then on recognizes all routines that support the I²C interface. That is when the real work can begin.

Useful for your own software experiments, the example programs on the diskette illustrate the use of the I²C driver routines in assembler as well as in the higher programming languages C and Pascal.

Hardware

The circuit diagram of the ADC/DAC and I/O card for the I²C bus is shown in Fig. 1. The main components in the circuit are the PCF8574 I/O port and the PCF8591 ADC/DAC. These I²C compatible building blocks prove that interface circuits with I²C control can be kept very simple indeed. The 8-bit I/O port is simplicity itself. Its eight I/O lines may be linked to external digital devices via connector K1. Address lines A0, A1 and A2 are connected to the positive supply line via pull-up resistors. Three switches in DIP switch block S1 are used to set the programmable part of the I/O address of the IC. The DIP switch allows up to eight PCF8574s

to be used simultaneously via the I²C bus.

As with all I²C devices, the addresses are partly fixed in the ICs. The two ICs on the present card are addressed as follows:

PCF8574: 0100 A2 A1 A0 R/$\overline{W}$
PCF8591: 1001 A2 A1 A0 R/$\overline{W}$

In both cases, the first four bits cannot be changed by the user. The next three bits can be set on the DIP switches, and the last bit selects between reading and writing of data. Read operations are enabled when R/$\overline{W}$ is '1', write operations when R/$\overline{W}$ is '0'. As regards the device driver routines found on the diskette, it is assumed that all DIP switches are closed, which means that the I/O port and the ADC/DAC are located at the address pairs 40_H-41_H and 90_H-91_H respectively. If other address pairs are set on the switches, the software requires to be changed accordingly. Given that the example files have a copious amount of comment, this should not cause problems.

The quasi-bidirectional I/O port Type PCF8574, of which the block diagram is shown in Fig. 2, has only one read/write register. Depending on the application, this device allows its output lines to be used as input lines. The output lines have a current sink and source specification of 25 mA and 0.4 mA respectively. If a port line is to be used as an input, it is first made logic '1'. Next, the level of the 'output line' is read back to see if it is still at '1'. If not, it is apparently pulled low (i.e., held at '0') by an external device. Thus, the low level supplied by an external device to the port line overrides the previously programmed '1', and is so recognized by the software. The open-drain outputs allow this to be done with impunity.

The circuit around the PCF8591 ADC/DAC is far more complex than that around the I/O IC. The analogue inputs of the ADC/DAC are protected against overvoltages by resistor-diode combinations R10-R13 and D1-D8. Here, too, the three address inputs are connected to DIP switches that enable the variable part of the address to be set by the user. The external voltage reference is set up around precision zener diode D9, a TLC431CLP. Resistors R1, R16, R17 and P1 are used to set a reference voltage between 2.8 V and 4.1 V. The user may set the value in this range required for the desired A-D/D-A step size. One step corresponds to $U_{ref}/256$. Capacitors C4 and C5 serve to suppress noise on the reference voltage. Jumper JP1 allows the ADC/DAC to be fed with an external reference voltage, which may be useful in certain cases when there is a danger of accurate measurements being spoilt by noise on the 5-V supply. For most applications, however, an external reference will not be required.

Figure 3 shows the block diagram of the I²C compatible ADC/DAC. Because it has many more possibilities, the PCF8591 is more complex to control than the PCF8574. Apart from data bytes, the converter IC requires a control byte to determine a number of settings as shown in Fig. 4. The highest nibble in the control byte determines the configuration of the analogue inputs (either two differential inputs, or four ordinary inputs), and in addition switches the analogue output on and off. The low nibble selects one of four A-D inputs, and may be used to enable the auto-increment flag.

