```
/*************************************************************************
****
 Title      :   HD44780U LCD library
 Author:    Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
 File:          $Id: lcd.c,v 1.14.2.1 2006/01/29 12:16:41 peter Exp $
 Software:  AVR-GCC 3.3
 Target:    any AVR device, memory mapped mode only for
AT90S4414/8515/Mega

 DESCRIPTION
       Basic routines for interfacing a HD44780U-based text lcd display

       Originally based on Volker Oth's lcd library,
       changed lcd_init(), added additional constants for lcd_command(),
       added 4-bit I/O mode, improved and optimized code.

       Library can be operated in memory mapped mode (LCD_IO_MODE=0) or
in
       4-bit IO port mode (LCD_IO_MODE=1). 8-bit IO port mode not
supported.

       Memory mapped mode compatible with Kanda STK200, but supports also
       generation of R/W signal through A8 address line.

 USAGE
       See the C include lcd.h file for a description of each function

*************************************************************************
****/
#include <inttypes.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include "lcd.h"



/*
** constants/macros
*/
#define DDR(x) (*(&x - 1))      /* address of data direction register of
port x */
#if defined(__AVR_ATmega64__) || defined(__AVR_ATmega128__)
    /* on ATmega64/128 PINF is on port 0x00 and not 0x60 */
    #define PIN(x) ( &PORTF==&(x) ? _SFR_IO8(0x00) : (*(&x - 2)) )
#else
      #define PIN(x) (*(&x - 2))    /* address of input register of port
x          */
#endif


#if LCD_IO_MODE
#define lcd_e_delay()   __asm__ __volatile__( "rjmp 1f\n 1:" );
#define lcd_e_high()    LCD_E_PORT  |=  _BV(LCD_E_PIN);
#define lcd_e_low()     LCD_E_PORT  &= ~_BV(LCD_E_PIN);
```

```
#define lcd_e_toggle()  toggle_e()
#define lcd_rw_high()   LCD_RW_PORT |=  _BV(LCD_RW_PIN)
#define lcd_rw_low()    LCD_RW_PORT &= ~_BV(LCD_RW_PIN)
#define lcd_rs_high()   LCD_RS_PORT |=  _BV(LCD_RS_PIN)
#define lcd_rs_low()    LCD_RS_PORT &= ~_BV(LCD_RS_PIN)
#endif

#if LCD_IO_MODE
#if LCD_LINES==1
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_4BIT_1LINE
#else
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_4BIT_2LINES
#endif
#else
#if LCD_LINES==1
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_8BIT_1LINE
#else
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_8BIT_2LINES
#endif
#endif

#if LCD_CONTROLLER_KS0073
#if LCD_LINES==4

#define KS0073_EXTENDED_FUNCTION_REGISTER_ON  0x24   /* |0|010|0100 4-bit
mode extension-bit RE = 1 */
#define KS0073_EXTENDED_FUNCTION_REGISTER_OFF 0x20   /* |0|000|1001 4
lines mode */
#define KS0073_4LINES_MODE                    0x09   /* |0|001|0000 4-bit
mode, extension-bit RE = 0 */

#endif
#endif

/*
** function prototypes
*/
#if LCD_IO_MODE
static void toggle_e(void);
#endif

/*
** local functions
*/



/*************************************************************************
*
 delay loop for small accurate delays: 16-bit counter, 4 cycles/loop
*************************************************************************
/
static inline void _delayFourCycles(unsigned int __count)
{
    if ( __count == 0 )
        __asm__ __volatile__( "rjmp 1f\n 1:" );    // 2 cycles
    else
        __asm__ __volatile__ (
          "1: sbiw %0,1" "\n\t"
          "brne 1b"                              // 4 cycles/loop
          : "=w" (__count)
          : "0" (__count)
         );
}


/*************************************************************************
*
delay for a minimum of <us> microseconds
the number of loops is calculated at compile-time from MCU clock
frequency
*************************************************************************
/
#define delay(us)  _delayFourCycles( ( ( 1*(XTAL/4000) )*us)/1000 )


#if LCD_IO_MODE
/* toggle Enable Pin to initiate write */
static void toggle_e(void)
{
    lcd_e_high();
    lcd_e_delay();
    lcd_e_low();
}
#endif


/*************************************************************************
*
Low-level function to write byte to LCD controller
Input:    data   byte to write to LCD
          rs     1: write data
                 0: write instruction
Returns:  none
