design ideas

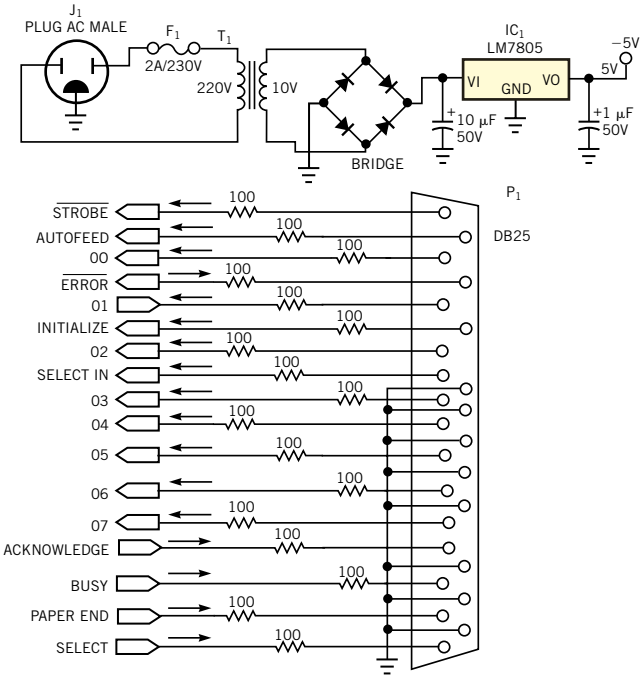
Edited by Bill Travis

Tricks increase utility of parallel port

Lorenzo Tazzari, Selex SNC, Alessandria, Italy

In this simple application of the 68HC68 microcontroller's serial-I/O utility, the goal is to configure a simple circuit, driven by any LPT parallel-printer port, which you can use as a remote I/O for a PC. You can independently program each I/O line as either an input or an output. The protocol in this application is an SPI (MISO/MOSI/SCK) type, using synchronous serial communications. **Figure 1** shows a circuit that effects the connection with the PC and power supply for all I/O signals. A bus carries signals of the SPI protocol, and the LPT port can drive all the $\overline{CE}$ (Chip Enable) signals. With this type of bus, you drive as many as five $\overline{CE}$ signals, and each $\overline{CE}$ line can address four 68HC68 chips. Each microcontroller can drive eight I/O lines, with each line independently programmable. Thus, the system can address as many as 160 I/O ports.

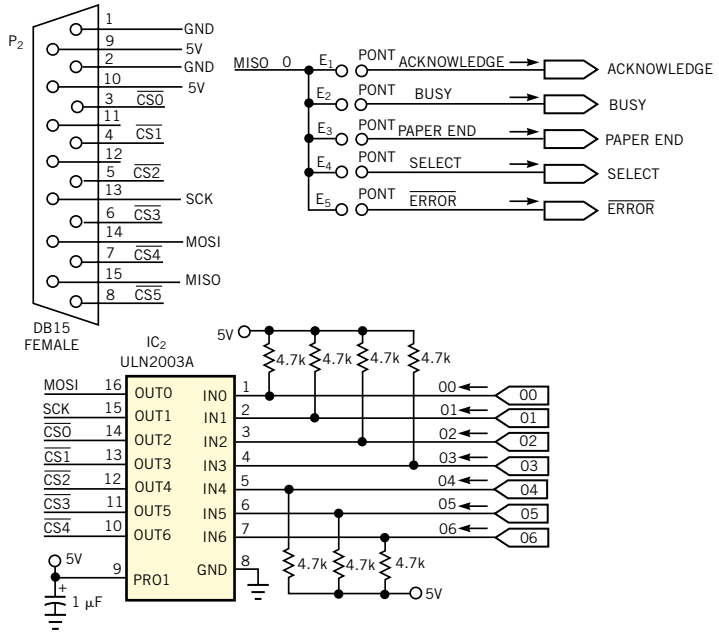
Figure 1



You address as many as 160 I/O signals with this simple connection of the LPT port.

