Light Switching Using RFID Protocol

Javier Onglao Sergio Shin

May 6, 2011

Contents

1 Introduction

1.1 Motivation

In the advent of the popular green revolution sweeping the nation, companies and individuals have adopted energy-efficient technologies. The incentives in saving energy equates to lower electric costs. Interestingly, most, if not all, people still forget to switch off lights when leaving; a trend which has been dominant in urban settings.

1.2 Goal

The goal of this project is to improve the human interaction with peripherals in a room. For simplicity, we choose to interface with the lights in the room. This project works with the assumption that most people walk through doors with their keys. As the person walks through the door sensors such as the radio frequency identification reader (RFID) determines which room lights to activate.

1.3 Concept

The RFID reader sends out radio signals. As people walk through the door (Figure 2) installed with RFID readers, the radio signals sent from the RFID reader powers the RFID tag through induction, and activates the microchip in the RFID tag to send its identification signal. The identification signal is captured by the RFID, and relayed to the Texas Instruments MSP430 Microcontroller. The microcontroller, then, switches the lights on if it determines that the unique ID belongs to its system. When the user walks out, the same protocol is implemented; the lights are switched off.

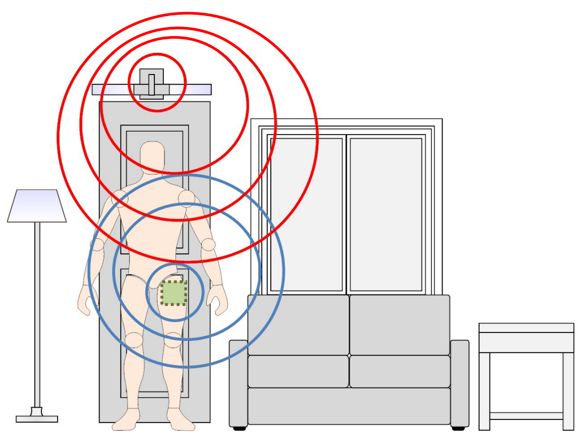

Figure 1: *Red circles indicate radio signals sent from the RFID reader. Blue circles indicate radio signals sent by the green RFID tag.*

2 Implementation

2.1 Design

The input to this device is the tag information sent to the RFID reader. The tag information has 12 characters which uniquely represent each tag. This information is sent to the MSP430 Microcontroller, and compared against the programmed tag values. If the input tag information matches one of the tags in the microcontroller, then the lights (one of the outputs) in the room turn on. When the user walks back again, the lights turn off. The other output is the LCD screen which welcomes the user, if a match is found. If no matches are found, it displays "Not recognized," and the tag information. This allows other tags to be programmed into the microcontroller.

2.2 System Level

The MSP430 Microcontroller is tasked to provide a corresponding action based on the input. The input is the tag information which goes to the MSP430 Microcontroller. The program in the MSP430 gets each character using the UART (9600 Baudrate) as simulated by the Timer A vector. Each received byte is stored into a buffer, which gets stored in the buffer array. Each array element is compared against all the programmed tag information arrays. The program determines which tag information array matches the stored buffer array, and then routes the action based on that tag. This leads to displaying the name of that person, and turning the associated light channel high. The number of times the user enters is incremented. Also, the program determines using modulos 2 if the user is entering or leaving. The associated message is displayed for leaving and entering, as well the action of turning the associated light channel high or low.

2.3 Components Level

2.3.1 RFID

The RFID uses one pin for the data, one pin for the 5V power, one pin for the ground, and one pin for reset. The 5V power is provided by the external battery source using 4 AAA batteries. The data pin is the main communication line of the RFID reader with the MSP430. Since the RFID reader has been programmed to communicate at 9600 Baudrate, the MSP430 had to be programmed to receive at 9600 Baudrate. The reset pin allows the RFID reader to read new tags again after a millisecond delay. The reset pin also tells the MSP430 via the data pin that all the tag information has been transmitted by appending a value of 3 at the end. The MSP430 then looks for the value of 3 in the receive buffer if the transmission has ended.

2.3.2 LCD Display

For the LCD panel implementation, I followed the schematic and the code provided online from LCD portion of the example programs. I powered up the LCD panel and the red LED backlight accordingly using the corresponding pins (pin 1: VSS, pin 2: VDD, pin 15: V+, pin 16: V-). The code utilizes an SPI interface, sending data continuously to the ST7600 controller connected to the LCD in a serial fashion. For this reason, we needed an 8-bit shift register to use a 4-bit mode data transmit from the MSP430 to the shift register using the OEbar and finally to the D7-D4 data lines of the LCD panel.

