Building a Smart-Necklace

Part 2: More Animations

by
Bruce E. Hall, W8BH

1) INTRODUCTION

In Part 1 of this series I described how I built a necklace for my daughter, based on the ATtiny4313 microcontroller and a 5x7 LED matrix. Since then I have been playing around with the code to see what interesting displays I could create. This article describes a few of the animated displays.

2) PIXELS, COLUMNS and ROWS

To create any graphics, we need a way of individually addressing each pixel. The first and simplest method would be to turn on (or off) the physical pins associates with the desired LED. In a non-multiplexed display, to turn on a cathode-column LED at (0,0) we would take the pin for column 0 low and the pin for row 0 high. A non-multiplexed matrix display cannot display arbitrary pixel patterns, however. In Part 1, we multiplexed the display, using an interrupt routine. The display pixels are mapped onto an array of 5 bytes, as shown here. Each byte represents a column of the display. In each byte/column, bits 0-6 represent the individual pixels. For example, the orange pixel at (5,2) is the 5th bit of byte 2.

	byte 0	byte 1	byte 2	byte 3	byte 4
Bit0					
Bit1					
Bit2					
Bit3					
Bit4					
Bit5					
Bit6					
Bit7					

To set an individual pixel, the code is simple:

```
buf[x] |= (1<<y);
```

Select the desired column, byte x, which is buf[x].

Then use a logical OR to set the bit. Clearing a pixel requires a logical AND with a 0 at the bit position. For this we use the negative function to create a bit mask:

```
void ClearPixel(byte x, byte y)
{
        buf[x] &= ~(1<<y);                      // turn off led at (x,y)
}
```

Setting and clearing columns is easy: just set buf[x] with all 1's or all 0'x. Setting a row requires setting the same (row) pixel in each column. You could write it like this:

```
void SetRow(byte n)
{
        for (byte i=0;i<COLS;i++)               // for each column
        {
                SetPixel(i,n);                  // turn on the n row pixel
        }
}
```

Or like this:

```
void SetRow(byte n)
{
        for (byte i=0;i<COLS;i++)
        {
                buf[i] |= (1<<n);               // turn on all leds in row
        }
}
```

The first method is more human-readable; the second method doesn't require a function call and may take less memory.

3) CYLONS

Here is a simple, fun animation: light a single LED in back & forth motion. Remember the Cylon creatures from Battlestar Galactica? The car KITT from Knight Rider also used this effect. Creating horizontal, back & forth motion takes 2 for loops: the first loop moves the LED to the right, and the second moves it back to the left:

```
void OnePixel(byte x,byte y)     // turn on led at (x,y)
{
        DisplaySymbol(0);        // everything off first
        SetPixel(x,y);           // then set the pixel
        DelayCS(8);              // and wait
}


void HorizontalCylon()           // side-to-side animation
        {
                int x;
                while(1)         // forever…
                {
                        for (x=0;x<4;x++)  OnePixel(x,0);       // left to right
                        for (x=4;x>0;x--)  OnePixel(x,0);       // right to left
```

```
                }
            }
```

The helper function, OnePixel, makes sure that only one pixel is lit at a time, and sets the animation speed.

4) BUT WAIT, THERE'S MORE

I added a bunch of simple animations to the code: alternating patterns, morse code, blinking eyes, raindrops, marquis animation, zigzags, etc. Check out the source code.

