SYMLINK.DK

Using HD44780-based LCDs with AVR microcontrollers

Update 2009-11-17: A mistake in the pinout information for UNIQ/eVision GC-1602I1 was pointed out to me by Tyler. This has now been updated (and corresponds to the posted schematics, which was correct).

On this page I will describe how to communicate with HD44780-based LCDs (Liquid Crystal Displays) using an Atmel AVR microcontroller. The page is mainly meant as a place for me to keep information on the pinouts of the display modules I have used, and to gather information about these displays, for me and others to use.

I will not go into too much detail about how to control these LCD modules, since there are already a lot of pages on this subject. The approach taken on this page is to use a library written by someone else for the low-level stuff. If you need to make modifications to such a library to get support for some extra functions, you should be able to find some helpful information in the Links-section at the end of this page.

The HD44780-based displays

In many microcontroller applications some sort of display is needed to present information and status of the controller to the user. Although a few LEDs can provide a lot of information, a more advanced display is sometimes needed. A solution which is quite often seen is using a HD44780-based character LCD module. These modules come in various configurations from one line of 8 chars to 4 lines of 40 characters.

HD44780-based displays can be operated in either 8-bit or 4-bit modes. Apart from the data-bus, 2 or 3 control-lines are needed, namely RS (Register Select) which selects if the data written to the module should be interpreted as data or an instruction, R/W (Read/Write) which determines the data direction (this pin can be tied to GND to save an I/O-pin on the controller, but this will make it impossible to read back status from the display, so all data must be sent slow enough to live up to the worst-case delays found in the data-sheet), end finally E (Enable), which is used to make the display read the data-bus. Even with 8-bit microcontrollers, the displays are often used in 4-bit mode, since this allows the data and control connections to occupy only one I/O-port (8 bits, of which 6 or 7 are used).

When power is applied to the display, it will start in the unitialized state, which is indicated by the top line being all black, while nothing is displayed in the bottom line. If nothing shows up, this could be due to the contrast setting (the voltage applied to V0 - sometimes called Vee) being wrong. Note that some displays (extended temperature models) use a negative voltage for Vee. All the displays I have worked with use a positive Vee, so I haven't had any problems with the diagram shown below.

Now the initialization sequence needs to be sent to the module. If the R/W-line is not connected, it is important to play safe on the timing, to ensure proper initialization. Note that some of the initialization commands take much longer time for the display to process than normal instructions and data. One of the commands sent during the initialization-sequence will configure the display for either 4 or 8-bit mode. In 4-bit mode the two nibbles are sent one after another, both to DB7-DB4 on the display. I will not go into details about the initialization, but you can find much more elaborate information in various places on the net, such as here, or in the datasheet for the HD44780-controller. Note that the initialization also configures various parameters of the displays behaviour, such as if the cursor will be automatically advanced when writing a character, and if the cursor should be shown etc. When the initialization is complete, the black characters in the first line of the display should dissapear.

After initialization, an instruction can be sent to the display, in order to move the cursor to a specific position. For the 2x16 char displays I have worked with, the first line has character addresses 0x80-0x8F while the second line has 0xC0-0xCF. Note that other displays may have other configurations, but this seems to be the standard for this size display. The really big displays have two HD44780-controllers, and so have two enable-lines, controlling different parts of the display.

Character set

The HD44780 has the font built into ROM, including normal latin letters, numbers and punctuation, as well as some japanese and scientific symbols etc. It also allows 8 user-defined characters to be created, which will be mapped into the character-codes 0-7 and repeated at 8-F. The repeated userdefined characters allows the user to avoid using certain characters (such as 0 which is used in C to denote the end of a string).



As it can be seen from the character-set, no ROM characters are defined for characters 0x10-0x1F and 0x80-0x9F. Since I live in Denmark, ocationally I need to display the danish characters æ, ø, å, Æ, Ø and Å, which are not available in the character set. I have defined these as custom characters as shown below:



The corresponding hex-codes for the user-defined characters are as follows:

æ	00 00 1B 05 1F 14 1F 00
ø	00 00 0E 13 15 19 0E 00
å	0E 0A 0E 01 0F 11 0F 00
Æ	0F 14 14 1F 14 14 17 00
Ø	0E 13 15 15 15 19 0E 00
Å	0E 0A 0E 11 1F 11 11 00

The addresses of the custom characters in the character generator memory of the HD44780 starts af 0x40, and each character uses 8 bytes (normally the last line is not used in the character, since it contains the cursor, if enabled). Only the lower 5 bits of each byte is used in the character generator, so the higher bits can be set in order to avoid problems with sending certain values

(such as 00 which would conflict with the 0-termination of strings in C). So to redefine the first userdefined character, one would first set the address (by sending command 0x40), and then send the eight bytes (as data). The user-defined character can then be shown on the display by using either character code 0 or 8 (of course the address should first be changed to one that is in the displays region, eg. 0x80 for the first character on the first line).

