CONVERT A LANGUAGE TRANSLATOR TO A LOW-COST COMPUTER TERMINAL

BY BILL PORTER

A simple hardware addition and some software allows portable language translator to be used as a computer peripheral

ALTHOUGH marketed as a language translator, the Nixdorf LK-3000 is actually a far more powerful device. It contains a microprocessor, RAM, ROM, an alphanumeric keyboard for data entry and an alphanumeric display for data readout; and it can apply these elements to a wide variety of other applications. One notable use of the LK-3000 is as an alphanumeric terminal. This article discusses hardware/software aspects of the translator and details how it can be converted for use as a hand-held computer terminal by adding simple hardware and some software.

Basic Operation. The language translator consists of a hand-held case that contains a keyboard, a 16-character one-line alphanumeric display, and a rechargeable battery. A module that contains an 8-bit CPU and a language-translation program in ROM plugs into the case via an edge connector. The combination thus formed is a dedicated microcomputer.

Since the edge connector carries data and address lines and the needed "handshake" (system control) hardware, it is possible to substitute a computer parallel I/O (input/output) port for the plug-in module. When this is done, the display section of the language translator can be used as a relatively low-cost (approximately $140) alphanumeric terminal. Since there are no hardware changes, the LK-3000 can still perform its language translation function when the language module is in place.

As shown in the photograph, the translator, which is sold in many retail establishments, is a hand-held, battery-powered device about 6" long, 3½" wide and 1" thick. The upper surface contains 27 dual-purpose data keys and six control keys with a window for displaying the 16-character alphanumeric readout. Each of the 16 characters is formed on a 16-segment (plus decimal point) LED readout.

All capabilities are determined by the plug-in module. A typical language translation module, shown in Fig. 1, consists of a 3870 CPU and the 64K ROM containing the language "package." The 3870 CPU includes an internal 2K ROM, 64 bytes of RAM, four I/O ports, a programmable timer, and a built-in crystal oscillator. It requires a +5-volt power supply.

System logic is shown in Fig. 2. If the plug-in module is ignored, a look at the circuit shows that, if the proper signals are applied to the edge connector, it is possible to cause this system to "look" like a simple alphanumeric terminal.

At present, there are three plug-ins that can be used to convert the translator into a terminal. Two are serial ports, while the easiest to implement is the parallel interface discussed here.

From a construction viewpoint, the parallel interface consists of a 16-conductor, color-coded ribbon cable interconnected between the LK-3000 edge connector and a conventional 16-pin DIP connector as shown in Table I, with the elements shown in Fig. 1.