2.4 Schematics

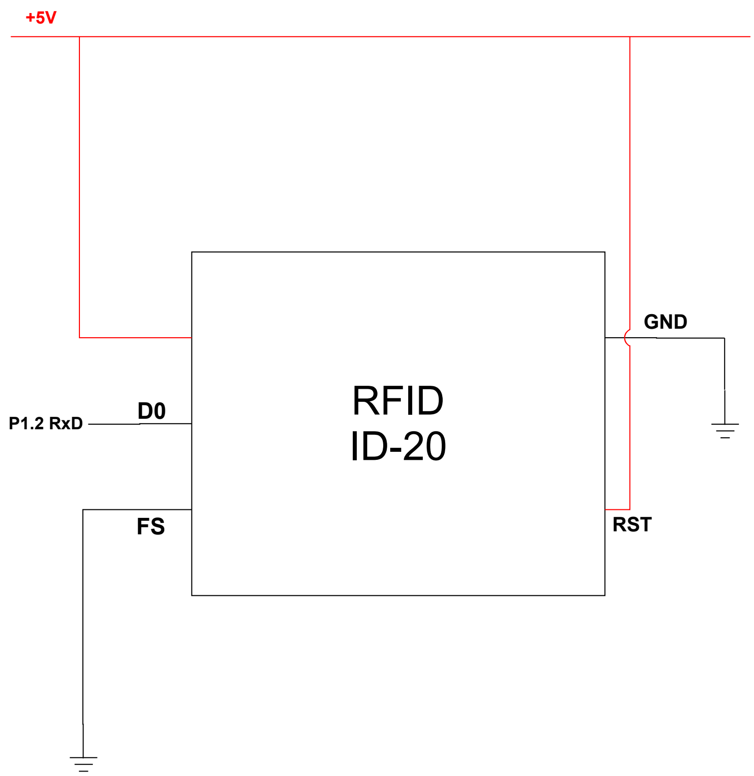

Figure 2: *RFID schematics linking up to Figure 3.*

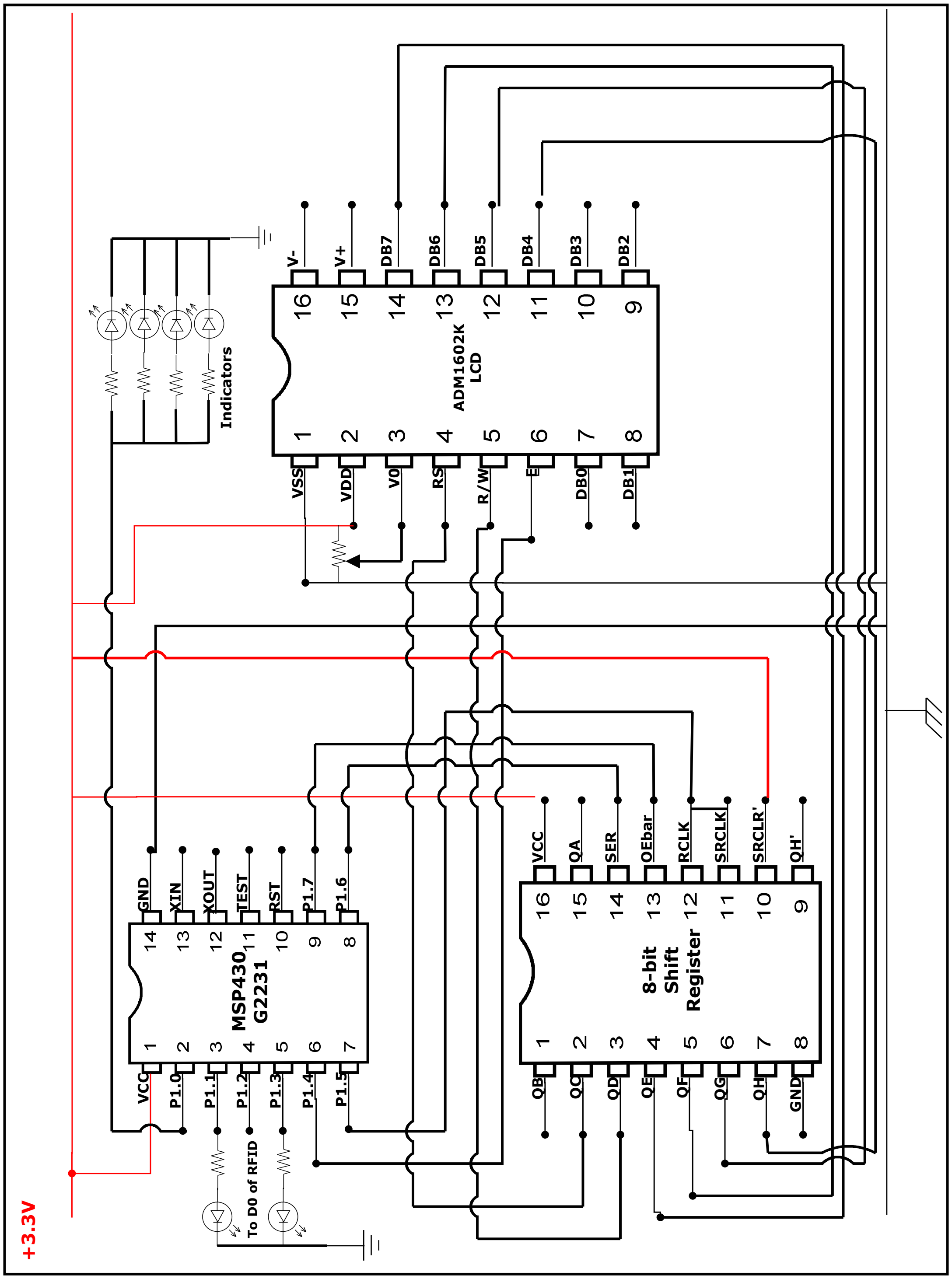

Figure 3: *LCD Schematics and the MSP430 Microcontroller.*

3 Assessment

3.1 Overall

The project is a major success. The components worked as expected, and the program performed with minimal technical difficulties. Although the project is successful, the range of the RFID reader limits the capacity of this device.

3.2 Challenges

Some of the difficulties we faced during the LCD panel interface with the MSP430 microcontroller were the hardware connections from the schematic. For example, the LCD panel itself was hard to power up considering the fact that the wiring was loose and the connections were not as good. After we soldered the pins using a male-to-male connection, we were able to solidify the wiring and successfully power the LCD panel and send the appropriate data for display.

The challenges faced in the RFID Reader includes the data bits transmitting bogus information. Later on we realized that the wire connecting the data pin to the MSP430 was loose, and we had to ensure that the connection was stable.

When it came to actually combining the two components together (RFID reader and LCD panel), the hardware was a little simpler to handle. We just had to modify some of the pins heading into the microcontroller since the RFID reader needed a pin that the LCD panel was already using. For the software (coding) interface, many modifications had to be made in order to make the LCD panel display a message when there was an actual tag detected.

3.3 Future Developments

Future developments for this include connecting this device to a door frame to have a much better user experience. To extend the range, we might consider using active tags instead of passive tags. Moreover, instead of just powering lights, we can consider controlling the power supply such that whenever the user leaves or enters, the power supply responds accordingly. This is useful for people who want to quickly unplug appliances that draw current when idle such as TV's.