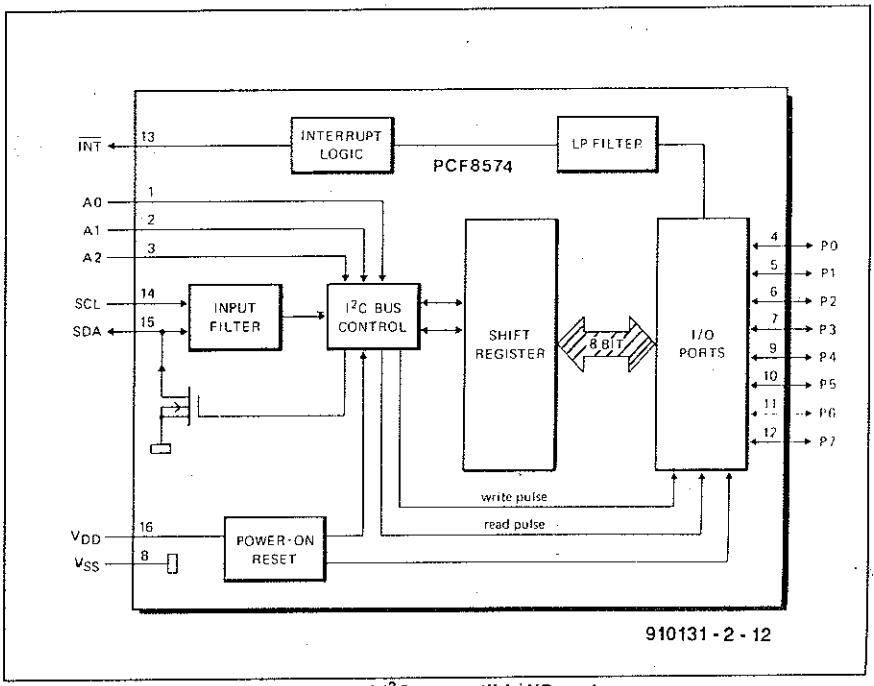


Fig. 2. Block diagram of the PCF8574 I²C compatible I/O port.

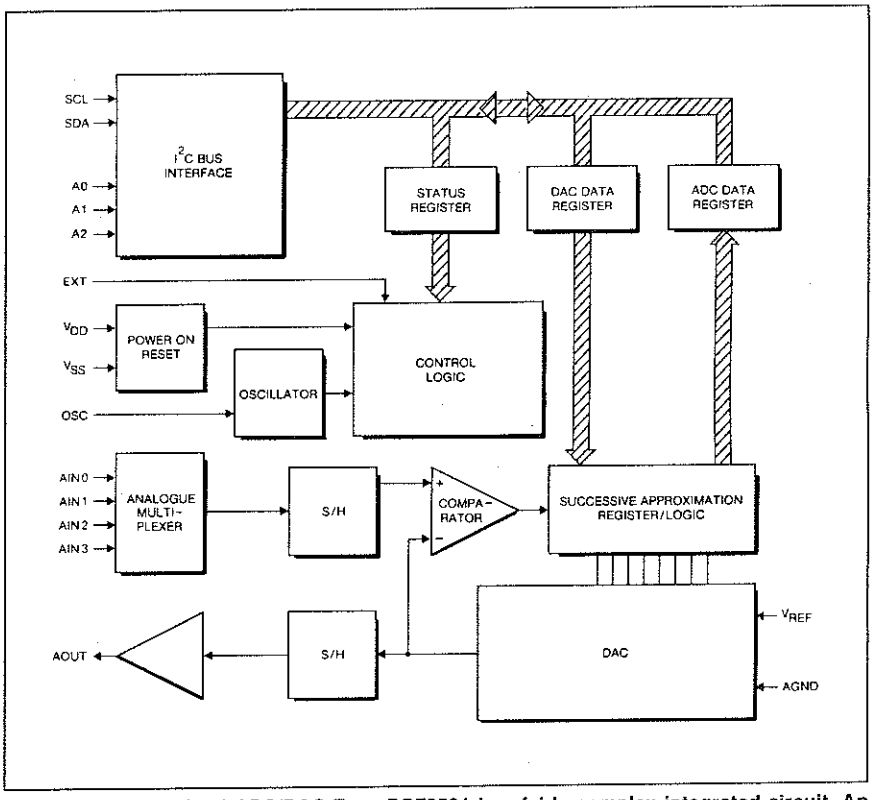


Fig. 3. The combined ADC/DAC Type PCF8591 is a fairly complex integrated circuit. An external voltage reference allows the conversion step size of the ADC and the DAC to be set as required.