Figure 2 shows a simple way to select the MISO signal. You can choose which pin of the LPT port to use in receiving the MISO signal. This circuit uses the ULN2003 as an amplifier to drive the critical $\overline{CE}$ signal. By joining the SIP and $\overline{CE}$ signals, you can configure a bus system in which all signals go to the bus connector. You need an external power supply, because the LPT port cannot supply sufficient current. You can connect as many as five I/O circuits. The circuit in **Figure 3** contains the 68HC68 chip with its $\overline{CE}$ and address selection. With jumpers E_{12} and E_{13}, you

Figure 2



This simple scheme facilitates selection of the MISO signal in the SPI protocol.

can select the chip's address. It is important to emphasize that the system has no check for an address conflict arising from choosing the same address, with the same $\overline{CE}$, in more than one chip. If the conflict arises, however, it does not cause hardware damage. Theoretically, using an LPT port at a baud rate of 1 Mbyte/sec, the technique can read or write 160 I/O signals in less than 16 msec.

Is this the best Design Idea in this issue? Vote at www.ednmag.com.

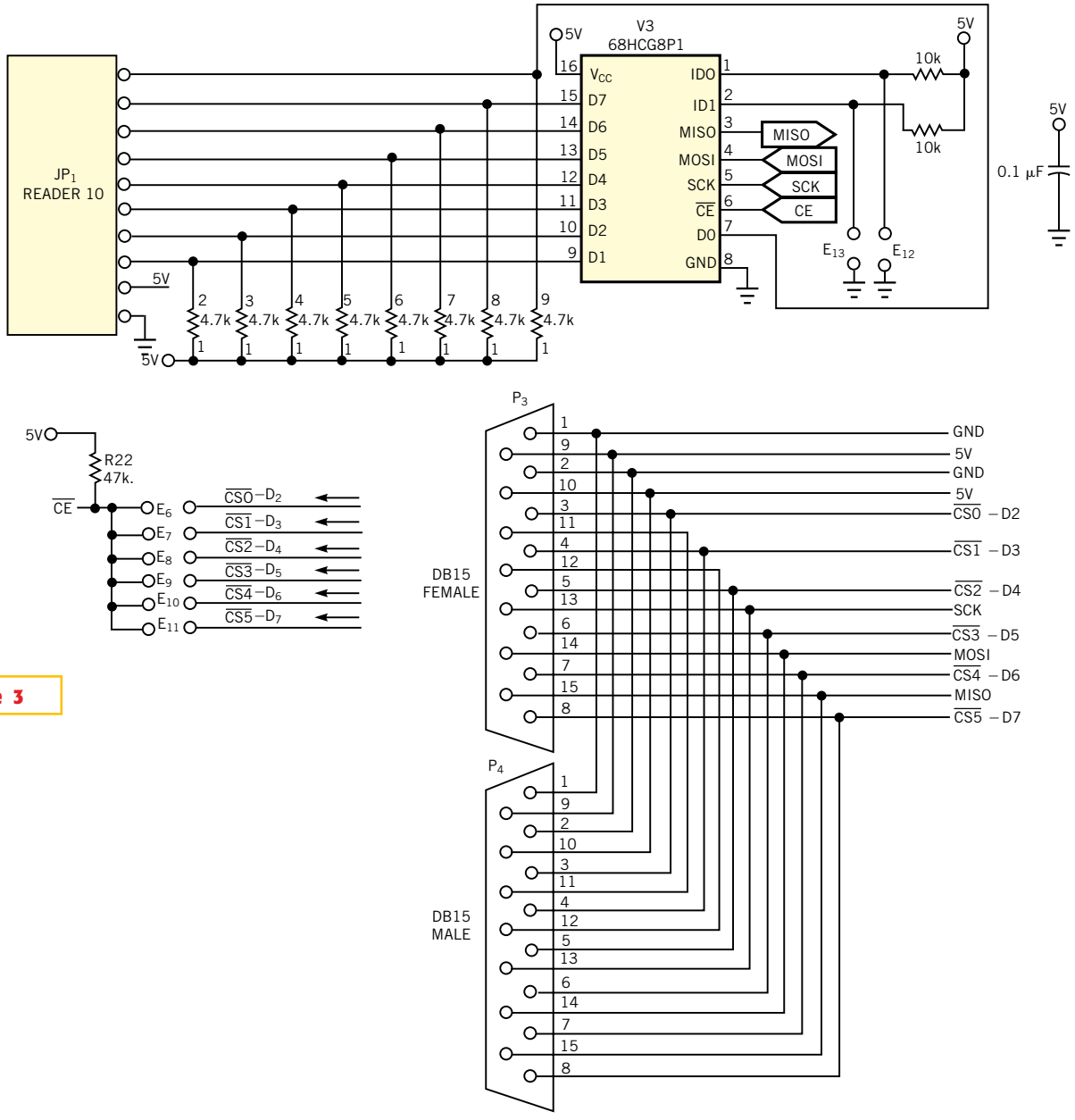


Figure 3

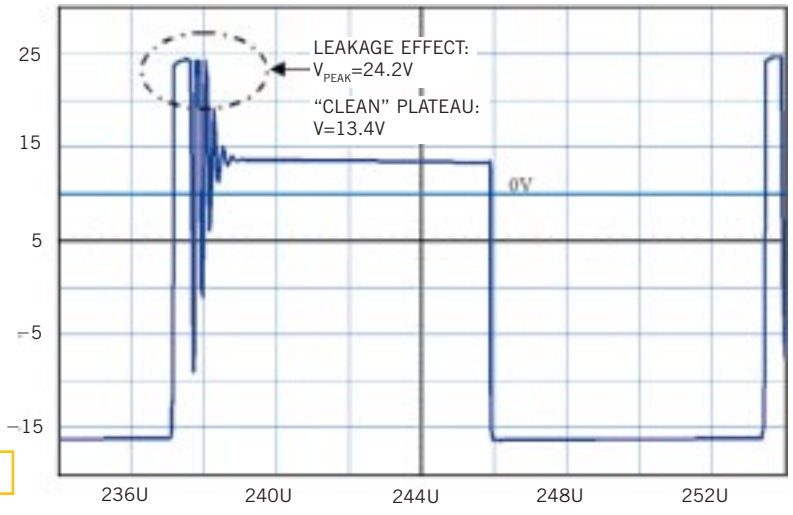
A 68HC68 microcontroller handles $\overline{CE}$ and address selection.

Transistors tame perfidious leakage inductance

Christophe Basso, On Semiconductor, Toulouse, France

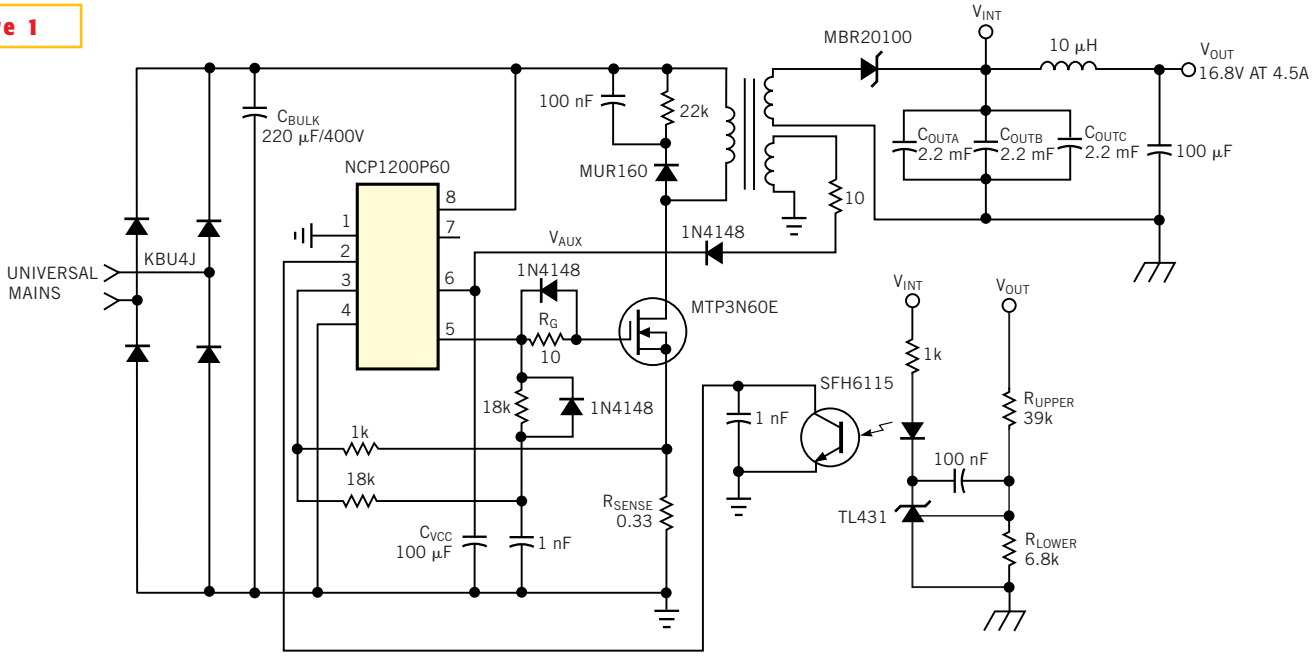
In flyback converters that use primary regulation, the loose coupling between the power secondary and the primary auxiliary windings often results in poor cross-regulation. This situation arises mainly from the leakage inductance but also comes from the level of the primary clamp voltage. **Figure 1** shows a typical application schematic using On Semiconductor's (www.onsemi.com) NCP1200 in an auxiliary-winding configuration. This IC uses a DSS (dynamic self-supply), but in some low-standby-power applications, it is desirable to permanently disconnect this feature through an auxiliary level. The DSS simply acts as a standard start-up current source until the auxiliary level takes over. In this application, the regulation takes place on the secondary side by means of the TL431, but the primary

