



needed a human interface to my home-brew automation system. This interface had to be simple, good looking, and fully programmable.

Since my home control system is based on a PC, I could have used the monitor and keyboard, but I didn't like the PC's bulkiness and wanted the option of multiple interfaces.

My other choices included dumb terminals (too big and ugly), off-the-shelf terminals (expensive), or a home-made display terminal (perfect in all aspects, except my time).

Build my own? A great idea! I decided to call it Display-8.

## REQUIREMENTS

Once I decided to design my own display terminal, a list of features soon appeared:

- LCD display for text display
- 8 buttons for user input
- 8 LEDs for system status
- attractive enclosure-wall or table mount
- 875 1 based (easy to wire-wrap)
- powered from PC.

Since the most flexible display terminal would use an RS-232 interface to the PC, the buttons and display would then appear as a dumb terminal interface. The buttons generate the ASCII numerals 1-8 and text from the PC is displayed on the LCD display. Escape codes access LCD features and LEDs.

## DISPLAY-S HARDWARE

Figure 1 shows the schematic for the Display-8 terminal. Display-8 is based on an Intel 875 1 (U1) because the internal RAM and EPROM simplifies wire-wrapping. With the 875 1, I can use all four I/O ports as I wish: port 0 reads the key switches, port 1 interfaces to the LCD display, port 2 controls the LEDs, and port 3 performs various control functions. I also have a large amount of code already written for this processor, which should save me some time.

# The Display-8

## Add a Human Interface to Your Home Automation System

The LEDs are controlled by software writes to port 2. The switches are debounced by multiple software reads from port 0.

Display-8 includes a Hitachi LM052L 2-line x 16-character display. Typically, you would use decode logic to have the LM052L appear as external data space. However, since I have ample pins on the 875 1 and do not enjoy wire-wrapping, I connected the LM052L directly to the processor. Software controls the pins directly.

The LM052L data bus is connected to port 1 of the 875 1. The LM052L register-select pin is tied to P3.4 (T1), the read/write pin is tied to P3.6 (WR), and the enable pin is tied to P3.7 (RD).

A MAX232 transceiver from Maxim provides the RS-232 interface. This chip internally generates the  $\pm 12$  V needed for RS-232 and requires only four capacitors. Display-8 uses the 875 1's built-in serial port on pins P3.0 (RXD) and P3.1 (TXD).

Connector J1 is an RJ-11 connector through which power is supplied and serial communications take place.

For Display-8, I used C&K switches that include a DPDT switch and LED in one housing. The switch fits into a standard 14-pin socket footprint. Everything is easily wire-wrapped onto a single protoboard. Photo 1 shows the component placement.

