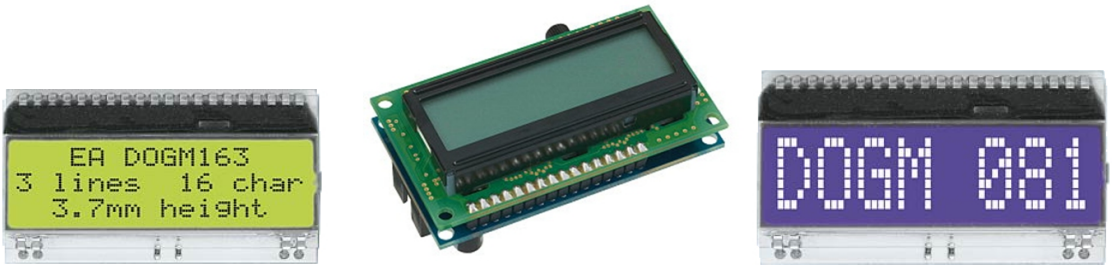
Exercise 8: Liquid Crystal Display Modules

Overview

During the past exercises, we have acquired and processed various sensor values. However, our only output channels so far were LEDs and the debug console. LEDs are very limited when it comes to visualization of information and the debug console requires the presence of a host PC. This is why we will now focus on visualization of information using liquid crystal displays (LCDs). LCDs are available in many different forms and shapes (compare figure 1) and can be used to display text and/or graphics. In this lab course, we will however focus on a text-based LCD.



(a) **(b)** **(c)**

Figure 1: Different LC display modules

Some LCDs have an integrated controller IC. Those devices are usually called *LCD modules*. An LCD module offers a communication interface which can be used to directly program the LCD using a digital protocol and is similar to the way we communicated with the digital temperature sensor from an architectural point of view (i.e., the LCD has a “local intelligence”). Most LCDs are also capable of being backlit. This improves readability.

In this exercise, we will use the text-based LCD module EA DOGM163L-A, which allows to display 3 rows with 16 characters each (figure 1 **(a)**). It incorporates an ST7036 LCD controller/driver which supports the character set that shown in figure 2. Note that for alphanumeric characters, the character code corresponds to the ASCII character code, meaning if you want to send a capital ’A’ to the display, instead of sending the number `0b01000001` (which corresponds to 65), you can simply write `'A'`. The first eight characters (address `0x00` to `0x07`) are programmable.

The EA DOG-M LCD Module Family

The DOG-M LCD Module Family features an integrated ST7036 display controller. The file `EA-DOG-M.pdf` on the lab course file server and the website describes the features and technical details of the controller and the display itself. The ST7036 controller provides four different communication protocols for driving the display of which three are accessible on the EA DOGM display family:

- 8 bit mode: In addition to common control lines, eight data lines are used for communication.
- 4 bit mode: In addition to common control lines, four data lines are used for communication. This means that a single data byte has to be split into two 4 bit “nibbles”.
- SPI mode: In addition to common control lines, two data lines are used for communication. The two lines implement the SPI protocol (*Serial Peripheral Interface*), a widely used inter-chip communication protocol in microcontroller applications.