*************************************************************************
/
#if LCD_IO_MODE
static void lcd_write(uint8_t data,uint8_t rs)
{
    unsigned char dataBits ;


    if (rs) {   /* write data        (RS=1, RW=0) */
       lcd_rs_high();
    } else {    /* write instruction (RS=0, RW=0) */
       lcd_rs_low();
    }
    lcd_rw_low();
    if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && ( &LCD_DATA1_PORT ==
&LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT == &LCD_DATA3_PORT )
      && (LCD_DATA0_PIN == 0) && (LCD_DATA1_PIN == 1) && (LCD_DATA2_PIN
== 2) && (LCD_DATA3_PIN == 3) )
    {
        /* configure data pins as output */
        DDR(LCD_DATA0_PORT) |= 0x0F;

        /* output high nibble first */
        dataBits = LCD_DATA0_PORT & 0xF0;
        LCD_DATA0_PORT = dataBits |((data>>4)&0x0F);
        lcd_e_toggle();

        /* output low nibble */
        LCD_DATA0_PORT = dataBits | (data&0x0F);
        lcd_e_toggle();

        /* all data pins high (inactive) */
        LCD_DATA0_PORT = dataBits | 0x0F;
    }
    else
    {
        /* configure data pins as output */
        DDR(LCD_DATA0_PORT) |= _BV(LCD_DATA0_PIN);
        DDR(LCD_DATA1_PORT) |= _BV(LCD_DATA1_PIN);
        DDR(LCD_DATA2_PORT) |= _BV(LCD_DATA2_PIN);
        DDR(LCD_DATA3_PORT) |= _BV(LCD_DATA3_PIN);

        /* output high nibble first */
        LCD_DATA3_PORT &= ~_BV(LCD_DATA3_PIN);
        LCD_DATA2_PORT &= ~_BV(LCD_DATA2_PIN);
        LCD_DATA1_PORT &= ~_BV(LCD_DATA1_PIN);
        LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);
      if(data & 0x80) LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
      if(data & 0x40) LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
      if(data & 0x20) LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
      if(data & 0x10) LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
        lcd_e_toggle();

        /* output low nibble */
        LCD_DATA3_PORT &= ~_BV(LCD_DATA3_PIN);
        LCD_DATA2_PORT &= ~_BV(LCD_DATA2_PIN);
        LCD_DATA1_PORT &= ~_BV(LCD_DATA1_PIN);
        LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);
      if(data & 0x08) LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
      if(data & 0x04) LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
      if(data & 0x02) LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
      if(data & 0x01) LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
        lcd_e_toggle();

        /* all data pins high (inactive) */
        LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
        LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
        LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
        LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
    }
}
#else
#define lcd_write(d,rs) if (rs) *(volatile uint8_t*)(LCD_IO_DATA) = d;
else *(volatile uint8_t*)(LCD_IO_FUNCTION) = d;
/* rs==0 -> write instruction to LCD_IO_FUNCTION */
/* rs==1 -> write data to LCD_IO_DATA */
#endif


/**********************************************************************
*
Low-level function to read byte from LCD controller
Input:    rs     1: read data
                 0: read busy flag / address counter
Returns:  byte read from LCD controller
**********************************************************************
/
#if LCD_IO_MODE
static uint8_t lcd_read(uint8_t rs)
{
    uint8_t data;


    if (rs)
        lcd_rs_high();                       /* RS=1: read data      */
    else
        lcd_rs_low();                        /* RS=0: read busy flag */
    lcd_rw_high();                           /* RW=1  read mode      */

    if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && ( &LCD_DATA1_PORT ==
&LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT == &LCD_DATA3_PORT )
      && ( LCD_DATA0_PIN == 0 )&& (LCD_DATA1_PIN == 1) && (LCD_DATA2_PIN
== 2) && (LCD_DATA3_PIN == 3) )
    {
        DDR(LCD_DATA0_PORT) &= 0xF0;         /* configure data pins as
input */

        lcd_e_high();
        lcd_e_delay();
        data = PIN(LCD_DATA0_PORT) << 4;     /* read high nibble first */
        lcd_e_low();

        lcd_e_delay();                       /* Enable 500ns low       */