The third byte, sent to the IC after the address byte and the control byte, is stored in the DAC register. Next, the previously stored value is converted into an analogue voltage that appears at output of the DAC. The output voltage increment equals $U_{ref}/256$. This means that a value of '00' results in 0 V at the output, and '255' in an output voltage of $255 \times U_{ref}/256$.

The reading back of ADC output values is performed in a slightly different manner. An A-D conversion cycle is started on the positive-going edge of the acknowledge pulse, which is returned to the master device

after the converter has been set to 'read' mode with the aid of a read command. The IC performs another A-to-D conversion cycle while it sends the data resulting from the previous conversion. At the start of the conversion, the voltage level at the selected input is sampled and subsequently converted into an 8-bit binary code. Input voltages supplied by a differential input are converted into an 8-bit two's complement code. The result is stored in the data register of the ADC, from which it can be transmitted. When the auto-increment flag is actuated, the next input is selected. In this manner, all inputs are selected in succession.

Construction

The ADC/DAC and I/O extension is easy to build on the printed circuit board of which the copper side layout and the component mounting plan are shown in Fig. 6. The 6-way miniature DIN-style connectors enable the extension card to be readily connected to the I²C interface in the PC. In principle, only one of the two mini-DIN sockets needs to be fitted on the board. The second socket is required only if further I²C boards are to be connected to form a chain. If a number of I²C extensions are fitted into a common enclosure, there is, of course, no objection against omitting the connectors, and using permanent wiring instead. The +5-V, ground, SCL, SDA and INT lines of the units are then connected from board to board.

The pinning of connector K1 is such that it can be linked direct to the 'Measurement amplifier' described last month (Ref. 2). All that is required to implement computer control on this amplifier is a short length of flat cable to link it to the ADC on the present board. By studying the source code of the test program 'ADIO', you will notice that the combination of the ADC and the measurement amplifier is readily turned into an autoranging measurement system.

The reference voltage is set to the required value by adjusting preset P1 and measuring the voltage across C5 with a digital multimeter. Since the program 'ADIO' on the diskette is based on a reference voltage of 4.0 V, it is advisable to set this value initially. Later, other values may be chosen, provided the relevant statements in the program are changed accordingly.

That completes the construction and adjustment of the I²C extension card, which is then ready to be tested. Testing is done in a 'hands-on' way with the aid of a well-documented test program, of which a Turbo Pascal and a C version is available on the diskette. Both versions of the test program cycle through a number of routines, including one that reads the levels at I/O port lines b4 to b7, and copies these to outputs b0 to b3. To run the test, force port lines b4 to b7 logic high with the aid of 10-kΩ pull-up resistors. Connect push-buttons that switch to ground to the same lines. Connect four LEDs between the b0 to b3 output lines and +5 V via 330-Ω series resistors. Run the test program, and check that one of the LEDs lights when the