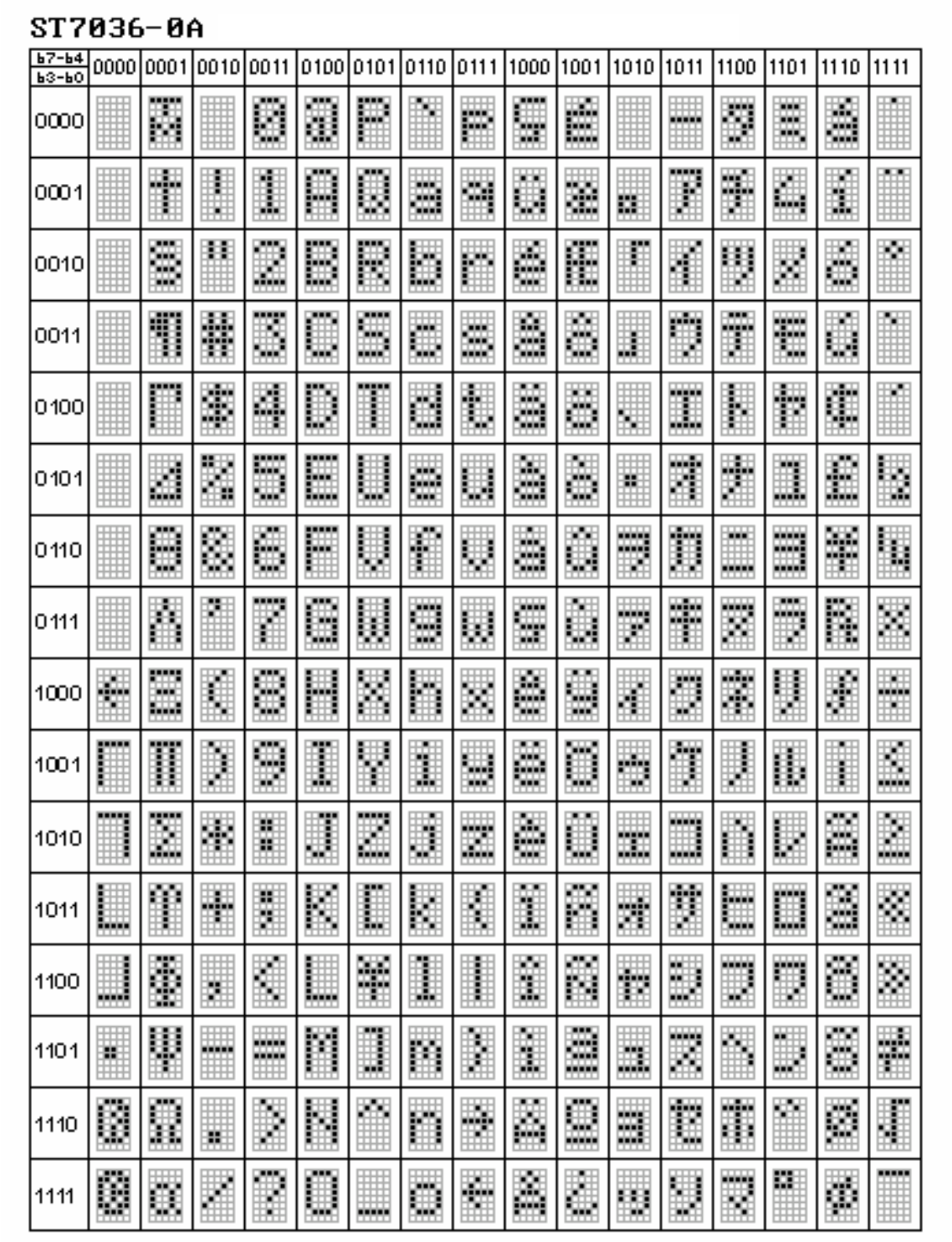
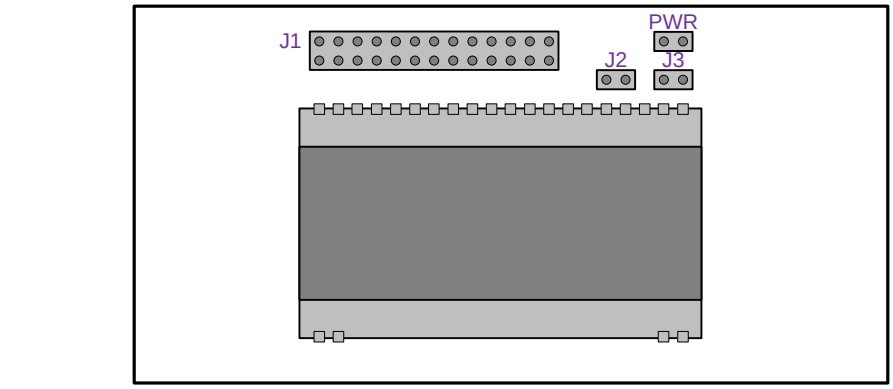


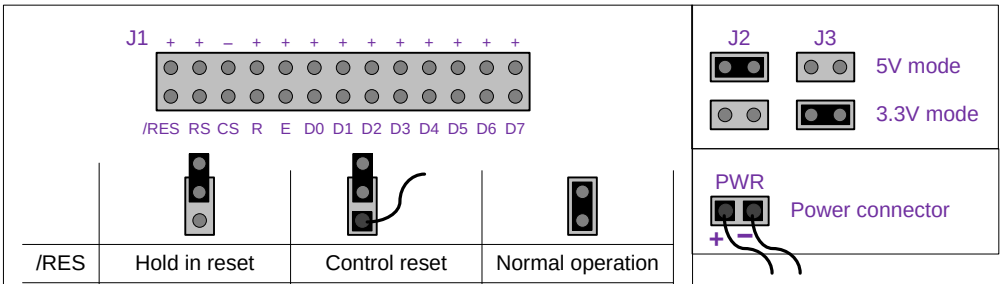
Figure 2: Character set of the EA DOGM LCD module family; characters `0x00` to `0x07` can be freely programmed by the user