Using HD44780 LCDs with AVR

This section describes the needed software and circuit for using an HD44780 LCD with an Atmel AVR microcontroller. The circuit depends a little on the software used, since there are a couple of modes the LCDs can be operated.

The HD44780-based displays can be operated in either 4-bit or 8-bit mode. When using 4-bit mode, the two nibbles of each byte is sent in sequence (high nibble first) on the high 4 bits of the data-bus (eg. DB7..DB4), and the low bits (DB3..DB0) are not connected (or they can be grounded). Selecting 4-bit or 8-bit mode is done during initialization (with a bit in the most siginficant nibble, which is sent first regardless of 4 or 8 bit). On microcontrollers it is quite common to operate HD44780 modules in 4-bit mode, since this conserves some I/O-pins on the microcontroller.

Apart from the 4 or 8 data-lines, an additional two or three signals are required. First of all is the Enable-line, which is used to tell the module to read data from the data-bus (or write to it). It can be thought of as the clock-line. If you wish to have more than one LCD-module connected to one micro-controller, they can share all signals except this enable-line.

Then there is the RS (Register Select) line. This makes it possible for the module to determine if the information should be interpreted as data (eg. characters) or as a command (such as moving the cursor, clearing the display etc).

Finally there is the R/W (Read/Write) line. This is used to switch the data-direction between the module and the mikrocontroller. If the amount of available I/O-pins on the microcontroller is very limited, this one can be spared, and the displays R/W-pin connected to ground. However, this means that any timing of the data sent must obey the restrictions (worst-case) from the LCD datasheet, and especially if the microcontroller uses an RC-oscillator for timing this can be a problem (because the timing must be set even slower to work in all cases). Using the R/W-line it is possible to request the status of the module, and determine if the previous command is done. This can speed up the access to the display, although polling this status-bit requires a little extra work from the microcontroller. One nice thing about this approach is that the code becomes unaffected by varying clock frequency.

Software

There are a couple of choices concering the software (or driver if you wish) for the LCD module. You can of course write your own. Normally I would only recommend this if you need some special feature or hardware setup, and if you can't find some existing code you can modify to suit

your needs. Or if you have severe constraints on the flash memory, and need to write everything in highly optimized assembler etc.

In other cases I recommend finding some existing code. There are a lot of good libraries to be found on the Net, ranging from very basic to quite advanced (with custom characters, bar-graphs etc). Different libraries exist for use with assembler or C (actually different versions for C exists, since there are different C-compilers that are not fully compatible). The example I will be using here is written in C, and made for use with avrgcc (a port of the GNU C-compiler and libraries for use with AVR). The easiest way to get this on windows is through the WinAVR-package, which bundles avrgcc avr-libc and a range of development tools, including an editor and software for device programming.

The library I used is lcd16 by Ron Kreymborg. It is very simple, and is only made for 2x16 character displays (although different sizes could be used with minor changes to the code).

To make lcd16 work with newer versions of avr-libc, some changes are required, since the inp, outp, sbi and cbi functions are now deprecated. I therefore created compat.h which defines a few macros to handle this. These should be included in main.c and lcd16.c.

Secondly, lcd16 has no support for user-defined characters, so I have made a funtion that adds this support. I have also changed the main.c-file to define and print some user-defined characters, and the counters are gone (but my changes to the library should work fine with Ron Kreymborg's original main.c-file).

Finally I have used a standard Makefile from WinAVR and adapted it to the lcd16-package. I use an updated AVRdude, so you may have to change the makefile to use a different programmer type, if you are using the standard distribution.

My updated version of the library and example-program can be downloaded as lcd16.zip.