In the breadboard stage, this technique was used to successfully interface

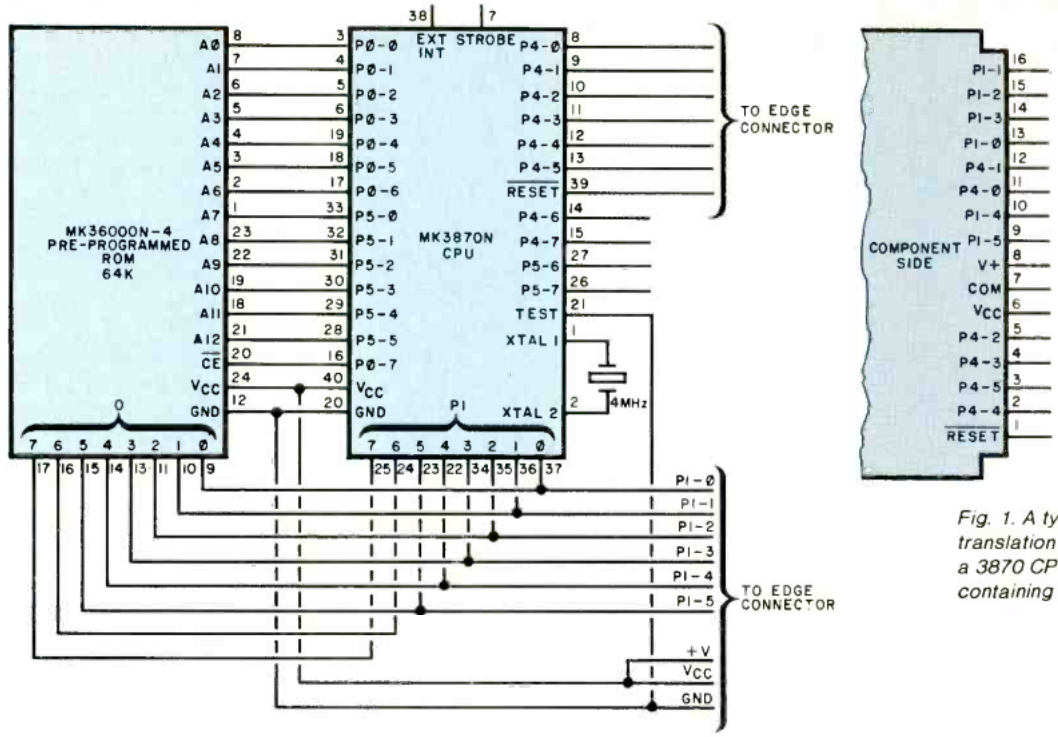

Fig. 1. A typical language translation module consists of a 3870 CPU and the 64K ROM containing the language package.

to 3870, 3872, F8, 8080, Z80, 8048, 8748, 8035, 8039 and 8085A systems.

Write Mode. Without the keyboard service routine contained in the plug-in language module, neither the LK-3000 keyboard, nor the external computer can write to the display. When using the computer parallel port, the ASCII data and position address must be latched to the data and address lines as defined in Table I. Once this data is correct, the display write strobe line $\overline{\text{DWSTB}}$, which is normally held in the high state, is dropped to the low state for at least 400 ns, then raised high. The display driver within the LK-3000 will do the rest. Display strobe subroutines for selected microprocessors are shown in Table II.

The following examples make use of the display. In both cases, the existence of a DISPLY (display) subroutine like that shown in Table II is presumed to be loaded within the computer.

For example, the 8085 code to write a space character to all 16 positions is:

```
CLDISP: MVI B, 15       ;position and
                         loop control
CLDLUP: MVI A, 20       ;an ASCII space
        CALL DISPLY     ;display and
                         increment
        MOV A, B        ;test if position
        INR A           ;went below zero
        JNZ CLDUP
        RET
```

The second example will show how to display a previously defined message on the right side of the display. This message is stored as a string of ASCII characters preceded by a length character. An example could be:

```
TITLE:  DB 6 'NIXDORF'
```

To use the following subroutine, the length parameter must be one less than the actual number of characters in the message. The routine will display the message found in memory location pointed to by H,L.

```
MESSAG: MOV B, M        ;get length and
                        ;starting position
MEXLUP: INX H           ;bump address
        MOV A, M        ;get next message
                        ;character
        CALL DISPLY     ;display and
                        ;decrement
        MOV A, B        ;test position
                        ;for below zero
        INR A
        JNZ MESLUP
        RET
```

Key Read. There are four basic steps required to read the keys on the display case. The first is to select one of the 8 rows of keyswitches, the second is to write a zero to the keyboard read strobe ($\overline{\text{KYRSTB}}$) which is normally kept high. Four column bits are then input on the data bus, with a zero indicating a closed switch. Finally, the keyboard read strobe is returned to the high state.

Although these four steps are all that is required for utilization of the keyboard, in practice there are other considerations. In the language translation mode, the problem of key scanning, code translation, buffering of data and keyswitch debounce are all handled by a short program within the language

TABLE I—
CONNECTIONS BETWEEN SYSTEM AND DIP SOCKET

Edge Contact	Name	Nominal Direction	DIP Pin Number
1	$\overline{\text{RESET}}$	←	1
2	$\overline{\text{KYRSTB}}$	→	16
3	$\overline{\text{DWSTB}}$	→	2
4	Address 3	→	15
5	Address 2	→	3
6	V+ IN	←	13
7	0 volts	→	4
8	+5 volts	→	14
9	Data 5	↔	5
10	Data 4	↔	12
11	Address 0	→	6
12	Address 1	→	11
13	Data 0	↔	7
14	Data 3	↔	10
15	Data 2	↔	8
16	Data 1	↔	9

The edge connector pads are 0.05" wide on 0.1" centers.
All signals TTL compatible.
Data 0-5: bidirectional data bus, 4 bits from keyboard, 6 bits to display.
Address 0-3: select one of 16 display positions, 1 of 8 keyboard rows.
$\overline{\text{DWSTB}}$: negative true display write strobe (400 ns min.) to write data to an addressed position.
$\overline{\text{KYRSTB}}$: negative true keyboard read signal to read a key from addressed keyboard row. This signal must be 0 (low) while column data is read.
$\overline{\text{RESET}}$: negative true indicating CRL key depressed. May be used as system reset, shift key, or as 33rd data key.