        lcd_e_high();
        lcd_e_delay();
        data |= PIN(LCD_DATA0_PORT)&0x0F;    /* read low nibble        */
        lcd_e_low();
    }
    else
    {
        /* configure data pins as input */
        DDR(LCD_DATA0_PORT) &= ~_BV(LCD_DATA0_PIN);
        DDR(LCD_DATA1_PORT) &= ~_BV(LCD_DATA1_PIN);
        DDR(LCD_DATA2_PORT) &= ~_BV(LCD_DATA2_PIN);
        DDR(LCD_DATA3_PORT) &= ~_BV(LCD_DATA3_PIN);

        /* read high nibble first */
        lcd_e_high();
        lcd_e_delay();
        data = 0;
        if ( PIN(LCD_DATA0_PORT) & _BV(LCD_DATA0_PIN) ) data |= 0x10;
        if ( PIN(LCD_DATA1_PORT) & _BV(LCD_DATA1_PIN) ) data |= 0x20;
        if ( PIN(LCD_DATA2_PORT) & _BV(LCD_DATA2_PIN) ) data |= 0x40;
        if ( PIN(LCD_DATA3_PORT) & _BV(LCD_DATA3_PIN) ) data |= 0x80;
        lcd_e_low();

        lcd_e_delay();                       /* Enable 500ns low       */

        /* read low nibble */
        lcd_e_high();
        lcd_e_delay();
        if ( PIN(LCD_DATA0_PORT) & _BV(LCD_DATA0_PIN) ) data |= 0x01;
        if ( PIN(LCD_DATA1_PORT) & _BV(LCD_DATA1_PIN) ) data |= 0x02;
        if ( PIN(LCD_DATA2_PORT) & _BV(LCD_DATA2_PIN) ) data |= 0x04;
        if ( PIN(LCD_DATA3_PORT) & _BV(LCD_DATA3_PIN) ) data |= 0x08;
        lcd_e_low();
    }
    return data;
}
#else
#define lcd_read(rs) (rs) ? *(volatile uint8_t*)(LCD_IO_DATA+LCD_IO_READ)
: *(volatile uint8_t*)(LCD_IO_FUNCTION+LCD_IO_READ)
/* rs==0 -> read instruction from LCD_IO_FUNCTION */
/* rs==1 -> read data from LCD_IO_DATA */
#endif


/*********************************************************************
*
loops while lcd is busy, returns address counter
*********************************************************************
/
static uint8_t lcd_waitbusy(void)

{
    register uint8_t c;

    /* wait until busy flag is cleared */
    while ( (c=lcd_read(0)) & (1<<LCD_BUSY)) {}

    /* the address counter is updated 4us after the busy flag is cleared
*/
    delay(2);
    /* now read the address counter */
    return (lcd_read(0));  // return address counter

}/* lcd_waitbusy */


/*************************************************************************
*
Move cursor to the start of next line or to the first line if the cursor
is already on the last line.
*************************************************************************
/
static inline void lcd_newline(uint8_t pos)
{
    register uint8_t addressCounter;


#if LCD_LINES==1
    addressCounter = 0;
#endif
#if LCD_LINES==2
    if ( pos < (LCD_START_LINE2) )
        addressCounter = LCD_START_LINE2;
    else
        addressCounter = LCD_START_LINE1;
#endif
#if LCD_LINES==4
#if KS0073_4LINES_MODE
    if ( pos < LCD_START_LINE2 )
        addressCounter = LCD_START_LINE2;
    else if ( (pos >= LCD_START_LINE2) && (pos < LCD_START_LINE3) )
        addressCounter = LCD_START_LINE3;
    else if ( (pos >= LCD_START_LINE3) && (pos < LCD_START_LINE4) )
        addressCounter = LCD_START_LINE4;
    else
        addressCounter = LCD_START_LINE1;
#else
    if ( pos < LCD_START_LINE3 )
        addressCounter = LCD_START_LINE2;
    else if ( (pos >= LCD_START_LINE2) && (pos < LCD_START_LINE4) )
        addressCounter = LCD_START_LINE3;
    else if ( (pos >= LCD_START_LINE3) && (pos < LCD_START_LINE2) )
        addressCounter = LCD_START_LINE4;
    else
        addressCounter = LCD_START_LINE1;
#endif
#endif
    lcd_command((1<<LCD_DDRAM)+addressCounter);

}/* lcd_newline */


/*
** PUBLIC FUNCTIONS
```

```
*/

/*************************************************************************
*
Send LCD controller instruction command
Input:   instruction to send to LCD controller, see HD44780 data sheet
Returns: none
*************************************************************************
/
void lcd_command(uint8_t cmd)
{
    lcd_waitbusy();
    lcd_write(cmd,0);
}


/*************************************************************************
*
Send data byte to LCD controller
Input:   data to send to LCD controller, see HD44780 data sheet
Returns: none
*************************************************************************
/
void lcd_data(uint8_t data)
{
    lcd_waitbusy();