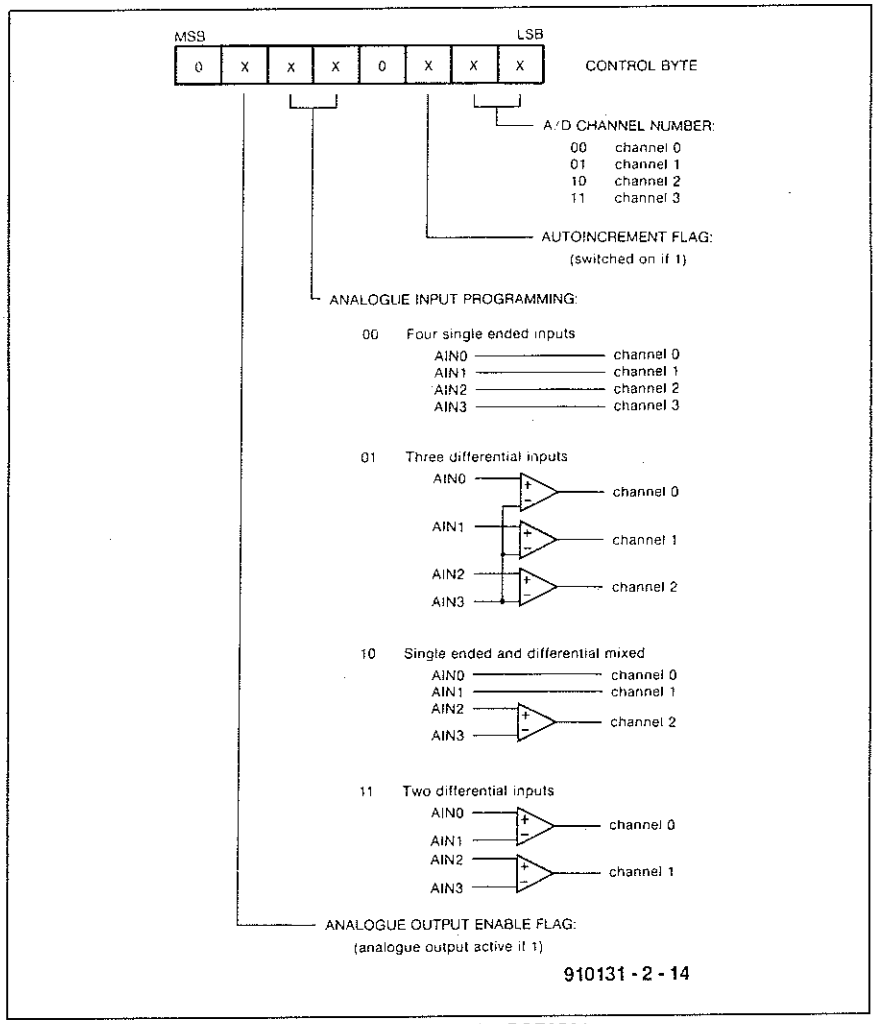


Fig. 4. Bit functions in the control byte sent to the PCF8591.

```
begin (* TestADDA *)
  Ctrl:=GetControlByte(1);          {-Load control byte with option# 1}
  Address(ADA_Addr);                {-Because the R/W_ bit (= LSB ADA_Addr)
                                     equals zero, the PCF8591 enters the
                                     write-mode. Therefore, the next trans-
                                     mitted byte is interpreted as a
                                     control byte.}
  write(bus,Ctrl);                  {-Transmit control byte.}
  with AD do                        {-The next bytes sent to the PCF8591
                                     would be stored in the DAC register.
                                     But at this point, we switch to
                                     read-mode...}
    read(Bus,Dummy,Data[0],Data[1],Data[2],Data[3]); {-I2CDRIV.SYS now
                                     generates a repeated start (same
                                     address, but with R/W_ = 1) and reads
                                     five AD-conversion bytes from the
                                     PCF8591. This is done by using the
                                     channel auto-increment function of
                                     the chip. The first read byte (Dummy)
                                     is the convertion result code of the
                                     previous cycle! We are not interested
                                     in that sample, so it is thrown away.}

  write(bus,Ctrl,AD.Data[Chan3]);   {-I2CDRIV.SYS generates a repeated
                                     start condition; same address, but
                                     with R/W_ = 0. Thus, the PCF8591 is
                                     in write-mode again and expects a
                                     control byte and one or more data-
                                     bytes. All databytes are stored in
                                     the DAC-register, but the analogue
                                     output voltage is always calculated
                                     from the previous DAC-register
                                     contents.}

  for Channel:=Chan0 to Chan3 do
    ShowVoltage(Channel,AD.Data[Channel]);
  ShowVoltage(4,AD.Data[Chan3]);
end;  (* TestADDA *)
```

Fig. 5. Extract from the I²C device driver listing. This source code is written in Turbo Pascal.

