Computer Architecture and Organization

8051 Project 4: LCD control

1 Project's goal

The goal of the projects is to create an assembly program to control an LCD text display. The program should consist of three independent modules:

- `wait` – implementing for timer based waiting,
- `LCDcomm` – implementing for low level communication with the LCD component, and
- `LCDcontrol` – main module used to program the desired text into the LCD and to move the text in the display from left to right (i. e. to *scroll the text*) and back in an endless loop.

2 LCD simulation module

As an add-on to Keil's μVision the LCD simulation module integrates into the μVision IDE and enables debugging of communication between the microcontroller and an LCD text display.

Important: Complete the steps of this section **before** opening the project file.

2.1 Installing the LCD simulator module

The directory `LCDsim` of the project zip file contains an LCD simulation module for Keil's μVision. Copy the `LCDsim\C51` directory to your μVision installation directory. Example: if μVision is installed in `C:\Keil` copy `LCDsim\C51` to `C:\Keil` (there is already a directory named `C51`).

Next open the file `TOOLS.ini` located in your μVision installation directory using a text editor. Add the following line to section `[C51]`:

```
AGSI2=LCD.DLL ("LCD simulation")
```

2.2 Configuring the LCD simulation module

The module is configured by editing the text file `LCD control\lcd-display.lcd`. You can use any text editor (e. g. Windows notepad) to edit the file.

When first opened the file should look as shown in Fig. 1.

```
[Version]
Version=0.2
[LCD_0]
Manufacturer=BATRON
OrderNr=BT21605 (2x16)
PortWidth=4
D0_NAME=
D1_NAME=
D2_NAME=
D3_NAME=
D4_NAME=P1^3
D5_NAME=
D6_NAME=
D7_NAME=
RS_NAME=
RW_NAME=
EN_NAME=
```

Figure 1: Contents of the LCD module configuration file.

In order to configure the module you have to complete the lines `D4_NAME` to `EN_NAME`. The lines above `D4_NAME` should not be changed.

The correct value for `D4_NAME` is already entered to show the syntax. You should be able to enter correct values for the remaining lines after studying Section 3.

3 System description

We assume the following system setup: an 8051 compatible microcontroller is connected to an LCD text display module using lines of 8051's parallel port P1, as shown in Fig. 2. The LCD text display controller is compatible to the Hitach HD44780U, the de facto standard for LCD text displays. In the `docs` directory you will find a copy of the HD44780U data sheet. The document explains the function of the pins RS, R/$\overline{\text{W}}$, DB7 to DB0 and EN, see pages 3, 8, 22 and 24-25.

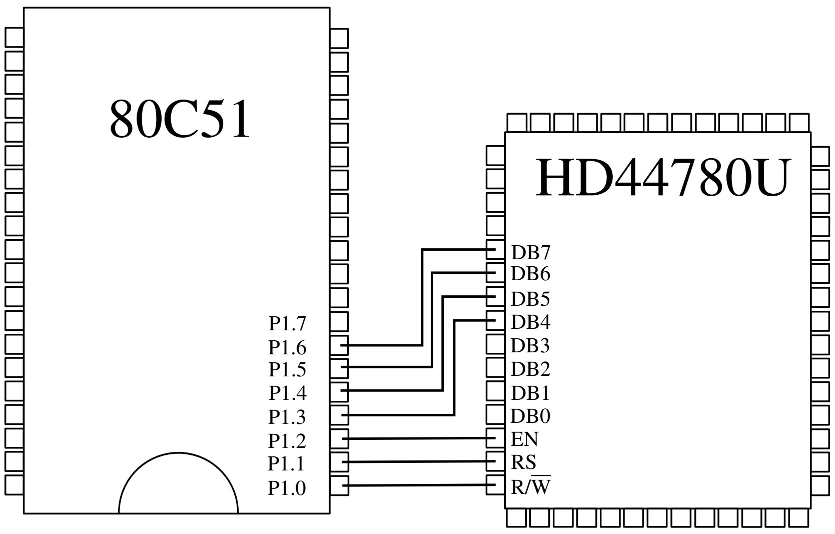


Figure 2: System setup: connections between the 8051 and the LCD text display controller chip.

