A-to-D via the COM port

Interfacing via the pc's COM port and accessible using simple Basic routines, David Gains' a-to-d converter can log up to four differential sensors with fast sampling.

My requirements were for a data acquisition system that would acquire analogue signals from transducers such as temperature sensing integrated circuits. The system had to use my PC's unused COM2 port, and it had to be capable of sampling a single channel at, say, 1-2kHz, or a few channels on demand at infrequent intervals. Further, to accommodate signals of different voltage levels, and to improve resolution, each channel had to have software programmable gain.

The resulting design, **Fig. 1**, provides the following features:

- four differential input channels,
- four gains of 1, 2, 4, and 8,
- single-conversion, or free-running conversion mode, and
- single-channel, or scanning channel mode.

All of the above features are software programmable. In addition, the capture module can be configured for either unipolar or bipolar input signals.

Table 1. Receiver output bits from the uart are used to configure the a-to-d conversion circuitry.

RBR_8 Unused	RBR_7 Unused	RBR_6 Conversion	RBR_5 Scan	$RBR_{4,3}$ Gain	$RBR_{2,1}$ Channel
x	x	0=single	0=off	0,0=×1	0,0=channel 1
x	x	1=running	1=on	0,1=×2	0,1=channel 2
x	x			1,0=×4	1,0=channel 3
x	x			1,1=×8	1,1=channel 4

Table 2. Three bits configure the uart serial data framing as follows.

CLS_1	CLS_2	SBS	Format bits data	stop
0	0	0	5	1
0	0	1	5	1.5
1	0	0	6	1
1	0	1	6	2
0	1	0	7	1
0	1	1	7	2
1	1	0	8	1
1	1	1	8	2

Table 3. Programmable oscillator – applicable bit-rate settings.

Input S_3	S_2	S_1	Frequency kHz	Data rate bps
1	1	1	76.8	4800
1	1	0	153.6	9600
1	0	1	307.2	19200

Serial interface

The *MAX232* line driver-receiver, IC_1, provides the communications interface between the computer and the data acquisition system. Ostensibly, it converts signals between RS-232 compatible levels of ±12V and 5V ttl levels, but only requires a single 5V rail.

A single byte-long character command is passed to the *CDP6402* universal asynchronous receiver-transmitter, IC_3. This uart takes the serial data from the receiver input RRI and converts it into a parallel word. Provided that this word has been received correctly, it then appears at the receiver buffer register output, $RBR_{1\text{-}8}$.

In my design, if a framing error or an overrun error occurs, it is ignored. In any case, if an error does occur, the $RBR_{1\text{-}8}$ outputs adopt a high impedance state. The $RBR_{1\text{-}8}$ outputs are then decoded to provide the functions shown in **Table 1**.

The uart is configured for a data format of eight data bits, and one stop bit, **Table 2**. In addition, there is no parity bit; the parity inhibit PI input is held high.

The serial data rate is set by programmable oscillator, IC_2, which is an *EXO-19.6608*. This device allows data rates of 4800baud up to 1228800baud. However, the *MAX232* supports RS-232C standard, and this is only guar-

Listing 2. Object-oriented implementation of the functions used to interface to pc and a-to-d converter, in Turbo C++

```
// Standard libraries
#include <bios.h>
#include <conio.h>
#include <dos.h>
#include <process.h>
#include <stdio.h>

// Function key codes
#define F1 0x3B
#define F2 0x3C
#define F3 0x3D
#define F4 0x3E
#define F5 0x3F
#define F6 0x40
#define F7 0x41
#define F8 0x42
#define F9 0x43
#define F10 0x44

// COM port settings
#define COM2 1
#define DATA_READY 0x100
#define SETTINGS
(_COM_9600 | _COM_CHR8 | _COM_STOP1 | _COM_NOPARITY)

// Implementation of interface to unit. No error checking.
class serial {
private:
  unsigned _port;     // Port identity
  unsigned _settings; // Port settings
  // Configuration
  unsigned _channel, _gain, _scan, _run;
  char str[40];
public:
  // Constructor - Configures serial port
  serial(unsigned p, unsigned s) : _port(p), _settings(s) {
    _bios_serialcom(_COM_INIT, _port, _settings);
    _channel=_gain=_scan=_run=0;
  };

  // Change run mode
  void run() {
    _run=!_run;        }

  // Change scan mode
  void scan() {
    _scan=!_scan;
  }

  // Change gain
  void gain(unsigned g) {
    _gain=g;
  }

  // Change channel
  void channel(unsigned c) {
    _channel=c;
  }

  // Get port status
  int status() {
    unsigned s=_bios_serialcom(_COM_STATUS, _port, 0);
    return s;
  }

  // Read value from port
  unsigned read(unsigned& v) {
    return v=_bios_serialcom(_COM_RECEIVE, _port, 0);
  }

  // Write a value to port
  void write() {
    _bios_serialcom(_COM_SEND, _port,
      _channel | _gain<<2 | _run<<4 | _scan<<5);
  }

  // Get a-to-d unit's configuration
  char* config() {
    sprintf(str, "Channel:%1x Gain:%1d %s %s", _channel, 1<<_gain,
      _scan?"Scanning":"Fixed", _run?"Running":"Single");
    // If scanning, keep track of channel
    if (_scan) {
      _channel++;
      _channel%=4;
    }
    return str;
  }
}; // End of class definition
```