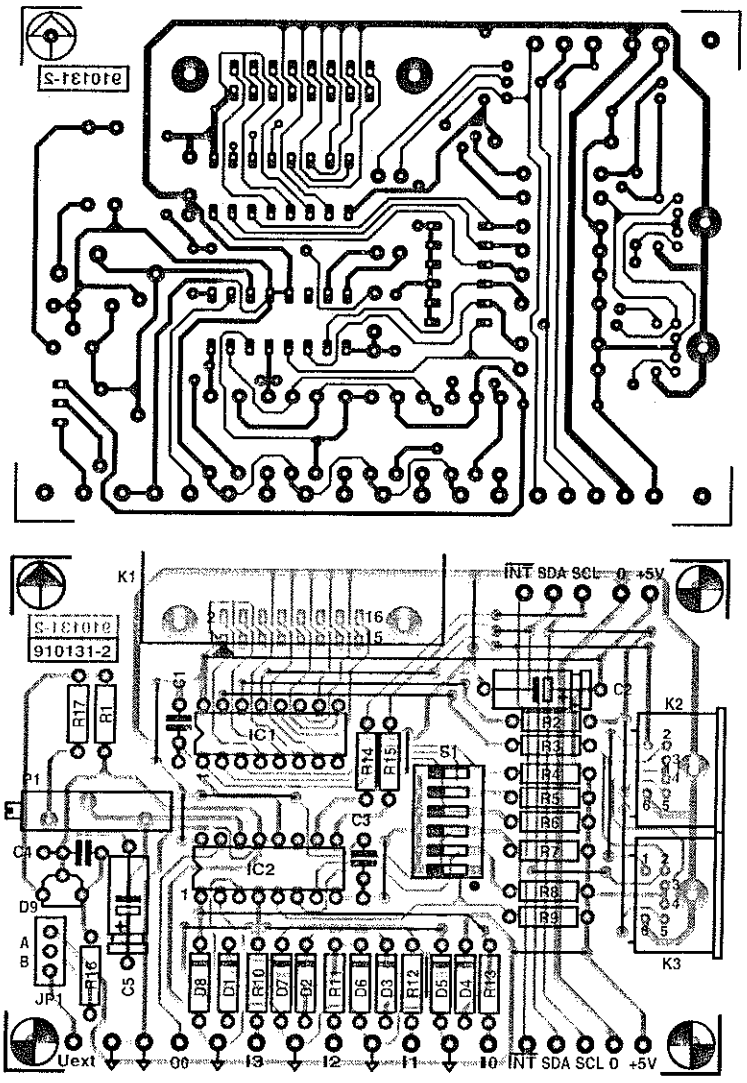


Fig. 6. Track side layout and component mounting plan of the PCB designed for the I²C extension card. Six-way mini-DIN connectors are used to connect I²C boards to the I²C interface fitted in the PC.

COMPONENTS List

Resistors:

7	10kΩ	R1;R4-R9
5	330Ω	R2;R3;R14; R15;R16
4	100Ω	R10;R13
1	1kΩ2	R17
1	5kΩ multiturn preset	P1

Capacitors:

3	100nF	C1;C3;C4
1	10µF 16V	C2
1	33µF 10V	C5

Semiconductors:

8	1N4148	D1-D8
1	TL431CLP*	D9
1	PCF8574*	IC1
1	PCF8591*	IC2

Miscellaneous:

1	16-way header, angled, with side latches	K1
2	6-way mini-DIN socket for PCB mounting	K2;K3
2	6-way mini-DIN plug	
2m (approx.)	6-wire cable	
1	6-way DIP switch	S1
1	Printed circuit board	910131-2
1	Control software on disk (MSDOS)	ESS1671

* Suggested supplier: C-I Electronics, P.O. Box 22089, 6360 AB Nuth, Holland. Fax: +31 45 241877.

corresponding push-button is pressed.