Here is a picture of an LCD module displaying some characters, including 8 user-defined ones. This is the output from my modified version of lcd16:



If you want to look at a more advanced library for HD44780-modules on AVR-controllers, take a look at Procyon AVRlib, which contains a lot of C functions for different tasks using AVR-controllers, including support for HD44780-based LCDs (in the lcd.c-file).

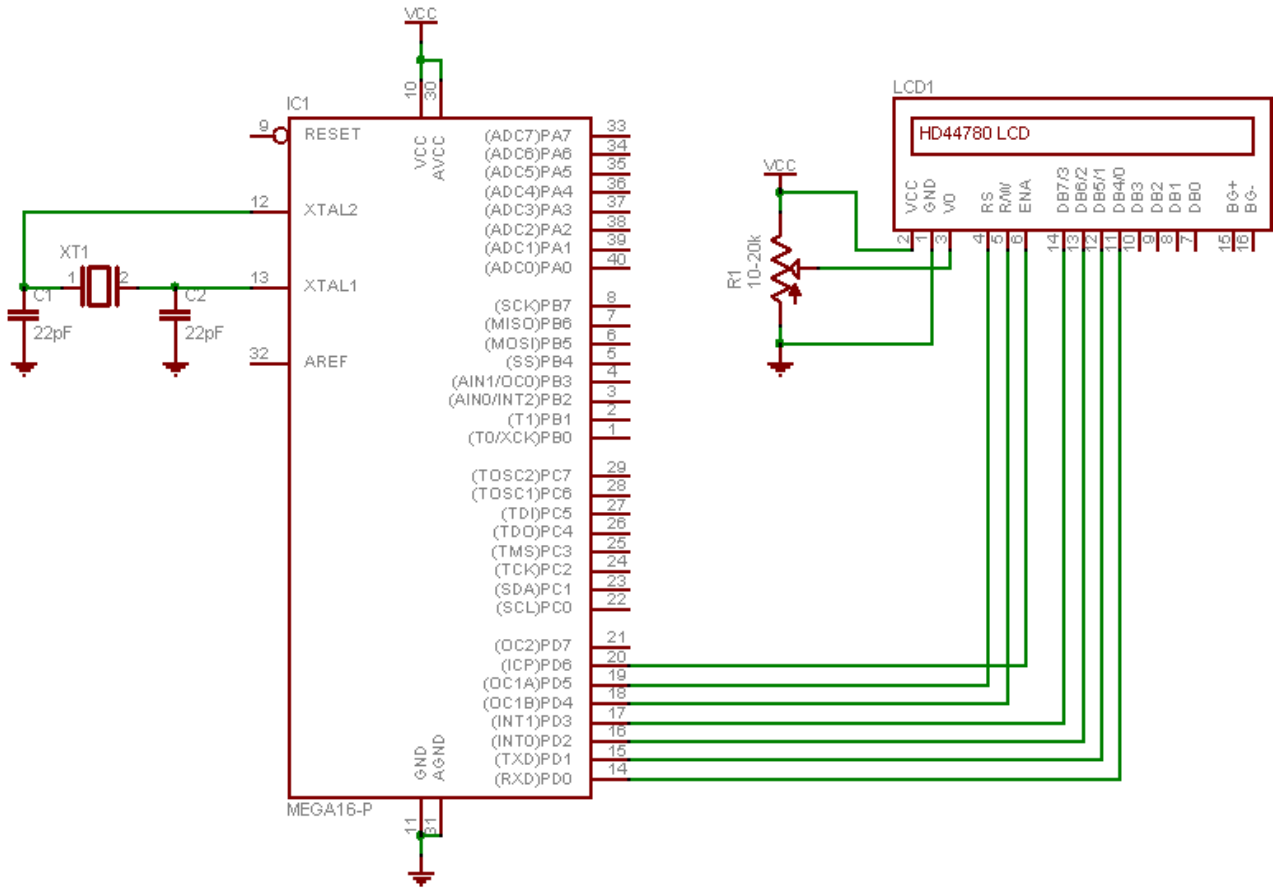
Circuit

The circuit I describe here is a very basic setup for driving a 2x16 character HD44780-based LCD from an Atmel ATmega16. It uses (a slightly modified version of) Ron Kreymborg's lcd16 (described above), and should work on most AVR controllers, when compiled with avrgcc (e.g. WinAVR).