    lcd_write(data,1);
}



/*************************************************************************
*
Set cursor to specified position
Input:    x  horizontal position  (0: left most position)
          y  vertical position    (0: first line)
Returns:  none
*************************************************************************
/
void lcd_gotoxy(uint8_t x, uint8_t y)
{
#if LCD_LINES==1
    lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
#endif
#if LCD_LINES==2
    if ( y==0 )
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
    else
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE2+x);
#endif
#if LCD_LINES==4
    if ( y==0 )
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
    else if ( y==1)
```

```
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE2+x);
    else if ( y==2)
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE3+x);
    else /* y==3 */
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE4+x);
#endif

}/* lcd_gotoxy */


/*************************************************************************
*
*************************************************************************
/
int lcd_getxy(void)
{
    return lcd_waitbusy();
}


/*************************************************************************
*
Clear display and set cursor to home position
*************************************************************************
/
void lcd_clrscr(void)
{
    lcd_command(1<<LCD_CLR);
}


/*************************************************************************
*
Set cursor to home position
*************************************************************************
/
void lcd_home(void)
{
    lcd_command(1<<LCD_HOME);
}


/*************************************************************************
*
Display character at current cursor position
Input:    character to be displayed
Returns:  none
*************************************************************************
/
void lcd_putc(char c)
{
    uint8_t pos;
    pos = lcd_waitbusy();   // read busy-flag and address counter
    if (c=='\n')
    {
        lcd_newline(pos);
    }
    else
    {
#if LCD_WRAP_LINES==1
#if LCD_LINES==1
        if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
        }
#elif LCD_LINES==2
        if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE2,0);
        }else if ( pos == LCD_START_LINE2+LCD_DISP_LENGTH ){
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
        }
#elif LCD_LINES==4
        if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE2,0);
        }else if ( pos == LCD_START_LINE2+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE3,0);
        }else if ( pos == LCD_START_LINE3+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE4,0);
        }else if ( pos == LCD_START_LINE4+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
        }
#endif
        lcd_waitbusy();
#endif
        lcd_write(c, 1);
    }

}/* lcd_putc */


/*************************************************************************
*
Display string without auto linefeed
Input:    string to be displayed
Returns:  none
*************************************************************************
/
void lcd_puts(const char *s)
/* print string on lcd (no auto linefeed) */
{
    register char c;

    while ( (c = *s++) ) {
        lcd_putc(c);
    }

}/* lcd_puts */
```

```
/*************************************************************************
*
Display string from program memory without auto linefeed
Input:     string from program memory be be displayed
Returns:   none
*************************************************************************
/
void lcd_puts_p(const char *progmem_s)
/* print string from program memory on lcd (no auto linefeed) */
{
    register char c;

    while ( (c = pgm_read_byte(progmem_s++)) ) {
        lcd_putc(c);
    }

}/* lcd_puts_p */


/*************************************************************************
*
Initialize display and select type of cursor
Input:    dispAttr LCD_DISP_OFF            display off
                   LCD_DISP_ON             display on, cursor off
                   LCD_DISP_ON_CURSOR      display on, cursor on
