12-bit analogue i/o via LPT

Although simple, Yongping Xia's LPT analogue interface resolves to 12 bits.

Equipped with a-to-d and d-to-a converters, a pc can perform many measurement and control functions. **Figure 1** shows an easy way of providing 12-bit a-to-d and d-to-a conversion. Driven by the software below, the circuit interfaces with the pc through its printer port.

The *MAX176* is a complete serial 12-bit a-to-d converter with a built-in track/hold circuit and a voltage reference. Two signals – clock and convert start – are needed to drive the chip. Once started, the a-to-d conversion result is sent out through the DATA pin in two's complement, high-to-low serial order. Analogue input is buffered by IC_{1B} with range of –5V to +5V. The *MAX176* needs +5V and –15V power supplies and provides a –5V reference output.

The *MAX543* is a 12-bit serial d-to-a converter. Its current output is converted to voltage by IC_{1A}. Required –5V reference is provided by a *MAX176*. Resistors $R_{1,4}$ adjust the d-to-a converter offset and gain respectively. The d-to-a converter's output range is also –5V to +5V.

The a-to-d and d-to-a conversion procedures shown are in C. In this application, two printer port addresses (0×37c and 0×37d) are used. One is for output and the other input. Note that the base address may differ between computers. You should find details in your user guide.

The d-to-a conversion procedure converts 12-bit data in serial order and sends it to *MAX543* through the printer port pin 5. Conversion data is stored in 'data out'. An output register named 'out' is used to map the base address printer port. The a-to-d conversion procedure generates *MAX176* required CL(oc)K and CONV(ersion start) signals through pins 2 and 3 of the printer port, reads serial data via printer port pin 15, and returns the reorganised a-to-d conversion result.

These procedures can be included in any C-based application program. If an a-to-d conversion is needed, call the a-to-d procedure and it will return the result. If a d-to-a conversion is required, simply call the d-to-a procedure and pass the data to the procedure. Conversion time depends on the type of pc is used. It takes around 75µs for a-to-d and 68µs for d-to-a on a 50MHz 486 machine. ■

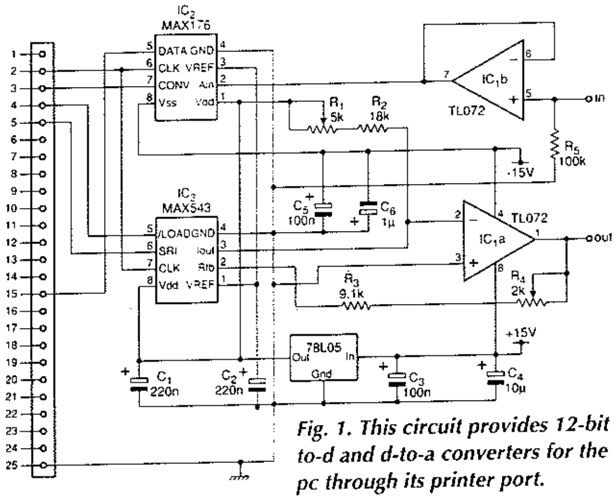

Fig. 1. This circuit provides 12-bit a-to-d and d-to-a converters for the pc through its printer port.

Assembly language for reading and writing the analogue data converters via the pc's LPT port.

```
#include cstdio.h~
#include <dos.h>
#include <conio.h~
#define OUT PORT    0x37c          /* base address */
#define IN PORT    0x37d           /* base address + 1 */
#define CLOCK HIGH 0x01            /* set clock bit high */
#define CLOCK LOW Oxfe             /* set clock bit low */
#define CONVST HIGH 0x02           /* set start conversion bit high */
#define CONVST LOW 0xfd            /* set start conversion bit low */
#define LOAD HIGH 0x04             /* set load bit high */
#define LOAD LOW 0xfb              /* set load bit low */
#define DAC HIGH 0x08              /* set data out bit high */
#define DAC LOW 0xf7               /* set data out bit low */
void dac(int data out);
int adc(void);

/* D-to-A conversion procedure */
void dac(int data out)
{
 int i, out;
 out = 0x04;            /* set DAC's LOAD to be high */
 for (i=0; i<12; i++)   /* send out 12-bit DAC data */
  {
   if (data out < 0x800)            /* if bit 11 is low, */
    {
    out = out & DAC LOW;            /* set output bit = 0 and */
    data out *= 2;         /* double DAC data */
   else     /* if bit 11 is high, */
    out = out | DAC HIGH;           /* set output bit = 1 and */
    data out = (data out-Ox800) * 2; /* double DAC data after set */
    }       /* bit 11 to be 0 */
   outportb(OUT PORT, out);         /* send out */
   out = out | CLOCK HIGH;          /* turn CLOCK high */
   outportb(OUT PORT, out);         /* send out */
   out = out & CLOCK LOW;           /* turn CLOCK low */
   outportb(OUT PORT, out);         /* send out */
  }
  out = out & LOAD LOW:           * turn LOAD low */
  outportb(OUT PORT, out:         * send out */
  out = out | LOAD HIGH;          * turn LOAD high */
  outportb(OUT PORT, out:         * send out */
}
/* A-to-D conversion procedure */
int adc(void)
{
 int i, data, out;
 data=0;    /* clean data
 out = 0x04 | CLOCK HIGH;         * set CLOCK and DAC's LOAD high */
 outportb(OUT PORT, out:          * send out */
 out = out & CLOCK LOW:           * turn CLOCK low */
 outportb(OUT PORT, out:          * send out */
 out = out | CLOCK HIGH:          * turn CLOCK high */
 outportb(OUT PORT, out:          * send out */
 out = out | CONVST HIGH;         * turn A/D CONVERT START high */
 outportb(OUT PORT, out):         * send out */
 out = out & CLOCK LOW:           * turn CLOCK low */
 outportb(OUT PORT, out):         * send out */
 out = out & CONVST LOW;          * turn A/D CONVERT START low */
 outportb(OUT PORT, out:          * send out */
 out = out | CLOCK HIGH:          * turn CLOCK high */
 outportb(OUT PORT, out:          * send out */
 out = out & CLOCK LOW:           * turn CLOCK low */
 outportb(OUT PORT, out:          * send out */
 for (i=0; i<l2; i++)             * get 12-bit conversion data */
  {
   out = out | CLOCK HIGH:        * CLOCK high */
   outportb(OUT PORT, out);       * send out */
   data = data * 2 + (inportb(IN PORT) & 0x08) / 8; /* update */
   out = out & CLOCK LOW;         * CLOCK low */
   outportb(OUT PORT, out);       * send out */
  }
 return (data);             /* return A D conversion result */
}
main()
{
/* your application */
```