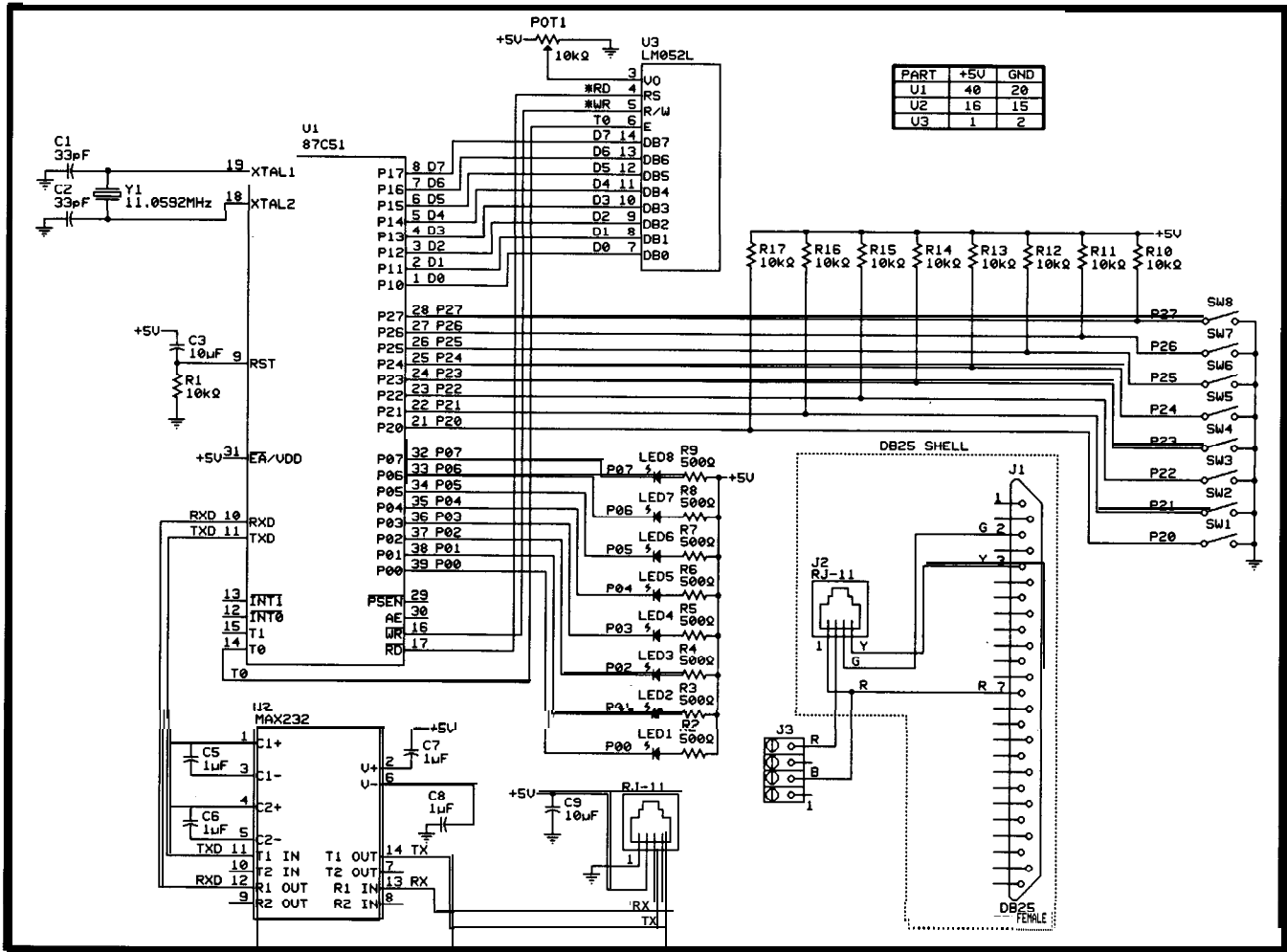
## DISPLAY-S ENCLOSURE

The most important issue of this display was that it needed to be attractive. I used a premade aluminum enclosure from Radio Shack, so I was able to drill and cut holes for the display and

## MITCH DRUMMOND

While many home control systems operate passively in the background, users **often** like some hands-on interaction with the system. Display-8 offers that interaction in a small, unobtrusive package.





**Figure 1:** Display-8 is based on the 87C51 microcontroller for simple assembly. The switch, LED, and LCD hardware decoding is performed by the microcontroller software.

key switches. (If you implement this idea, remember to make your package safe by filing the cut edges.)

For a tabletop unit, the enclosure comes with rubber feet and the cable exits in the rear. My unit is wall mounted and includes a cutout so it can be mounted over a standard electrical box. I painted the finished display white to match the decor of the house.

**PC CONNECTION**

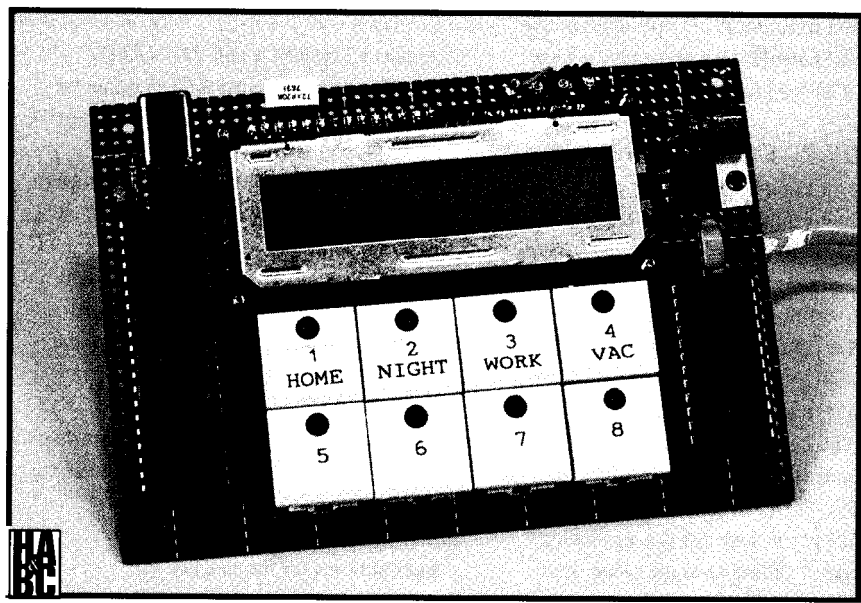
Since this display is powered from the PC, I used an adapter cable at the PC to combine the serial interface and the power supply (see Figure 1). I used a DB-25-to-RJ-11 converter and a disk drive power supply extension cable from Radio Shack.