Listing 3. Turbo C++ routine applying the objected-oriented software, Liting 2.

```
void main(void) {

unsigned in, out;
class serial s(COM2, SETTINGS);
// Define and setup port

clrscr();
for(;;){
  if (s.status() & DATA_READY)
    // Print unit's configuration and
value read
    cprintf ("%s %3d\r", s.config(),
s.read(in));

  if (kbhit()) {
    out=getch();
    if (out=='\x1B')
     // Escape key pressed. Quit.
     exit(1);
    else {
     if (out=='\x00') {
       // Extended key pressed
       out = getch();
       switch (out) {
         case F1: s.channel(0); break;
         case F2: s.channel(1); break;
         case F3: s.channel(2); break;
         case F4: s.channel(3); break;
         case F5: s.gain(0); break;
         case F6: s.gain(1); break;
         case F7: s.gain(2); break;
         case F8: s.gain(3); break;
         case F9: s.run(); break;
         case F10: s.scan(); break;
       }
       s.write();
     }
    }
  }
}
}
```

anteed to work up to 19200baud. Switches, S_{1-3}, program the oscillator, **Table 3**.

Selecting channels

Selection of the analogue channel is carried out by the *MPC509* four-channel differential multiplexer, IC_6. This device offers up to $70V_{p-p}$ over-voltage protection, and should it lose power, it does not cause problems for the signal sources.

Channel addressing for the multiplexer is produced by the asynchronous presettable two-bit counter-latch formed by the JK bistable devices, IC_{10} and the steering logic, IC_{11}. Operation is as follows.

When scan mode is off, i.e. RBR_5 is low, the JK bistable device clocks are disabled, and the channel address on RBR_{1-2} is used to preset the counter-latch outputs. The PRESET signal, created by data-received signal DR going high and triggering monostable IC_{8b}, enables the NAND gates, IC_{11}. These then derive suitable logic levels for the bistable clear and set inputs. These inputs are clock independent, or asynchronous. Since the multiplexer's ENABLE line is tied high, the analogue input channel is immediately selected.

In channel scanning mode, with RBR_5 high, the counter-latch is preset with the channel address, as before. In this case, however, after each conversion has completed, the counter is incremented by the NEXT pulse. This pulse is derived from the BUSY signal going inactive, and enabled by the inputs RBR_{5-6} on NAND gate IC_{12c}. It results in the next analogue channel being selected.

Programmable-gain amplifiers

The *PGA205* programmable amplifier, IC_5, provides fixed programmable gains of 1, 2, 4, and 8. Its gain-selection inputs are ttl-compatible and bits RBR_{3-4} are connected directly. With the a-to-d converter configured with a reference voltage of 2.5V, the *PGA205* gives the system the full scale ranges and resolutions shown in **Table 4**.

