Tutorial: Introduction to LCD Programming 2: Driving 4-bit Mode

Contributed by H. en C. van den Broek

This second LCD-tutorial is based on 'Introduction to LCD Programming', so you should to read that first.

In the first tutorial we have seen that character LCD's based on the HD44780 chip can be driven in 8bits mode, which requires in total 11 lines from you microcontroller. If we want (or need) to spare some lines for other purposes it is possible to drive the display in 4bits mode, which requires 7 lines. It is possible to use only 6 lines, in which case R/W is tied to GrouND. This configuration is seen many times in projects. Instead of reading the busy flag (which is somewhat trickier than it is in 8 bit modus) we have to use delay loops. These delayloops have to be recalculated when using other oscillator frequencies. In this tutorial we use the somewhat harder solution, making use of the busy flag and being independent of the oscillator frequency. The only drawback using 4 bits is that commands and data have to be sent in two nibbles (4bit parts) to the display, which take slightly more time. In many cases that won't be a problem.

To keep things simple, I will take the examples from the first tutorial, make the necessary changes and only explain the differences.

HARDWARE CONFIGURATION

The only difference with the 8bit version is DB0, DB1, DB2 and DB3 on the displaymodule side. These lines are not connected to the processor. Leave those lines unconnected, DON'T SHORT THEM TO GROUND as seen in projects where R/W is tied to ground.

So the initial equates are:

Message Board
Tutorials
8052 FAQ
News
Code Library
Chips

```
DB4     EQU   P1.4
DB5     EQU   P1.5
DB6     EQU   P1.6
DB7     EQU   P1.7
EN      EQU   P3.7
RS      EQU   P3.6
RW      EQU   P3.5
DATA    EQU   P1
```

Again, if you want to know how to handle the control lines in your programs, please read the first tutorial on LCD displays. In 4-bit mode, we have to read and write databytes and commandbytes in two separate ’nibbles’ (4bit parts). To make only minor changes to the original example, we make two subroutines; one to read two nibbles from the LCD, and the other to write to nibbles to the LCD. Furthermore, the toggling of the EN-line is also taken to these subroutines, because we have to toggle for each nibble.

```
READ_2_NIBBLES:
 ORL    DATA,#0F0      ;Be sure to release datalines (set outputlatches
                       ;to '1') so we can read the LCD
 SETB   ENMOV A,DATA   ;Read first part of the return value (high nibble)
 CLR    ENANL A,#0F0h  ;Only high nibble is usablePUSH ACC
 SETB   EN
 MOV    A,DATA         ;Read second part of the return value (low nibble)
 CLR    EN
 ANL    A,#0F0h        ;Only high nibble is usable
 SWAP   A              ;Last received is actually low nibble, so put it in place
 MOV    R7,A
 POP    ACC
 ORL    A,R7           ;And combine it with low nibble
 RET
WRITE_2_NIBBLES:
 PUSH   ACC            ;Save A for low nibbleORL DATA,#0F0h ;Bits 4..7 <- 1
 ORL    A,#0F0h        ;Don't affect bits 0-3
 ANL    DATA,A         ;High nibble to display
 SETB   EN
 CLR    EN
 POP    ACC            ;Prepare to send
 SWAP   A              ;...second nibble
 ORL    DATA,#0F0h     ; Bits 4...7 <- 1
 ORL    A,#0F0h        ; Don't affect bits 0...3
 ANL    DATA,A         ;Low nibble to display
 SETB   EN
 CLR    EN
 RET
```

As we see in the WRITE_2_NIBBLES routine, there are some logic instructions (ORL, ANL) so I/O lines P1.0 to P1.3 are not affected. These lines are thus free for input or output and not affected by this routine.

CHECKING THE BUSY STATUS OF THE LCD

```
WAIT_LCD:
  CLR RS ;It's a command
  SETB RW ;It's a read command
  LCALL READ_2_NIBBLES ;Take two nibbles from LCD in A
  JB ACC.7,WAIT_LCD ;If bit 7 high, LCD still busy
  CLR RW ;Turn off RW for future commands
  RET
```

INITIALIZING THE LCD

Before you may really use the LCD, you must initialise and configure it. This is accomplished by sending a number of initialisation instructions to the LCD.

The first instruction we send must tell the LCD we'll be communicating with it with a 4-bit data bus. We also select a 5x8 dot character font. These two options are selected by sending the command 28h to the LCD as a command. After powering up the LCD, it is in 8-bit mode. Because only four bits are connected, the first command has to be send twice; the first time to switch to 4-bits mode, (the lower 4 bits of the command are not seen), the second time to send it as two nibbles so the lower part is received, too.

```
CLR   RS
CLR   RW
CLR   EN
SETB  EN
MOV   DATA,#28h
CLR   EN
LCALL WAIT_LCD
MOV   A,#28h
LCALL WRITE_2_NIBBLES ;Write A as two separate nibbles to LCD
LCALL WAIT_LCD
```

Programming Tip: The LCD command 28h is really the sum of a number of option bits. The instruction itself is the instruction 20h ("Function set"). However, to this we add the values 0h to indicate a 4-bit data bus plus 08h to indicate that the display is a two-line display.

