LCD tutorial(Be Learner and Become Trainer)

Contents

Introduction

Many of you have used LCD with Arduino by including “LiquidCrystal.h” header file and lcd.print(), lcd.setCursor() etc. You feel your job is done as you can see the output being printed in the LCD screen as desired. But what if I told you to do the same in other microcontrollers or in Arduino without using the header file, or you suppose instead of setCursor() function I would like to use display_at() as function name or make your own header file that can be used with your desired microcontroller. All you can do with the help of this article.

To be specific let’s say “Character LCD” as used to display text/characters like (A, B, 1, 2, or even characters like heart, emoji) but not graphics (like we see in TV). It is one of the most used display by hobbyists and generally for the purpose of testing outputs.

We have heard 16*2 LCD display in common. Here 16 refers to the number of columns and 2, the number of rows i.e. we can display characters in two lines and each line can contain maximum of 16 characters. Closely observing, we can see the small rectangles similar one as we see when we pressed the calculator screen during our childhood. Each character is made up of 5*8 such small rectangles.Be sure I am telling 5*8 not 5*7 because you may be in confusion when reading further. These LCDS come in many configurations each with between 8 to 80 viewable characters arranged in 1, 2, or 4 rows. i.e. beside 16*2 we have other options also available like 16*1, 16*4, 20*4 etc. We can choose among them according to our requirements. We can have color choice as well like white text on blue background, black text on green and many more.

For general information almost all of the Character LCD’s controller are made by Hitachi or based upon the architecture introduced by Hitachi. The HD44780 type controller chip is used with a wide variety of Liquid Crystal Displays.

Use of LCD over LEDs

Previously, LEDs where common for the output purpose but in recent days LCDs are general for output purpose as people are concerned about their declining price, their ability to display numbers, characters. To display a word, CPU need to refresh the LEDs to display next letter of the word, but LCD can display at once.

16*2 Character LCD Pinout

There are 16 pins.

The pins can be at any positions shown below. (Note last two pins are not shown in below figure).

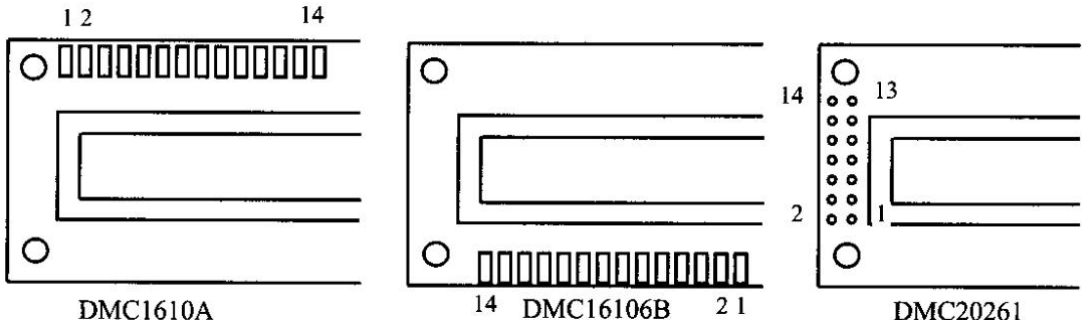


Fig 1

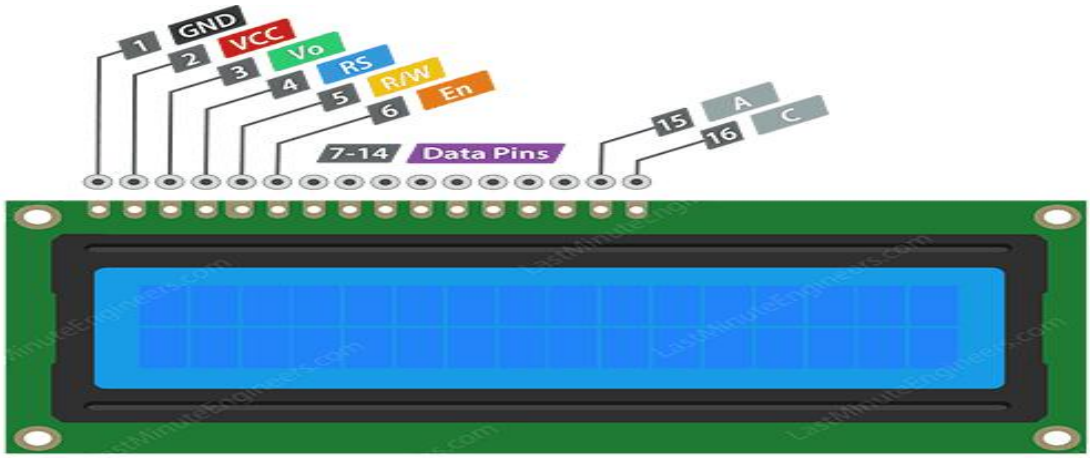


Fig 2

1. GND or V_{SS} - should be connected to the ground.
2. +5v or V_{CC} - should be connected to the power supply or +5V pin of microcontroller.
3. Vo or V_{EE} - controls the contrast and brightness of the LCD. We can use potentiometer which helps for adjustment to the contrast.
4. RS (Register Select) - used to differentiate between the commands like (clear screen, show cursor…) and the data like (A, 1 …). RS pin when low means we are sending command to the LCD and when high means data.

That's for simple understanding. Look at the full form of RS which means it selects one of the two register when set 1 and another register when set 0. LCD has two built in registers namely data register and command register. Data register is for placing the data to be displayed, and the command register is to place the commands. If we make RS pin low and put a data on the data line, the module will recognize it as a command.

