Purpose
To understand the hardware structure of a typical alphanumeric display unit, having an in build display controller. We will implement a C driver for the LCD display.

Material
- Manual for the LCD controller HD44780U (Hitachi).
- Data sheet for the LCD display unit PC1602-F (PowerTip).
- “AVR Demo Board” diagrams.

The exercise
In this exercise, we will implement a C driver for the LCD display.
The basic bus timing demands has to be respected.
Therefore, part of the exercise will be to study the data book for the LCD display controller.

Background
Mounted at the “AVR Demo Board” you will find a 2 line x 16 character LCD display (PowerTip PC1602-F), and the display terminals are wired to the connector "Display" (eventually study the “AVR Demo Board” diagrams at the Campus Net).

Using this connector, it is possible to establish connection between the display unit and one of the STK500 ports (doing this, the display will be power supplied from the STK500, as well).

The display unit itself is representative for most common alphanumeric display units, because the LCD controller HD44780U (also called LCD II standard) controls it. The controller is an integrated part of the display unit, controlling the individual pixels of the display. Besides, HD44780 has an external bus interface, enabling us to control the display from the AVR microcontroller I/O ports.

Having the necessary knowledge of the HD44780 functionality, we will be able to implement a driver, useful for many other similar displays. Because of that, it is very important to study the HD44780U data book (AMS Blackboard).

As can be seen from the HD44780data book, it is possible to communicate with the display using either an 8 bit data bus or a 4 bit data bus. Besides, in both cases, we need 3 control signals (E, RW and RS).

At the "AVR Demo Board" the 4 bit mode is chosen, because this enables us to control the display from an 8 bit port (only 4+3 port pins are used). As a minor disadvantage, the driver will be just slightly more complicated, than if 8 bit mode was chosen.

In our case, the most relevant HD44780 pins are the bus interface pins.
The next pages figure describes these bus interface pins.

Notice: The lower part of the data bus (DB0 to DB3) is not used in 4 bit mode, and therefore not physically wired to the "AVR Demo Board" connector named "Display".

Pin Functions

Signal	No. of Lines	I/O	Device Interfaced with	Function
RS	1	I	MPU	Selects registers. 0: Instruction register (for write) Busy flag: address counter (for read) 1: Data register (for write and read)
R/$\overline{W}$	1	I	MPU	Selects read or write. 0: Write 1: Read
E	1	I	MPU	Starts data read/write.
DB4 to DB7	4	I/O	MPU	Four high order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. DB7 can be used as a busy flag.
DB0 to DB3	4	I/O	MPU	Four low order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. These pins are not used during 4-bit operation.

The signals RS and R/W select the type of operation performed, when the signal E is pulsed. Commands can be given to HD44780 by writing the "instruction register" IR.
Notice: A command can be relatively slow executing (compared to the microcontroller execution time). Command execution can is some cases last several milliseconds.

Because it is prohibited to write the HD44780, when it is executing any internal command, we have to ensure HD44780 not being "BUSY", before we write a new command.
This can be ensured by polling the "BUSY flag". The BUSY flag is output at the DB7 pin, if we select RS = 0 and R/W = 1.

Table 1 Register Selection

RS	R/$\overline{W}$	Operation
0	0	IR write as an internal operation (display clear, etc.)
0	1	Read busy flag (DB7) and address counter (DB0 to DB6)
1	0	DR write as an internal operation (DR to DDRAM or CGRAM)
1	1	DR read as an internal operation (DDRAM or CGRAM to DR)

Obviously, when reading and writing the HD44780, certain bus timing requirements has to be observed. The timing diagrams and timing demands are shown at the following pages.

