Driving a Seven Segment Display with the Neuron® Chip

January 1995

LONWORKS® Engineering Bulletin

Introduction

This engineering bulletin describes how the Neuron Chip can be used to drive a seven-segment display controller chip, the Motorola MC14489. This device is used in Echelon's Gizmo2 and Motorola's Gizmo 3 multi-function I/O devices. The MC14489 can control up to five LED digits, each consisting of seven segments and a decimal point. No external current limiting resistors or drive transistors are required. The chip has a Serial Peripheral Interface (SPI), allowing for easy connection to the Neuron Chip's Neurowire port. This port can drive devices conforming to Motorola's SPI device interface and National Semiconductor's Microwire™ device interface. The engineering bulletin also presents software drivers written in the Neuron C programming language that display decimal numbers from binary data.

Schematic

The MC14489 can be connected to the Neuron 3150® or 3120xx® Chips as indicated in the following schematic. The Neuron Chip's Neurowire port in master mode uses pin `IO_8` as the clock pin, and `IO_9` as the serial output data pin. In this case, pin `IO_2` is used as the enable pin for the MC14489 display controller, but any of pins `IO_0` through `IO_7` could have been chosen, with the appropriate modification to the driver software.

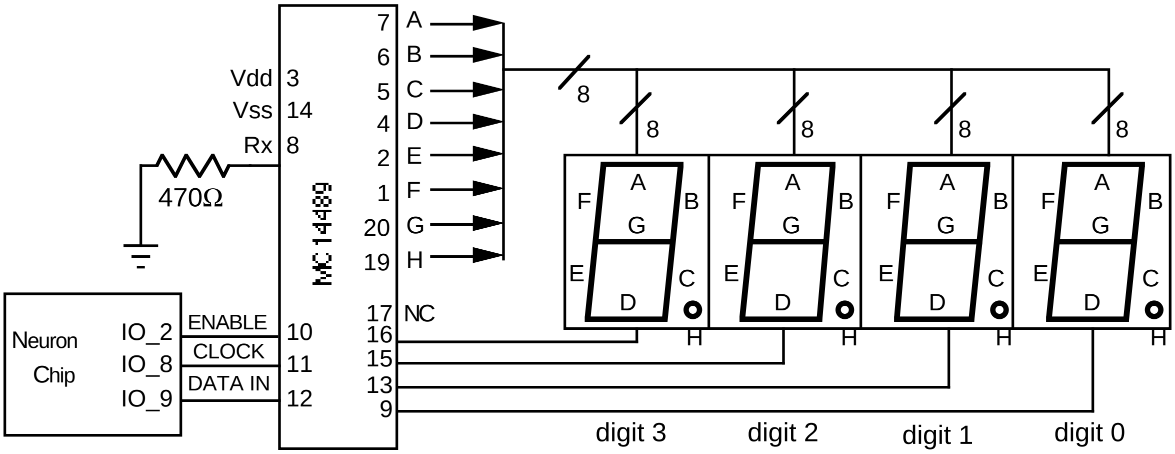


Figure 1. Seven-segment LEDs controlled by the Neuron Chip

The MC14489 is connected to four common-cathode seven-segment LED display devices. These are available from most manufacturers of opto-electronic devices, such as General Instruments, Hewlett-Packard, Industrial Electronic Engineers and William J. Purdy. The value of the current-limiting resistor connected between the Rx pin and ground depends on the application. If more than five digits are desired, several MC14489 devices may be connected in a cascade configuration, with the serial data being shifted out of one device into the next. See the Motorola MC14489 data sheet for more details. Other SPI or Microwire devices may be connected at the same time to the Neuron Chip's Neurowire port, provided each device has its own Enable pin.

Programming

The MC14489 has two write-only device registers controlled by the software on the Neuron Chip. The eight-bit configuration register shown in figure 2 contains bits that affect the decoding of the data in the 24-bit display register.