**Photo 1:** IC sockets protect the components during assembly. The resistors are located under the LCD display. Once placed in its enclosure, Display-8 is ready for table or wall mounting.

**DISPLAY-8 SOFTWARE**

The more complicated issue was the 875 1 software. I wanted to be sure the program fit into the 4 KB of program space,

but more importantly, the 128 bytes of data space. For simplicity, I chose to use the Archimedes 805 1 C cross-development tools. This choice



**Listing 1:** Interrupts are only used for input with the 8031 serial I/O routines.

```

interrupt void serial_isr(void)

if (RI) {
    rx_buff[rx_buff_head] = SBUF; /* store char in buffer */
    RI = 0;
    ri_flag = 1;
    rx_buff_head++; /* increment head pointer */
    rx_buff_head &= RX_BUFFER_MASK; /* Wrap around the end */
    if (rx_buff_head == rx_buff_tail) { /* If at tail, */
        rx_buff_head; /* drop character instead */
        rx_buff_head &= RX_BUFFER_MASK;
    } /* end if RI */

if (TI) {
    TI = 0; /* tell foreground routine */
    ti_flag = 1;
}

return; /* end serial_isr */
}

char get_char(char * c)

if (rx_buff_head != rx_buff_tail) { /* check for empty buf */
    *c = (rx_buff[rx_buff_tail]);
    rx_buff_tail++; /* increment tail pointer */
    rx_buff_tail &= RX_BUFFER_MASK; /* wrap-around if needed */
    return TRUE;
}

else {
    ri_flag = 0; /* empty buf, clear flag */
    return FALSE; /* and fail. */
} /* end get_char */

void send_char(unsigned char c)

while (ti_flag == 0);
ti_flag = 0;
SBUF = c;
return; /* end send-byte */
}

```

worked out very nicely in that I did not have to do any assembly language programming. All interrupt service routines (ISRs) and output routines were written in C.

The software includes ISRs for both the serial port and the 5-ms timer tick. The timer ISR simply sets a global flag that is checked by the main routine. The main routine calls `scan_input` when the timer flag is set. The `scan_input` routine first reads the current state of the input pins and stores the current states in an array of characters.

The routine then integrates the input values for a count of `MAX_CNT` (three reads or 30 ms). If the integrated input state is 0, then a button is

pressed and a character is sent to the host. A hex 31 is added to the switch number (`cn_t`) to indicate an ASCII 1-8.

The serial ISR is not as simple. Listing 1 shows the serial port routines. The serial ISR first checks the RI flag. If a character has been received, `SBUF` is read and stored in a circular buffer and the head (input) pointer is incremented.

The OR function checks for wrap around. If the head pointer is equal to the tail pointer, the buffer has overflowed. In this case, the head pointer is decremented, and the last character is lost. The ISR sets `ri_flag` to indicate that a new character is in the input buffer. If the serial ISR detects a transmit interrupt, the output buffer is available and `ti_flag` is set for the foreground routine.

The main routine checks the variable `ri_flag` and calls the routine `get_char` to get the character from the serial buffer. This routine checks that the head pointer does not equal the tail pointer. The character is read from the address of the tail pointer, and the tail pointer is incremented. If the tail pointer equals the head pointer, the buffer is empty and the `ri_flag` variable is cleared.

The software in Display-8 supports several control characters including formfeed (0Ch), new line (0Ah), carriage return (0Dh), and backspace/delete (08h). Also, escape codes control the LEDs and the special functions of the LCD driver.

Listing 2 shows the routines involved in character output. The `display()` routine is called for each character that is received. Based on the last character received (possibly an escape code), the correct action is taken. If an escape code is received, a flag is set and the routine returns while waiting on the next character.

The normal characters are sent to the `lcd_putchar()` function, which controls the LCD. This function looks for control characters and sends characters to the LCD using `lcd_char()` and control commands using `lcd_control`.

The secret to the `lcd_putchar()` routine is to keep a copy of the display in memory (variable `display_copy[1]`). The LCD controller has memory for a 2 x 40 display, but I am using only a 2 x 16 display. The software only stores an image of the 2 x 16 memory.