Timing Characteristics

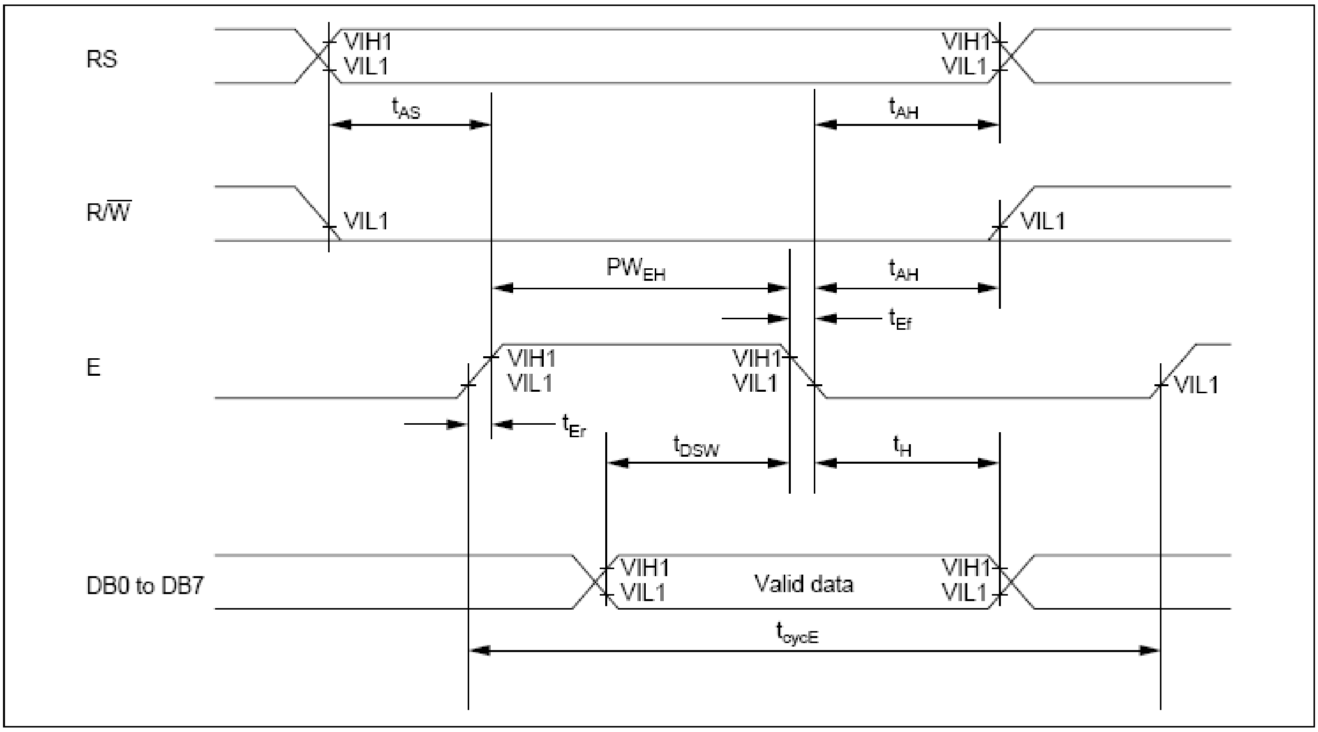

Figure 25 Write Operation

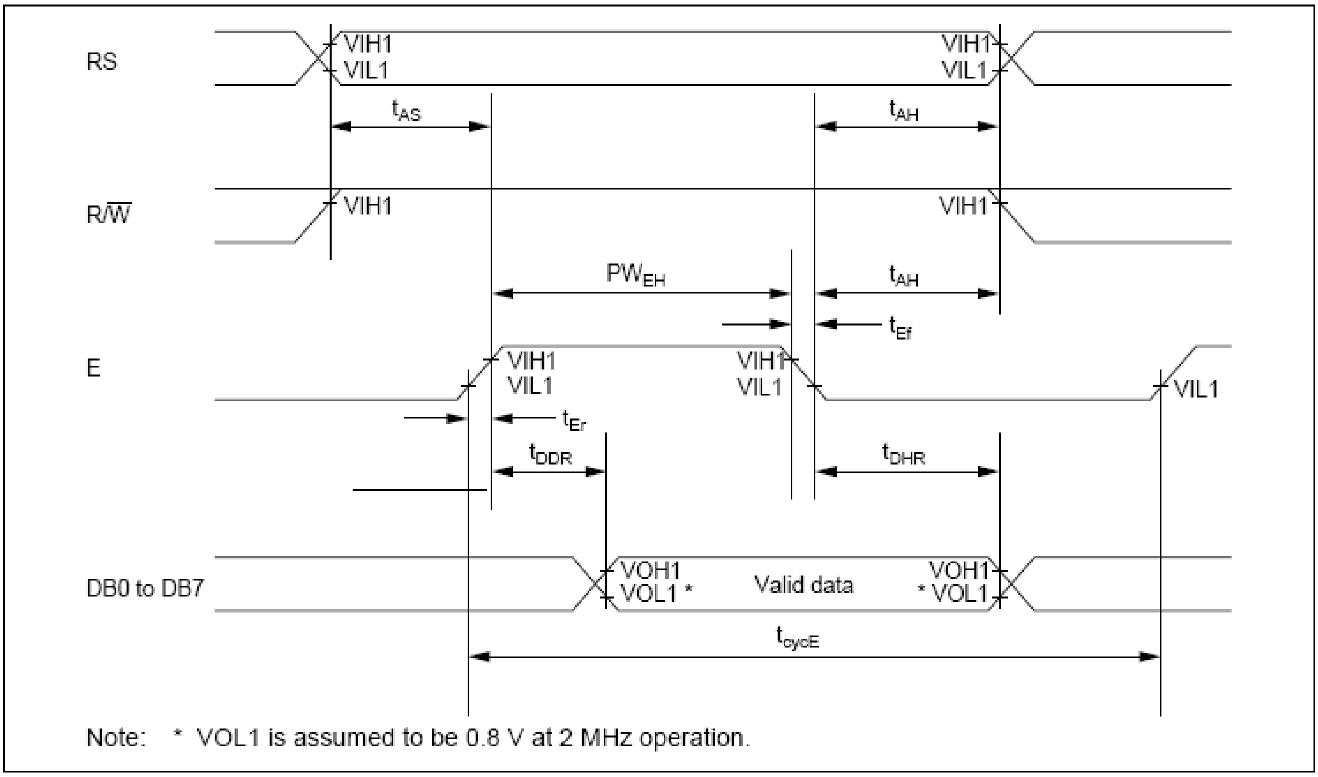

Figure 26 Read Operation

Bus Timing Characteristics

Write Operation

Item	Symbol	Min	Typ	Max	Unit	Test Condition
Enable cycle time	t_{cycE}	500	—	—	ns	Figure 25
Enable pulse width (high level)	PW_{EH}	230	—	—		
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	20		
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	40	—	—		
Address hold time	t_{AH}	10	—	—		
Data set-up time	t_{DSW}	80	—	—		
Data hold time	t_{H}	10	—	—		

Read Operation

Item	Symbol	Min	Typ	Max	Unit	Test Condition
Enable cycle time	t_{cycE}	500	—	—	ns	Figure 26
Enable pulse width (high level)	PW_{EH}	230	—	—		
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	20		
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	40	—	—		
Address hold time	t_{AH}	10	—	—		
Data delay time	t_{DDR}	—	—	160		
Data hold time	t_{DHR}	5	—	—		

When using the 4 bit interface, we need 2 read- or write-operations for transferring 8 bits.
The figure shows an example doing this (notice, the BUSY flag is read onDB7):