We've now sent the first byte of the initialisation sequence. The second byte of the initialisation sequence is the instruction 0Eh. Thus we must repeat the initialisation code from above, but now with the instruction. Thus the next code segment is:

initialisation code from above, but now with the instruction. Thus the next code segment is:

```
MOV   A,#0Eh
LCALL WRITE_2_NIBBLES ;Write A as two separate nibbles to LCD
LCALL WAIT_LCD
```

Programming Tip: The command 0Eh is really the instruction 08h plus 04h to turn the LCD on. To that an additional 02h is added in order to turn the cursor on.

The last byte we need to send is used to configure additional operational parameters of the LCD. We must send the value 06h.

```
MOV   A,#06h
LCALL WRITE_2_NIBBLES ;Write A as two separate nibbles to LCD
LCALL WAIT_LCD
```

Programming Tip: The command 06h is really the instruction 04h plus 02h to configure the LCD such that every time we send it a character, the cursor position automatically moves to the right.

So, in all, our initialisation code is as follows:

```
INIT_LCD:
 CLR   RS
 CLR   RW
 CLR   EN
 SETB  EN
 MOV   DATA,#28h
 CLR   EN
 LCALL WAIT_LCD
 MOV   A,#28h
 LCALL WRITE_2_NIBBLES
 LCALL WAIT_LCD
 MOV   A,#0Eh
 LCALL WRITE_2_NIBBLES
 LCALL WAIT_LCD
 MOV   A,#06h
 LCALL WRITE_2_NIBBLES
 LCALL WAIT_LCD
 RET
```

Having executed this code the LCD will be fully initialised and ready for us to send display data to it.

CLEARING THE DISPLAY

```
CLEAR_LCD:
 CLR   RS
 MOV   A,#01h
 LCALL WRITE_2_NIBBLES ;Write A as two separate nibbles to LCD
 LCALL WAIT_LCD
 RET
```

WRITING TEXT TO THE LCD

```
WRITE_TEXT:
 SETB  RS
 LCALL WRITE_2_NIBBLES
 LCALL WAIT_LCD
 RET
```

THE "HELLO WORLD" PROGRAM

```
 LCALL INIT_LCD
 LCALL CLEAR_LCD
 MOV A,#'H'
 LCALL WRITE_TEXT
 MOV A,#'E'
 LCALL WRITE_TEXT
 MOV A,#'L'
 LCALL WRITE_TEXT
 MOV A,#'L'
 LCALL WRITE_TEXT
 MOV A,#'O'
 LCALL WRITE_TEXT
 MOV A,#' '
 LCALL WRITE_TEXT
 MOV A,#'W'
 LCALL WRITE_TEXT
 MOV A,#'O'
 LCALL WRITE_TEXT
 MOV A,#'R'
 LCALL WRITE_TEXT
 MOV A,#'L'
 LCALL WRITE_TEXT
 MOV A,#'D'
 LCALL WRITE_TEXT
```

The above "Hello World" program should, when executed, initialise the LCD, clear the LCD screen, and display "Hello World" in the upper left-hand corner of the display. As you see, no difference in this routine with the 8bit drive routines.

CURSOR POSITIONING

Let's again, write the word 'world' on the second line now, from the tenth position:

```
LCALL INIT_LCD
LCALL CLEAR_LCD
MOV   A,#'H'
LCALL WRITE_TEXT
MOV   A,#'E'
LCALL WRITE_TEXT
MOV   A,#'L'
LCALL WRITE_TEXT
MOV   A,#'L'
LCALL WRITE_TEXT
MOV   A,#'O'
LCALL WRITE_TEXT
CLR   RS
MOV   A,#0C9h
LCALL WRITE_2_NIBBLES
LCALL WAIT_LCD
MOV   A,#'W'
LCALL WRITE_TEXT
MOV   A,#'O'
LCALL WRITE_TEXT
MOV   A,#'R'
LCALL WRITE_TEXT
MOV   A,#'L'
LCALL WRITE_TEXT
MOV   A,#'D'
LCALL WRITE_TEXT
```

SUMMARY

This tutorial has presented the underlying concepts of programming an LCD display in 4bit modus. If things in this document are not clear, please be sure to read again the first tutorial on using LCD modules.