In order to initialize and program the LCD text display the communication protocol described in the HD44780U data sheet needs to be implemented. For the purpose of this project we assume that[1]:

- $R/\overline{W}$ is connected to a logical '0' signal. This means that:
  - busy flag function of the LCD module cannot be used,
  - the correct timing of the LCD communication must be ensured by using sufficiently long waiting times (cf. Table 6 on page 24 of the data sheet),
  - the initialization routine you will write in Section 5.4 has to set the $R/\overline{W}$ line to '0'.
- Lines DB3 to DB0 are not connected to the 8051. This means that we can communicate to the LCD component in 4-bit mode only (see pages 32 to 33 of the data sheet for the explanation of 8-bit and 4-bit modes).

μVision LCD simulation module configuration
Complete now the configuration file of the μVision LCD simulation module as explained in Section 2.2.

4 Module `wait`

Files: `wait.a51` and `wait.inc`

When controlling the LCD module different waiting times need to be achieved: some are in microsecond range, others in millisecond range.

The `wait` module should provide the `wait` routine which can be configured to wait for a specified number of time units. The routine supports two time units: 1 microsecond and 1 millisecond. The waiting should be done in idle mode as much as possible.

Parameters:

- number of time units to wait – specified in R3,
- time unit selection – specified by PSW.1: 1 – millisecond, 0 – microsecond.

4.1 Time unit configuration

Assume 12 MHz oscillator clock. Make sure to set the correct value in Keil (*Options for Target→ Target→ Xtal*) in order to obtain meaningful results from the simulation.

4.1.1 Microsecond range

Assume that a single timer tick takes 1 μs. Compute the initial timer value based on the assumption that R3 timer ticks are a good enough approximation for R3 μs.

[1]This is the most commonly encountered configuration.

4.1.2 Millisecond range

Configure the initial timer value in such a way that an interrupt is generated 1 ms after the start. Re-trigger the timer R3 times.

4.2 Timer and interrupt setup

Complete the `wait` module by writing the code to configure the timers and interrupts, and an interrupt service routine. You can use the code from project 3 as starting point. Write the code in such a way that a single routine can be used for both time units.

4.3 Exporting routines

The routine `wait` is exported from the module using the `PUBLIC` assembler statement in `wait.a51`.

The include file `wait.inc` contains the declaration of the `wait` routine. By including the file in other modules the `wait` routine can be used in those modules.

5 Module `LCDcomm`

Files: `LCDcomm.a51` and `LCDcomm.inc` This module contains low level routines for communicating with the LCD module. Make sure you understand the basic function principle of the LCD module, as well as the system setup. Refer to Section 3 for more information.

5.1 Importing routines

The `wait` routine is imported from the `wait` module. Also two constants are defined in `lcdhw.inc`, which is also included.

5.2 `LCD_send_nibble` routine

The `LCD_send_nibble` routine is used to perform the actual data transfer to the LCD display controller using the 4-bit interface. The routine implements the timing diagram of Fig. 9, page 22 of the HD44780U data sheet. However, in our setup $R/\overline{W}$ is bound to ground (logic '0'). Thus busy flag check cannot be done. Instead, after sending a command to the LCD, we must allow sufficient time for its processing.

The routine first extracts the lower nibble (the 4 LSbits) from the accumulator and shifts them so that they correspond to the four signal lines used for transferring data to the LCD (cf. Fig 2). To make the code easily adaptable to other setups three constants are introduced in file `LCDcomm.a51`.