5) SOURCE CODE:

```
// ---------------------------------------------------------------------------
//  BMATRIX:  NECKLACE with LED-MATRIX that displays text messages
//
//  Based on Nuts&Volts Jul 2013 article: "Smart Necklace", p. 40
//
//  Author   :  Bruce E. Hall <bhall66@gmail.com>
//  Website  :  w8bh.net
//  Version  :  1.1
//  Date     :  04 Aug 2013
//  Target   :  ATTINY4313 microcontroller
//  Language :  C, using AVR studio 6
//  Size     :  2666 bytes, using -O1 optimization
//
// ---------------------------------------------------------------------------
//
//     Uses 5x7 LED MATRIX Displays LTP-757 or LTP-747
//
//                    ATTINY4313            b  PORTA   PORTB   PORTD
//  LED FUNCTION  to  PORT                  ------------------------
//  ----------------------------             7  -       -       -
//  Col 0  - PD1    // Row 0 - PB6           6  -       row0    -
//  Col 1  - PA0    // Row 1 - PB5           5  -       row1    -
//  Col 2  - PB4    // Row 2 - PA1           4  -       col2    row6
//  Col 3  - PB1    // Row 3 - PB3           3  -       row3    row5
//  Col 4  - PB2    // Row 4 - PD2           2  -       col4    row4
//  Col 4  - PB2    // Row 5 - PD3           1  row2    col3    col0
//                  // Row 6 - PD4           0  col1    -       -
//
//  For the LTP-757 (cathode column) display,
//  Columns are active LOW;  to set a col, PORTx &= ~(1<<bit)
//  Rows are active HIGH; to set a row, PORTx |= (1<<bit)
//
//  For the LTP-747 (anode column) display,
//  Columns are active HIGH;  to set a col, PORTx |= (1<<bit)
//  Rows are active LOW; to set a row,  PORTx &= ~(1<<bit)
//
//  Fuse settings:  4 MHz osc with 65 ms Delay, SPI enable; *NO* clock/8


//     ---------------------------------------------------------------------------
//     DEFINES

#define F_CPU          4000000L              // run CPU at 4 MHz
#define ROWS           7                     // LED matrix has 7 rows, 5 columns
#define COLS           5
#define CHARSPACING    1                     // spacing, in columns, between chars
#define SCROLLDELAY    15                    // delay in cs between column shifts
#define FLASHDELAY     17                    // delay in cs between symbol flashes
```

```
#define BEATDELAY      30                    // delay in cs between heartbeats
#define HEARTCHAR      99
#define SMILECHAR      107
#define FROWNCHAR   108
#define TEXT0 "--Sarah Hall-- "
#define TEXT1 "I Love You! "
#define TEXT2 "Ich Liebe Dich! "

#define ClearBit(x,y) x &= ~_BV(y)           // equivalent to cbi(x,y)
#define SetBit(x,y) x |= _BV(y)              // equivalent to sbi(x,y)

//      ------------------------------------------------------------------------
//      INCLUDES

#include <avr/io.h>                          // deal with port registers
#include <avr/interrupt.h>                   // deal with interrupt calls
#include <avr/pgmspace.h>                    // put character data into progmem
#include <util/delay.h>                      // used for _delay_ms function
#include <string.h>                          // string manipulation routines
#include <avr/sleep.h>                       // used for sleep functions

//      ------------------------------------------------------------------------
//      TYPEDEFS

typedef uint8_t byte;                        // I just like byte & sbyte better
typedef int8_t sbyte;

//      ------------------------------------------------------------------------
//      GLOBAL VARIABLES

byte buf[10];                                // display buffer; each byte = 1 column
                                             // buf[0] is the left-most column (col0)
                                             // buf[4] is the right-most column (col4)
                                             // buf[5]..buf[9] are scrolled onto display

byte curCol;                                 // current column; values 0-4