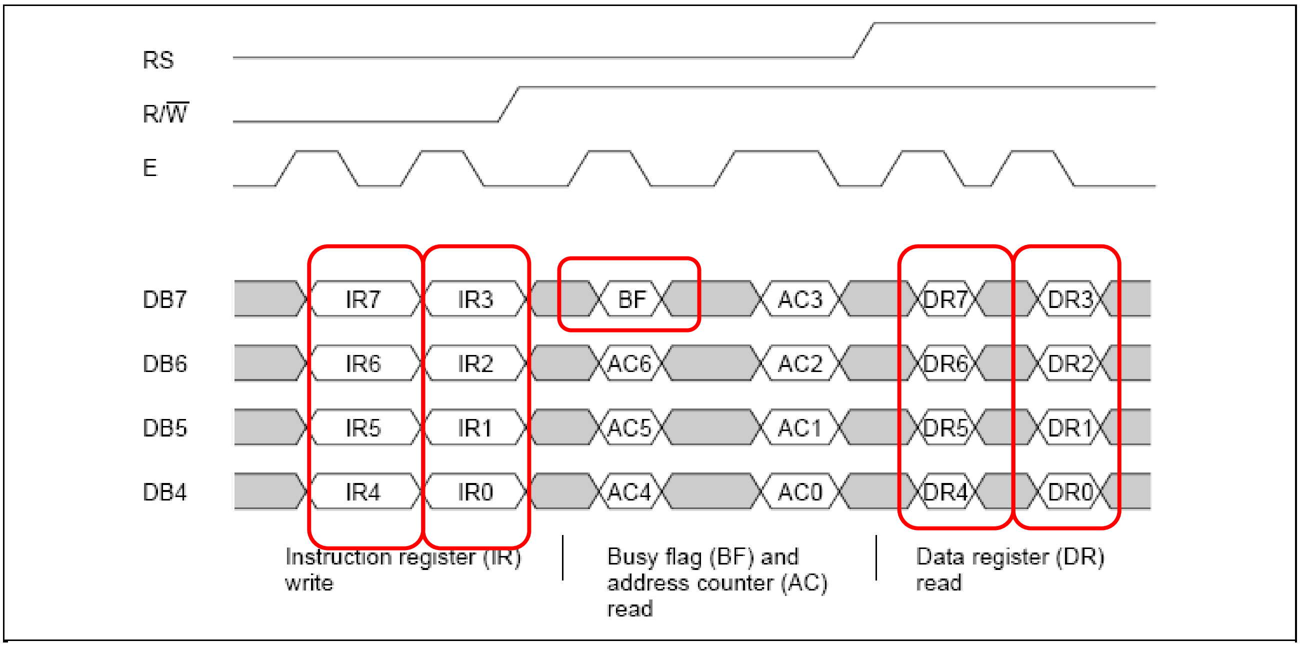

Figure 9 4-Bit Transfer Example

The data book explains how different sized displays (various number of lines and number of characters per line) are addressed concerning the individual display characters.
Each character at the display has a **DDRAM-address** (Display Data Ram address).

A 2 line x 16 character display (like the one, we are using) has these DDRAM-addresses (hex):

Display position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DDRAM address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Notice: Line number 2 has the starting DDRAM address 0x40.

During the display initialization, there is an option of selecting "auto increment", meaning the DDRAM-address will be automatically incremented after each display write.
Using this option, we don't have to set the DDRAM-address (using a dedicated command) before every display write when writing a string to the display (but only to set the first address).

HD44780 has an embedded ROM with bit patterns, generating the most commonly used characters. Consult the data book for details (our display unit uses the European fonts).

Important note: The **character codes 0x00 to 0x08** have a special meaning:

These are not associated to the character generator ROM, but to a special memory, called the **"character generator RAM" (CGRAM)**. The CGRAM contents define the character dot patterns for the character codes 0x00 to 0x08.

Because the **CGRAM** contents can be modified by the user, it enables him/her to define his/hers own special characters (for example the special Danish characters Æ, Ø and Å).
The character dot definitions can be done dynamically (during program execution), as well. This enables us to do (very simple) display animations.

The figure at next page shows the relationship between CGRAM addresses, Character codes (DDRAM) and Character Patterns (CGRAM Data).

Consult the pages19 and 20 in the HD44780U data book for further information!

Table 5 Relationship between CGRAM Addresses, Character Codes (DDRAM) and Character Patterns (CGRAM Data)

For 5 × 8 dot character patterns

Character Codes (DDRAM data)								CGRAM Address						Character Patterns (CGRAM data)								
7	6	5	4	3	2	1	0	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
High							Low	High					Low	High							Low	
0	0	0	0	*	0	0	0	0	0	0	0	0	0	*	*	*	1	1	1	1	0	Character pattern (1)
											0	0	1				1	0	0	0	1	
											0	1	0				1	0	0	0	1	
											0	1	1				1	1	1	1	0	
											1	0	0				1	0	1	0	0	
											1	0	1				1	0	0	1	0	
											1	1	0				1	0	0	0	1	
											1	1	1	*	*	*	0	0	0	0	0	Cursor position
0	0	0	0	*	0	0	1	0	0	1	0	0	0	*	*	*	1	0	0	0	1	Character pattern (2)
											0	0	1				0	1	0	1	0	
											0	1	0				1	1	1	1	1	
											0	1	1				0	0	1	0	0	
											1	0	0				1	1	1	1	1	
											1	0	1				0	0	1	0	0	
											1	1	0				0	0	1	0	0	
											1	1	1	*	*	*	0	0	0	0	0	Cursor position
											0	0	0	*	*	*						
											0	0	1									
0	0	0	0	*	1	1	1	1	1	1												
											1	0	0									
											1	0	1									
											1	1	0									
											1	1	1	*	*	*						

At the following page, you will find an <u>instruction outline</u> for the HD44780 (notice the execution times).

The above description is not sufficient!
It is also highly recommended to consult the HD44780U data book.