Source Code

```
//*****************************************************************************
// Light Switching Using RFID Protocol
// 1.5.1.2
// April 24, 2010
//
//  Changelog available at changelog.txt
//
//  Made for RFID ID-20 (ID Innovations)
//
//  By Javier Onglao and Sergio Shin
//
//  UART Transmit and Receive by D. Dang (Texas Instruments, October 2010)
//  LED Display Driver by R. Giles (Boston University, April 2011)
//  RFID Read/Write Protocol by Bildr Blog (http://bildr.org, February 2011)
//
//  Built with CCS Version 4.2.0 and IAR Embedded Workbench Version: 5.10
//*****************************************************************************

#include "msp430g2231.h"

//------------------------------------------------------------------------------
// Channel Tag Definitions
// - Determines which tag id's activate which channel
//------------------------------------------------------------------------------
char channel1[] = "3D002154377F"; // Channel1 is Sergio
char channel2[] = "4400E6B60E1A"; // Channel2 is Javie

//------------------------------------------------------------------------------
// MSP430 Hardware Port Definitions
//------------------------------------------------------------------------------
#define UART_RXD     0x04                              // RXD on P1.2 (Timer0_A.CCI1A)
#define LED1       1              // LED1 on P1.0
#define ENABLE_READ 0x10             // LCD Enable on P1.5
#define OEbar      0x80              // LCD OE on P1.7

#define CH1 0x02               // Channel 1 on P1.1
#define CH2 0x08               // Channel 2 on P1.3

#define BIC(location,mask) ((location) &= ~(mask))
#define BIS(location,mask) ((location) |= mask)

//------------------------------------------------------------------------------
// Conditions for 9600 Baud SW UART, SMCLK = 1MHz
//------------------------------------------------------------------------------
#define UART_TBIT_DIV_2     (1000000 / (9600 * 2))
```

```c
#define UART_TBIT           (1000000 / 9600)

//------------------------------------------------------------------------------
// Global variables
//------------------------------------------------------------------------------
unsigned char rxBuffer;             // Received UART character
unsigned int i = 0;         // Buffer write index control
unsigned int j = 0;         // Buffer read index
char tagId[12];           // Buffer for storing the ID
char hasRead = 0;           // Done Flag
char reading = 0;           // Read Flag

unsigned int entryA = 1;      // how many times person A entered
unsigned int entryB = 1;      // how many times person B entered

unsigned int personA = 0;       // select person id
unsigned int personB = 0;       // select person id

unsigned delay_counter;       // variable for delays by watchdog interval timer
char needs_init = 1;        // global flag to avoid multiple poweron inits

//------------------------------------------------------------------------------
// Function prototypes
//------------------------------------------------------------------------------
void TimerA_UART_init(void);      // Initializes Timer A for UART

void IndicatorOn(void);         // Sets an indicator light
void IndicatorOff(void);        // Sets an indicator light
void delay(unsigned long);        // delay a number of microseconds

void SR_put_byte(unsigned char);      // send a byte out thru the shift register
void LCD_put(int value);         // send a value to the LCD
void LCD_init(void);           // initialize LCD
void LCD_put_string(char *);      // sends a string into the LCD
//------------------------------------------------------------------------------
// main()
//------------------------------------------------------------------------------
void main(void)
{
    BCSCTL1 = CALBC1_1MHZ;
    DCOCTL = CALDCO_1MHZ;

    //===========================================
  // Watchdog Timer
  //===========================================
  WDTCTL =(WDTPW + // (bits 15-8) password
                          // bit 7=0 => watchdog timer on
```

```
                        // bit 6=0 => NMI on rising edge (not used here)
                        // bit 5=0 => RST/NMI pin does a reset (not used here)
            WDTTMSEL +  // (bit 4) select interval timer mode
          WDTCNTCL + // (bit 3) clear watchdog timer counter
                    // bit 2=0 => SMCLK is the source
          WDTIS1+WDTIS0// bits 1-0 = 11 => source/64 => 8 microsec
);

  IE1 |= WDTIE; // enable the WDT interrupt (in the system interrupt register IE1)

//==================================================================
// LCD Screen Initialize
//==================================================================
// setup output ports for output to the SR
BIS(P1OUT,OEbar);                    // SR output disabled
BIC(P1OUT,ENABLE_READ);                // controller reads on rising edge of Enable
BIS(P1DIR,OEbar|ENABLE_READ);            // out pins
// setup SPI interface
USICTL0=USIPE6+USIPE5+USILSB+USIMST+USIOE;     // master, LSB first, enable SPI clk, out
USICTL1=USICKPH;                     // write on first transition
USICKCTL=USIDIV_7+USISSEL_2;              // clock divisor 7; SM clock source

//================================
// I/O Pins
//================================
  P1OUT = 0x00;                      // Initialize all GPIO
  P1SEL = UART_RXD;             // Timer function for TXD/RXD pins
  P1DIR = 0xFF & ~UART_RXD;            // Set all pins but RXD to output

  //==============================================
  // Initialization
  //  - informs the user that the system is ready
  //==============================================
  _bis_SR_register(GIE);          // enable interrupts
  TimerA_UART_init();                 // Start Timer_A UART

  LCD_init();               // Set Welcome Message
LCD_put_string("Welcome!");
LCD_put(0x80+40); // cursor to line 2
LCD_put_string("EC450 Project");

  for(;;)
  {
        // Wait for incoming character
        __bis_SR_register(LPM0_bits);

       if(rxBuffer == 2)
```

```
            {
               IndicatorOff();                 // Turn on the Red Indicator Light
               reading = 1;
            }
            if(rxBuffer == 3)
            {
               i = 0;
               reading = 0;
               hasRead = 1;

               while(j < 12)
               {
                 // Determine person using weights
                 if(channel1[j] == tagId[j])
                   personA++;
                 if (channel2[j] == tagId[j])
                   personB++;

                 j++;
               }

               if(personA == 12)
               {
                 IndicatorOn();                  // Turn on the Red Indicator Light

                 if(entryA [mod using percent] 2)
                 {
                   P1OUT |= CH1;

               delay(16000); // must wait 1.52ms after clear!
               LCD_init();
               LCD_put_string("Welcome Sergio!");
               LCD_put(0x80+40); // cursor to line 2
               LCD_put_string("Colts Rule");

                 }
                 else
                 {
                   delay(16000); // must wait 1.52ms after clear!
               LCD_init();
               LCD_put_string("Bye Sergio!");
               LCD_put(0x80+40); // cursor to line 2
               LCD_put_string("See you later");

               P1OUT &= ~CH1;
                 }