`LCD_DATA` specifies the port used for transferring the data (connecting to lines DB7 to DB4 of the LCD),

`LCD_DATA_SHIFT` specifies by how many bits the four lines of port `LCD_DATA` connecting to DB7:4 are shifted from bit 0 (it is 3 in case of the system from Fig. reffig-system where P1.3 to P1.6 are used),

LCD_DATA_MASK mask applied to the port LCD_DATA before writing a new value to the LCD; contains a '1' for every bit NOT used for communicating with DB lines, '0' for the lines used.

Refer to the comments in the file for more information.

Implementation Complete the code of the routine LCD_send_nibble, so that it performs the specified function. The interface is documented in the file LCDcomm.a51.

Test your code.

5.3 LCD_send_byte routine

The LCD_send_byte routine is used to perform the actual data enables data transfer of while bytes using the 4-bit interface.

Implementation Complete the code of the routine. The routine interface is documented in the file LCDcomm.a51
Make use of the routine LCD_send_nibble.

Hint: SWAP instruction can be helpful.

5.4 LCD_init routine

This routine implements the initialization sequence of the LCD according to the diagram of Fig. 24, page 46 of the HD44780U data sheet,

Hint: The initial configuration of the LCD is slightly different than in the diagram. Refer to the comments in the code to determine the correct configuration, and to Table 6, page 24 of the data sheet to determine the correct coding.

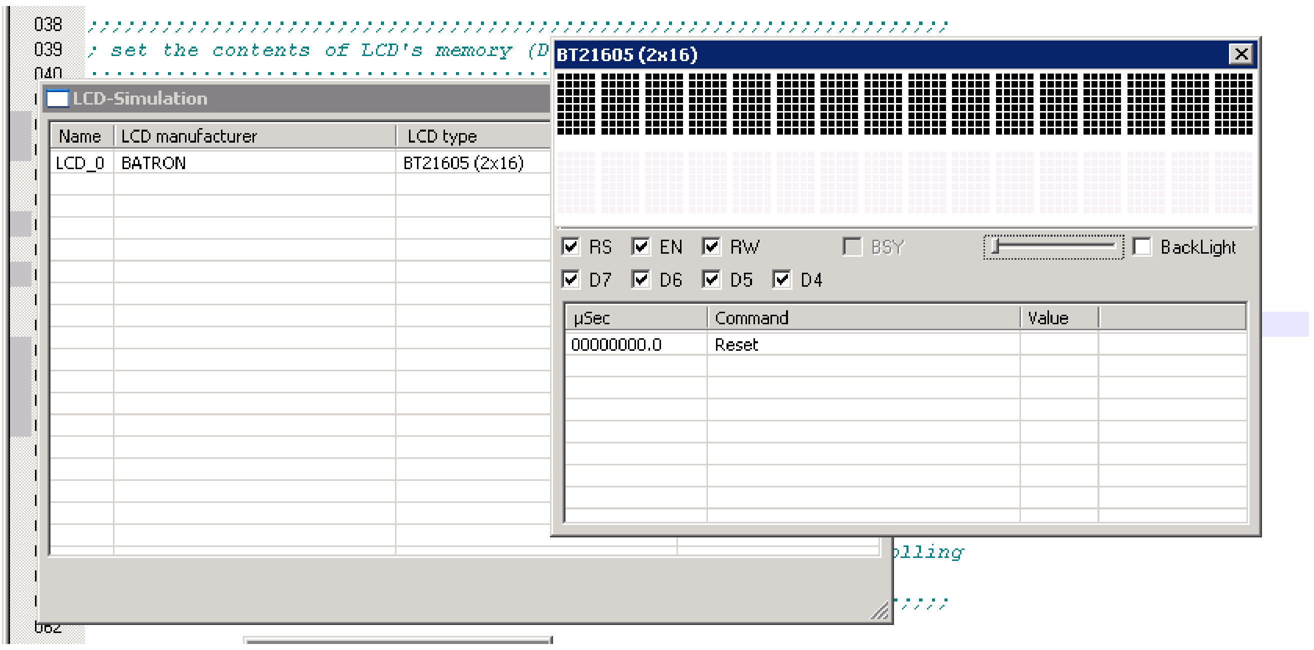
5.4.1 Using the LCD simulator module

To bring up the LCD simulator module window, make sure you have the LCD-Simulation window open (see Fig. 3). You can toggle it by selecting *Peripherals→LCD Simul*.