const byte FONT_CHARS[113][5] PROGMEM =
{
        { 0x00, 0x00, 0x00, 0x00, 0x00 },    // (space)
        { 0x00, 0x00, 0x5F, 0x00, 0x00 },    // !
        { 0x00, 0x07, 0x00, 0x07, 0x00 },    // "
        { 0x14, 0x7F, 0x14, 0x7F, 0x14 },    // #
        { 0x24, 0x2A, 0x7F, 0x2A, 0x12 },    // $
        { 0x23, 0x13, 0x08, 0x64, 0x62 },    // %
        { 0x36, 0x49, 0x55, 0x22, 0x50 },    // &
        { 0x00, 0x05, 0x03, 0x00, 0x00 },    // '
        { 0x00, 0x1C, 0x22, 0x41, 0x00 },    // (
        { 0x00, 0x41, 0x22, 0x1C, 0x00 },    // )
        { 0x08, 0x2A, 0x1C, 0x2A, 0x08 },    // *
        { 0x08, 0x08, 0x3E, 0x08, 0x08 },    // +
        { 0x00, 0x50, 0x30, 0x00, 0x00 },    // ,
        { 0x08, 0x08, 0x08, 0x08, 0x08 },    // -
        { 0x00, 0x60, 0x60, 0x00, 0x00 },    // .
        { 0x20, 0x10, 0x08, 0x04, 0x02 },    // /
        { 0x3E, 0x51, 0x49, 0x45, 0x3E },    // 0
        { 0x00, 0x42, 0x7F, 0x40, 0x00 },    // 1
        { 0x42, 0x61, 0x51, 0x49, 0x46 },    // 2
        { 0x21, 0x41, 0x45, 0x4B, 0x31 },    // 3
        { 0x18, 0x14, 0x12, 0x7F, 0x10 },    // 4
        { 0x27, 0x45, 0x45, 0x45, 0x39 },    // 5
        { 0x3C, 0x4A, 0x49, 0x49, 0x30 },    // 6
        { 0x01, 0x71, 0x09, 0x05, 0x03 },    // 7
        { 0x36, 0x49, 0x49, 0x49, 0x36 },    // 8
        { 0x06, 0x49, 0x49, 0x29, 0x1E },    // 9
        { 0x00, 0x36, 0x36, 0x00, 0x00 },    // :
        { 0x00, 0x56, 0x36, 0x00, 0x00 },    // ;
        { 0x00, 0x08, 0x14, 0x22, 0x41 },    // <
        { 0x14, 0x14, 0x14, 0x14, 0x14 },    // =
        { 0x41, 0x22, 0x14, 0x08, 0x00 },    // >
        { 0x02, 0x01, 0x51, 0x09, 0x06 },    // ?
```

```
{ 0x32, 0x49, 0x79, 0x41, 0x3E },     // @
{ 0x7E, 0x11, 0x11, 0x11, 0x7E },     // A
{ 0x7F, 0x49, 0x49, 0x49, 0x36 },     // B
{ 0x3E, 0x41, 0x41, 0x41, 0x22 },     // C
{ 0x7F, 0x41, 0x41, 0x22, 0x1C },     // D
{ 0x7F, 0x49, 0x49, 0x49, 0x41 },     // E
{ 0x7F, 0x09, 0x09, 0x01, 0x01 },     // F
{ 0x3E, 0x41, 0x41, 0x51, 0x32 },     // G
{ 0x7F, 0x08, 0x08, 0x08, 0x7F },     // H
{ 0x00, 0x41, 0x7F, 0x41, 0x00 },     // I
{ 0x20, 0x40, 0x41, 0x3F, 0x01 },     // J
{ 0x7F, 0x08, 0x14, 0x22, 0x41 },     // K
{ 0x7F, 0x40, 0x40, 0x40, 0x40 },     // L
{ 0x7F, 0x02, 0x04, 0x02, 0x7F },     // M
{ 0x7F, 0x04, 0x08, 0x10, 0x7F },     // N
{ 0x3E, 0x41, 0x41, 0x41, 0x3E },     // O
{ 0x7F, 0x09, 0x09, 0x09, 0x06 },     // P
{ 0x3E, 0x41, 0x51, 0x21, 0x5E },     // Q
{ 0x7F, 0x09, 0x19, 0x29, 0x46 },     // R
{ 0x46, 0x49, 0x49, 0x49, 0x31 },     // S
{ 0x01, 0x01, 0x7F, 0x01, 0x01 },     // T
{ 0x3F, 0x40, 0x40, 0x40, 0x3F },     // U
{ 0x1F, 0x20, 0x40, 0x20, 0x1F },     // V
{ 0x7F, 0x20, 0x18, 0x20, 0x7F },     // W
{ 0x63, 0x14, 0x08, 0x14, 0x63 },     // X
{ 0x03, 0x04, 0x78, 0x04, 0x03 },     // Y
{ 0x61, 0x51, 0x49, 0x45, 0x43 },     // Z
{ 0x00, 0x00, 0x7F, 0x41, 0x41 },     // [
{ 0x02, 0x04, 0x08, 0x10, 0x20 },     // "\"
{ 0x41, 0x41, 0x7F, 0x00, 0x00 },     // ]
{ 0x04, 0x02, 0x01, 0x02, 0x04 },     // ^
{ 0x40, 0x40, 0x40, 0x40, 0x40 },     // _
{ 0x00, 0x01, 0x02, 0x04, 0x00 },     // `
{ 0x20, 0x54, 0x54, 0x54, 0x78 },     // a
{ 0x7F, 0x48, 0x44, 0x44, 0x38 },     // b
{ 0x38, 0x44, 0x44, 0x44, 0x20 },     // c
{ 0x38, 0x44, 0x44, 0x48, 0x7F },     // d
{ 0x38, 0x54, 0x54, 0x54, 0x18 },     // e
{ 0x08, 0x7E, 0x09, 0x01, 0x02 },     // f
{ 0x08, 0x14, 0x54, 0x54, 0x3C },     // g
{ 0x7F, 0x08, 0x04, 0x04, 0x78 },     // h
{ 0x00, 0x44, 0x7D, 0x40, 0x00 },     // i
{ 0x20, 0x40, 0x44, 0x3D, 0x00 },     // j
{ 0x00, 0x7F, 0x10, 0x28, 0x44 },     // k
{ 0x00, 0x41, 0x7F, 0x40, 0x00 },     // l
{ 0x7C, 0x04, 0x18, 0x04, 0x78 },     // m
{ 0x7C, 0x08, 0x04, 0x04, 0x78 },     // n
{ 0x38, 0x44, 0x44, 0x44, 0x38 },     // o
{ 0x7C, 0x14, 0x14, 0x14, 0x08 },     // p
{ 0x08, 0x14, 0x14, 0x18, 0x7C },     // q
{ 0x7C, 0x08, 0x04, 0x04, 0x08 },     // r
{ 0x48, 0x54, 0x54, 0x54, 0x20 },     // s