MORE BACKGROUND INFORMATION ABOUT DIFFERENT SIZED LCD MODULES

The HD44780 or compatible controller is basically designed to build LCDisplays with one or two lines with a maximum of 40 characterpositions each. A single HD44780 is able to display two lines of 8 characters each. If we want more, the HD44780 has to be expanded with one or more expansion chips, like the HD 44100 (2 x 8 characters expansion) or the HD 66100 (2 x 16 characters expansion). Seen from the HD44780, the first line starts with 00h; the second line with 40h.

LAYOUT OF DISPLAY MODULES *WITHOUT* EXPANSION CHIP(S):

1. **8 characters x 1 line**
   First Line: `00 01 02 03 04 05 06 07`
   The module has to be initialised as a ONE line display.

2. **8 characters x 2 lines**
   First Line: `00 01 02 03 04 05 06 07`
   Second Line: `40 41 42 43 44 45 46 47`
   To use the second line, don’t forget to initialise the display as a TWO lines display.

3. **16 characters x 1 line**
   First Line: `00 01 02 03 04 05 06 07 40 41 42 43 44 45 46 47`
   In fact, in this case the two lines are placed one after another. So when we want to use the display from the ninth position, it has to be initialised as if it were a TWO lines display! Mind the ninth position is addressed as 40h, not 08h.

LAYOUT OF DISPLAY MODULES *WITH* EXPANSION CHIP(S)

1. **16 characters x 1 line**
   First Line: `00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F`
   On the outside this module looks the same like #3 above, but all characterpositions are addressed continuously. The module has to be initialised as a ONE-line display.

2. **20 characters x 1 line**
   First Line: `00 01 02 03 04 05 06 07 08 09 40 41 42 43 44 45 46 47 48 49`
   In fact, from HD44780 point of view, in this case two lines are placed one after another. So when we want to use the display from the eleventh position, it has to be initialised as if it were a TWO lines display! Mind the eleventh position is addressed as 40h, not 0Ah.

3. **40 characters x 1 line**
   First Line: `00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11...27`

The module has to be initialised as a TWO lines display, if we also want to use the second line.

4. **24 characters x 2 lines**

First Line:	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17
Second Line:	40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57

The module has to be initialised as a TWO lines display, if we also want to use the second line.

5. **40 characters x 2 lines**

First Line:	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11... 27
Second Line:	40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51... 67

The module has to be initialised as a TWO lines display, if we also want to use the second line. This is also the maximum configuration which is possible with one HD44780 + extension chips (80 characters).

6. **16 characters x 4 lines**

First Line:	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
Second Line:	40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
Third Line:	10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
Fourth Line:	50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F

To use the second and the fourth line, the module has to be initialised as a TWO lines display (strange, no?). In fact, the third line is continuous to the first line, and the fourth line is continuous two the second line (from addressing point of view).

7. **20 characters x 4 lines**

First Line:	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13
Second Line:	40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53
Third Line:	14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27
Fourth Line:	54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67

To use the second and the fourth line, the module has to be initialised as a TWO lines display. In fact, the third line is continuous to the first line, and the fourth line is continuous two the second line (from addressing point of view).

8. **24 characters x 4 lines**

First Line:	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17
Second Line:	20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37
Third Line:	40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57
Fourth Line:	60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77

To use the THIRD and the FOURTH line, the module has to be initialised as a TWO lines display. Look out! There is a small ’view’-gap between the address ing of the first and the second line (and the third and fourth line respectively).

9. **40 characters x 4 lines**

```
First Line:   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12... 27
Second Line:  40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52... 67
Third Line:   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12... 27
Fourth Line:  40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52... 67
```

These modules uses two HD44780's (each with expansion chips) and can be seen as two 40 x 2 modules in one. All wiring is common, except for the EN (enable) lines, which are separate to drive each HD44780 apart.

> **Tip:**If you are not sure what configuration your display has, you can do use the following trick: Imagine you have a 16 characters x 1 line display. You want to know if it has configuration #3 in the first section or #1 in this section. Just connect the contrast input of your module to GND (= maximum contrast) and apply only power (+5V) to the module. The module will be initialised by an internal routine, which puts it in single line modus. Only the positions that belong to the first line will be black.
>
> **Conclusion:** If all positions are black, we have a module as described under #1 of this section. If only the first 8 characters are black, you have a module as described under # of the previous section.

Not all possible module configurations are described here, but with the help of this information you must be able to work with different modules. Some (functional, not necessary pin-) compatible chips; to help you to determine your module:

Controller:
HD44780 (Hitachi)
KS0066 (Samsung)
SED1278 (Epson)

Expansion 8 x 2:
HD44100 (Hitachi)
KS0061 (Samsung)
M5259 (OKI)

Expansion 12 x 2:
SED1181 (Epson)

Expansion 16 x 2:
HD66100 (Hitachi)

Note: Some modules have black blobs which are chips direct mounted on the pc-board, covered with some resin substantion, so the chips are not recognisable.
