Here's another tutorial for driver code of the HD44780 controller. R/W pin is grounded. Code offers :

4 / 8 bit I/F for Mega, Xmega and some tiny series MCUs . I've interfaced a 5V LCD with 3.0V Xmegas directly .

Option to use the virtual ports feature for Xmega parts .

I've used this code on Mega series and 'A3_'A4 Xmega series .

Compiler : WinAvr20100110
IDE : Studio 4.18 SP3

slow_clk()
fast_clk()

are functions to change CPU frequency when it's too fast for the LCD, but are specific to Xmegas . Change according to your system clock, and oscillator, if you use these .

I hope people find it useful .

Jerome

lcd.c :

```
#include <util/delay.h>
#include   <avr/io.h>
#include <avr/pgmspace.h>

#include   "lcd.h"

extern  void ccp_write( void *ccp_reg, uint8_t value );


// Clearing a row has similar code to lcd_xy, so I factored out the common parts and so this is a combo function . See CLR_LINE () macro in lcd.h .

void   set_lcd_xy( uint8_t row, uint8_t col )

{

static  uint8_t   xx[4] = { 0, 0x40, 0x14, 0x54 };  // Should be changed to switch construct else it eats 4 RAM bytes.

uint8_t   x = 0;
```

```
        x = xx[row];

        if ( col < CLR_COLS )   {
                x += col;
        }

        x |= 0x80;
        lcd_cmd( x );

// This section provides a clr_row op. if needed .

        if ( col >=  CLR_COLS ) // Put col > CLR_COLS for 2nd parameter to clr a row.
   {
                        for ( uint8_t i = Max_col; i; i-- )
                                lcd_putc( ' ' );
                        //set_lcd_xy( row, 0 ); // Now set cursor to 1st column on same
row.
   }

}// End set_lcd_xy


void   lcd_putc( char a )
{

uint8_t   temp;

        if ( a > 9)
                temp = a;
        else  temp = a + 0x30; // Convert  to hex.

        set_rs;

#if defined   MODE_4

uint8_t   hi, lo;

        #if   defined   ( PORTA_OUT )

                slow_clk();
        #endif

        hi = (temp & UPPER_MASK );
        lo = (temp & LOWER_MASK );
```

```
        set_e;
        lcd_data = hi;
        clr_e;

        set_e;
        lcd_data = ( lo << 4 );
        clr_e;

#elif  defined   MODE_8
        //slow_clk();

        set_e;
        lcd_data = temp;
        clr_e;
#endif

        _delay_us(40);
        clr_rs;

        #if   defined   ( PORTA_OUT )

                fast_clk();

        #endif

}// End lcd_putc


void   lcd_puts( char *s )
{

        while( *s != '\0' )
        {
           lcd_putc( *s++ );
        }

}// End lcd_puts


void   lcd_cmd( uint8_t c )
{
        clr_rs;

#if defined   MODE_4

uint8_t   hi, lo;
```

```
        #if   defined   ( PORTA_OUT )

                slow_clk();
        #endif

        hi = ( c & UPPER_MASK );
        lo = ( c & LOWER_MASK );

        set_e;
        lcd_data = hi;
        clr_e;

        set_e;
        lcd_data = ( lo << 4 );
        clr_e;

#elif  defined   MODE_8

        set_e;
        lcd_data = c;
        clr_e;

#endif

        if ( c < 4 ){

                _delay_ms(2);
        }

        else{

                _delay_us(40);
        }

        #if   defined   ( PORTA_OUT )

                fast_clk();
        #endif

}// End lcd_cmd


void   lcd_putstr_P( PGM_P s )
{
```

```
char   temp;


        while(  ( temp = pgm_read_byte( s++ ) ) != '\0' )

        {       lcd_putc( temp );
        }

}// End lcd_putstr_P

#if   defined  ( PORTA_OUT ) // Xmegas only .

void   slow_clk( void ){

        OSC_CTRL &= ~( 1<< 4 );    // disable PLL

        ccp_write( (void * )&CLK_CTRL, 0x00 );  // Int. 2MHz as temp. system clk.
}


void   fast_clk( void ){

        //OSC_PLLCTRL = ( 8<< 0 );  Set this in main() so it doesn't get done multiple
times. Fout = (2 MHz * PLL_of_8X ) = 16 MHz .

        OSC_CTRL |= ( 1<< 4 ); // Enable PLL .
        while( !( OSC_STATUS & 0x10 ) ); // Wait unitl PLL as sys_clk is rdy to go.

        ccp_write( (void * )&CLK_CTRL, 0x04 );
}

#endif
```