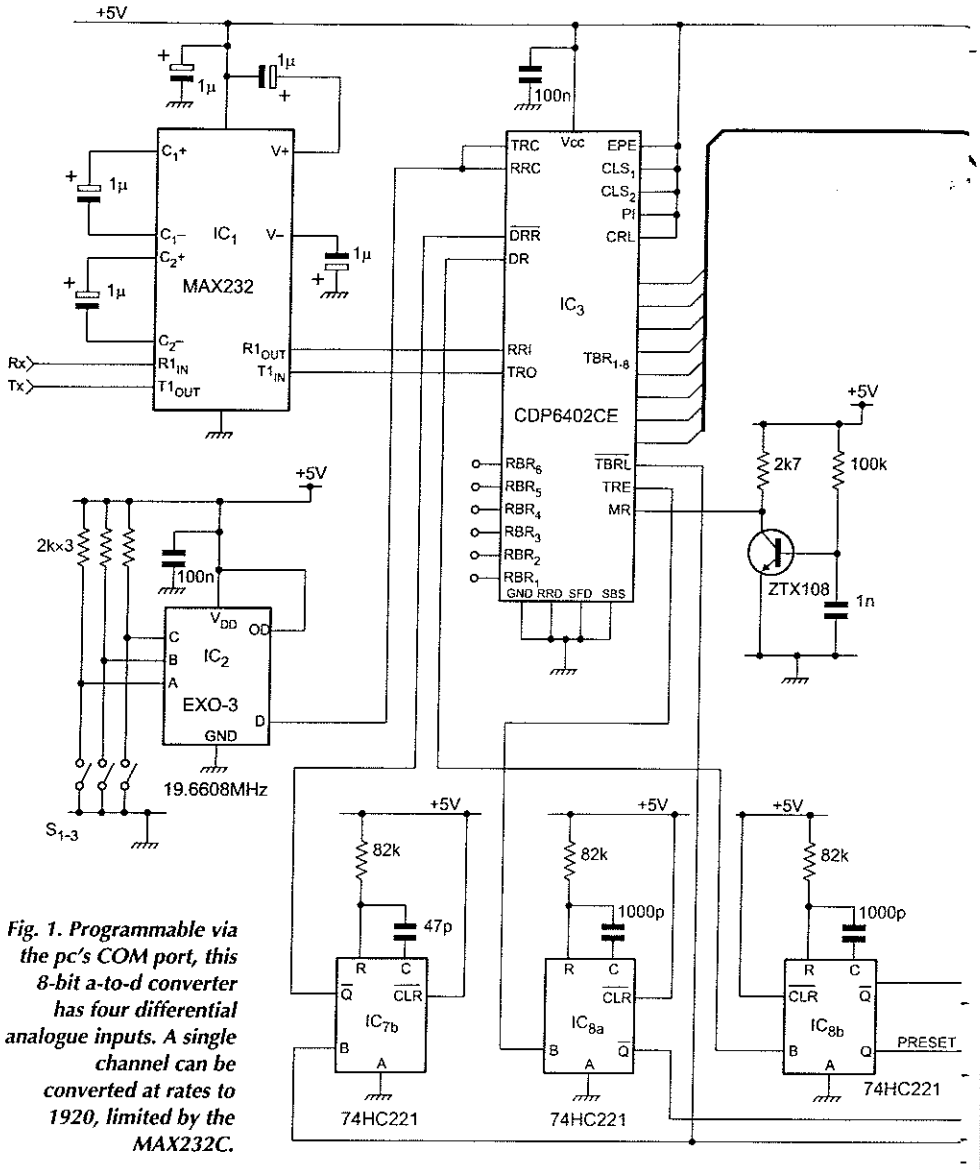


Fig. 1. Programmable via the pc's COM port, this 8-bit a-to-d converter has four differential analogue inputs. A single channel can be converted at rates to 1920, limited by the MAX232C.

Conversion mode

The conversion mode allows the unit to make either a single conversion when requested, or continually provide conversions, i.e. free run, at a rate governed by the uart.

If the single conversion mode is selected, i.e. RBR_6 is low, the a-to-d converter starts converting the selected channel shortly after the command is received from the computer. The data received status line DR goes high, and edge-triggered monostable IC_{8b} provides a low level pulse of about 82µs duration to the WRITE input of the a-to-d.

On the rising edge of the same low-level pulse, the a-to-d converter starts converting. The duration of the pulse is long enough to ensure the *PGA205* gain network and multiplexer have settled. It is also fast enough for conversions to be performed and transmitted at up to about 19.2 kilobaud, i.e. the sampling rate is about 1.9kHz.