{ 0x04, 0x3F, 0x44, 0x40, 0x20 },     // t
{ 0x3C, 0x40, 0x40, 0x20, 0x7C },     // u
{ 0x1C, 0x20, 0x40, 0x20, 0x1C },     // v
{ 0x3C, 0x40, 0x30, 0x40, 0x3C },     // w
{ 0x44, 0x28, 0x10, 0x28, 0x44 },     // x
{ 0x0C, 0x50, 0x50, 0x50, 0x3C },     // y
{ 0x44, 0x64, 0x54, 0x4C, 0x44 },     // z
{ 0x00, 0x08, 0x36, 0x41, 0x00 },     // {
{ 0x00, 0x00, 0x7F, 0x00, 0x00 },     // |
{ 0x00, 0x41, 0x36, 0x08, 0x00 },     // }
{ 0x08, 0x08, 0x2A, 0x1C, 0x08 },     // ->
{ 0x08, 0x1C, 0x2A, 0x08, 0x08 },     // <-
{ 0xFF, 0x41, 0x5D, 0x41, 0xFF },     // 096:  psycho 2
{ 0x00, 0x3E, 0x22, 0x3E, 0x00 },     // 097:  psycho 1
{ 0x06, 0x15, 0x69, 0x15, 0x06 },     // 098:  nuke
{ 0x0C, 0x1E, 0x3C, 0x1E, 0x0C },     // 099:  solid heart
{ 0x0C, 0x12, 0x24, 0x12, 0x0C },     // 100:  outline heart
{ 0x0A, 0x00, 0x55, 0x00, 0x0A },     // 101:  flower
{ 0x08, 0x14, 0x2A, 0x14, 0x08 },     // 102:  diamond
```

```
        { 0x07, 0x49, 0x71, 0x49, 0x07 },     // 103:  cup
        { 0x22, 0x14, 0x6B, 0x14, 0x22 },     // 104:  star2
        { 0x36, 0x36, 0x08, 0x36, 0x36 },     // 105:  star3
        { 0x0F, 0x1A, 0x3E, 0x1A, 0x0F },     // 106:  fox
        { 0x20, 0x44, 0x40, 0x44, 0x20 },     // 107:  smile
        { 0x40, 0x24, 0x20, 0x24, 0x40 },     // 108:  frown
        { 0x20, 0x40, 0x40, 0x40, 0x20 },     // 109:  mouth
        { 0x00, 0x04, 0x00, 0x04, 0x00 },     // 110:  eyes
        { 0x55, 0x2A, 0x55, 0x2A, 0x55 },     // 111:  50% I
        { 0x2A, 0x55, 0x2A, 0x55, 0x2A }      // 112:  50% II
};

//      ------------------------------------------------------------------------
//      INTERRUPT SERVICE ROUTINES
//
//  Function:  Light a column on the LED matrix display, according to contents
//             of display buffer.  buf[0] = leftmost column; buf[4] = rightmost
//
//  Each ISR below corresponds to a display type.   Choose the ISR for your
//  display and comment out the others.


/*
//  INTERRUPT SERVICE ROUTINE for LTP757 (Cathode Column) Displays
//

ISR (TIMER0_COMPA_vect)
{
        if (++curCol >= COLS)                 // advance column counter
                curCol = 0;

        // turn off all LEDS, by taking cathode (column) pins high
        PORTA = 0x01;
        PORTB = 0x16;
        PORTD = 0x02;

        // turn on individual row bits in this column
        char i = buf[curCol];
        if (i & _BV(0))  SetBit(PORTB,6);                    //  PORTB |= _BV(6);
        if (i & _BV(1))  SetBit(PORTB,5);
        if (i & _BV(2))  SetBit(PORTA,1);
        if (i & _BV(3))  SetBit(PORTB,3);
        if (i & _BV(4))  SetBit(PORTD,2);
        if (i & _BV(5))  SetBit(PORTD,3);
        if (i & _BV(6))  SetBit(PORTD,4);

        // turn selected column on
        switch(curCol)
        {
                case 0: ClearBit(PORTD,1);    break;         // PORTD &= ~_BV(1)
                case 1: ClearBit(PORTA,0);    break;
                case 2: ClearBit(PORTB,4);    break;
                case 3: ClearBit(PORTB,1);    break;
                case 4: ClearBit(PORTB,2);    break;
        }
}
*/

//      INTERRUPT SERVICE ROUTINE for LTP747 (Anode Column) Displays
//

ISR (TIMER0_COMPA_vect)
{
        if (++curCol >= COLS)                 // advance column counter
                curCol = 0;

        // turn off all LEDS, by taking cathode (column) pins high
        PORTA = 0x02;
        PORTB = 0x68;
        PORTD = 0x1C;
```

```
        // turn on individual row bits in this column
        byte i = buf[curCol];
        if (i & _BV(0))  ClearBit(PORTB,6);                 //  PORTB &= ~_BV(6);
        if (i & _BV(1))  ClearBit(PORTB,5);
        if (i & _BV(2))  ClearBit(PORTA,1);
        if (i & _BV(3))  ClearBit(PORTB,3);
        if (i & _BV(4))  ClearBit(PORTD,2);
        if (i & _BV(5))  ClearBit(PORTD,3);
        if (i & _BV(6))  ClearBit(PORTD,4);

        // turn selected column on
        switch(curCol)
        {