lcd.h :

```
#ifndef  LCD_H

// Driver for HD44780 controller. R/W pin is gnd'ed.

        #define    LCD_H

        #include   <avr/io.h>
        #include <util/delay.h>
       #include <avr/pgmspace.h>
```

```
#define   MODE_4    1  // Set to 4 bit mode
//#define   MODE_8    1  // Set to 8 bit mode

#define   VP_ON   1  // Use Virtual Port
//#define   VP_OFF  1  // Don't use Virtual Port

#define   Max_rows   4
#define   Max_col    20

#define   EN     2
#define   RS     3

#if defined DDRA  // To identify Mega or Tiny series being used.

    #define   lcd_ddr     DDRA
    #define   lcd_data    PORTA

    #define   lcd_ctrl_ddr   DDRD
    #define   lcd_ctrl     PORTD

    #define   set_e     lcd_ctrl |= (1 << EN)  //;  sets a bit
    #define   clr_e     lcd_ctrl &= ~( 1 << EN )   //;  clr a bit

    #define   set_rs    lcd_ctrl |= (1 << RS)
   #define   clr_rs    lcd_ctrl &= ~( 1<< RS)

#elif  defined ( PORTA_OUT )  // To identify Xmega series MCU being used.

        #ifdef   VP_ON

                #define   lcd_ddr   VPORT0.DIR
                #define   lcd_data  VPORT0.OUT

                #define   lcd_ctrl_ddr    VPORT1.DIR
                #define   lcd_ctrl   VPORT1.OUT

                #define   set_e     VPORT1.OUT  |= ( 1<< EN )
                #define   clr_e     VPORT1.OUT  &= ~( 1<< EN )

                #define   set_rs     VPORT1.OUT  |= ( 1<< RS )
                #define   clr_rs     VPORT1.OUT  &= ~( 1<< RS )

        #elif   VP_OFF

                #define   CTRL_PORT  PORTE
```

```
                #define   lcd_ddr   PORTD.DIRSET
                #define   lcd_data  PORTD.OUTSET

                #define   lcd_ctrl_ddr    CTRL_PORT.DIRSET
                #define   lcd_ctrl   CTRL_PORT.OUTSET

                #define   set_e     CTRL_PORT.OUTSET = ( 1<< EN )
                #define   clr_e     CTRL_PORT.OUTCLR = ( 1<< EN )

                #define   set_rs     CTRL_PORT.OUTSET = ( 1<< RS )
                #define   clr_rs     CTRL_PORT.OUTCLR = ( 1<< RS )

        #endif  // End #ifdef   VP_ON

#endif // End #if defined DDRD

// LCD cmds

#define   disp_on  2
#define   cursor_on  1
#define   blink_on  0
#define   CLR_COLs   50  // Used as 2nd param. in set_lcd_xy to clr screen.

#define   I_D   1
#define   S     0
#define   S_C   3
#define   R_L   2
#define   DL    4
#define   N     3
#define   F     2

// 4 or 8 bit I/F

#define   _8_bit    0x30
#define   _4_bit   0x28

#define   UPPER_MASK      0xF0
#define   LOWER_MASK    0x0F

#define   CLR_LCD       0x01
#define   GO_HOME     0x02
#define   ONE_LINER  0x30
#define   TWO_LINER 0x38
#define   disp_off  0x08
#define   FAST_DELAY   50
#define   CURSOR_ON   ( 1<< cursor_on )
```

```
        #define   BLINK_ON    ( 1<< blink_on )
        #define   CURSOR_OFF   0

        #define   CLR_LINE( row )    set_lcd_xy( row, CLR_COLS )


/******** Prototypes ************************/

        void   set_lcd_xy( uint8_t , uint8_t );
        void   lcd_putc( char );
        void   lcd_puts( char * );
        void   lcd_cmd( uint8_t );
        void   lcd_putstr_P( PGM_P s );

        static  inline void   init_lcd( uint8_t );

        void   slow_clk ( void );   // Needed if CPU freq...
        void   fast_clk ( void );  //... too fast for LCD.


static  inline  void init_lcd( uint8_t cursor )
{

#if defined   MODE_4  // Init. Code for 4 bit I/F.


uint8_t   lo_nibble, hi_nibble;


        #if defined  PORTA_OUT  // Setup for Xmegas .

                #if defined   VP_ON

                        PORTCFG.VPCTRLA = PORTCFG_VP1MAP_PORTE_gc |
PORTCFG_VP0MAP_PORTD_gc; // VP0 for PORTD, VP1 for PORTE