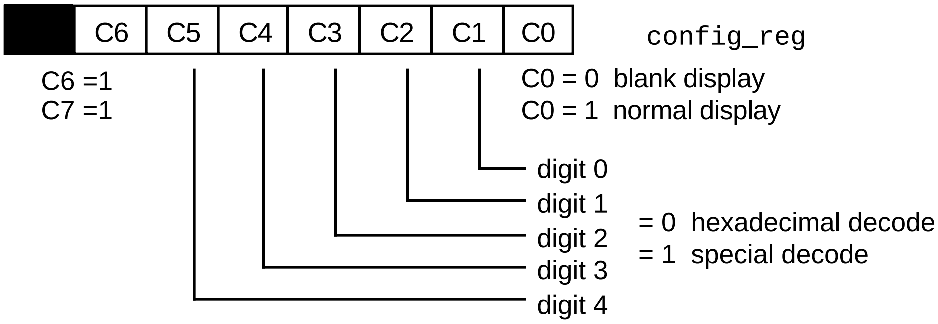
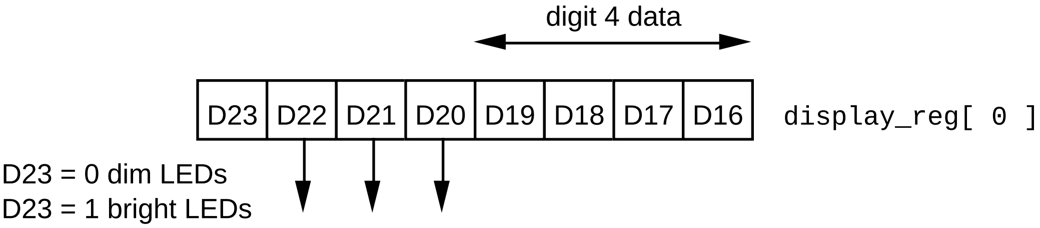


Figure 2. MC 14489 configuration register

The display register contains bits that define the display pattern of the LED digits and the decimal points as defined in figure 3.



0	0	0	No Decimal Points
0	0	1	DP - digit 0
0	1	0	DP - digit 1
0	1	1	DP - digit 2
1	0	0	DP - digit 3
1	0	1	DP - digit 4
1	1	0	DP - digits 0 and 1
1	1	1	DP - all digits

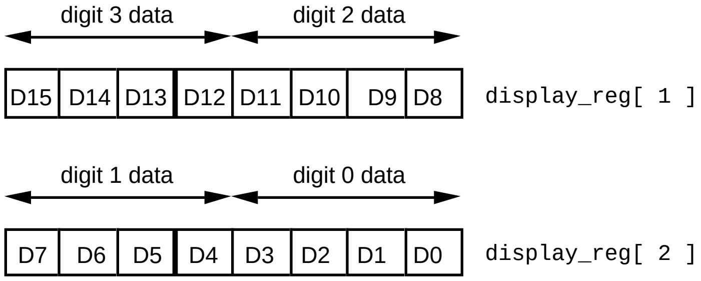


Figure 3. MC14489 display register

For the purposes of this application, there are two modes in which the data can be displayed. In hexadecimal mode, the four bits of data displayed in each digit are decoded as the hexadecimal digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. In special mode, certain other characters such as a space and a minus sign can be displayed. The table in figure 4 shows the patterns displayed in the two modes for all possible values of the data in the digit.

digit data	hex decode	special decode
0	0	
1	1	c
2	2	H
3	3	h
4	4	J
5	5	L
6	6	n
7	7	o
8	8	P
9	9	r
A	A	U
B	b	u
C	C	y
D	d	-
E	E	=
F	F	°

Figure 4. MC14489 display decoding

The software uses the Neurowire (SPI) function of the Neuron C programming language to write data to these two registers. When the application program issues an `io_out()` function call to the Neurowire device, the system software activates the chip select pin (in this case `IO_2`), and then clocks the data out on pin `IO_9`, using pin `IO_8` for the clock. The default rate of this serial data clock is 20kbps when the Neuron Chip input clock is 10MHz. When eight bits of data are clocked into the MC14489, these data bits are written to the configuration register. When 24 bits of data are clocked in, they are written to the display register. The software drivers presented in listings 1 and 2 always write both the configuration and the display register sequentially.