5. R/W(Read/Write) - this tells whether we are writing to LCD or reading from LCD. Normally we use LCD to output data rather then read the content of it, so we can connect it to ground as 0 to R/W pin indicating write mode.
6. E(Enable) - used to enable the display. When this pin is set to LOW, the LCD does not care what is happening with RS, R/W and data bus lines; when HIGH, the LCD is processing the incoming data. Its use is made cleared at the coding section.
7. D0
8. D1
9. D2

10. D3 --> D0 to D8 pins are the data pins and carry the 8 bit data we sent to display. Similarly the commands, which are also of 8 bits, are sent through these pins.
11. D4
12. D5
13. D6
14. D7
15. A(Anode) or LED+

--> A-K pins are used for background light of the LCD.

16. C or K(Cathode) or LED-

Our desire in this documentation(Interfacing with microcontroller)

There are lots of things to talk before entering into interfacing but it will be monotonous. So lets dive into the character LCD interfacing with microcontroller. For reference we will consider ATmega32A microcontroller and 16*2 LCD. But don't fell that this documentation is not for me because I'm having different microcontroller or different LCD. This will surely help for others as well. Note that Arduino Uno uses ATmega328p is which is similar to ATmega32a, so it also consists of PORTB, PORTC, PORTD but lacks PORTA. To run the below program with Arduino, replace PORTA with PORTC or PORTD in the definition section.

/*-- For those who have played with LCD, one of the common mistakes when testing an LCD module is to either not draw any characters to the screen or to not connect the contrast pin. In both cases, the screen will be blank and you'll assume that either your protocol is wrong or the display is broken. --*/

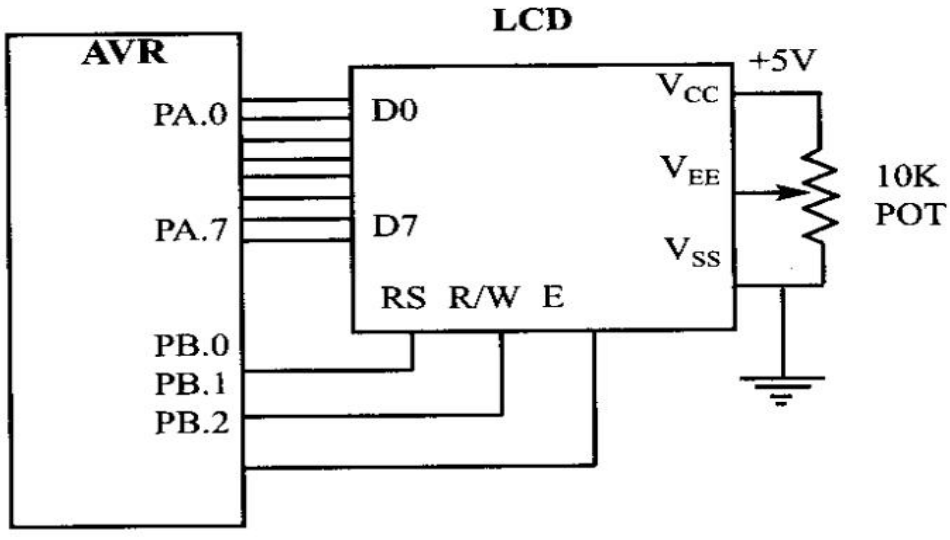


Fig 3

*The connection between ATmega32A and 16*2 LCD are given below:*
PIN1 or VSS - ground
PIN2 or VDD or VCC +5v power
PIN3 or VEE or Vo - ground(gives maximum contrast best for beginner) or as shown in fig use potentiometer to control contrast

PIN4 or RS(Register Select) - PB0
PIN5 or RW (Register Selection) - PB1
Pin6 or E (Enable) - PB2
PIN7 or D0 - PA0
PIN8 or D1 - PA1
PIN9 or D2 - PA2
PIN10 or D3 - PA3
PIN11 or D4 - PA4
PIN12 or D5 - PA5
PIN13 or D6 - PA6
PIN14 or D7 - PA7
(PIN15 or A (Anode) - +5V power -> *for backlight if necessary i.e. making non character displaying portion also light.*
PIN 16 or C or K(Cathode) - Gnd) *May require resistor for protection as well, can be found out from data sheet or use 220ohm if not mentioned, but most LCD comes with protection connected internally.*

We have use 8-bit communication but is not compulsory as 4-bit method also available. If you have not understood this line then don't worry, just follow the next steps and soon you will be familiar with it.

Coding:

```
/*-- C Program for LCD display --*/
/*-- Using 8-bit mode --*/

#include <avr/io.h>
#include <util/delay.h>

/*-- Definition Section --*/
#define LCD_DPRT PORTA     // Data PORT (For pins D0 - D7 of LCD)
#define LCD_CPRT PORTB     // Command PORT (For RS, R/W and E pins of LCD)
#define LCD_DDDR DDRA      // Data DDR
#define LCD_CDDR DDRB      // Command DDR
#define LCD_DPIN PINA  // Data PIN
#define LCD_CPIN PINB  // Command PIN
#define LCD_RS   0   // Connected to PB0
#define LCD_RW         1   // Connected to PB1
#define LCD_EN   2   // Connected to PB2

/*-- User-defined function declaration section --*/
void delay_us(unsigned int d);      // Although predefined function for
      // delay is available we made out own because different compiler
      // have their own delay functions like _delay_us() for WinAVR,
      // delay_ms() for CodeVision and shows error while using diff
      // compiler if used predefined (not necessary).
void lcd_command(unsigned char cmd);
```