                   VPORT0.DIR = 0xF0; // Set Upper nibble as output .
                   VPORT1.DIR = ( 1<< EN | 1<< RS );  //Set lcd control pins as output .

                #elif   defined  VP_OFF

                        lcd_ddr |= 0xF0;
                        lcd_ctrl_ddr |= (1<<EN) | (1<<RS);
                #endif

        #elif   defined  DDRA  // Setup for all other MCU types .
```

```
            lcd_ddr |= 0xF0; // Set Upper nibble as output .
            lcd_ctrl_ddr |= (1<<EN) | (1<<RS); // Set lcd control pins as output .

        #endif // End #if defined  PORTA_OUT

        // Must send MSN ONLY and 3X 1st. and Enable line must toggle for each
nibble.

        lcd_data = 0;  // Data and ...
        lcd_ctrl &= ~( (1<<EN) | (1<<RS) );  //... control lines init. to zero .

        _delay_ms(60);  // PORst. delay, needed.

        hi_nibble =  _8_bit;

        set_e;
        lcd_data = hi_nibble;   // Send 1st function set, must be ( 0x3x ) to lcd... first time.
        clr_e;
        _delay_ms(6);      // Must be > 4.1 ms

        set_e;            //  2nd send of function set
        lcd_data = hi_nibble;
        clr_e;
        _delay_us(105);   // Must be > 100 uS

        set_e;          // 3rd @@!@!!!! send of function set.
        lcd_data = hi_nibble;
        clr_e;
        _delay_us(105);

//************ Code above this line cannot be changed, except TWO_LINER.

        hi_nibble = _4_bit;
        set_e;              // This section sets desired data bit size I/F to 4 bits.
        lcd_data = hi_nibble;  //TWO_LINER;
        clr_e;
        _delay_us(FAST_DELAY);

        hi_nibble = ( ( _4_bit | 0x08 ) & ( UPPER_MASK ) );
        lo_nibble = ( ( _4_bit | 0x08 ) & ( LOWER_MASK ) );

        set_e;              // This part preserves data bit size I/F, and sets font and # of
lines.
        lcd_data = hi_nibble;
        clr_e;
```

```
//_delay_us(FAST_DELAY); NO !! Delay after a byte, NOT nibble !

        set_e;
        lcd_data = ( lo_nibble << 4 );
        clr_e;
        _delay_us(FAST_DELAY);
                // 1 byte sent.

        hi_nibble = ( disp_off & UPPER_MASK );
        lo_nibble = ( disp_off & LOWER_MASK );

        set_e;
        lcd_data = hi_nibble;
        clr_e;

        set_e;
        lcd_data = ( lo_nibble << 4 );
        clr_e;
        _delay_us(FAST_DELAY);

        hi_nibble = ( CLR_LCD & UPPER_MASK );
        lo_nibble = ( CLR_LCD & LOWER_MASK );

        set_e;
        lcd_data = hi_nibble;
        clr_e;

        set_e;
        lcd_data = ( lo_nibble << 4 );
        clr_e;
        _delay_us(FAST_DELAY);

        hi_nibble = ( ( 0x04 | 1 << I_D ) & UPPER_MASK );
        lo_nibble = ( ( 0x04 | 1 << I_D ) & LOWER_MASK );

        set_e;          // Set entry mode -- 0x06 is cursor shift right
        lcd_data = hi_nibble;
        clr_e;

        set_e;
        lcd_data = ( lo_nibble << 4 );
        clr_e;
        _delay_us(FAST_DELAY);

/********** end of 4 bit init. **********************/
```

```
hi_nibble = ( 0x14 & UPPER_MASK );
lo_nibble = ( 0x14 & LOWER_MASK );

set_e;                  // 0x14 is enable cursor right shift
lcd_data = hi_nibble;
clr_e;

set_e;
lcd_data = ( lo_nibble << 4 );
clr_e;
_delay_us(FAST_DELAY);

hi_nibble = ( CLR_LCD & UPPER_MASK );
lo_nibble = ( CLR_LCD & LOWER_MASK );

set_e;
lcd_data = hi_nibble;
clr_e;

set_e;
lcd_data = ( lo_nibble << 4 );
clr_e;
_delay_ms(2);

hi_nibble = ( GO_HOME & UPPER_MASK );
lo_nibble = ( GO_HOME & LOWER_MASK );

set_e;
lcd_data = hi_nibble;
clr_e;

set_e;
lcd_data = ( lo_nibble << 4 );
clr_e;
_delay_ms(2);

hi_nibble = ( ( 0x08 |  1<< disp_on  |  cursor  ) & UPPER_MASK );