When you choose a library for driving an LCD, it is important to note if it makes any constraints on the I/O-ports used. lcd16 uses bi-directional communication with the LCD, so we need to use the R/W-line. Furthermore the module uses 4-bit mode and the data-bits must be on the low for bits of a port (the port used can be changed in a header-file before compilation). The three additional I/O-lines are in the same port, so apart from being able to use the last bit for something else, this library takes up an entire 8-bit port of the microcontroller.

In the example presented here, I have simply used PORTD, which was the default configuration in the lcd16-library I downloaded. You may want to use a different port for your application, since the two low bits of PORTD are the ones used for the UART (in case your application needs to use serial communication). If you need to connect the module to other pins (apart from changing the entire port), you should probably consider using a different library.

The test-setup I made is shown in the figure below.



Actually I used the STK500-kit and mounted the LCD module on a bread-board, connected to the PORTD-connector of the STK500. I used a separate 5V power-supply for the LCD, since the backlight can be a little heavy on the power-consumption, and I do not know how much power the STK500 can supply.

Notice the trimmer potentiometer R1, which makes a voltage-divider between VCC and ground. This is connected to the V0 (or Vee)-pin of the display and allows the contrast of the display to be adjusted. The value of the potentiometer is not very important, but 10-20k seems to work well.

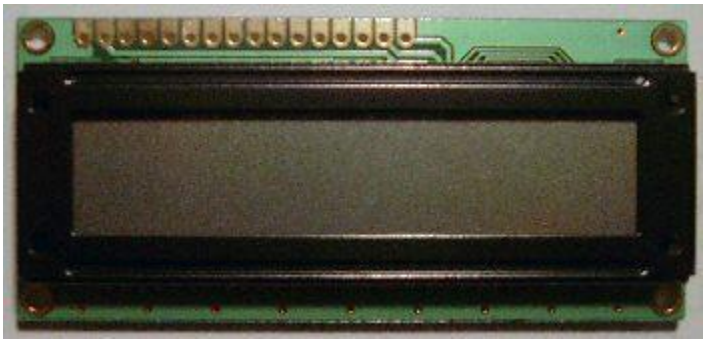
The LCD modules

I will now list the pin-configurations of a few types of diplay modules, which I have been using for various things. Both are 2x16 chars (with the Japanese character ROM). I have some other modules (4x20 and 2x40 chars), but I have not yet found any use for these, so I havent looked much at the pinout for these modules.

Basically, if you have an LCD module, which appears to be character based, there is a high probability that it is HD44780-based (and if it uses FP-80 footprint, you can just look at the type-number of the controller). If your module has 14 or 16 pins, it shouldn't be too hard to figure out the pinout. 14-pin versions either have no backlight, or the backlight is supplied from the VCC

and ground-pins. For 16-pin models, normally pin 15 and 16 are for the backlight, although some modules do not use these pins, and supply the backlight from the main supply (and on some models this can be selected, typically by solder-jumpers on the back). Note that the industry standard pin-out is the one described for the UNIQ/eVision. The pin-out of the KFC-model displays are reversed, and the VCC and ground pins are swapped as well. Normally you will be able to tell which pins are VCC and GND by looking at the size of the PCB traces. Also, the GND-pin is often connected to the metal-frame arround the actual LCD. If you have found VCC and GND in either end of the connector, there is a good chance that the remaining pins fit the industry standard.

UNIQ/eVision GC-1602I1



I bought a box of 48 pcs of these 2x16 char LCD modules on eBay. The modules have standard green backlight (with black characters). PCB dimensions are 80x36mm, and the viewable area of the display is 65x14mm.

The displays have single-row 16-pin connections, which should be on the top to have the characters appear correctly on the display. Although the 16-pin connector suggests separate connections for the backlight, the displays I have are wired to supply the backlight from the main power-pins. This can be configured by some solder-jumpers on the back of the module. My tests have shown that J1 and J2 connect the anode of the LED backlight to pin 15 of the display, with J2 using a current-limiting resistor, suitable for 5V backlight operation. Otherwise J3 can be used, connecting the anode to pin 2 (+5V) again with the resistor. Only one of J1, J2 and J3 should be used. Similarly, J4 connects the cathode to pin 16, and J5 to pin 1. Again, only one of J4 and J5 should be used. The standard configuration with the backlight powered from the main supply is achieved with J3 and J5.