Software

Two listings are presented here. The first listing is for a simple decimal display function for positive numbers. Leading zeroes are not suppressed, so that, for example, the number 123 is displayed as 0123. There is also no error check for numbers that are out of range of the display. The built-in Neuron C library function `bin2bcd()` performs the actual data conversion.

The function `DspDisplayNumber(number, dpDigit)` displays unsigned decimal numbers with a decimal point to the right of the specified digit. Note that the special values `NO_DP` and `ALL_DPS` may be used as the digit number to illuminate none or all, respectively, of the decimal points.

Listing 1. Seven-segment display driver for positive numbers

```
///////////////// SEVEN SEGMENT DISPLAY DRIVER  ///////////////////

// This Neuron C #include file contains code to drive the Motorola MC14489
// seven-segment display controller chip, interfaced to the Neuron Chip
// using Neurowire master output mode.

// Pin IO_8 is the Neurowire clock, pin IO_9 is the serial output data
// Pin IO_2 is the chip select (may be modified).

// The function display_number( ) displays unsigned numbers
// with leading zeroes

/////////////////////// DECLARATIONS ////////////////////////////////

IO_8 neurowire master select(IO_2) ioSevenSeg;
IO_2 output bit io7SegSelect = 1;
#pragma ignore_notused io7SegSelect

struct bcd dspDataReg;           // 24 bits for 7-seg display reg
unsigned dspCfgReg;             //  8 bits for 7-seg  config reg

////////////////////////// DISPLAY DECIMAL NUMBER ///////////////////

void DspDisplayNumber(unsigned long number, int dpDigit) {
    dsoCfgReg = 0xc1;                        // Decimal decode all digits
    bin2bcd( number, &dspDataReg );          // Convert binary to decimal
    dspDataReg.d1 = 0x80 + dpDigit + 1;      // Set MS nibble for dec. pt.
    io_out (ioSevenSeg, &dspCfgReg, 8 );     // Update device registers
    io_out( ioSevenSeg, &dspDataReg, 24 );
}

#define NO_DP   -1
#define ALL_DPS  6
```

The second listing is for a more user-friendly decimal display. It handles positive, negative and out-of-range numbers, it suppresses leading zeroes where appropriate, and it allows the caller to specify the rightmost digit position. There is also a function to display strings consisting of the characters shown in Fig. 4, and a function to display temperatures values.

Before using this code, make sure that the number of digits in your display is specified correctly by the `NUM_DIGITS` parameter. For the display in the Gizmo 2 Multifunction I/O device, `NUM_DIGITS` should be set to 4. For the display in the Gizmo 3, set `NUM_DIGITS` to 5.

The software functions are divided into three groups; low-level functions, display image update functions, and high-level functions.

Low-level functions

The first group of functions provides low-level access to the display controller chip.

The function `DspClearImage()` clears a RAM copy of the configuration and display registers (the variables `dspCfgReg` and `dspDataReg`) to a state that displays all blank characters. It does this by setting all digits to special decode mode, and writing the data for the blank character to all digits.

The function `DspUpdateDisplay()` uses the Neurowire I/O device to write the contents of the RAM copy of the configuration and display registers to the actual MC14489 device registers. The Neurowire device is full-duplex, so that the `io_out()` operation which updates the hardware registers in the MC14889 also causes data to be shifted in from pin `IO_10` and stored in the RAM variables. Therefore a local copy of these variables is used for the `io_out()` operation, so that `DspUpdateDisplay()` may be called repeatedly without having to refresh the RAM copy of the variables.