Table 6 Instructions

Instruction	RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs

Instruction	RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8 bits, DL = 0: 4 bits N = 1: 2 lines, N = 0: 1 line F = 1: 5 × 10 dots, F = 0: 5 × 8 dots BF = 1: Internally operating BF = 0: Instructions acceptable										DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (corresponds to cursor address) AC: Address counter used for both DD and CGRAM addresses	Execution time changes when frequency changes Example: When f_{cp} or f_{OSC} is 250 kHz, $37\ \mu s \times \frac{270}{250} = 40\ \mu s$

Note: — indicates no effect.

The exercise

We are now going to implement a driver for the LCD display mounted at the “AVR Demo Board”.

The driver consists of the two files “lcd162.c” and ”lcd162.h”.

The display unit will be connected to PORT C (using the displays 4 bit mode):

```
[LCD]           [PORTC]
RS  (pin4) ----- bit 0
RW  (pin 5) ---- bit 1
E   (pin 6) ---- bit 2
DB4 (pin 11) --- bit 4
DB5 (pin 12) --- bit 5
DB6 (pin 13) --- bit 6
DB7 (pin 14) --- bit 7
```

Write the driver implementation in the file "lcd162.c", and specify the interface (function prototypes) in the header file "lcd162.h".

Totally implementing the driver is quite time consuming (many timing requirements have to be taken into account). Therefore, part of the driver is already implemented (and tested).

The header file "lcd162.h" and the partly implemented source file "lcd162.c" can be downloaded from Blackboard.

The content of these files is shown at the pages to follow.

```
/*------------------------------------------------------------------------
  File name: "lcd162.h"

  Driver for the PowerTip PC1602-F alphanumeric display.
  Display controller = HD44780U (LCD-II).

  Max. uC clock frequency = 16 MHz (Tclk = 62,5 ns)

  Connection : PORTx (4 bit mode) :
  [LCD]              [Portx]
  RS (pin4)  ------  bit 0
  RW (pin 5) ------  bit 1
  E  (pin 6) ------  bit 2
  DB4 (pin 11) ---   bit 4
  DB5 (pin 12) ---   bit 5
  DB6 (pin 13) ---   bit 6
  DB7 (pin 14) ---   bit 7

  Henning Hargaard, January 25, 2017
-------------------------------------------------------------------------*/

// Defining the used PORT:
// (here PORTC is used) :
#define DDR_lcd  DDRC
#define PIN_lcd  PINC
#define PORT_lcd PORTC

 //  Public:
 void LCDInit();
 void LCDClear();
 void LCDGotoXY(unsigned char x, unsigned char y);
 void LCDDispChar(char ch);
 void LCDDispString(char *str);
 void LCDDispInteger(int i);
 void LCDLoadUDC(unsigned char UDCNo, const unsigned char *UDCTab);
 void LCDOnOffControl(unsigned char cursor, unsigned char blink);
 void LCDCursorLeft();
 void LCDCursorRight();
 void LCDShiftLeft();
 void LCDShiftRight();
 //-----------------------------------------------------------------------
```

The implementation is based on a maximum microcontroller clock frequency of 16 MHz.
Based on this worst-case consideration, all lower clock frequencies then will operate as well.

Some of the necessary delays are implemented using the NOP instruction, executing in 1 CPU clock cycle.

In the header file, the Mega32 port, used for connecting the display is defined (in this example we use PORTC).

The only public function already implemented in the c-file, is the LCDInit() function.
The rest of the functions have to be implemented as part of the exercise.

The functionalities have to be as follows:

void LCDInit( ) must initialize the display, blank it and set "current display position" to the upper line leftmost position (invisible cursor).

void LCDClear( ) blanks the display and sets "current display position" to the upper line leftmost position.

void LCDGotoXY(unsigned char x, unsigned char y) sets "current display position" to character position x (0-15) and line number y (0-1).

void LCDDispChar(char ch) displays the character "ch" at "current display position".

void LCDDispString(char *str) displays the string "str" starting at "current display position". The 0-terminated string is pre-stored in memory, and the parameter is a pointer to this string.

void LCDDispInteger(int i) displays the value of the integer "i" at "current display position".
Hint: The standard C function itoa() is useful for converting an integer to a local ASCII string (demands including <stdlib.h>). After converting, the string can be send to the display using LCDDispString( ).

void LCDLoadUDC(unsigned char UDCNo, const unsigned char *UDCTab) loads one of the 8 user defined characters (in CGRAM) with a dot pattern, defined in an 8 byte array stored in flash.
The parameter UDCNo (0 to 7), defines which of the 8 CGRAM characters is to be loaded.
The parameter *UDCTab is a pointer to the array containing the bit (dot) pattern.
Hint: In the HD44780 data sheet (especially page 19), study how to load bit patterns to CGRAM.