Also, the position of the cursor (row and column) is tracked by the global variables `row` and `col`. If a new line is found, the cursor advances to the next line. If it is already on the second line, the second line is copied to the first line, and the cursor is positioned at the start of the second line.

After each character is sent to the display, the position is incremented and checked for a wrap-around condition. When a wrap-around condition occurs, the cursor is repositioned, and the second line is copied to the first if necessary.

A formfeed character executes a display home command that clears the display and positions the cursor in the first position. A `backspace` command repositions the cursor back one space, prints a blank, and repositions the cursor again. A carriage return simply repositions the cursor at the beginning of the current row.



## OPERATION

When plugged into a PC, Display-8 can be used with any communications program. Characters sent to Display-8 are echoed (so you can see what you're typing). Consult the LCD data manual for a complete list of the characters available. Carriage return, new line, formfeed, and backspace characters are all supported by Display-%

When buttons on the Display-S are pressed, the ASCII characters 1-8 are sent to the PC communications program.

To control the LEDs, send an **LED\_CHAR** command (hex OE, dec 14) and then a bitmap of which LEDs to turn on. LED 1 corresponds to the least-significant bit of the byte.

Other special functions of the LCD can be executed by issuing a **CONT\_CHAR** command (hex OF, dec 15) and then the LCD control byte. This byte is written to the control register of the LCD display.

Graphics characters can also be defined in this way. Data read-back capabilities of the LCD display are not available to the serial port user.

## FINALE

Display-8 provides an excellent user interface to my home control system. In fact, I like the design so much that I am planning on building more units to use around for other control functions.

*Mitch Drummond is a design engineer for the cable television industry. Mitch has been developing embedded systems for military and commercial applications for over ten years. He's currently developing a home automation system in his spare time. Mitch may be reached at [mitchd@uvsg.com](mailto:mitchd@uvsg.com).*

## SOFTWARE

Software for this project is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" in this issue for downloading and ordering information.

## I R S

- 419 Very Useful
- 420 Moderately Useful
- 421 Not Useful

Listing 2: *The function `lcd_putchar` controls the LCD's I/O pins for writing data to the display.*

```
void lcd_putchar(unsigned char c_out)

    int i;

    if (c_out == '\a') {          /* No bell on display, ignore. */

    else if (c_out == '\b'){     /* backspace, delete char. */
        if (col > 0){
            lcd_control((char) (DS_ADDRESS+((row*40)+col-1)));
            lcd_char(' ');
            lcd_control((char) (DS_ADDRESS+((row*40)+col-1)));
            disp_copy[row*COL_CNT+col-1] = ' ';
            col --;
        }

    else if (c_out == '\f'){     /* formfeed, so clear screen */
        lcd_control(DS_HOME);
        row = 0;
        col = -1;
        for (i=0; i<ROW_CNT*COL_CNT; i++)
            disp_copy[i] = ' ';
    }
    else if (c_out == '\n'){     /* new line. If on 2nd row, */
        if (++row == ROW_CNT){   /* copy to first. */
            lcd_control(DS_HOME); /* Clear the display */
            for (i=0; i<COL_CNT; i++){ /* copy 2nd row to 1st row and */
                disp_copy[i] = disp_copy[i+COL_CNT]; /* clear 2nd row memory */
                lcd_char(disp_copy[i]);
                disp_copy[i+COL_CNT] = ' ';
            }
        }
        row = 1;

        col = -1;                /* put cursor at start of 2nd row. */
        lcd_control((char) (DS_ADDRESS+40));

    else if (c_out == '\r'){     /* Insert carriage return, put */
        col = -1;                /* cursor at first column */
        lcd_control((char) (DS_ADDRESS+(row*40)));

    else {
        lcd_char(c_out);
        disp_copy[row*COL_CNT+col] = c_out;

    if (++col == COL_CNT){      /* inc ptr; process next char */
        if (++row == 2){        /* copy 2nd row to 1st row and */
            lcd_control(DS_HOME); /* clear 2nd row memory */
            for (i=0; i<COL_CNT; i++){
                disp_copy[i] = disp_copy[i+COL_CNT];
                lcd_char(disp_copy[i]);
                disp_copy[i+COL_CNT] = ' ';
            }
        }
        row = 1;
    }
    col = 0;                    /* reposistion cursor in correct place */
    lcd_control((char) (DS_ADDRESS+((row*40)+col)));
    }
    return;
}

void lcd_char (unsigned char c_out)
{
    wait_bf();                  /* Write character to display */
    DATA_P = c_out;           /* wait for the LCD ready flag */
    RW = WRITE;                /* place the data on the bus */
    RS = DATA;                /* set to write */
                                /* select the data register */
}
```