Display image update functions

The second group of functions are routines that write into the RAM copy of the configuration and display registers. They do not update the hardware device registers.

The function `DspInsertData(int digitNumber, int data, boolean isNumeric)` writes the data nibble into the specified digit position in the RAM copy of the display register. Digits are numbered from right to left, with digit 0 being the rightmost (units) digit. If `isNumeric` is `TRUE`, numeric display decode is enabled for that digit.

The function DspInsertDecimal(int digitNumber, int number) writes the specified decimal (0 - 9) into the specified digit position in the RAM copy of the display register.

The function DspInsertDecimal2(int rightDigit, unsigned number) writes a two-digit decimal number (00 - 99) into the specified digit positions in the RAM copy of the display register.

The function DspInsertDP(int digitNumber) writes the appropriate bit in the RAM copy of the display register to illuminate the decimal point to the right of the specified digit.

The function DspInsertMinus(iny digitNumber) writes the appropriate values in the RAM copy of the display register to illuminate the minus sign (segment G) in the specified digit.

The function DspInsertChar(int digitNumber, char ch) writes the data value for an ASCII character in the RAM copy of the display register. If the ASCII character does not appear in Fig. 4, nothing is written, leaving the display blank for that digit.

The function DspInsertNumber(long number, int dpDigit, int rightDigit) updates the RAM copy of the display register to display a signed decimal number. The function illuminates a decimal point to the right of the specified digit. If the dpDigit parameter is NO_DP, no decimal point is illuminated. If the dpDigit parameter is ALL_DPS, all decimal points will be illuminated. The caller also specifies the right-most digit position of the displayed number. Display data to the right of this position are unchanged. If the number to be displayed does not fit in the specified field, all digits are set to the minus character.

High-level functions

The third group of functions forms complete display images and updates the display hardware.

The function DspDisplayBlanks() clears the display.

The function DspDisplayString(const char * pString, int dpDigit) causes the first 4 or 5 ASCII characters in the specified string to be displayed. Four characters are displayed for a Gizmo 2, and five characters are displayed for a Gizmo 3. The letters available in upper case are "A, C, E, F, H, I, J, L, O, P, S, U, Y, Z", and in lower case "b, c, d, h, l, n, o, r, u, y". Digits 0-9 are displayed, as well as the space ' ', degree '°', minus '-', and equals '=' special characters. If other letters, or more elegant letters are desired, an alphanumeric display should be used instead of a seven-segment display.

The function `DspDisplayNumber(long number, int dpDigit, int rightDigit)` displays positive or negative decimal numbers with a decimal point to the right of the specified digit, with suppression of leading zeroes. The special values `NO_DP` and `ALL_DPS` may be used as the decimal point digit number to illuminate none or all, respectively, of the decimal points. The parameter `rightDigit` indicates the digit position for the least significant digit of the displayed number. Numbers that do not fit into the specified field are displayed as all minus characters '-----'.

Table 1 shows some examples of the display produced by different input values with a four-digit display.

```
DspDisplayNumber(123, 0 ,0)         =>      1 2 3.
DspDisplayNumber(123, 1 ,0)         =>      1 2.3
DspDisplayNumber(123, 2 ,0)         =>      1.2 3
DspDisplayNumber(123, 3 ,0)         =>    0.1 2 3
DspDisplayNumber(123, NO_DP ,0)     =>      1 2 3
DspDisplayNumber(123, ALL_DPS ,0)   =>     .1.2.3.

DspDisplayNumber(-123, 0 ,0)        =>    - 1 2 3.
DspDisplayNumber(-123, 1 ,0)        =>    - 1 2.3
DspDisplayNumber(-123, 2 ,0)        =>    - 1.2 3
DspDisplayNumber(-123, 3 ,0)        =>    -.1 2 3
DspDisplayNumber(-123, NO_DP ,0)    =>    - 1 2 3
DspDisplayNumber(-123, ALL_DPS ,0) =>    -.1.2.3.
```

Table 1. Display produced by various input parameters to the DspDisplayNumber function.