TABLE II—DISPLAY STROBE SUBROUTINES

```
'80/Z80/8085A                    8048                    3870

;n A                    ;data in A              ;data in SP 0
;ion in B               ;position in R1         ;position in SP1
LY: OUT DISPOT          DISPLY: XCH A, R1       DISPLY: LR A, 0
    MOV A, B                    ANL A, #15              COM
    ANI 0FH                     SWAP A                  OUTS 1
    ORI 30H                     OUTL P2, A              LR A, 1
    OUT DKCONT                  SWAP A                  COM
    ANI 0DFH                    XCH A, R1               NI 15
    OUT DKCONT                  ORL A, 0C0H             OUTS 4
    ORI 20H                     OUTL P1, A              OI H '20'
    OUT DKCONT                  ANL A, #7FH             OUTS 4
    DCR B                       OUTL P1, A              NI 15
    RET                         ORL A, #80H             OUTS 4
                                OUTL P1, A              OS 1
                                DEC R1                  POP
                                RET

;does display for one position
;OISPOT= display output
;DKCONT= display and keyboard control
;16 display position, 15 is left, 0 is right
```

ROM. Since this ROM is not used in this application, the remainder of this section will discuss how these items can be handled by the associated computer and its keyboard. No actual machine-language code will be listed, and the emphasis will be placed on flow charts so that the process can be implemented on any system.

The following assumptions control the design of the keyboard implementation: (1) all data keys should generate ASCII codes when operated; (2) the keyboard need not be fast; (3) key noise should be eliminated; (4) only one key at a time should be used; and (5) there should be a buffer between the key switch service function and the mainline program in the user computer system.

Figure 3 is the flow-chart required to satisfy the assumptions. Only 4 bytes of RAM (5 if debounce logic is used) are required to implement the keyboard active flag, keyboard current row, keyboard active column and the keyboard data buffer.

Translation of keyswitch row and column into ASCII is accomplished by constructing an index into a table like that shown in Table III. The translation value used for modules is row+8 (column)+32 (mode) where row is 0 through 7, column is 0 through 3 and mode is 0 or 1 depending on whether a shift-lock key is implemented.

The following example will help to explain how Table III and Fig. 3 make the keyboard useful. One presumption is that the keyswitch service subroutine will be called every 2 ms. When no key is depressed and the 2-ms interval expires, the routine is invoked. The routine examines its own flags and determines that the keyswitches are not active.

The next step is to add 1 to the working storage register that holds the representation of the current row. Since there are 8 rows on the keyboard, and since a 1-byte memory location can contain values up to 255, the results of the addition are "wrapped around." By ANDing the sum with a 7 (binary 0000 0111), the new value must be in the set 0,1,2,3,4,5,6,7. Each of these eight values represents a row. This current row value is used to enable (send power to) four switches representing the columns. The switches are wired to connect a column to ground.

Thus, open switches are input as binary 1 and closed switches as binary 0, with a single port used to read the data thus presented. If all four column bits are 1, no switches are closed and control is returned to the main program. As long as no keyswitches are depressed, the keyswitch service will continue to scan all eight rows (one row every 2 ms) looking for a new closure.

As soon as a keyswitch is found closed, the routine sets an internal flag to indicate an active state. This flag will be used to ensure that the key is fully released before any other key is accepted. The remainder of the new key processing is easier to explain if a typical example is used, as follows.

Assume that the "Y" key is found to be newly closed. This key could only be found when the current row is 5. (Table III, the ASCII code translation table, shows the mapping of the keys into the switch matrix.) When the columns are