For ease of use, we use a special carrier board for the LCD module that allows to configure it for different usage scenarios, including 8 bit mode, 4 bit mode and SPI mode. The configuration is applied by setting various jumpers. Figure 3 shows the valid configurations.

The different columns in the table at the bottom describe three different usage scenarios per jumper on the `J1` connector. The first column shows the configuration when the jumper is left open (to prevent the jumper from being lost, please attach it to the upper contact on connector `J1` in this case). The second column shows the configuration when the microcontroller is used to set the respective pin to high or low using the attached cable. The third column shows the configuration when the jumper is used to preset the value for the respective connector.

`J2` and `J3` allow configuring the display for 5V or 3.3V applications. Since ATmega168 runs with 5 V, the jumper has to be set on `J2`. Finally, the `PWR` connector is for powering the LCD module. Please respect the correct polarity!





/RES	Hold in reset	Control reset	Normal operation
RS	–[1]	Control Cmd/Data	–[1]
CS	Chip not selected	Control CS	Chip selected
R	–[1]	Control Read/Write	SPI mode
E	–[1]	Control Enable	SPI mode
D0	–[1]	Control D0	SPI / 4 Bit mode
D1	–[1]	Control D1	SPI / 4 Bit mode
D2	–[1]	Control D2	SPI / 4 Bit mode
D3	–[1]	Control D3	SPI / 4 Bit mode
D4	–[1]	Control D4	SPI mode
D5	–[1]	Control D5	SPI mode
D6	–[1]	Control D6	–[1]
D7	–[1]	Control D7	–[1]

[1] Do not use

Figure 3: Valid jumper configurations on the LCD module carrier board

Serial Peripheral Interface

Exercise 8.1

a) Configure the LCD carrier board for SPI mode. Connect the board to STK500 using the correct pins for SPI communication and set up the jumpers.

b) Implement a routine `spi_init()` that initializes the SPI interface using the following configuration: Master mode, most significant bit first, data mode 0, data rate 2 Mbit/s.

LCD Driver Implementation

Exercise 8.2

a) Implement a routine `lcd_init()` that initializes the display over SPI. Include `util/delay.h` to be able to use the `_delay_ms()` and `_delay_us()` functions. Make sure that the CPU frequency is set up correctly in the project configuration options.

b) Write a routine `int lcd_putchar(char c, FILE* unused)` and a test program to write a string to the display. In addition to the `stdout` redirection to the debug console, you might want to assign the `stderr` channel to print to the LCD as follows:

```
/* File handle for "error" output on LCD */
static FILE lcd_stderr =
    FDEV_SETUP_STREAM(lcd_putchar, 0, _FDEV_SETUP_WRITE);
stderr = &lcd_stderr;
```

Then you can use the functionality as follows:

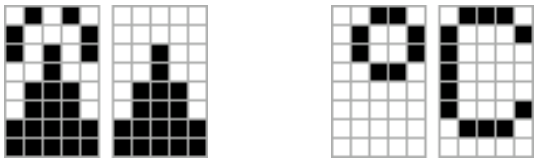
```
fprintf(stderr, "Hello world!"); /* Print to display */
```

LCD Applications

Exercise 8.3

a) Use one of the applications from the previous sessions to visualize data in the LCD (e.g., the current DCF77 time or the current temperature). Document your solution appropriately.

b) Read the document `st7036.pdf` to learn how to program custom characters and use them to prettify your output. For the DCF77 time example, you might want to switch between the two "radio controlled" icons shown in figure 4 **(a)** as soon as the time is synchronized. For the temperature sensor application, you might want to add a custom "degree" sign (aligned to the right in contrast to character `0b11011111`) to be able to display the text "°C" (compare figure 4 **(b)**; you can use a normal "C" as second character in this case).



(a) **(b)**

Figure 4: Custom LCD characters