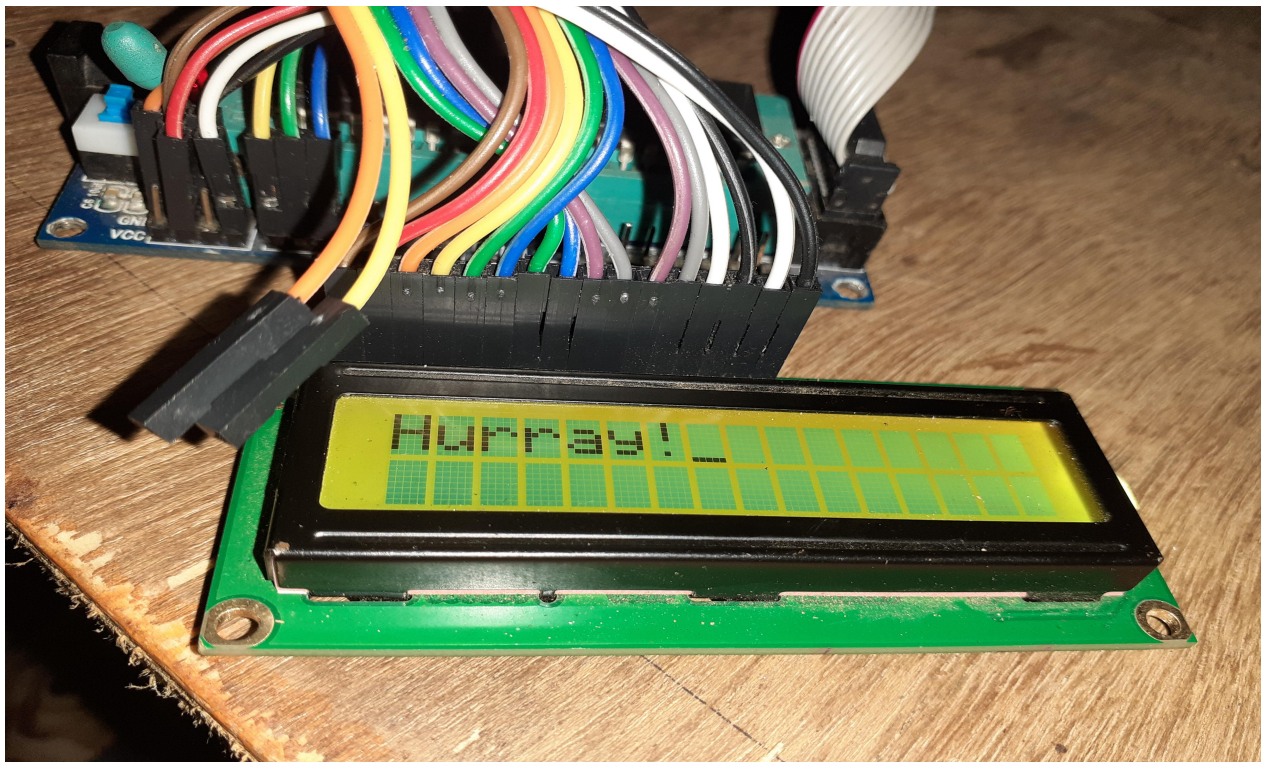
```
void lcd_data(unsigned char data);
void lcd_init();
void lcd_print (unsigned char *str);
void lcd_gotoxy(unsigned char x, unsigned char y);
void custom_char();


/*-- Start of main function --*/
int main (void) {
   unsigned char i;
   lcd_init();
   lcd_gotoxy(1,1);     // Printing starts from column 1, row 1 of LCD
   lcd_print("Hurray!"); // You can pass any string like "Hello World!"
      // which you are used to in programming as there is believe that
      // luck comes in your favour starting with this. But I've used
      // "Hurray!" bc it's your expression when you see that being
      // printed on the LCD.

   while(1)
       // This time I've kept while loop empty but if you want your
      // LCD to change frequently you can write your code inside while
      // loop.
   return 0;
}


/*-- User-defined function definition section --*/
void delay_us(unsigned int delay) {
   while(delay --)
      _delay_us(1);      // Note I'm not passing delay variable
            // inside the _delay_us() function because this function
            // does not take the variable as parameter and should be
            // used with only constant value like 1.
}


void lcd_command(unsigned char cmnd) {
      // Function to send command. Note command is also data of 8 bit
      // sent through D0-D7 pins .
   LCD_DPRT = cmnd;            // Make the command available at PORT
   LCD_CPRT &= ~(1 << LCD_RS);  // RS = 0 for cmd mode
   LCD_CPRT &= ~(1 << LCD_RW); // RW = 0 for write operation
   LCD_CPRT |= (1 << LCD_EN);    // Sending high pulse to EN telling LCD
      // to receive data available at the PORT
   delay_us(1);                // According to datasheet minimum pulse
      // should be 450ns, making it of 1us should work fine for almost
```

```
    // LCDs.
  LCD_CPRT &= ~(1 << LCD_EN);  // Sending low pulse to    EN indicating
    // LCD we are done sending data and LCD process the receive data

  delay_us(50);              // Giving time for LCD to process the
    // data
}


void lcd_data(unsigned char data) {
  LCD_DPRT = data;       // Make the data available at the PORT
  LCD_CPRT |= (1 << LCD_RS);     // RS = 1 for data mode
  LCD_CPRT &= ~(1 << LCD_RW); // RW = 0 for write operation
  LCD_CPRT |= (1 << LCD_EN);     // Sending high pulse to EN
  delay_us(1);               // Increasing the pulse width. (If you
    // are confuse in the terms time period and pulse then remember
    // time period is the sum of on and off time whereas pulse
    // width refers only on time.
  LCD_CPRT &= ~(1 << LCD_EN);  // Sending low pulse to    EN
  _delay_us(50);
}


void lcd_init() {
  LCD_DDDR = 0xFF;           // Set for output mode
  LCD_CDDR = 0x0F;
  LCD_CPRT &= ~(1 << LCD_EN);  // LCD EN = 0
  _delay_us(2000);           // Wait for stable power
    // After power-up you should wait for sometime before sending
    // initialization commands to the LCD. If the LCD initializer
    // function is not the first function in your code    you can
    // omit this delay.
  lcd_command(0x38);         // init. LCD 2 line, 5*7 matrix
  lcd_command(0x0E);         // display on, cursor on (if F instead of E then
    // cursor blinking ON as well
  lcd_command(0x01);         // Clear LCD
  delay_us(2000);          // Clear LCD cmnd requires delay of about
    // 2ms (takes longer time than other commands)
}


void lcd_gotoxy(unsigned char x, unsigned char y) {
  // x and y are the value of row and column respectively which starts
  // from 1. To make it start from 0 replace y-1 and x-1 with y and x.
  unsigned char first_char_adr[] = {0x80, 0xC0, 0x94, 0xD4};
                  // These are the starting addresses of
```

```
                        // the cursor of 4 rows
    lcd_command(first_char_adr[y-1] + x-1);
    delay_us(100);
}


void lcd_print (unsigned char *str) {
    unsigned char i = 0;
    while(str[i] != 0) {
        lcd_data(str[i]);
        i++;
    }
}
```