                case 0: SetBit(PORTD,1);      break;         // PORTD |= _BV(1)
                case 1: SetBit(PORTA,0);      break;
                case 2: SetBit(PORTB,4);      break;
                case 3: SetBit(PORTB,1);      break;
                case 4: SetBit(PORTB,2);      break;
        }
}

//      ---------------------------------------------------------------------------
//      PROGRAM INITIALIZATION CODE

void init ()
{
                                              // set 12 output pins
        DDRA   =  0x03;                       // 0000.0011
        DDRB   =  0x7E;                       // 0111.1110
        DDRD   =  0x1E;                       // 0001.1110
                                              // setup Timer/Counter0 for LED refresh
        TCCR0A = _BV(WGM01);                  // Set CTC mode
        TCCR0B = _BV(CS02);                   // Set prescaler clk/256 = 15625 Hz
        OCR0A  =  40;                         // 15625/40 = 390 interrupts/sec (by 5 cols = ~78fps)
        TIMSK  = _BV(OCIE0A);                 // Enable T/C 0A interrupt

        MCUCR  =  0x30;                       // 0011.0000  (sleep enabled, power down)

                                              // To turn off the watchdog timer:
        WDTCR  =  0x18;                       // set WD enable + WD turn-off bits
        WDTCR  =  0x10;                       // quickly reset WD enable to complete the WD turnoff.

        sei();                                // enable interrupts
}


void DelayCS(byte cs)
//      Delays CPU for specified time, in centiseconds (1/100 sec)
//  Calling _delay_ms in a routine prevents inlining, reducing code size,
//  at the expense of slight timing inaccuracies.
{
        for (byte i=0; i<cs; i++)
                _delay_ms(10);
}

void DelaySecond()
{
        DelayCS(100);
}

//      ---------------------------------------------------------------------------
//      CHARACTER SCROLLING ROUTINES

void ShiftLeft()
//      shifts the entire display buffer one column to the left
{
        for (byte i=0; i<9; i++)
        {
                buf[i] = buf[i+1];            // shift each column to left
        }
        buf[9] = 0x00;                        // last column becomes blank
```

```
}

void Scroll()
//      scrolls a character onto the display
{
        for (byte i=0; i<COLS+CHARSPACING; i++)
        {
                ShiftLeft();                    // shift display 1 column to left
                DelayCS(SCROLLDELAY);           // and wait a while
        }                                       // repeat for whole character + spacing
}

void LoadSymbol(byte index)
//  loads a font symbol into the non-visible part of display buffer
{
        for (byte y = 0; y < COLS; y++)
        {
                buf[y+5] = pgm_read_byte(&(FONT_CHARS[index][y]));
        }
}

void MakeVisible()
//      copies char from non-visible to visible part of buffer
{
        for (byte i=0; i<COLS; i++)
        {
                buf[i] = buf[i+5];
        }
}

void DisplaySymbol(byte index)
//  loads a font symbol into the visible display buffer
{
        LoadSymbol(index);
        MakeVisible();
}


void ScrollText(const char *text)
//  scrolls given text across matrix, right to left
{
        for (byte i=0; i<strlen(text); i++)
        {
                LoadSymbol(text[i]-' ');        // get char
                Scroll();                       // and scroll it
        }                                       // repeat for all chars
}

void DisplayText(const char *text)
//      displays given text, one character at a time
{
        for (byte i=0; i<strlen(text); i++)
        {
                DisplaySymbol(text[i]-' ');     // display char
                DelaySecond();                  // wait a while
        }                                       // repeat for all chars
}


//      ------------------------------------------------------------------------