The ADC/DAC is tested similarly. The program reads the voltage levels at the analogue inputs I0, I1, I2 and I3, and puts the level of I3 on output O0.

If the circuit passes the above tests, it is ready for use with your own applications. ■

References:

1. "I²C interface for PCs", *Elektor Electronics* January 1992.
2. "Measurement amplifier", *Elektor Electronics* January 1992.

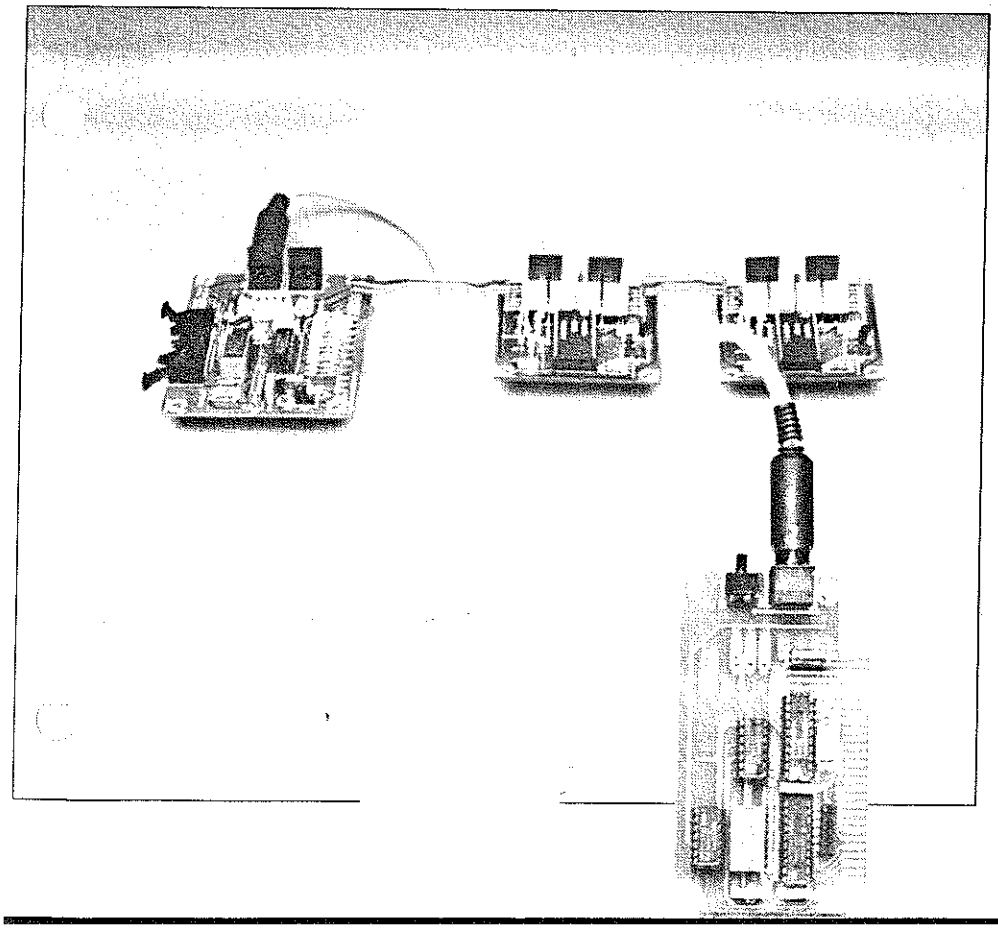


Fig. 7. This demonstration set-up shows how different modules can work with a single interface. The display driver shown will be discussed in a future publication.