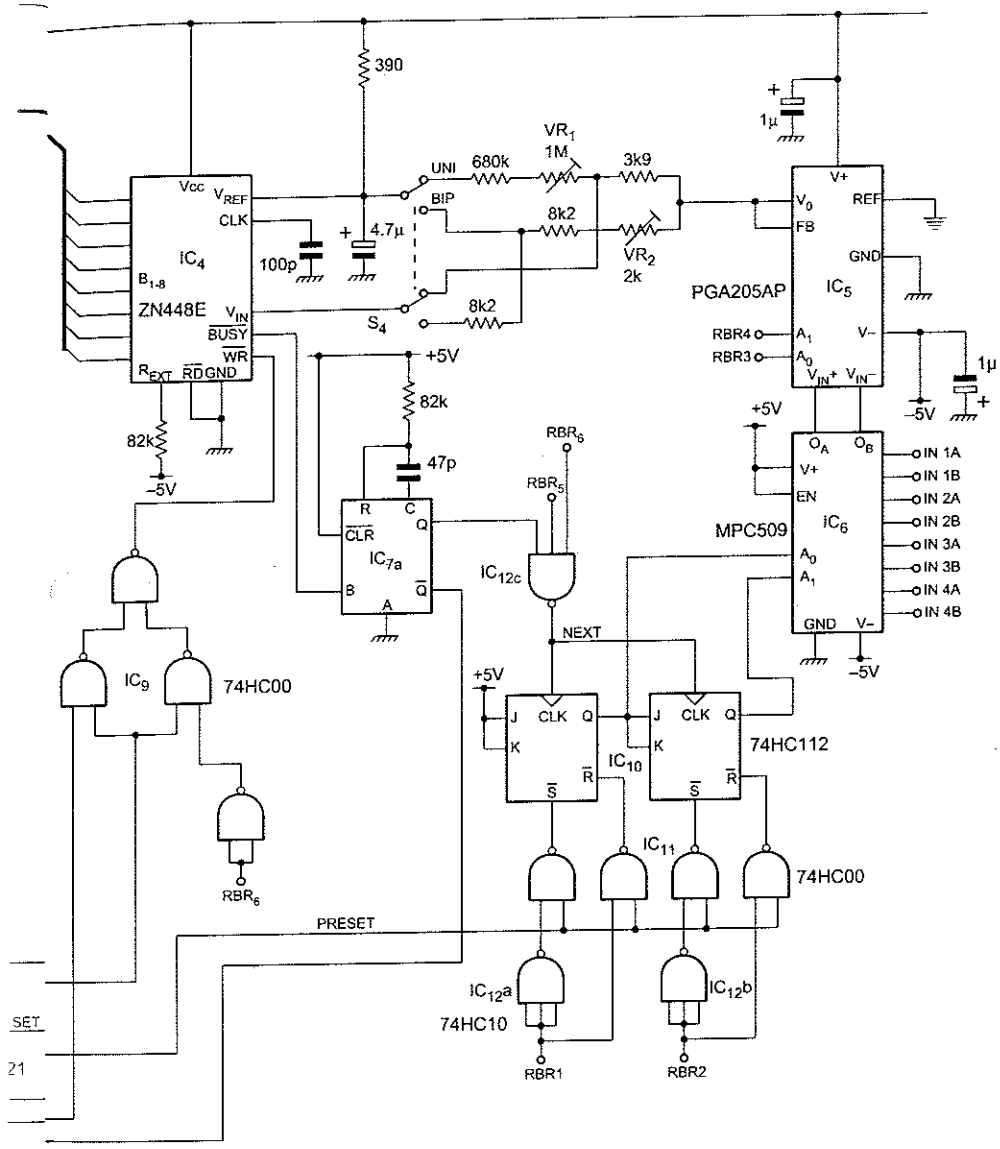
NAND gates within IC_9 select the source to be used for the start conversion signal. With RBR_6 low, only DR is used as the basis for the WRITE signal. When RBR_6 is high however, the start conversion signal is derived initially

Table 4. Full scale range, or fsr, and resolution at different gains.