void LCDOnOffControl(unsigned char cursor, unsigned char blink) selects visible cursor (or not), and selects whether the character at "current display position" is to blink (or not).
If the parameter "cursor" is 0, the cursor is set to invisible – otherwise it is set to visible.
If the parameter "blink" is 0, the character at the "current position" should not blink – otherwise it should blink.
Hint: Study the instruction "Display ON/OFF control" in the HD44780 data sheet.

void LCDCursorLeft( ) left-shifts the cursor.
Hint: Study the instruction "Cursor and Display Shift" in the HD44780 data sheet.

void LCDCursorRight( ) right-shifts the cursor.
Hint: Study the instruction "Cursor and Display Shift" in the HD44780 data sheet.

void LCDShiftLeft( ) left-shifts the text on the display.
Hint: Study the instruction "Cursor and Display Shift" in the HD44780 data sheet.

void LCDShiftRight( ) right-shifts the text on the display.
Hint: Study the instruction "Cursor and Display Shift" in the HD44780 data sheet.

Start the exercise by studying (+ understanding) the partly implemented driver:

```
/*----------------------------------------------------------------------
  File name: "lcd162.c"

  Description: Driver for the PowerTip PC1602-F alphanumeric display.
  Display controller = HD44780U (LCD-II).

  Max. uC clock frequency = 16 MHz (Tclk = 62,5 ns)

  Connection : PORTx (4 bit mode) :
  [LCD]                 [Portx]
  RS (pin4)  ------  bit 0
  RW (pin 5) ------  bit 1
  E  (pin 6) ------  bit 2
  DB4 (pin 11) ---   bit 4
  DB5 (pin 12) ---   bit 5
  DB6 (pin 13) ---   bit 6
  DB7 (pin 14) ---   bit 7

  Henning Hargaard, January 25, 2017
--------------------------------------------------------------------*/
#include <avr/io.h>
#define F_CPU 3686400
#include <util/delay.h>
// Enabling us to use macro _NOP() to insert the NOP instruction
#include <avr/cpufunc.h>
#include "lcd162.h"

// library function itoa() is needed
#include <stdlib.h>

// Defining the used port pin numbers
// (the used port is defined in "lcd.h")
#define RS   0
#define RW   1
#define E    2
#define BUSY 7

// Display = 2 lines x 16 characters
#define NUMBER_OF_LINES 2
#define NUMBER_OF_CHARS 16
#define LINE2_START_ADR 0x40
```

```
//********************** PRIVATE (static) operations ********************
static void E_High()
{
  // Set the E pin high
  PORT_lcd |= 1<<E;
  // Min 230 ns E-pulse-width : PWEH
  _NOP();
  _NOP();
  _NOP();
  _NOP();
}

static void E_Low()
{
  // Set the E pin low
  PORT_lcd &= ~(1<<E);
  // Enable cycle time : Min 500 ns
  _NOP();
  _NOP();
}

static void waitBusy()
{
unsigned int counter = 0;
unsigned char BusyStatus;

  // DB7-DB4 = input
  DDR_lcd &= 0b00001111;
  // RW = 1, RS = 0
  PORT_lcd |= 1<<RW;
  PORT_lcd &= ~(1<<RS);
  do
  {
    // Set pin E high (tAS > 40 ns gained via the call of E_High() )
    // - and wait tDDR (min. 160 ns)
    E_High();
    // Read BUSY flag (DB7)
    BusyStatus = PIN_lcd & 1<<BUSY;
    // Min 230 ns E-pulse-width : (PWEH > 230 ns is gained)
    E_Low();
    // Dummy "reading" AC3-AC0
    E_High();
    E_Low();
    // "Counter" used for implementing timeout:
    // If the Busy flag is not reset within (appr.) 100 ms, the loop is broken
    counter++;
  } while( BusyStatus && counter );
  // DB7-DB4 = output
  DDR_lcd |= 0b11110000;
}
```

```
static void sendInstruction( unsigned char data )
{
  // Wait for display controller ready
  waitBusy();
  // Write high nibble ::
  // RW = 0, RS = 0, E = 0, DB7-DB4 = Data high nibble
  PORT_lcd = (data & 0b11110000);
  // Set pin E high (tAS > 40 ns gained via calling E_High() )
  E_High();
  // Set pin E low (PWEH > 230 ns is gained)
  E_Low();

  // Write low nibble ::
  // RS = 0, RW = 0, E = 0, DB7-DB4 = Data low nibble
  PORT_lcd = (data & 0x0F)<<4;
  // Set pin E high (tAS > 40 ns is gained via calling E_High() )
  E_High();
  // Set pin E low (PWEH > 230 ns is gained)
  E_Low();