//      ANIMATION ROUTINES

void FlashHeart()
{
        DisplaySymbol(HEARTCHAR);               // flash heart on
        DelayCS(FLASHDELAY);                    // wait
        DisplaySymbol(0);                       // flash heart off
        DelayCS(FLASHDELAY);                    // wait
}

void HeartBeat()
```

```
{
        FlashHeart();                           // heart on/off
        FlashHeart();                           // heart on/off
        DelayCS(BEATDELAY);                     // wait
        FlashHeart();                           // do it again!
        FlashHeart();
        DelayCS(BEATDELAY);
}


//      ---------------------------------------------------------------------
//      PIXEL ROUTINES

void SetColumn(byte n)
{
        buf[n] = 0xFF;                          // turn on all leds in column
}

void ClearColumn(byte n)
{
        buf[n] = 0;                             // turn off all leds in column
}

void SetRow(byte n)
{
        for (byte i=0;i<COLS;i++)
        {
                buf[i] |= (1<<n);               // turn on all leds in row
        }
}

void ClearRow(byte n)
{
        for (int i=0;i<ROWS;i++)                // turn off all leds in row
        {
                buf[i] &= ~(1<<n);
        }
}

void SetPixel(byte x,byte y)
{
        buf[x] |= (1<<y);                       // turn on led at (x,y)
}

void ClearPixel(byte x, byte y)
{
        buf[x] &= ~(1<<y);                      // turn off led at (x,y)
}


void InvertPixel(byte x, byte y)
{
        buf[x] ^= (1<<y);                       // invert led at (x,y)
}


//      ---------------------------------------------------------------------
//      NEW STUFF

void Pause()                                    // blank display for brief period
{
        DisplaySymbol(0);
        DelayCS(50);
}

void PutPixel(byte x, byte y)                   // helper function for Marquis()
{
        InvertPixel(x,y);
        DelayCS(3);
}
```

```
void Marquis()
{
	byte x,y;
	for (byte count=0;count<8;count++)
	{
		for (x=0;x<5;x++)  PutPixel(x,0);		// move top-left to top-right
		for (y=1;y<7;y++)  PutPixel(4,y);		// move top-right to bottom-right
		for (x=3;x>0;x--)  PutPixel(x,6);		// move bottom-rt to bottom-left
		for (y=6;y>0;y--)  PutPixel(0,y);		// move bottom-left to top-left
	}
}

void OnePixel(byte x,byte y)			// turn on a single led at (x,y)
{
	DisplaySymbol(0);			// everything off first
	SetPixel(x,y);				// then set the pixel
	DelayCS(8);				// and wait
}

void Mouse()
{
	sbyte x,y;
	for (byte count=0;count<2;count++)
	{
		// first, do all of the rows
		for (x=0;x<=4;x++)  OnePixel(x,0);
		for (x=4;x>=0;x--)  OnePixel(x,1);
		for (x=0;x<=4;x++)  OnePixel(x,2);
		for (x=4;x>=0;x--)  OnePixel(x,3);
		for (x=0;x<=4;x++)  OnePixel(x,4);
		for (x=4;x>=0;x--)  OnePixel(x,5);
		for (x=0;x<=4;x++)  OnePixel(x,6);

		// now do all of the columns
		for (y=5;y>=0;y--)  OnePixel(4,y);
		for (y=0;y<=6;y++)  OnePixel(3,y);
		for (y=6;y>=0;y--)  OnePixel(2,y);
		for (y=0;y<=6;y++)  OnePixel(1,y);
		for (y=6;y>=0;y--)  OnePixel(0,y);
	}
}

void Pattern(byte n)
//
{
	for (byte i=0; i<8;i++)
	{
		DisplaySymbol(n);		// 50% lit, pattern A
		DelayCS(15);
		DisplaySymbol(n+1);		// 50% lit, pattern B
		DelayCS(15);
	}
	Pause();
}

void Box(byte size)
// creates a filled-box; size 1-5 OK
{
	byte value = (1<<size)-1;
	for (byte i=0;i<size;i++)
	{
		buf[i] = value;
	}
}

void Beat()
{
	for (byte count=0;count<8;count++)
	{
		for (byte i=1;i<5;i++)
		{
```

```
                        Box(i);
                        DelayCS(2);
                }
                for (byte i=4;i>0;i--)
                {
                        Box(i);
                        DelayCS(2);
                        DisplaySymbol(0);
                }
                DelayCS(40);
        }
}

#define BAR 0x3C
void HorizontalCylon()                          // bar goes side-to-side