```

```
          entryA++;

          delay(99999);
          IndicatorOff();
          delay(16000); // must wait 1.52ms after clear!
          LCD_init();
          LCD_put_string("Welcome!");
          LCD_put(0x80+40); // cursor to line 2
          LCD_put_string("EC450 Project");
            }
            else if(personB == 12)
            {
              IndicatorOn();              // Turn on the Red Indicator Light

          if(entryB [mod using percent] 2)
              {
            P1OUT |= CH2;

            delay(16000); // must wait 1.52ms after clear!
            LCD_init();
            LCD_put_string("Welcome Javie!");
            LCD_put(0x80+40); // cursor to line 2
            LCD_put_string("Mabuhay");
              }
              else
              {
                delay(16000); // must wait 1.52ms after clear!
            LCD_init();
            LCD_put_string("Bye Javie!");
            LCD_put(0x80+40); // cursor to line 2
            LCD_put_string("See you later");

            P1OUT &= ~CH2;
              }

          entryB++;

          delay(99999);
          IndicatorOff();
          delay(16000); // must wait 1.52ms after clear!
          LCD_init();
          LCD_put_string("Welcome!");
          LCD_put(0x80+40); // cursor to line 2
          LCD_put_string("EC450 Project");
            }
            else
            {
```

```
                IndicatorOff();              // Turn on the Red Indicator Light

                delay(16000); // must wait 1.52ms after clear!
            LCD_init();
            LCD_put_string("NOT RECOGNIZED");
            LCD_put(0x80+40); // cursor to line 2
            LCD_put_string(tagId);

            delay(99999);
            IndicatorOff();
            delay(16000); // must wait 1.52ms after clear!
            LCD_init();
            LCD_put_string("Welcome!");
            LCD_put(0x80+40); // cursor to line 2
            LCD_put_string("EC450 Project");
              }

              j = 0;
              personA = 0;
              personB = 0;
              hasRead = 0;
          }

          if(hasRead == 0 && reading && rxBuffer != 2 && rxBuffer != 10 && rxBuffer != 13)
          {
              tagId[i] = rxBuffer;
              i++;
          }

   }
}

//------------------------------------------------------------------------------
// Function configures Timer_A for full-duplex UART operation
//------------------------------------------------------------------------------
void TimerA_UART_init(void)
{
    TACCTL0 = OUT;                          // Set TXD Idle as Mark = '1'
    TACCTL1 = SCS + CM1 + CAP + CCIE;       // Sync, Neg Edge, Capture, Int
    TACTL = TASSEL_2 + MC_2;                // SMCLK, start in continuous mode
}

//------------------------------------------------------------------------------
// Indicator Light
//  - Switches indicator lights to inform user of what's happening
//------------------------------------------------------------------------------
void IndicatorOn(void)
```

```
{
  P1OUT |= LED1;
}

//------------------------------------------------------------------------------
// Indicator Light
//  - Switches indicator lights to inform user of what's happening
//------------------------------------------------------------------------------
void IndicatorOff(void)
{
  P1OUT &= ~LED1;
}

//------------------------------------------------------------------------------
// Timer_A UART - Receive Interrupt Handler
//------------------------------------------------------------------------------
#pragma vector = TIMERA1_VECTOR
__interrupt void Timer_A1_ISR(void)
{