The function `DspDisplayTemp(SNVT_temp temp, boolean dspFahrenheit)` displays temperature values in either Celsius or Fahrenheit, with one decimal place. For more details on the Standard Network Variable Type `SNVT_temp`, see the *SNVT Master List and Programmer's Guide.* As an example, `DspDisplayTemp(2940, FALSE)` displays '`20.0C`', and `DspDisplayTemp(2940, TRUE)` displays '`68.0F`'.

Listing 2. General-purpose seven-segment display driver

```
// DISPLAY.H -- Display handler for Gizmo 2 and Gizmo 3 LED displays
//
// Date last modified: 29-Dec-94
//
// Driver for the Motorola MC14489 seven-segment display controller chip
// on the Gizmo 2 and Gizmo 3
//
////////////////////////////// Include Files ////////////////////////////////

#include <stdlib.h>

/////////////////////////////// I/O Objects /////////////////////////////////

IO_8 neurowire master select(IO_2) ioSevenSeg;
IO_2 output bit io7SegSelect = 1;    // Initially unselected
#pragma ignore_notused io7SegSelect

//////////////////////////////// Constants //////////////////////////////////

//  The constant NUM_DIGITS depends on the display.
//  Gizmo 2 has 4 digits, Gizmo 3 has 5 digits.
//  The rightmost digit is numbered 0.   The default device is Gizmo 3

//  #define NUM_DIGITS 4    // uncomment this line for Gizmo 2

#ifndef NUM_DIGITS
#define NUM_DIGITS  5       // for Gizmo 3
#endif

//  The following constants may be used as the dpDigit argument for
//  DspDisplayNumber( ) and DspDisplayString( )

#define NO_DP   -1    // display number without  decimal point
#define ALL_DPS -2    // display number with all decimal points

///////////////////////////// Global Variables ///////////////////////////////

unsigned dspCfgReg;         //  8 bits for 7-seg  config reg
unsigned dspDataReg[3];     // 24 bits for 7-seg display reg

///////////////////////////////// Functions /////////////////////////////////

//  DspUpdateDisplay( ) -- update device hardware registers

void DspUpdateDisplay(void) {
    static unsigned cfgRegCopy;
    static unsigned dataRegCopy[3];

    cfgRegCopy = dspCfgReg;
    memcpy(dataRegCopy, dspDataReg, 3);     // copy images
    io_out(ioSevenSeg, &cfgRegCopy, 8);     // shift out
    io_out(ioSevenSeg, dataRegCopy, 24);
}
```

```
/////////////////////////////////////////////////////////////////////////////

//  DspClearImage( ) -- clear image of device registers to all blanks

void DspClearImage(void) {
    dspDataReg[0] = 0x80;   // max brightness
    dspDataReg[1] = 0;      // blanks on all digits
    dspDataReg[2] = 0;
    dspCfgReg = 0xFF;       // special decode on banks 1-5, normal mode
}

/////////////////////////////////////////////////////////////////////////////

//  DspInsertData( ) -- insert a nibble (0-F) in specified digit

void DspInsertData(int digitNumber, int data, boolean isNumeric) {
    dspDataReg[2 - digitNumber / 2] |=              // insert nibble
        (digitNumber & 1) ? (data << 4) : data;
    if (isNumeric)
        dspCfgReg &= ~(1 << (digitNumber + 1));     // set numeric decode
}

/////////////////////////////////////////////////////////////////////////////

//  DspInsertDecimal( ) -- insert a decimal number (0-9) in specified digit

void DspInsertDecimal(int digitNumber, int number) {
    DspInsertData(digitNumber, number, TRUE);
}

///////////////////////////////////////////////////////////////