==========

Output



Output 1

Note if this does not work then try increasing the delay times.

If it's your first time working with LCD then the program is not clear to you. I've included the programming part to avoid the continuous theory portion which I feel mind-numbing too. But don't worry, clarification begins from here. So continue reading.....

I won't go line by line explanation of each explanation in code considering that you won't enter directly into using character LCD without knowing the C programming and playing with the LEDs in that microcontroller.

As observing the pin connections, if not considering the pins connected to either GND or +5v you should find 11 pins connected to GPIO(General Purpose I/0) pins(if unknown about GPIO pins better to have a self study or consult me).
Among 11 pins also RS, R/W and E are command pins whose functions as described earlier. The remaining 8 pins connected to PORTA are for receiving or sending the 8-bit data to or from LCD.

All these 11 pins are considered output as can be seen in lcd_init() function. We have sent some hex data like 0x38, 0x01 etc as commands. What that commands do is given in brief as comment in the code itself. For more detail you will see a table and explanations below for all the commands for LCD. Don't worry about lcd_gotoxy() function as well. You will have knowledge about it soon.

Next we will see the table of some commands frequently used in LCD some of which can be seen in the above code as well. Then we will enter some more depth in the instructions.

LCD Commands

Code (Hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to beginning to 1st line
C0	Force cursor to beginning to 2nd line
38	2 lines and 5x7 matrix

Table 1

It seems as if we need to remember the hex number but not really. To clarify more I've presented you with another table. Note that the table shows the value of the RS as well as R/W pin but it is required only the value of D0-D7 to be passed to lcd_command() and lcd_data() functions.

Note that the table and explanation is for depth information. If you are not interested you can leave that portion. But if you enter into that try shifting display or cursor left or right, 5*7 or 5*10 dots character font, 1 line or two line which I won't teach you. But don't worry about 4 bit mode, it will be included.

/*-- At the beginning I've talked about 5*7 and 5*8 dots character. From now be clear that among 5*8 dots 5*7 are available for displaying characters or data we passed like 'A', '1'. And remaining last row for displaying the cursor.
--*/

List of LCD Instructions

	Code											Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)
Instruction	**RS**	**R/$\overline{W}$**	**DB7**	**DB6**	**DB5**	**DB4**	**DB3**	**DB2**	**DB1**	**DB0**	**Description**	
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

Table 2

Bit names

Bit	*Settings*	
I/D	0 = Decrement cursor position	1 = Increment cursor position
S	0 = No display shift	1 = Display shift
D	0 = Display off	1 = Display on
C	0 = Cursor off	1 = Cursor on
B	0 = Cursor blink off	1 = Cursor blink on
S/C	0 = Move cursor	1 = Shift display
R/L	0 = Shift left	1 = Shift right
DL	0 = 4-bit interface	1 = 8-bit interface
N	0 = 1/8 or 1/11 Duty (1 line)	1 = 1/16 Duty (2 lines)
F	0 = 5x7 dots	1 = 5x10 dots
BF	0 = Can accept instruction	1 = Internal operation in progress

Table 3

Clear Display

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	0	1

Clears all display and returns the cursor to the home position (Address 0).

Details

Writes space code (20h) into all DD RAM addresses. Sets address counter to DDRAM location 0.

What does this mean?

The Display Data RAM (DDRAM) is a RAM that stores the ASCII code for the characters that we send to the LCD module. The DDRAM can store up to 80 characters. However, only some of these 80 characters are displayed on the LCD. For example, in the case of a 16*2 LCD, only 32 of these memory locations are displayed. The relationships between the displayed DDRAM addresses and the LCD positions are shown in Figure

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

If we write a particular character to the DDRAM address 0x00, it will be displayed in the first cell of the upper line. If in 0x40, appear in the first cell of lower line. To go to a particular address of the DDRAM, we can write the desired address to the Address Counter(AC). Moreover, the AC determines the position on the LCD that a character entered by a write operation goes to.