Figure 2



The leakage inductance on the auxiliary-winding side causes high rectified voltages.

Figure 1



In this circuit, leakage inductance in the auxiliary winding can invalidate the controller's short-circuit-protection circuitry.

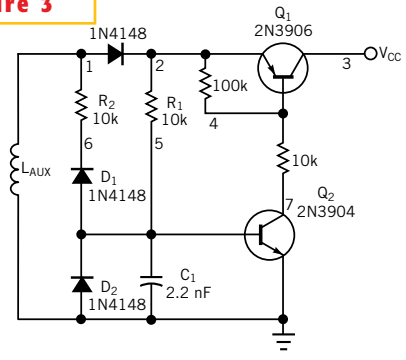
level assumes importance in short-circuit conditions. Each time the NCP1200's V_{CC} crosses 10V while dropping, the internal logic senses the eventual presence of a short circuit through the feedback pin. Should the circuit confirm a short circuit, the NCP1200 emits a safe-autorecovery, low-frequency burst. However, if poor coupling prevents the auxiliary winding from collapsing, in the presence of a secondary short circuit, V_{CC} never crosses the 10V threshold, and damage to the circuit may ensue.

Figure 2 details the effects of the leakage inductance when a short circuit occurs at the output. As you can see, the leakage spike pushes the auxiliary level well above its regular plateau voltage, which is the value you'd like to obtain. With the rectifying diode playing the role of an envelope detector, the result is a final level close to 24V, far from the 13.4V you would expect. As a result, a possibly destructive condition exists if the levels exceed the maximum ratings in the controller's data sheet. You need to clamp the auxiliary voltage using a dissipative element, such as a zener diode. **Figure 3** shows the circuitry you adopt to avoid the leakage-inductance problems. The component arrangement actually implements a self-contained sample-and-hold system. When the main power switch is on, capacitor C_1 discharges through R_2 and D_1, and D_2 avoids a large reverse bias of Q_2's base-emitter junction. When the main switch opens, the secondary voltage sharply rises, and Node 1 becomes positive. However, because C_1 discharges, Q_1 remains open, and V_{CC} does not increase.

After a short period (adjustable via R_1 or C_1), Q_2 closes and brings Q_1's base closer to ground. V_{CC} now increases and catches up to the level at Node 2, minus Q_1's $V_{CE(SAT)}$. If you correctly select the time delay, V_{CC} is devoid of any voltage spike, because you have sampled the plateau. **Figure 4** shows the final result. Performing some measurements on a 70W application board featuring low standby power yields the final tracking results in **Figure 5**. You can see that a 4.3A change in I_{OUT} results in a change of only 420 mV in V_{OUT}. You can use the circuit in a primary-regulation application in which you need a precise level without either heavily filtering the secondary winding (and thus lowering the available auxiliary energy in standby mode) or reducing the primary clamp voltage to a higher dissipative value. In the NCP1200 application, when a short circuit appears at the output, the auxiliary winding properly triggers the short-circuit protection.

