

```

/*
Miscellaneous support functions for watch.
*/

#USE I2C(master, slow, sda=PIN_C0, scl=PIN_C1) // Set up
I2C port

// Read ASCII data in the buf[] screen buffer, convert to dots in the
// letter[] constant array, and
// Load five column words (32 bits each, right justified.)
// Note - I forgot why I expanded the outer loop, but putting it back saves 4%
of
// the total code space!!
void display(void)
{
    signed int8 i,j;
    for(i=3;i>=0;--i)
    {
        srcol[0].bits32 = srcol[0].bits32 << 7;
        srcol[0].bits32 += (letter[(buf[i]-'0')*5 + 0]);
    }
    for(i=3;i>=0;--i)
    {
        srcol[1].bits32 = srcol[1].bits32 << 7;
        srcol[1].bits32 += (letter[(buf[i]-'0')*5 + 1]);
    }
    for(i=3;i>=0;--i)
    {
        srcol[2].bits32 = srcol[2].bits32 << 7;
        srcol[2].bits32 += (letter[(buf[i]-'0')*5 + 2]);
    }
    for(i=3;i>=0;--i)
    {
        srcol[3].bits32 = srcol[3].bits32 << 7;
        srcol[3].bits32 += (letter[(buf[i]-'0')*5 + 3]);
    }
    for(i=3;i>=0;--i)
    {
        srcol[4].bits32 = srcol[4].bits32 << 7;
        srcol[4].bits32 += (letter[(buf[i]-'0')*5 + 4]);
    }
} // End of display()

// Increment 2 bcd digits packed into one byte, roll over at 'max'
int8 bcdinc(int8 initial, int8 max)
{
    ++initial;
    if((initial & 0x0F) > 0x09){initial = (initial & 0xF0) + 0x10;} // detect
carry into tens
    if(initial > max){initial = 0;} // detect
max value
    return(initial);
} // End of bcdinc()

// Write characters to screen buffer. Called by printf.
void to_screen(char x)
{

```

```

static int8 i;
if(i<4)
{
    buf[i] = x;
    if(x==' '){buf[i]='@';} // Deal with space character. @ is 'A' -1, which
is all zeros in table
    ++i;
}
if(x=='\n')
{
    i=0;
}
} // End of to_screen

// Reset general purpose timer
void reset_timer(void)
{
    disable_interrupts(INT_TIMER1);           // Reset timer to zero.
    timeout = 0;
    enable_interrupts(INT_TIMER1);
} // End of reset_timer()

// Read or write RTC data to and from the local "time"
// variable.
void rtc(int8 reg)
{
    int8 x;
    switch(reg)
    {
        case (WRITE):
            i2c_start();
            i2c_write(RTC_ADDR | WRITE);
            i2c_write(BURST | WRITE);
            for(x=0;x<=6;++x)
            {
                i2c_write(time[x]);
            }
            i2c_write(0);
            i2c_stop();
            break;

        case (READ):
            i2c_start();
            i2c_write(RTC_ADDR | WRITE);
            i2c_write(BURST | READ);
            i2c_start();
            i2c_write(RTC_ADDR | READ);
            for(x=0;x<=6;++x)
            {
                time[x] = i2c_read(1);
            }
            i2c_stop();
            break;
        default:
            }
} // End of rtc()

```

```

// This is where we end up when button is pushed.
#int_RB
RB_isr()
{
    int8 x;
    x = input_B();    // Clear mismatch condition.
} // End of RB ISR

// General purpose timer to tell how long to display things, etc.
#int_TIMER1
TIMER1_isr()
{
    if(timeout<255) ++timeout;
} // End of TIMER1 ISR

// This is the display driver ISR.
#int_TIMER2
TIMER2_isr()
{
    static unsigned int8 q;
    ++q;
    if(q>4) q=0;
    output_A(0x00);    // shut off all column lines. Doing this here
reduces "ghosting"
    spi_write(srcol[q].by.te3);    // Send shift register data to the display
    spi_write(srcol[q].by.te2);
    spi_write(srcol[q].by.te1);
    spi_write(srcol[q].by.te0);

    switch(q)    // Turn on next column line.
    {
        case 0: output_high(COL1); break;
        case 1: output_high(COL2); break;
        case 2: output_high(COL3); break;
        case 3: output_high(COL4); break;
        case 4: output_high(COL5); break;
        default:
    }
} // End of TIMER2 ISR

// Initialize hardware.
void initialize(void)
{
    setup_adc_ports(NO_ANALOGS);
    setup_adc(ADC_CLOCK_DIV_2);
    setup_spi(SPI_MASTER|SPI_L_TO_H|SPI_CLK_DIV_4|SPI_SS_DISABLED);
    CKP = 0;    // These are tweaks. The compiler has screwed up
    CKE = 1;    // SPI setup in the past, just making sure. Anyway,
    SPEN = 0;    // set up for data valid on rising clock, transitions on falling
clock.
    setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
    setup_timer_1(T1_INTERNAL|T1_DIV_BY_1);    // 16 bit, (4*2*2^16)/10MHz = 52.4
ms period
    setup_timer_2(T2_DIV_BY_4,255,4);
    setup_ccp1(CCP_OFF);
    setup_ccp2(CCP_OFF);

```

```

port_b_pullups(TRUE);
// Set up data direction on I/O ports.
set_tris_b(0xF0); // We can do this because of the pullups. (Nothing
floats.)
set_tris_a(0x00); // Column lines. (And that pesky open drain RA4.)
//set_tris_c(0x13); // This is the only port that we switch from
peripherals to GPIO.
output_a(0x00); // shut off all column lines.
output_low(PIN_C7); // Make sure this guy doesn't float.
i2c_start(); // Reset I2C bus.
i2c_stop();
time[SECONDS] = 0x00; // Starting time so you don't have to count up from
0:00 all the time.
time[MINUTES] = 0x31;
time[HOURS] = 0x17;
time[DATE] = 0x26;
time[MONTH] = 0x08;
time[WEEKDAY] = 0x05;
time[YEAR] = 0x03;

i2c_start(); // set century
i2c_write(RTC_ADDR | WRITE); i2c_write(0x92 | WRITE); i2c_write(20);
i2c_stop();

rtc(WRITE); // set clock

i2c_start(); // Disable temperature sensor, micropower shutdown.
i2c_write(FM75_ADDR | WRITE); i2c_write(0x01); i2c_write(0x61);
i2c_stop();
}

```