Returns display to its original state if it was shifted. In other words, the display disappears and the cursor or blink goes to the left edge of the display (the first line if 2 lines are displayed). Sets entry mode I/D to 1 (Increment Mode). Entry mode shift (S) bit remains unchanged.

May be you are not clear with the table. It is explained below in LCD Addressing.

Cursor Home

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	1	*

Returns the cursor to the home position (Address 0). Returns display to its original state if it was shifted.

Details

Sets the address counter to DD RAM location 0 in the address counter. Returns display to its original state if it was shifted. DD RAM contents remain unchanged. The cursor or blink goes to the left edge of the display (the first line if 2 lines are displayed).

Entry Mode Set

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1	I/D	S

Sets the effect of subsequent DD RAM read or write operations. Sets the cursor move direction and specifies or not to shift the display. These operations are performed during data read and write.

Details

Specifies whether to increment (I/D = 1) or decrement (I/D = 0) the address counter after subsequent DD RAM read or write operations.

If S = 1 the display will be shifted to the left (if I/D = 1) or right (if I/D = 0) on subsequent DD RAM write operations. This makes it looks as if the cursor stands still and the display moves when each character is written to the DD RAM. if S = 0 the display will not shift on subsequent DD RAM write operations.

Display ON/OFF

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	1	D	C	B

Controls display of characters and cursor.

Details

D: The display is ON when D = 1 and OFF when D = 0. The DD RAM contents remain unchanged.

C: The cursor is displayed when C = 1 and is not displayed when C = 0.

The cursor is displayed as 5 dots in the 8th line when the 5 x 7 dot character font is selected and as 5 dots in the 11th line when the 5 x 10 dot character font is selected.

B: The character at the the cursor position blinks when B = 1.

Blinking is performed by switching between all blank dots and the display character every 409.6 ms.

Cursor and Display Shift

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	1	S/C	R/L	*	*

Moves the cursor and shifts the display without changing DD RAM contents.

Details

Shifts cursor position or display to the right or left without writing or reading display data. This function is used to correct or search for the display. In a 2-line display, the cursor moves to the 2nd line when it passes the 40th digit of the 1st line. Notice that the 1st and 2nd line displays will shift at the same time. When the displayed data is shifted repeatedly each line only moves horizontally. The 2nd line of the display does not shift into the 1st line position.

S/C R/L
---
0 0 Shifts the cursor position to the left
(Address Counter is decremented by 1)
0 1 Shifts the cursor position to the right
(Address Counter is incremented by 1)
1 0 Shifts the entire display to the left The cursor follows the display shift
1 1 Shifts the entire display to the right The cursor follows the display shift

Function Set

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	DL	N	F	*	*

Sets interface data length (DL), number of display lines (N) and character font (F)

Details

This command should be issued only after automatic power-on initialization has occurred, or as part of the module initialization sequence.

DL: Sets interface data length

Data sent or received in 8 bit lengths (DB7-DB0) when DL = 1

Data sent or received in 4 bit lengths (DB7-DB4) when DL = 0

When the 4 bit length is selected, data must be sent or received in pairs of 4-bits each. The most-significant 4 bits are sent or received first.

N: Sets number of display lines

F: Sets character font

N	F	lines	Font	Remarks
0	0	1	5x 7 dots	-
0	1	1	5x10 dots	-
1	*	2	5x 7 dots	Cannot display 2 lines with 5x10 dot character font

Note that a 1 line x 16 character display is treated as a 2 line x 8 character display for some of the display(It will be discussed in LCD addressing).

Set CG RAM Address

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	AC	AC	AC	AC	AC	AC

Sets the CG RAM address. Subsequent read or write operations refer to the CG RAM.

Details

Sets the specified value (ACACACACACAC) into the address counter. Subsequent read or write operations transfer data from, or to, the character generator RAM.

Set DDRAM Address

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	AC	AC	AC	AC	AC	AC	AC

Sets the DD RAM address. Subsequent read or writes refer to the DD RAM.

Details

Sets the specified value (ACACACACACACAC) into the address counter. Subsequent read or write operations transfer data from, or to, the display RAM. Note: Adjacent display RAM locations do not necessarily refer to adjacent display positions.

Read busy flag and address counter

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	1	BF	AC	AC	AC	AC	AC	AC	AC

Reads the state of the busy flag (BF) and the contents of the address counter.

Details

Reads the busy flag (BF) that indicates the state of the LCD module. BF = 1 indicates that the module is busy processing the previous command. BF = 0 indicates that the module is ready to perform another command.

The value of the address counter is also returned. The same address counter is used for both CG and DD RAM transfers.

This command can be issued at any time. It is the only command which the LCD module will accept while a previous command is still being processed.

Write data to CG or DD RAM

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	0	D	D	D	D	D	D	D	D

Writes data into DD RAM or CG RAM

Details

Writes a byte (DDDDDDDD) to the CG or the DD RAM. The destination (CG RAM or DD RAM) is determined by the most recent `Set RAM Address' command. The location to which the byte will be written is the current value of the address counter. After the byte is written the address counter is automatically incremented or decremented according to the entry mode. The entry mode also determines whether or not the display will shift.

Read data from CG or DD RAM

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	1	D	D	D	D	D	D	D	D

Reads data from DD RAM or CG RAM.

Details

Reads a byte (DDDDDDDD) from the CG or DD RAM. The source (CG RAM or DD RAM) is determined by the most recent `Set RAM Address' command. The location from which the byte will be read is the current value of the address counter. After the byte is ready the address counter is automatically incremented or decremented according to the entry mode.

LCD Addressing

We are known of DDRAM. The DDRAM in the HD44780 controller if of 80 bytes.