Is this the best Design Idea in this issue? Vote at www.ednmag.com.

Figure 3

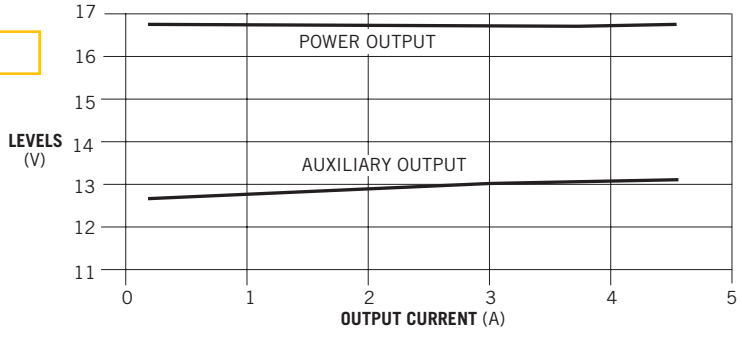


This component arrangement creates a discrete sample-and-hold system.

Figure 4

By delaying the sampling time, you obtain a clean auxiliary level that is devoid of any leakage-inductance effects.

Figure 5



Thanks to the circuit in Figure 3, the auxiliary winding better tracks the primary winding.

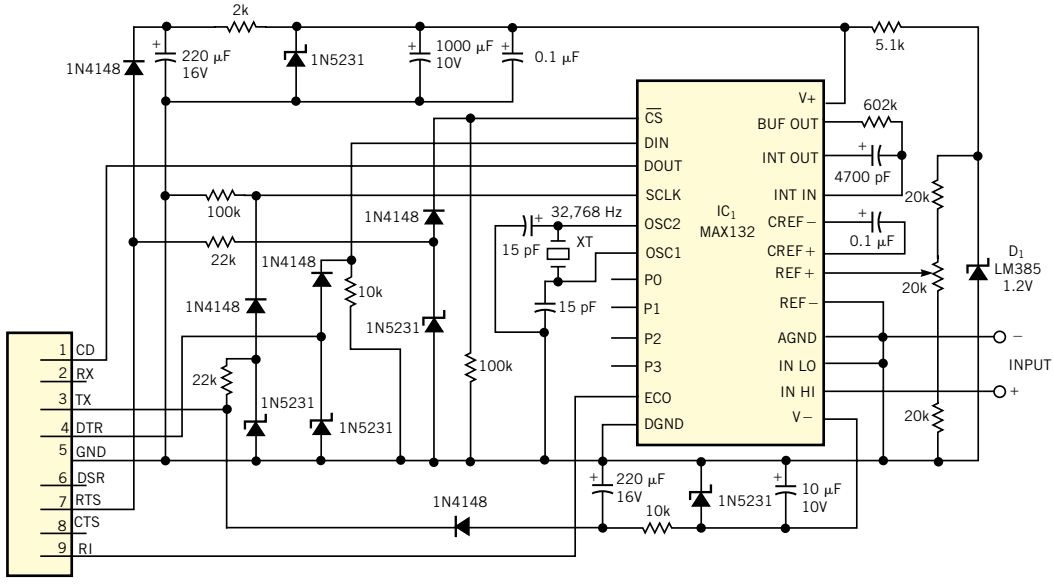
18-bit ADC uses PC's serial port

Yongping Xia, Teldata, Los Angeles, CA

A PC usually requires a plug-in ADC card to process analog signals. However, with the circuitry in **Figure 1**, a PC can communicate with an 18-bit ADC through its serial port. The port provides both positive and negative power supplies as well as control signals. IC_1 is an 18-bit MAX132 ADC with a serial interface. It requires three input control signals, $\overline{CS}$, DIN, and SCLK, and emits serial data, DOUT, and EOC (end-of-conversion) signals. An RS-232 port has three output lines: Pin 3 (TX), Pin 4 (DTR), and Pin 7 (RTS). TX generates the clock signal for the MAX132 and provides the negative power supply. DTR transmits serial data. RTS provides the CS signal and the positive power supply. Both the positive and the negative supplies use large capacitors for energy storage. When TX generates a clock signal or DTR sends a CS logic-low signal, the capacitors provide power to the MAX-132. The MAX132 integrates everything except a reference that comes from a 1.2V LM385 voltage-reference diode, D_1. The input-voltage range of the MAX132 is −512 to +512 mV. **Listing 1** is a C program that displays the analog-to-digital-conversion result on-screen. You can download **Listing 1** from the Web version of this Design Idea at www.ednmag.com.