TABLE III—ASCII TRANSLATION TABLE

```
LOC  OBJ    SEQ          SOURCE STATEMENT

            243 ;        KEYBOARD TRANSLATE TABLE
            244 ;
            245 ;
            246 ;  LOOKUP VALUE IS ROW + (8*COLUMN) + (32*MODE)
            247 ;
            248 ;                              ROW   COLUMN   GRAPHIC
            249 KBTBL:
00EE  41    250          DB      'A'     ;0    0        A
00EF  4A    251          DB      'J'     ;1    0        J
00F0  53    252          DB      'S'     ;2    0        S
00F1  45    253          DB      'E'     ;3    0        E
00F2  4E    254          DB      'N'     ;4    0        N
00F3  57    255          DB      'W'     ;5    0        W
00F4  3F    256          DB      3FH     ;6    0        ?
00F5  3D    257          DB      3DH     ;7    0        DEFINE
00F6  42    258          DB      'B'     ;0    1        B
00F7  4B    259          DB      'K'     ;1    1        K
00F8  54    260          DB      'T'     ;2    1        T
00F9  46    261          DB      'F'     ;3    1        F
00FA  4F    262          DB      'O'     ;4    1        O
00FB  58    263          DB      'X'     ;5    1        X
00FC  3C    264          DB      3CH     ;6    1        BACKSPACE
00FD  3B    265          DB      3BH     ;7    1        STEP
00FE  43    266          DB      'C'     ;0    2        C
00FF  4C    267          DB      'L'     ;1    2        L
0100  55    268          DB      'U'     ;2    2        U
0101  47    269          DB      'G'     ;3    2        G
0102  50    270          DB      'P'     ;4    2        P
0103  59    271          DB      'Y'     ;5    2        Y
0104  52    272          DB      'R'     ;6    2        R
0105  20    273          DB      ' '     ;7    2        SPACE
0106  44    274          DB      'D'     ;0    3        D
0107  4D    275          DB      'M'     ;1    3        M
0108  56    276          DB      'V'     ;2    3        V
0109  48    277          DB      'H'     ;3    3        H
010A  51    278          DB      'Q'     ;4    3        Q
010B  5A    279          DB      'Z'     ;5    3        Z
010C  49    280          DB      'I'     ;6    3        I
010D  3A    281          DB      3AH     ;7    3        FUNCTION
```

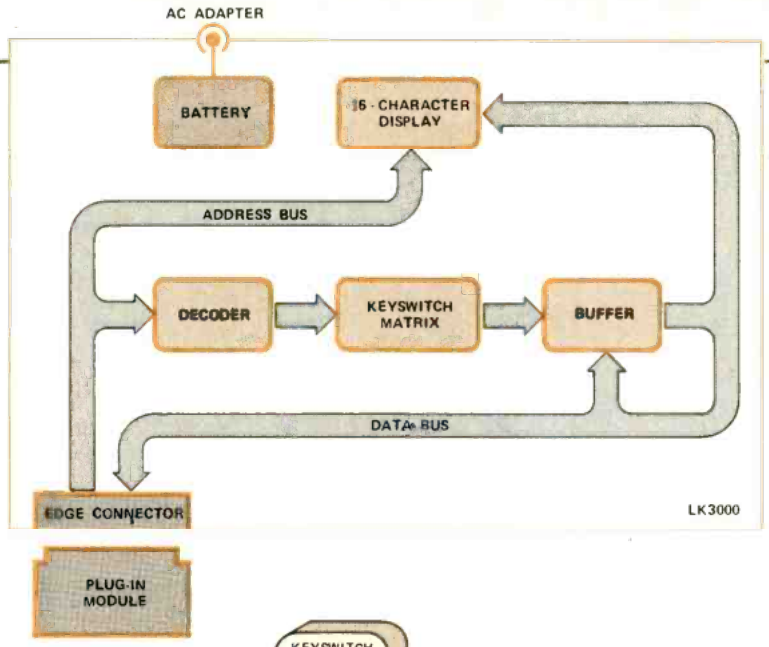

Fig. 2. Without the plug-in module, the syste... the appearance of a "dumb" alphanumeric t...

input, the value will be ... where the underlines indica... able (don't care) bit values. ... stores the row (5) and the ... The row and column are con... an index into the ASCII cod... tion table by first multiplying ... umn by 8 to yield 16, adding t... yield 21, then adding the base ... table in system memory to yield ... tual address of the character "Y".

The code value for "Y" is binary ... 1001, and this value is left in a mem... buffer for use by the main program. T... routine then returns control to the main ... program. The keyswitch will be auto... matically debounced by the scanning ... process during the active state process... ing as described next.

The next time that the 2-ms interval expires, the keyswitch service routine will determine that the keyboard is active. However, the current row will be 6 at that time, instead of the 5 where the key was found. This means that there will be more processing during that scan. After six more scans, the routine will find that the current row is the same 5 that was stored when the switch closure was found to be newly closed. The columns are read to see if column 2 is still a zero and if it is, no more processing is required since the key is still held down.

If a 1 is encountered, the key is released. In this case, the flag is set to indicate that the keyboard is inactive and that new keys may be accepted. Row 5 will not be examined again for 8 scan cycles, and each of the other rows will be examined first. If the "Y" key was bouncing upon release, it will not be examined again for 16-ms after it was found open. Since the bounce time for the switches is about 6 ms, debouncing has been successfully accomplished.