{
        sbyte x;
        for (byte count=0;count<8;count++)
        {
                for (x=0;x<5;x++)               // left to right
                {
                        DisplaySymbol(0);
                        buf[x] = BAR;
                        DelayCS(8);
                }
                for (x=4;x>=0;x--)              // right to left
                {
                        DisplaySymbol(0);
                        buf[x] = BAR;
                        DelayCS(8);
                }
        }
}

void VerticalCylon()                            // bar goes up-and-down
{
        byte i,count;
        for (count=0;count<8;count++)
        {
                for (i=0;i<6;i++)               // top to bottom
                {
                        DisplaySymbol(0);
                        SetRow(i);
                        DelayCS(8);
                }
                for (i=6;i>0;i--)               // bottom to top
                {
                        DisplaySymbol(0);
                        SetRow(i);
                        DelayCS(8);
                }
        }
}

void ZigZag()
// creates an animated 'triangle wave', single pixel width
{
        sbyte bit=3, dir=-1;                    // starting bit & direction
        for (byte n=0;n<150;n++)
        {
                buf[5] = (1<<bit);              // place 1 dot in buffer
                DelayCS(10);                    // set animation speed
                ShiftLeft();                    // shift it into view
                if ((bit==6)|(bit==0))          // dot at top/bottom of display?
                        dir=-dir;               // yes, reverse direction
                bit += dir;                     // advance to next dot
        }
}

void ZigZag2()
// creates an animated 'triangle wave', 3-pixel width
```

```
{
        sbyte bit=3, dir=-1;                    // starting bit & direction
        for (int n=0;n<150;n++)
        {
                buf[5] = (7<<bit);              // place 3 dots in buffer
                DelayCS(10);                    // set animation speed
                ShiftLeft();                    // shift it into view
                if ((bit==4)|(bit==0))          // dots at top or bottom of display?
                        dir=-dir;               // yes, reverse direction
                bit += dir;                     // advance to next dot
        }
}

void DoubleZigZag()
//  creates two animated triangle waves, superimposed
{
        sbyte bit1=3, dir1=-1;                  // bit 1 starting point/dir
        sbyte bit2=5, dir2=1;                   // bit 2 starting/direction
        for (byte n=0;n<150;n++)
        {
                buf[5] = (1<<bit1)|(1<<bit2); // superimpose both waves
                DelayCS(10);                    // set animation speed
                ShiftLeft();                    // animate the display
                if ((bit1==6)|(bit1==0))
                        dir1=-dir1;             // reverse direction of #1
                if ((bit2==6)|(bit2==0))
                        dir2=-dir2;             // reverse direction of #2
                bit1 += dir1;                   // update bits
                bit2 += dir2;
        }
}


void Rain()
//  animated 3-column falling rain display
//  a,b,c are vertical positions of 'raindrops' = lit LEDs
{
        byte a=2, b=5, c=0;                     // don't start drops together
        for (byte count=0;count<100;count++)
        {
                DisplaySymbol(0);
                if (a<ROWS) SetPixel(0,a);      // raindrop on column 0
                if (b<ROWS) SetPixel(2,b);      // raindrop on column 2
                if (c<ROWS) SetPixel(4,c);      // raindrop on column 4
                DelayCS(8);                     // animation speed set here
                a++; if (a>=19) a=0;            // 3 different repeat counts
                b++; if (b>=11) b=0;