lo_nibble = ( ( 0x08 |  1<< disp_on  |  cursor  ) & LOWER_MASK );

set_e;
lcd_data = hi_nibble;  //( 1<< cursor_on)  | ( 1<< blink_on );
clr_e;
          //_delay_us(FAST_DELAY);

set_e;
lcd_data = ( lo_nibble << 4 );
```

```
        clr_e;
        _delay_us(FAST_DELAY);
        // 1 byte sent.

#elif  defined   MODE_8  // Init. Code for 8 bit I/F.

        #if defined  PORTA_OUT  // Xmega series setup

                PORTCFG_VPCTRLA = PORTCFG_VP1MAP_PORTE_gc |
PORTCFG_VP0MAP_PORTD_gc; // VP0 for PORTD,  VP1 for PORTE

       VPORT0.DIR = 0xF0; // Set Upper nibble as output .
       VPORT1.DIR = ( 1<< EN | 1<< RS );  //Set lcd control pins as output .

        #elif   defined  DDRA  // Setup for all other MCU types .

                lcd_ddr = 0xFF;
                lcd_ctrl_ddr |= (1<<EN) | (1<<RS);

        #endif

                lcd_data = 0;  // Clr data and...
                lcd_ctrl &= ~( (1<<EN) | (1<<RS) );  //... control lines .

                _delay_ms(60);  // PORst. delay needed.

                set_e;                  // Send 1st function set, 0x3x to lcd... first time.
                lcd_data = TWO_LINER;    // 0x38
                clr_e;
                _delay_ms(6);  // Must be > 4.1 Ms

                set_e;                  //  2nd send of function set
                lcd_data = TWO_LINER;
                clr_e;
                _delay_us(105);  // Must be > 100 uS

                set_e;                 // 3rd @@!@!!!! send of function set.
                lcd_data = TWO_LINER;
                clr_e;
                _delay_us(105);

/********** Code above this line cannot be changed **********************/

                set_e;                  // This line actually sets desired data bit size I/F, font and
# of lines.
                lcd_data = TWO_LINER;
```

```
                clr_e;
                _delay_us(FAST_DELAY);

                set_e;              // 0x08
                lcd_data = disp_off;
                clr_e;
                _delay_us(FAST_DELAY);

                set_e;              // 0x01
                lcd_data = CLR_LCD;
                clr_e;
                _delay_us(FAST_DELAY);

                set_e;              // Set entry mode -- 0x06 is cursor shift right
                lcd_data = ( 0x04 | 1 << I_D ); // 0x01
                clr_e;
                _delay_us(FAST_DELAY);

/****************** end of 8 bit init. ******************************/

                set_e;              // 0x14 is enable cursor right shift
                lcd_data = 0x14;
                clr_e;
                _delay_us(FAST_DELAY);

                set_e;              // 0x01
                lcd_data = CLR_LCD;
                clr_e;
                _delay_ms(2);

                set_e;              // 0x02
                lcd_data = GO_HOME;
                clr_e;
                _delay_ms(2);

                set_e;
                lcd_data = ( 0x08) | ( 1<< disp_on) | ( cursor );  //( 1<< cursor_on)  | ( 1<<
blink_on );
                clr_e;
                _delay_us(FAST_DELAY);

#endif // #if defined   MODE_4

        }// End init_LCD

#endif  // End  #ifndef  LCD_H
```

ccp.S :

```
#define  _SFR_ASM_COMPAT  1
#define  _SFR_OFFSET  0

#include  <avr/io.h>


#define  CCP_SIGN  0xD8


.global  ccp_write

.func  ccp_write

        ccp_write:

   movw  r26, r24  // Copy register address into X ptr.
        ldi  r24, CCP_SIGN
        out  CCP, r24
   st  X, r22  // Write value to protected register X_ptr.
ret
.endfunc

// Must use registers shown, BUT you can use Y ( must preserve registers ) or Z ptr.
instead of X .
```