    static unsigned char rxBitCnt = 24; // Using Wiegand24 Protocol
    static unsigned char rxData = 0;

    switch (__even_in_range(TAIV, TAIV_TAIFG)) { // Use calculated branching
        case TAIV_TACCR1:                        // TACCR1 CCIFG - UART RX
            TACCR1 += UART_TBIT;                 // Add Offset to CCRx
            if (TACCTL1 & CAP) {                 // Capture mode = start bit edge
                TACCTL1 &= ~CAP;                 // Switch capture to compare mode
                //__delay_cycles(4000);
                TACCR1 += UART_TBIT_DIV_2;       // Point CCRx to middle of D0
            }
            else {
                rxData >>= 1;
                if (TACCTL1 & SCCI) {            // Get bit waiting in receive latch
                    rxData |= 0x80;
                }
                rxBitCnt--;
                if (rxBitCnt == 0) {             // All bits RXed?
                    rxBuffer = rxData;           // Store in global variable
                    rxBitCnt = 8;                // Re-load bit counter
                    TACCTL1 |= CAP;              // Switch compare to capture mode
                    __bic_SR_register_on_exit(LPM0_bits);  // Clear LPM0 bits from 0(SR)
                }
            }
            break;
    }
}
//------------------------------------------------------------------------------
```

```
void delay(unsigned long n){ // delays at least n microseconds using WDT

  if (delay_counter==0) { // only delay once
    delay_counter=(n+7)/8; // set rounded up number of wdt ticks
    _bis_SR_register(GIE+LPM0_bits);  // go into low power mode
  }
}

void SR_put_byte(unsigned char b){
  USISRL=b;  // transfer data to register
  USICNT=9+USI16B;  // 9 bits since the read and serial clocks on SR are connected
  while (!(USICTL1&USIIFG)) {}; // wait for flag
}

void LCD_put(int value){  // send a regular command or data to the LCD
  unsigned char high,low;
  high = value/16;  // 0 0 RS R/W DB7 DB6 DB5 DB4
  low = (value & 0x0F) | (high & 0x30); // 0 0 RS R/W DB3 DB2 DB1 DB0
  SR_put_byte(high);
  BIC(P1OUT,OEbar);   // assert data
  BIS(P1OUT,ENABLE_READ); // latch into controller
  delay(80);
  BIC(P1OUT,ENABLE_READ);
  BIS(P1OUT,OEbar);
  SR_put_byte(low);
  BIC(P1OUT,OEbar);   // assert data
  BIS(P1OUT,ENABLE_READ); // latch into controller
  delay(80);
  BIC(P1OUT,ENABLE_READ);
  BIS(P1OUT,OEbar);
}

void LCD_put_string (char *s){
  char c;
  int value;

  while ((c = *s++)!=0){
    value=0x200+c;
    LCD_put(value);
    delay(100);
  }
}

void LCD_init(){
  if (needs_init){
    delay(15000); // insure that 1.5 ms have elapsed
    // 3 function sets with delays
```

```
        SR_put_byte(3);
        BIC(P1OUT,OEbar); BIS(P1OUT,ENABLE_READ);
        delay(4100);
        BIC(P1OUT,ENABLE_READ); BIS(P1OUT,OEbar);
        SR_put_byte(3);
        BIC(P1OUT,OEbar); BIS(P1OUT,ENABLE_READ);
        delay(100);
        BIC(P1OUT,ENABLE_READ); BIS(P1OUT,OEbar);
        // set mode to 4 bit mode
        SR_put_byte(2);
        BIC(P1OUT,OEbar); BIS(P1OUT,ENABLE_READ);
        delay(80);
        BIC(P1OUT,ENABLE_READ); BIS(P1OUT,OEbar);
        needs_init=0; // only do this stuff once per session
    }
    // now we can send using the regular instructions
    LCD_put(0x28); // 2 lines, 5x8 characters
    LCD_put(0x0F); // display, cursor, blink on
    LCD_put(1);    // clear display
    delay(16000); // must wait 1.52ms after clear!
}

interrupt void WDTHandler()
{

    if ((delay_counter!=0) && (--delay_counter==0)){
        _bic_SR_register_on_exit(LPM0_bits);
    }
}

ISR_VECTOR(WDTHandler,".int10")
```