I recently bought some other displays with the type number JHT162A (No manufacturer name, although there is a small logo which looks like a Y inside a circle). These modules are with white LED backlight and blue background. The size and pin-configuration of this module is identical to the UNIQ/eVision-types, although these have no solder jumpers to configure the backlight (this is always supplied on pins 15 and 16, with the module having an appropriate current-limiting resistor for 5V operation). My experiments show that it is extremely difficult to read anything on the display with the backlight off, due to the dark blue filter in front of the display, so I don't think you would want to run the display without the backlight.

The pin-configuration of these display are as follows:

pin	symbol	description
1	GND	Ground
2	Vcc	Vcc (+5V) also powers backlight
3	V0	Contrast adjustment
4	RS	Register select: low = instruction, high = data
5	R/W	low = write, high = read
6	E	Enable (active high)
7	DB0	Data-bus bit 0 (not used in 4-bit mode)
8	DB1	Data-bus bit 1 (not used in 4-bit mode)
9	DB2	Data-bus bit 2 (not used in 4-bit mode)
10	DB3	Data-bus bit 3 (not used in 4-bit mode)
11	DB4	Data-bus bit 4
12	DB5	Data-bus bit 5
13	DB6	Data-bus bit 6
14	DB7	Data-bus bit 7
15	LED+	Positive backlight supply (if used)
16	LED-	Negative backlight supply (if used)

The KFC-model



I got a hold of these modules when I worked at Zitech Computer, way back. These modules were used in 17" (and later 15") monitors from KFC (which has no affiliation with the fastfood gigant using the same TLA). I don't even know if the brand of computer monitors called KFC still exists. My searches seems to say that KFC stands for Kuo Feng Corp, and the brand is now known as Smile. So why do I have a bunch of these displays? Well the guy working in monitor repair at Zitech said that a common error on those monitors was that the front-panel switches failed, and since it was easiest to replace the entire front panel, the old ones were often just thrown away. So I asked him to put some aside, and I still have a bunch of them.

The modules are standard 2x16 char with green LED backlight, and aparently come from at least three different manufacturers:

- Solomon LM1230SYL - These exists both with standard HD44780 in FP-80 package as well as a version with PCB mounted, epoxy-bonded chips
- Wayton M.I.T. MC1602GSYLU/PC1602B4 - with epoxy-bonded chips
- PVC160210AYL - HD44780 in FP-80 and an Oki M5259 instead of the normal HD44100H. This module has a gray frame instead of the normal black one.

All of these modules have the same dimensions, with a PCB size of 85x30mm and a display area of 64x16mm. They have a 2x7 pin connector to the left side of the display (when viewed from the front). Note that the text on the back of the PCB is upside-down compared to the text on the LCD.

The power for the backlight is supplied from the Vcc and Gnd connections. If one wants to be able to turn off the backlight, the displays have small solder jumpers (or the current-limiting resistors can be removed).

NOTE: When comparing to the pinout of other LCD modules, these models seem to have reversed the pin numbering, and also reversed the Vcc and Gnd connections. The pins are numbered on the PCB. When viewed from the front of the display, pin 1 is in the top left corner.

pin	symbol	description
1	DB7	Data-bus bit 7
2	DB6	Data-bus bit 6
3	DB5	Data-bus bit 5
4	DB4	Data-bus bit 4
5	DB3	Data-bus bit 3 (not used in 4-bit mode)
6	DB2	Data-bus bit 2 (not used in 4-bit mode)
7	DB1	Data-bus bit 1 (not used in 4-bit mode)
8	DB0	Data-bus bit 0 (not used in 4-bit mode)
9	E	Enable (active high)
10	R/W	low = write, high = read
11	RS	Register select: low = instruction, high = data
12	V0	Contrast adjustment
13	GND	Ground
14	Vcc	Vcc (+5V) also powers backlight

Links

- Peter Ouwehand's How to control a HD44780-based Character-LCD.
- LCD Module FAQ by Christopher Burian.
- Ron Kreymborg's lcd16 library for AVR's with avrgcc.
- Procyon AVRlib. Library of a lot of C-functions for AVR-controllers, including a more comprehensive HD44780-library.
- AVR-Tutorial - 4. LCD. Describes how to use a HD44780 with an AVR microcontroller programmed in assembler (in German).
- Using An LCD In 4 bit Mode. Another introduction to HD44780's and AVR in assembler.
- Scott Bruck's LCD FAQ Describes the physics and principles of operation of LCDs.