Is this the best Design Idea in this issue? Vote at www.ednmag.com.

Figure 1



You can use a PC's serial port to communicate with an 18-bit A/D converter.

LISTING 1—SCREEN-DISPLAY ROUTINE FOR ANALOG-TO-DIGITAL-CONVERSION RESULTS

```
#include <stdio.h>
#include <dos.h>
#include <time.h>
#include <conio.h>
#include <bios.h>
#include <conio.h>

#define COM1    0
#define MCR     4                       /* control register    */
#define MSR     6                       /* status register     */

int i, j, base_add1=0x3f8, base_add2=0x2f8, out_data=0x03, in_data[4];

float data;

void send_clk(void)
{
  delay(1);
  outportb(base_add1, 0x00);
  delay(3);
}

void read_port(void)
{
int control[4], out_control;
data=0;
for (i=0; i<4; i++)
  in_data[i]=0;
control[0]=0x82;
control[1]=0x04;
control[2]=0x00;
control[3]=0x00;
out_data|=0x02;
outportb(base_add1+MCR, out_data);              /* CS high
      */
delay(10);
out_data&=0x01;
outportb(base_add1+MCR, out_data);              /* CS low
      */
delay(10);
out_control=control[0];
for (i=0; i<8; i++)
 {
   if (out_control>=0x80)
     out_data|=0x01;
   else
```

Continued on pg 86

LISTING 1—SCREEN-DISPLAY ROUTINE FOR ANALOG-TO-DIGITAL-CONVERSION (CONTINUED)

```
      out_data&=0x02;
    outportb(base_add1+MCR, out_data);
    send_clk();                                  /* clock out      */
    out_control&=0x7f;
    out_control=out_control*2;
  }
 out_data|=0x02;
 outportb(base_add1+MCR, out_data);                    /* CS high
        */
 delay(10);
 do{
  }while((inportb(base_add1+MSR)&0x40)==0);      /* waiting for EOC=high
        */
 for (j=1; j<4; j++)
 {
    out_control=control[j];
    in_data[j]=0;
    out_data&=0x01;
    outportb(base_add1+MCR, out_data);   /* CS low         */
    delay(10);
    for (i=0; i<8; i++)
      {
        if (out_control>=0x80)
          out_data|=0x01;
        else
          out_data&=0x02;
        outportb(base_add1+MCR, out_data);
        in_data[j]=in_data[j]*2+(inportb(base_add1+MSR)&0x80)/0x80;
        send_clk();                              /* clock out      */
        out_control&=0x7f;
        out_control=out_control*2;
     }
    out_data|=0x02;
    outportb(base_add1+MCR, out_data);           /* CE high        */
    delay(10);
 }
 if ((in_data[1]&0x08)==0)

data=(float)(in_data[1]&0x07)+(float)(in_data[2])*2048+(float)(in_data[3])*8;
  else                                           /* reading is negative      */
    {
      in_data[1]=in_data[1]&0x07;
      in_data[1]=(8-in_data[1])&0x07;
      in_data[2]=(256-in_data[2])&0xff;
      in_data[3]=(256-in_data[3])&0xff;
     data=-((float)(in_data[1])+(float)(in_data[2])*2048+(float)(in_data[3])*8);
    }
}

void dis_data(void)
{
  float show_data;
  show_data=0.000002*data;
  gotoxy(1,1);
  printf("%.5f  ", show_data);
  gotoxy(1,1);
}

void init(void)
{
   bioscom(0, 255, COM1);                        /* set up COM1
          */
   out_data=0x02;
   outportb(base_add1+MCR, out_data);              /* CS=high, DIN=low
          */
   delay(100);
}

 void main(void)
{
   clrscr();
   init();
   gotoxy(60,24);
   printf("Hit any key to quit");
   do{
          read_port();
          dis_data();
          delay(500);
     } while(!kbhit());
}
```