Gain	Unipolar		Bipolar	
	FSR	Resolution	FSR	Resolution
1	2.5V	9.8 mV	±2.5V	19.5 mV
2	1.25V	4.9mV	±1.25V	9.8 mV
4	0.625V	2.4mV	±0.625V	4.9mV
8	0.3125V	1.2mV	±0.3125V	2.4mV

Listing 1. QuickBasic routine showing how simple it is to grab an analogue sample from channel 0.

```
REM Continually poll channel 0 (unity gain)
CLS : PRINT "Press any key to stop."
OPEN "COM2:9600,N,8,1" FOR RANDOM AS #1
A$ = ""
WHILE (A$ = "")
   PRINT #1, CHR$(0);
   LOCATE 2, 1: PRINT "Channel 0: "; ASC(INPUT$(1,
1))
   A$ = INKEY$
WEND
CLOSE #1
```



from DR, but then from the transmitter register empty, TRE, status flag of the uart. This signals that the last conversion has been sent to the computer, and that the UART is ready for new data. Again, the pulse is about 82µs duration.

Analogue-to-digital conversion

The a-to-d converter, IC_4, is a *ZN448* 8-bit successive approximation converter with internal band-gap reference and clock.

The converter is configured, by connecting the 100pF capacitor to the clock input, pin3, for conversion times of about 100ns. The input to the a-to-d converter can be either unipolar or bipolar according to the position of switch S_4. The resistor network sets the input voltage range to either 2.5V for unipolar operation or ±2.5V for bipolar.

During a conversion, the BUSY signal, active low, goes low, and when finished it goes high. On this rising edge, monostable multivibrator IC_{7a} creates a pulse that:

- automatically increments the channel address of the multiplexer,
- load the converted data into the uart's TBRL transmitter buffer register,
- reset the data received status flag of the uart, by taking DRR low.

Configuration and control

When the unit is powered up, the transistor, Tr_1 and associated passive components apply a low going pulse of 15µs to the uart's master reset MR input, and so ensures all the error/status flags, and transmitter buffers are reset.

The unit is easily configured and controlled by outputting byte commands – or appropriate ASCII characters – to the serial port.

An example *Quick-Basic* program is given in **Listing 1**. It shows how samples can be acquired from one channel, namely channel 0 with unity gain.

A further example is given by way of an object-oriented program using Borland *Turbo C++*, **Listings 2, 3**. Listing 2 gives the class implementation of the functions used to interface with the unit, while Listing 3 is an example of the class being used. The pc's function keys are used to configure the unit's operation, and the escape key exits the program.

Neither of these examples check for framing errors nor overrun errors, which would be necessary to ensure samples are not missed, or, if scanning channels, that the channel being sampled does not become misaligned with what the program thinks is being sampled.

The serial communications functions provided by *Turbo C++* are implemented with hardware handshaking. In this case, a null modem can be used; link the request-to-send line RTS and clear to send, CTS, together, and link data set ready DSR, data carrier detect, DCD, and data terminal ready, DTR. In *Quick Basic*, setting parameters in the OPEN statement that ignore handshaking is possible.

Setting up

The only setting up required is that of the *ZN448*. This is easily achieved with either of the above example programs, set to sample a channel continually.

Zero adjustment is required for the unipolar range. This is done by applying 5mV to a channel, and adjusting VR_1 until the most significant bit flickers between one and zero with all the other bits at zero. No gain adjustment is provided in this design.

Only offset adjustment is required for the bipolar range, as this design offers no gain adjustment. In a similar way to the unipolar zero adjustment, apply –2.49V to a channel, and adjust VR_2 until the most significant bit flickers between one and zero with all the other bits at zero.

Further development

There are two bits of the uart's received data word spare, namely RBR_{7-8}. These could easily be used to expand the unit's capability. Obvious enhancements are to provide eight channels of differential input, or 16 channels of single-ended input, or to make the unipolar/bipolar modes software selectable.

If greater accuracy is required, gain adjustment for both unipolar and bipolar input ranges could be added to the *ZN448*. ■

Further reading

MAXIM Integrated Products, *MAX230-241* Data Sheet, pp 2-25 to 2-40.
Universal asynchronous receiver transmitter, RS Data Sheet 4046, March 1985.
Crystal Oscillators – KSS Kinseki, *KSS-EXO-3* Series Data Sheet.
MPC508A, Burr Brown Data Sheet.
8-bit A to D converter *ZN448*, RS Data Sheet 5291, March 1985.
PGA205, Burr Brown Data Sheet, pp 4.175 to 4.187.
MM74HC221A Dual Non-retriggerable monostable mulitvibrator, National Semiconductor Data Sheet, pp 3:204-3:208.