                c++; if (c>=15) c=0;
        }
        Pause();
}

void FlashSmile()
{
        Pause();
        DisplaySymbol(SMILECHAR);               // on
        DelayCS(200);                           // wait
        DisplaySymbol(0);                       // off
        DelayCS(FLASHDELAY);                    // wait
}

#define NOEYES 109
void BlinkSmile()
{
        Pause();
        DisplaySymbol(SMILECHAR);               // flash heart on
        DelayCS(200);                           // wait
        DisplaySymbol(NOEYES);
        DelayCS(30);
        DisplaySymbol(SMILECHAR);
        DelayCS(30);
```

```
        DisplaySymbol(NOEYES);
        DelayCS(30);
        DisplaySymbol(SMILECHAR);
        DelayCS(200);
        DisplaySymbol(0);                       // flash heart off
        DelayCS(FLASHDELAY);                    // wait
}

//      ---------------------------------------------------------------------
//      MORSE CODE ROUTINES

#define DITLENGTH 16
void Dit()
{
        Box(2);
        DelayCS(DITLENGTH);                     // dit is one element long
        DisplaySymbol(0);
        DelayCS(DITLENGTH);                     // single element space
}

void Dah()
{
        Box(2);
        DelayCS(DITLENGTH*3);                   // dah is 3 elements long
        DisplaySymbol(0);
        DelayCS(DITLENGTH);                     // single element space
}

void CQ()
{
        for (byte count=0;count<3;count++)
        {
                Dah();Dit();Dah();Dit();        // morse 'C'
                DelayCS(DITLENGTH*2);           // 1+2 = 3 elements between chars
                Dah();Dah();Dit();Dah();        // morse 'Q'
                DelayCS(DITLENGTH*6);           // 1+6 = 7 elements between words
        }
}


//      ---------------------------------------------------------------------
//      MAIN PROGRAM LOOP
//

void NewLoop()
{
        while(1)
        {
                FlashSmile();
                CQ();
                ScrollText("from w8bh ");
                Pattern(96);
                Rain();
                Pattern(111);
                Mouse();  FlashSmile();
                HorizontalCylon();
                VerticalCylon(); FlashSmile();
                Beat(); BlinkSmile();
                Marquis();
                ZigZag(); FlashSmile();
                ZigZag2(); FlashSmile();
                DoubleZigZag(); Pause();
                DisplaySymbol(FROWNCHAR); DelayCS(200);
                Pause();
                sleep_cpu();
        }
}

void OldLoop ()
{
        while(1)
```

```
        {
                DisplaySymbol(100);             // display a fun symbol
                DelaySecond();
                DelaySecond();
                DisplaySymbol(0);
                ScrollText(TEXT0);              // intro text
                for (byte i=100; i<107; i++)
                {
                        DisplaySymbol(i);       // display a fun symbol
                        DelaySecond();
                        DelaySecond();
                        HeartBeat();            // heartbeats
                        HeartBeat();
                        DisplayText(TEXT1);     // display text1
                        DelaySecond();
                        HeartBeat();            // more heartbeats
                        ScrollText(TEXT2);      // scroll text2
                        DelaySecond();
                }                               // repeat 7 times
                sleep_cpu();                    // turn off display
        }
}

//      ---------------------------------------------------------------------------
//      MAIN

int main (void)
{
        init();                                 // set up ports, CPU registers
        NewLoop();                              // do the display, then sleep
        //OldLoop();
        return (0);                             // that's all, folks!
}
```