                   LCD_DISP_CURSOR_BLINK   display on, cursor on flashing
Returns:  none
*************************************************************************
/
void lcd_init(uint8_t dispAttr)
{
#if LCD_IO_MODE
    /*
     *  Initialize LCD to 4 bit I/O mode
     */

    if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && ( &LCD_DATA1_PORT ==
&LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT == &LCD_DATA3_PORT )
      && ( &LCD_RS_PORT == &LCD_DATA0_PORT) && ( &LCD_RW_PORT ==
&LCD_DATA0_PORT) && (&LCD_E_PORT == &LCD_DATA0_PORT)
      && (LCD_DATA0_PIN == 0 ) && (LCD_DATA1_PIN == 1) && (LCD_DATA2_PIN
== 2) && (LCD_DATA3_PIN == 3)
      && (LCD_RS_PIN == 4 ) && (LCD_RW_PIN == 5) && (LCD_E_PIN == 6 ) )
    {
        /* configure all port bits as output (all LCD lines on same port)
*/
        DDR(LCD_DATA0_PORT) |= 0x7F;
    }
    else if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && ( &LCD_DATA1_PORT
== &LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT == &LCD_DATA3_PORT )
           && (LCD_DATA0_PIN == 0 ) && (LCD_DATA1_PIN == 1) &&
(LCD_DATA2_PIN == 2) && (LCD_DATA3_PIN == 3) )
```

```
    {
        /* configure all port bits as output (all LCD data lines on same
port, but control lines on different ports) */
        DDR(LCD_DATA0_PORT) |= 0x0F;
        DDR(LCD_RS_PORT)    |= _BV(LCD_RS_PIN);
        DDR(LCD_RW_PORT)    |= _BV(LCD_RW_PIN);
        DDR(LCD_E_PORT)     |= _BV(LCD_E_PIN);
    }
    else
    {
        /* configure all port bits as output (LCD data and control lines
on different ports */
        DDR(LCD_RS_PORT)    |= _BV(LCD_RS_PIN);
        DDR(LCD_RW_PORT)    |= _BV(LCD_RW_PIN);
        DDR(LCD_E_PORT)     |= _BV(LCD_E_PIN);
        DDR(LCD_DATA0_PORT) |= _BV(LCD_DATA0_PIN);
        DDR(LCD_DATA1_PORT) |= _BV(LCD_DATA1_PIN);
        DDR(LCD_DATA2_PORT) |= _BV(LCD_DATA2_PIN);
        DDR(LCD_DATA3_PORT) |= _BV(LCD_DATA3_PIN);
    }
    delay(16000);        /* wait 16ms or more after power-on       */

    /* initial write to lcd is 8bit */
    LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);  // _BV(LCD_FUNCTION)>>4;
    LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);  // _BV(LCD_FUNCTION_8BIT)>>4;
    lcd_e_toggle();
    delay(4992);         /* delay, busy flag can't be checked here */

    /* repeat last command */
    lcd_e_toggle();
    delay(64);           /* delay, busy flag can't be checked here */

    /* repeat last command a third time */
    lcd_e_toggle();
    delay(64);           /* delay, busy flag can't be checked here */

    /* now configure for 4bit mode */
    LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);   //
LCD_FUNCTION_4BIT_1LINE>>4
    lcd_e_toggle();
    delay(64);           /* some displays need this additional delay */

    /* from now the LCD only accepts 4 bit I/O, we can use lcd_command()
*/
#else
    /*
     * Initialize LCD to 8 bit memory mapped mode
     */

    /* enable external SRAM (memory mapped lcd) and one wait state */
    MCUCR = _BV(SRE) | _BV(SRW);

    /* reset LCD */
    delay(16000);                           /* wait 16ms after power-on
*/
    lcd_write(LCD_FUNCTION_8BIT_1LINE,0);   /* function set: 8bit
interface */
    delay(4992);                            /* wait 5ms
*/
    lcd_write(LCD_FUNCTION_8BIT_1LINE,0);   /* function set: 8bit
interface */
    delay(64);                              /* wait 64us
*/
    lcd_write(LCD_FUNCTION_8BIT_1LINE,0);   /* function set: 8bit
interface */
    delay(64);                              /* wait 64us
*/
#endif

#if KS0073_4LINES_MODE
    /* Display with KS0073 controller requires special commands for
enabling 4 line mode */
      lcd_command(KS0073_EXTENDED_FUNCTION_REGISTER_ON);
      lcd_command(KS0073_4LINES_MODE);
      lcd_command(KS0073_EXTENDED_FUNCTION_REGISTER_OFF);
#else
    lcd_command(LCD_FUNCTION_DEFAULT);      /* function set: display
lines  */
#endif
    lcd_command(LCD_DISP_OFF);              /* display off
*/
    lcd_clrscr();                           /* display clear
*/
    lcd_command(LCD_MODE_DEFAULT);          /* set entry mode
*/
    lcd_command(dispAttr);                  /* display/cursor control
*/

}/* lcd_init */
```