In the LCD-Simulation window click on the line listing the LCD model (LCD_0, BATRON, etc.) to open the BT21605 window. It shows the text display, values of the controlling signal lines, as well as the log of activity on the interface.

5.4.2 Implementation

Complete the code of the routine. Test your code using the LCD simulator module. After completing the initialization you should see both lines of the display cleared and the horizontal bar of the cursor in the leftmost field.


```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; set the contents of LCD's memory (D
```

LCD-Simulation

Name	LCD manufacturer	LCD type
LCD_0	BATRON	BT21605 (2x16)

BT21605 (2x16)

☑ RS ☑ EN ☑ RW ☐ BSY ☐ BackLight
☑ D7 ☑ D6 ☑ D5 ☑ D4

µSec	Command	Value
00000000.0	Reset	

Figure 3: Windows of the LCD simulation module.

5.5 Utility routines

Complete the routines:
`LCD_scroll_right`, `LCD_scroll_left`,
`LCD_activate_1stline`, `LCD_activate_2ndline`.
Refer to the comments in the file for more information. The correct values that need to be sent to the LCD can be found in Table 6, page 24 of the data sheet, for an explanation of the difference and the meaning of the RS line.

5.6 `LCD_send_text` routine

This routine is used to transfer the text to be displayed to the LCD. This is different from previous transfers, as we are now sending data to the LCD and not a command anymore. See pages 8 to 9 and Table 6, page 24 of the data sheet.

The text is read from the code memory. Consult the comments in `LCDcomm.a51` for more information.

Hint: Use the base register addressing mode `MOVC A, A+DPTR` to access the text stored in the code memory.

Implementation Complete the `LCD_send_text` routine.

5.7 Exporting routines

Export the following routines from the `LCDcomm` module:
`LCD_init`, `LCD_send_text`,

`LCD_scroll_right`, `LCD_scroll_left`,
`LCD_activate_1stline`, `LCD_activate_2ndline`.

Also declare them in the file `LCDcomm.inc` so that they can be used from other modules.

6 Module `LCDcontrol`

Files: `LCDcontrol.a51`

6.1 Importing routines

Import both `wait` and `LCDcomm` modules.

6.2 Program entry and initialization

The `main` label marks the program entry. First, the stack is configured and the LCD transfer is initialized. Then, the text defined at the labels `TXT_LINE1` and `TXT_LINE2` is transferred to LCD. Finally, the program enters the endless loop of the `do_scrolling` routine.

6.3 Copying text

In order to copy text data to the LCD, first one of the two memory areas in LCD's data memory (DDRAM, cf. HD44780U Block Diagram, page 3 of the data sheet) needs to be selected. DDRAM area starting at the address 0x00 corresponds to the first line of the display, the DDRAM area at 0x40 to the second line.

Use the `LCD_activate_1stline`, `LCD_activate_2ndline` routines to achieve the desired effect, and the `LCD_send_text` routine to do the actual data transfer. Make sure that `DPTR` points to the correct text (`TXT_LINE1` and `TXT_LINE2`).

Implementation Complete this routine. Test your code using the LCD simulation module. After calling this routine from `main` you should see the first 16 characters of both lines in the LCD display.

6.4 Scrolling routine

We use the feature of the LCD controller which enables us to select the area of the text memory (DDRAM) which is shown in the display. By changing the address continuously (with some delay) we achieve the effect of sliding text (scrolling).

Implementation Complete the routine. Refer to the comments in the source code for more information. Test your code. In the LCD simulation window you should be able to see the scrolling text.