}

static void sendData( unsigned char data )
{
  // To be implemented
}
```

```
//*********************** PUBLIC operations *****************************

// Initializes the display, blanks it and sets "current display position"
// at the upper line, leftmost character (cursor invisible)
void LCDInit()
{
  // Initializing the used port
  DDR_lcd = 0xFF;  // bits 0-7 output
  PORT_lcd = 0x00; // bits 0-7 low

  // Wait 50 ms (min.15 ms demanded according to the data sheet)
  _delay_ms(50);
  // Function set (still 8 bit interface)
  PORT_lcd = 0b00110000;
  E_High();
  E_Low();

  // Wait 10 ms (min.4,1 ms demanded according to the data sheet)
  _delay_ms(10);
  // Function set (still 8 bit interface)
  PORT_lcd = 0b00110000;
  E_High();
  E_Low();

  // Wait 10 ms (min.100 us demanded according to the data sheet)
  _delay_ms(10);
  // Function set (still 8 bit interface)
  PORT_lcd = 0b00110000;
  E_High();
  E_Low();

  // Wait 10 ms (min.100 us demanded according to the data sheet)
  _delay_ms(10);
  // Function set (now selecting 4 bit interface !)
  // - and polling the busy flag will now be possible
  PORT_lcd = 0b00100000;
  E_High();
  E_Low();

  // Function Set : 4 bit interface, 2 line display, 5x8 dots
  sendInstruction( 0b00101000 );
  // Display, cursor and blinking OFF
  sendInstruction( 0b00001000 );
  // Clear display and set DDRAM adr = 0
  sendInstruction( 0b00000001 );
  // By display writes : Increment cursor / no shift
  sendInstruction( 0b00000110 );
  // Display ON, cursor and blinking OFF
  sendInstruction( 0b00001100 );
}
```

```c
// Blanks the display and sets "current display position" to
// the upper line, leftmost character
void LCDClear()
{
}

// Sets DDRAM address to character position x and line number y
void LCDGotoXY( unsigned char x, unsigned char y )
{
}

// Display "ch" at "current display position"
void LCDDispChar( char ch )
{
}

// Displays the string "str" starting at "current display position"
void LCDDispString( char *str )
{
}

// Displays the value of integer "i" at "current display position"
void LCDDispInteger( int i )
{
}

// Loads one of the 8 user definable characters (UDC) with a dot-pattern,
// pre-defined in an 8 byte const array
void LCDLoadUDC( unsigned char UDCNo, const unsigned char *UDCTab )
{
}

// Selects, if the cursor has to be visible, and if the character at
// the cursor position has to blink.
// "cursor" not 0 => visible cursor.
// "blink" not 0 => the character at the cursor position blinks.
void LCDOnOffControl( unsigned char cursor, unsigned char blink )
{
}

// Moves the cursor to the left
void LCDCursorLeft()
{
}

// Moves the cursor to the right
void LCDCursorRight()
{
}
```

```
// Moves the display text one position to the left
void LCDShiftLeft()
{
}

// Moves the display text one position to the right
void LCDShiftRight()
{

}

//------------------------------------------------------------------
```

The basic private function `static void` sendInstruction(unsigned char data) are implemented so that correct timing is ensured.

You should implement the basic private function static void sendData(unsigned char data) in a similar way (actually only RS has to be controlled differently).

These two functions sends instructions and display data (respectively) to the HD44780.

void waitBusy( ) waits for the display not to be BUSY.

void E_High( ) and void E_Low( ) are basic help functions.

Notice how the LCD data port "changes direction" in the function void waitBusy( ).

Implement the missing functions and c*arefully* test the driver by means of a test program, relevantly calling the various functions. Remember testing using different parameters.
If you do not want to write your own test program, a proposal is available for download at Blackboard.

Test the function LCDLoadUDC( ) by loading the display with the special Danish characters æ, ø, å, Æ, Ø and Å and then displaying them.