Thus, it can now be seen that with almost no hardware addition, and just using software routines, the LK-3000 can be used as a low-cost computer terminal. The basic language translator, including the recharger and case (but excluding a language module) is available for $140 from many retail outlets. The parallel interface, including ribbon cable and DIP connector, will be available in late summer for $35 from various computer stores. ◇

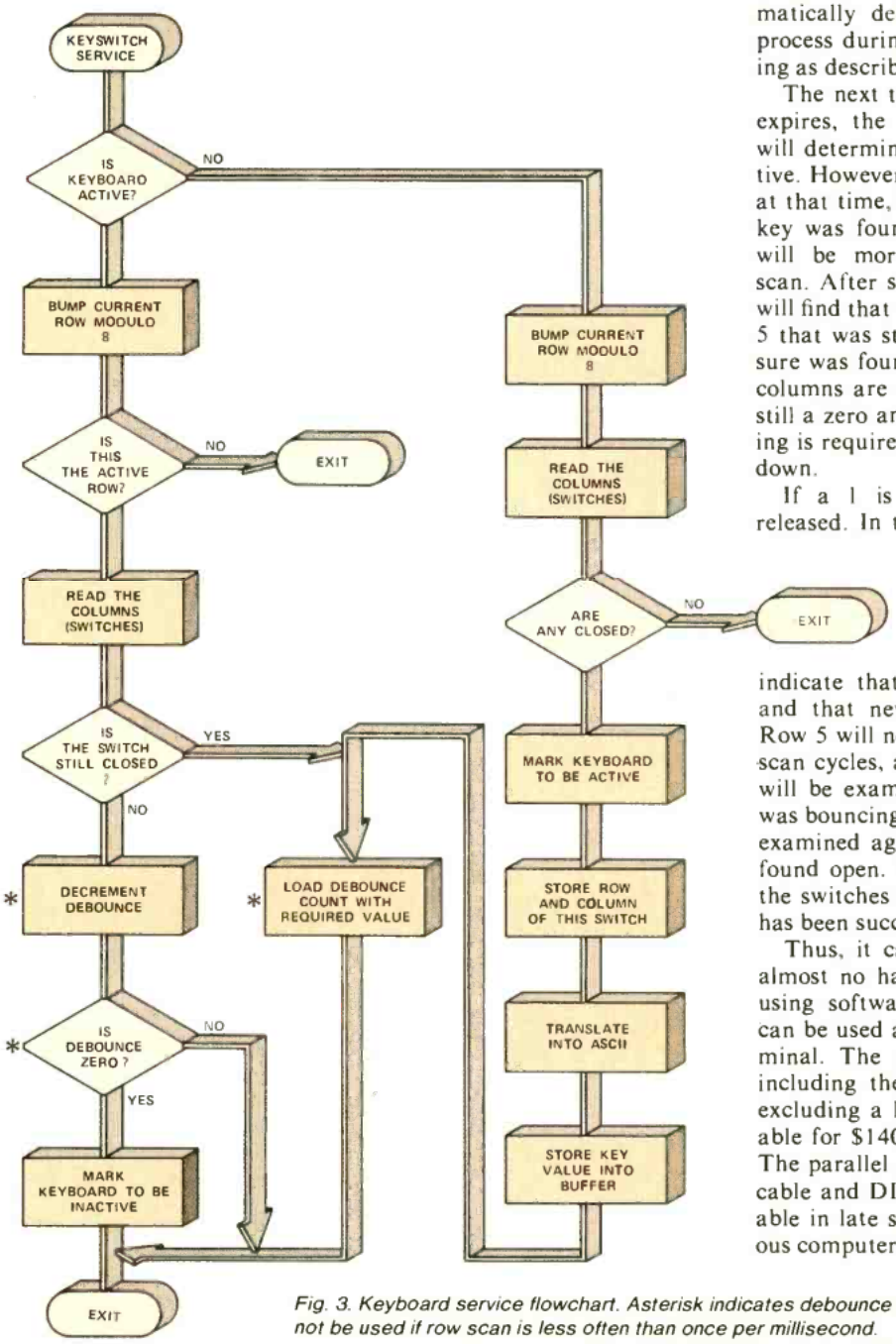

Fig. 3. Keyboard service flowchart. Asterisk indicates debounce logic need not be used if row scan is less often than once per millisecond.