//  DspInsertDecimal2( ) - insert a two-digit decimal number (00-99)

void DspInsertDecimal2(int rightDigit, unsigned number) {
    DspInsertDecimal(rightDigit++, number % 10);
    DspInsertDecimal(rightDigit, number / 10);
}

#pragma ignore_notused DspInsertDecimal2

/////////////////////////////////////////////////////////////////////////////

//  DspInsertDP( ) -- insert a decimal point to right of specified digit

void DspInsertDP(int digitNumber) {
    dspDataReg[0] |= (digitNumber + 1) << 4;        // set DP bit
}

///////////////////////////////////////////////////////////////

//  DspInsertMinus( ) -- insert a minus sign in specified digit

void DspInsertMinus(int digitNumber) {
    DspInsertData(digitNumber, 0xD, FALSE);
}
```

```
///////////////////////////////////////////////////////////////

//  DspInsertChar( ) -- insert a character from the Gizmo character set

void DspInsertChar(int digitNumber, char ch) {
    // Table of ASCII characters that may be displayed in 7 segments

    typedef struct {
         unsigned      charCode  : 4;
         unsigned      isNumeric : 1;
         char          asciiChar;
    } charTable;       // Gizmo character set table

    static const charTable CHAR_TABLE[ ] = {
    // Sorted in ASCII collating sequence.  '%' is displayed as a degree sign

        { 0x0, FALSE, ' ' }, { 0xF, FALSE, '%' }, { 0xD, FALSE, '-' },
        { 0x0, TRUE,  '0' }, { 0x1, TRUE,  '1' }, { 0x2, TRUE,  '2' },
        { 0x3, TRUE,  '3' }, { 0x4, TRUE,  '4' }, { 0x5, TRUE,  '5' },
        { 0x6, TRUE,  '6' }, { 0x7, TRUE,  '7' }, { 0x8, TRUE,  '8' },
        { 0x9, TRUE,  '9' }, { 0xE, FALSE, '=' },
        { 0xA, TRUE,  'A' }, { 0xC, TRUE,  'C' }, { 0xE, TRUE,  'E' },
        { 0xF, TRUE,  'F' }, { 0x2, FALSE, 'H' }, { 0x1, TRUE,  'I' },
        { 0x4, FALSE, 'J' }, { 0x5, FALSE, 'L' }, { 0x0, TRUE,  'O' },
        { 0x8, FALSE, 'P' }, { 0x5, TRUE,  'S' }, { 0xA, FALSE, 'U' },
        { 0xC, FALSE, 'Y' }, { 0x2, TRUE,  'Z' }, { 0xB, TRUE,  'b' },
        { 0x1, FALSE, 'c' }, { 0xD, TRUE,  'd' }, { 0x3, FALSE, 'h' },
        { 0x1, TRUE,  'l' }, { 0x6, FALSE, 'n' }, { 0x7, FALSE, 'o' },
        { 0x9, FALSE, 'r' }, { 0xB, FALSE, 'u' }, { 0xC, FALSE, 'y' }
    };
    const charTable *pLow, *pHigh, *pHalf;
    char tableChar;

    pLow = CHAR_TABLE;              // set up for binary search
    pHigh = CHAR_TABLE + sizeof(CHAR_TABLE) / sizeof(charTable);
    while (pHigh > pLow + 1) {
        pHalf = pLow + (pHigh - pLow) / 2;  // probe the mid-point
        tableChar = pHalf->asciiChar;
        if (ch == tableChar) {
            DspInsertData(digitNumber, pHalf->charCode, pHalf->isNumeric);
            return;
        }
        if (ch > tableChar) pLow = pHalf;
        else pHigh = pHalf;
    }
}

/////////////////////////////////////////////////////////////////////////////

// DspInsertNumber( ) -- insert signed decimal number

void DspInsertNumber(long number, int dpDigit, int rightDigit) {
    static const struct rangeTable {      // min and max displayable numbers
        long   minNum;
        long   maxNum;
    } RANGE_TABLE[] = {
        {  0, 9 },          // 1 digit
        { -9, 99 },         // 2 digits
        { -99, 999 },       // 3 digits
        { -999, 9999 },     // 4 digits
        { -9999, 32767 }    // 5 digits
    };
    const struct rangeTable *pRange;