(continued)

**Listing 2: continued**

```

E = 1;          /* pulse the enable line */
E = 0;
return;        /* return, assume it worked */

void lcd_control(unsigned char c_out)
{
    wait_bf(); /* Write ctrl byte to display */
    DATA_P = c_out; /* wait for the LCD ready flag */
    RW = WRITE; /* place the data on the bus */
    RS = CONTROL; /* set to write */
    E = 1; /* select the control register */
    E = 0; /* pulse the enable line */
    return; /* return, assume it worked */
}

void wait_bf(void)

char bf;

DATA_P = 0xff;
RS = CONTROL;
RW = READ;
do {
    E = 1;
    bf = DATA_P.7;
    E = 0;

while (bf);
return;

void init_disp(void)

char i, j;

for (i=0; i<125; i++){ /* wait for power to settle and*/
    delay(); /* init display. Pause 15 ms */

RS = CONTROL;
RW = WRITE;
DATA_P = 0x38; /* 8 bit, 2 lines. 5 x 7 font */
for (j=0; j<4; j++){
    E = 1;
    E = 0;
    for (i=0; i<35; i++)
        delay(); /* 4.1 ms */

lcd_control(0x08); /* display/cursor/blink off */
lcd_control(0x01); /* clear display */
lcd_control(0x06); /* increment dd ram, no shift */
lcd_control(0x0C); /* disp on, cursor/blink off */

display( 'H'); /* Just a test for the PROM */
display( 'e');
display( 'l');
display( 'l');
display( 'o');
display( ' ');
display( 'W');
display( 'o');
display( 'r');
display( 'l');
display( 'd');
display( '!');
display( '\r'); /* return home */
return;

```



#214

## Motion Control

From the software to the motor shaft! Everything you need to automate a machine tool or an OEM motion job.

Artisan-CNC software is a true 4-axis machine controller w/user interface, not just a 'driver'. It takes tool-path files in industry std G-codes, HPGL, or Excellon PCB drill codes. It's been refined & updated since 1989.

Fancy features include: Continuous contouring, feedrate override, cutter radius compensation, bio 1" screen read & toolpath graphics; PLC macro language for automation & I/O xtrol, & much, much more.

Very fast assy language runs on a 386/486 VGA to 40K steps/sec/axis! Indexer-I/O card for ISA bus included in prii. 130 pg tech-ref manual & 500K Help.

Make a PCB drill, CNC a Mill in your shop, or use as OEM for machine tools, cut-to-length, labeling, etc..

Software from \$349 - \$849. Systems from \$2K-\$5K. Call us for the whole scoop. Use 800# to get catalog or to order \$15 60pg Tech Info-pak + demo disk. Call 612-641-1797 for a sales engineer.



**ah-ha!**  
Design Group, Inc

We answer questions. Take the first CNC step & call us at 612-641-1797  
Box 14519 Mpls, MN 55414, 800-473-7650, 10-6pm CST

#212

## Tech·Arts Home Automation

• **Text to Speech Board** serial I/O \$ 89

• **Temperature boards:** w/16 sensors \$239  
w/8 sensors plus 8 analog inputs \$179  
both boards: -40° to 146°F, serial I/O

• **Digital I/O ISA cards:** 48 I/O ports \$125  
96 I/O ports \$ 169  
192 I/O ports \$249

• **4-Port ISA Serial Board** w/1 6550s \$129  
com 1-8 & irq's 2,3,4,5,10,11,12,15

• **J-Servo controller board** serial I/O \$ 89

• **Windows NT TelcomFAX Personal** \$129

-Automatic Drapery Controller \$ 139

**Call for our complete catalog!**

support 315-455-1003

308 E. Molloy Rd Fax 315-455-5838

Syracuse, NY 13211 BBS 315-455-8728

Promotions and prices are subject to change without notice. Call for guarantee and warranty information.

**800-455-9853**  
Visa • MC • Amex • COD

#213

## Home Automation



## Worthington Distribution

**1-800-282-8864**

NO MINIMUMS

NO HANDLING FEES

TRUE DEALER PRICING

36 Gumbetown Road. Paupack, PA 18451

#214