    DspInsertDP(dpDigit);       // display decimal point

    pRange = &RANGE_TABLE[NUM_DIGITS - 1 - rightDigit];
    if (number > pRange->maxNum || number < pRange->minNum) {
        while (rightDigit < NUM_DIGITS)    // display "-----" for overrange
            DspInsertMinus(rightDigit++);
        return;
    }

    dpDigit = max(dpDigit, rightDigit);
    if (number < 0) {
        DspInsertMinus(NUM_DIGITS - 1);   // leading minus sign
        number = - number;      // get absolute value
        if (dpDigit == NUM_DIGITS - 1) dpDigit--;    // allow -.xxxx format
    }

    while (number || rightDigit <= dpDigit) {
                // convert binary to decimal with leading zero suppress
        DspInsertDecimal(rightDigit++, (int)(number % 10));
        number /= 10;
    }
}

/////////////////////////////////////////////////////////////////////////////

//  DspDisplayNumber( ) -- display signed decimal number

void DspDisplayNumber(long number, int dpDigit, int rightDigit) {
    DspClearImage();       // clear image of display registers
    DspInsertNumber(number, dpDigit, rightDigit);   // convert number
    DspUpdateDisplay() ;   // update hardware
}

#pragma ignore_notused DspDisplayNumber

///////////////////////////////////////////////////////////////

//  DspDisplayTemp( ) -- Display a temperature in Fahrenheit or Celsius
//                       For Gizmo 2, range is  -9.9 to  99.9 degrees
//                       For Gizmo 3, range is -99.9 to 999.9 degrees

void DspDisplayTemp(SNVT_temp temp, boolean dspFahrenheit) {
    long value;
    value = temp - 2740;       // tenths of degrees C
    if (dspFahrenheit)         // display in tenths of degrees F
        value = value * 9 / 5 + 320;
    DspClearImage();
    DspInsertNumber(value, 2, 1);      // xxx.xX
    DspInsertData(0, dspFahrenheit ? 0xF : 0xC, TRUE);   // display F or C
    DspUpdateDisplay();
}

#pragma ignore_notused DspDisplayTemp

///////////////////////////////////////////////////////////////

//  DspDisplayString( ) -- display first NUM_DIGITS characters of a string

void DspDisplayString(const char *pString, int dpDigit) {
    int digitNumber;

    DspClearImage();        // all non-displayable chars show as blank
    for (digitNumber = NUM_DIGITS - 1; digitNumber >= 0; digitNumber--)
        DspInsertChar(digitNumber, *pString++);

    DspInsertDP(dpDigit);  // display decimal point
    DspUpdateDisplay();    // update hardware
}

#pragma ignore_notused DspDisplayString

/////////////////////////////////////////////////////////////////////////////

//  DspDisplayBlanks( ) -- display all blanks

void DspDisplayBlanks(void) {
    DspClearImage();
    DspUpdateDisplay();
}

#pragma ignore_notused DspDisplayBlanks
```

Disclaimer

Echelon Corporation assumes no responsibility for any errors contained herein. No part of this document may be reproduced, translated, or transmitted in any form without permission from Echelon.

Part Number 005-0014-01 Rev. C

Echelon Corporation
4015 Miranda Avenue
Palo Alto, CA 94304
Telephone (415) 855-7400
Fax (415) 856-6153

Echelon Europe Ltd
Elsinore House
77 Fulham Palace Road
London W6 8JA
England
Telephone +44-81-563-7077
Fax +44-81-563-7055

Echelon Japan K.K.
Kamino Shoji Bldg. 8F
25-13 Higashi-Gotanda 1-chome
Shinagawa-ku, Tokyo 141
Telephone (03) 3440-7781
Fax (03) 3440-7782