The most memory map of it is shown below:

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

This shows that DDRAM consists of two lines of with some gap between two lines. Note that this gap is because DDRAM addressing uses seven bit addressing and the highest bit signifies which row of memory is involved.

0 in MSB indicates first row

0	1	0	0	1	1	1

6-leftmost bits indicating 27H

1 in MSB indicates second row

1	1	0	0	1	1	1

6-rightmost bits indicating 67H

Here 0b0101000 i.e. 28H is not in place of 40H because the MSB of 28H is 0 which indicates the first row although we are in second row due to which it is 40H.

Each of these DDAM memory addresses corresponds to a character position in the display.

Now you may be in confusion that in 16*2 LCD we can display only 32 characters but we can have 80 characters to display. How are the remaining characters displayed or like that. You will see this how. For now we consider 40*2 LCD first and slowly move on.

40*2 LCD

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

Here we can display all the 80 characters and looks as above. To display the characters, we need to first specify the starting DDRAM address and then start sending string. When first character is sent after specifying the address, the character is stored in that address and next character of the string is stored to the next address if we have set the address counter(AC) to increment mode in “Entry mode set” , I/D = 1. If we reach to the address 27H then the next character of the string will be automatically incremented and put to 40H address. Similarly it increments from address 67H back to 00h.

20*4 LCD

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

First line starts with 00H address, second with 40H address, third with 14H and the last with 54H. Note the memory map is always the same regardless of the display configuration. So if we continue writing in the first line then remaining portion appears at the third line instead of second. To display to

the second line after the first line we have to manually set the address counter location to 40H after 13H as there is no way of telling the LCD controller that there are now 4 rows of 20 characters instead of 2 rows of 40 characters.

20*2 LCD

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53

Although this display has only 40 characters there are still 80 bytes of DDRAM and they are still configured the same as they were before. If characters written to the addresses not shown here, they do not appear at the screen but the characters are preserved at that location and can be displayed by shifting the display. (Just for information)

16*2 LCD

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

Most commonly found LCD. Its operation is similar to that of 20*2 LCD besides it can display only 32 characters.

LCDs support shift operations that can change the relationships that were shown in above figure. For example, a left shift applied to the default status of above figure will lead to the figure as shown below.

Display position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DDRAM address	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50

Similarly for right shift:

Display position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	27	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
DDRAM address	67	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

16*1 LCD

It can be found in two different variations and the initialization and operation of each is different.

/*-- Remember that when you power the multi-word LCD modules, before the controller is initialized you will see the character locations corresponding to the first line of memory are dark and the others are light (proper contrast adjustment required). */

If you apply power to a 16*1 LCD module you will see only the left 8 character or right 8 character locations are dark or you will see the whole 16 character locations dark. If only 8 character locations either left or right are dark, it is of one type and if all 16, it is of another type.

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F

If all 16 locations dark.

00	01	02	03	04	05	06	07

40	41	42	43	44	45	46	47

If only 8 locations dark.

Here although the display has only one row but as far as the LCD controller is concerned it is using two lines of memory and it must be considered to be a 2 line device when initializing the controller.

16*4 LCD

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

10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F

50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F

40*4 LCD

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27

40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27

40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67

It is the combination of two 40*2 devices stacked one on top of another in the same glass enclouser. It has two separate memory maps each with the same address range. The memories are accessed individually by strobing the desired Enable pin.

lcd_gotoxy() Function Explanation

This function is used to point the DDRAM address to which the characters are to be stored and displayed. There you can see the array of the commands 80H, CO, 94H and D4H. The 80H is the command for the AC to point to 00H address of the DDRAM. But why 80H?

` Set DDRAM Address

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	AC	AC	AC	AC	AC	AC	AC

If you look at the “Set DDRAM Address” command you can see DB0-DB6 contains the value of AC i.e. the address of the DDRAM to which AC should point to. To point to the starting character location of LCD the DDRAM address is 00H. If we substitute the value of AC and consider only DB0-DB7 we get 0b10000000 i.e. 80H, similarly for next DDRAM address i.e. 01H we get 81H and so on. For others you can check yourself. Note that this function does not works correctly for 16*4 LCD because the starting address of third row for this LCD is not C0H but 90H and that of fourth row is C0H. These hex number is passed to lcd_command() function and passing character causes to store that character at that location of DDRAM. This function is designed such that to represent first row and first column of the LCD need to pass (1,1) as parameter consider column first.

LCD initialization

from web,alfredstate.edu

This is based on the LCD modules controlled by HD44780 or equivalent.

Initialization by Internal Reset Circuit

According to the data sheet:

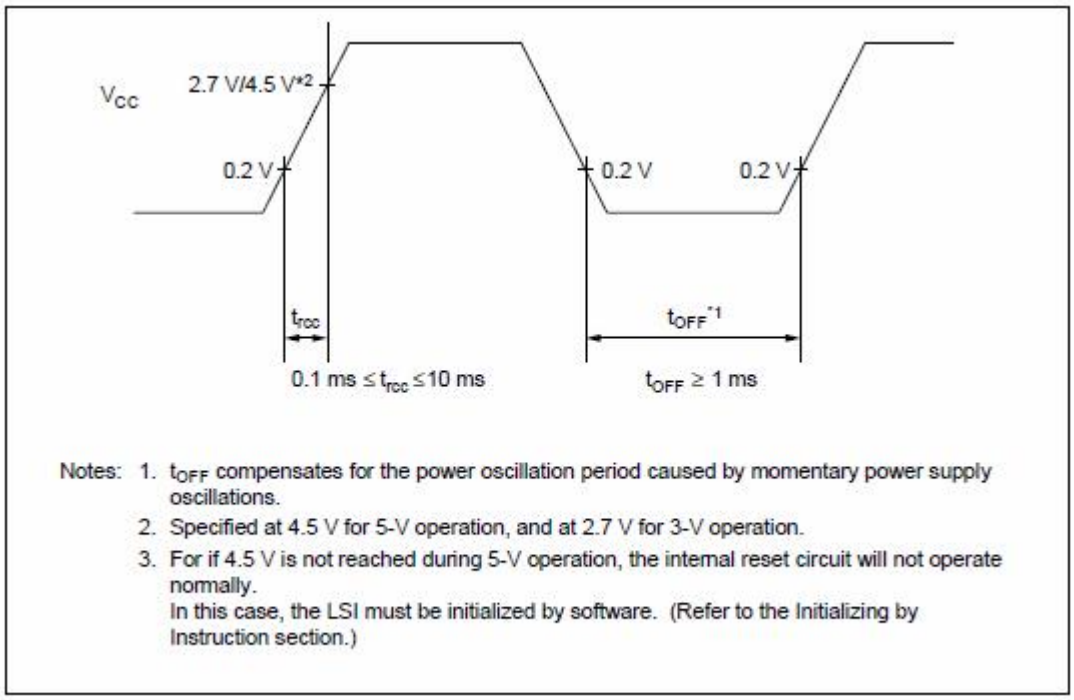
Initializing by Internal Reset Circuit

An internal reset circuit automatically initializes the HD44780U when the power is turned on. The following instructions are executed during the initialization. The busy flag (BF) is kept in the busy state until the initialization ends (BF = 1). The busy state lasts for 10 ms after V_{CC} rises to 4.5 V.

1. Display clear
2. Function set:
   DL = 1; 8-bit interface data
   N = 0; 1-line display
   F = 0; 5 × 8 dot character font
3. Display on/off control:
   D = 0; Display off
   C = 0; Cursor off
   B = 0; Blinking off
4. Entry mode set:
   I/D = 1; Increment by 1
   S = 0; No shift

Note: If the electrical characteristics conditions listed under the table Power Supply Conditions Using Internal Reset Circuit are not met, the internal reset circuit will not operate normally and will fail to initialize the HD44780U. For such a case, initial-ization must be performed by the MPU as explained in the section, Initializing by Instruction.

Figure 4



Notes: 1. t_{OFF} compensates for the power oscillation period caused by momentary power supply oscillations.
2. Specified at 4.5 V for 5-V operation, and at 2.7 V for 3-V operation.
3. For if 4.5 V is not reached during 5-V operation, the internal reset circuit will not operate normally.
In this case, the LSI must be initialized by software. (Refer to the Initializing by Instruction section.)

Internal Power Supply Reset

Figure 5

Relying on this internal reset may be satisfactory for an LCD module that is part of a system that also includes the power supply, such as the display on a printer. This method of initialization is not recommended. For more information visit data sheet.

Initialization by Instruction

There are separate initialization flowcharts for the 8-bit interface and the 4-bit interface, but the actual sequence of instructions sent to the LCD controller is essentially the same in each case. First there are a series of what are technically **Function Set** instructions whose purpose is to effectively 'reset' the LCD controller. Next, if the 4-bit interface is desired, there is an additional **Function Set** instruction to change the interface from the default 8-bit configuration. Finally there are four more instructions, the 'real' **Function Set**, the **Display on/off Control**, the **Clear Display**, and the **Entry Mode Set**.

When power is applied to the LCD module the LCD controller always comes up in the 8-bit interface mode. This means that the LCD controller reads all eight of its data pins each time the Enable pin is pulsed. This is fine if an 8-bit data interface is actually being used, but what about the other possibility, where a 4-bit data interface is connected? In this second case there may be indeterminate data on the lower four bits, especially if those pins have not been grounded as recommended. The answer is that the controller has been set up to ignore those lower four bits throughout the early part of the initialization process, until the actual interface has been established by **Function Set.**

It is important to make sure that the LCD controller has finished executing an instruction before sending it another one, otherwise the second instruction will be ignored. The data sheets give specific times for the delays during the beginning.

8-Bit Interface, Initialization by Instruction

The flowchart according to Hitachi data sheet is as shown:

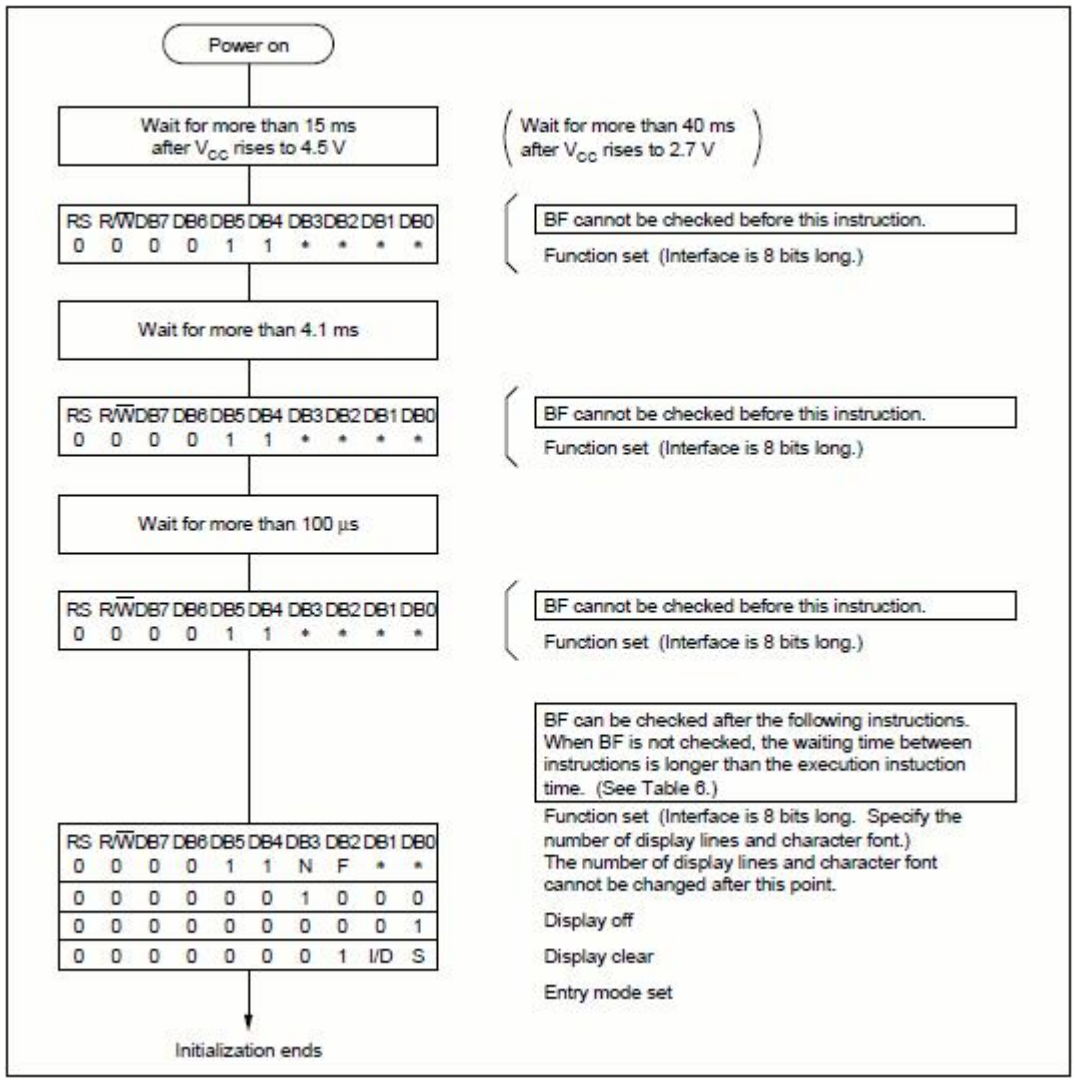


Figure 6

Character Mode Liquid Crystal Display Module
Initialization by Instruction (8-bit data interface)

Notes:
RS = 0 to select the Instruction register.
R/W = 0 so that data is written to the LCD module.

The second 100 µs time delay is not documented, this figure is speculation, it may be possible to check the busy flag here.

N and F must be set in the first non-special Function Set instruction and cannot be changed subsequently

All time delays specified after the Function Set are based on worst case instruction execution time (clock may be as low as 190 kHz).

The first Display ON/OFF Control instruction should probably be performed as specified (some programmers set D, C, and B here).

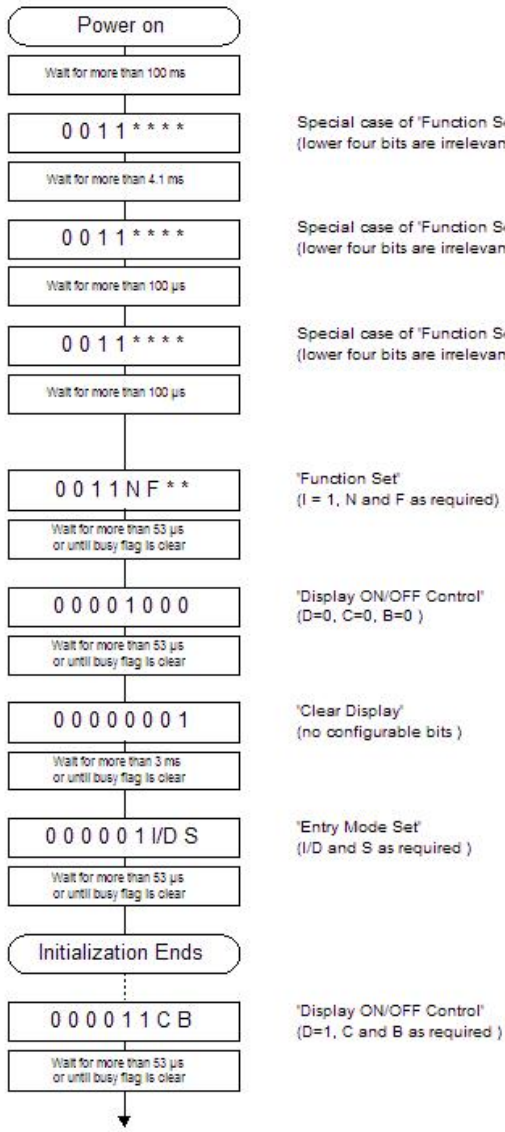


Figure 7

You may be in confusion that the flowchart shows different way of initialization than the coding shown above for 8-bit mode. This is done knowingly. Some of you may have read some other articles and they follow this sort of way of initialization. To make you feel comfortable with the LCD I also followed the same way. Again why against the steps given in data sheet. As you are aware that initially the LCD is in 8-bit mode and we are using 8-bit mode as well. So it can accepts the 8-bit command from initial after LCD receives stable power. Avoiding the initial delays of the initialization process the operation is little bit fast. But it's better you follow the initialization sequence. If you see inside the header files also you can this initialization sequence being followed.

4-Bit Interface, Initialization by Instruction

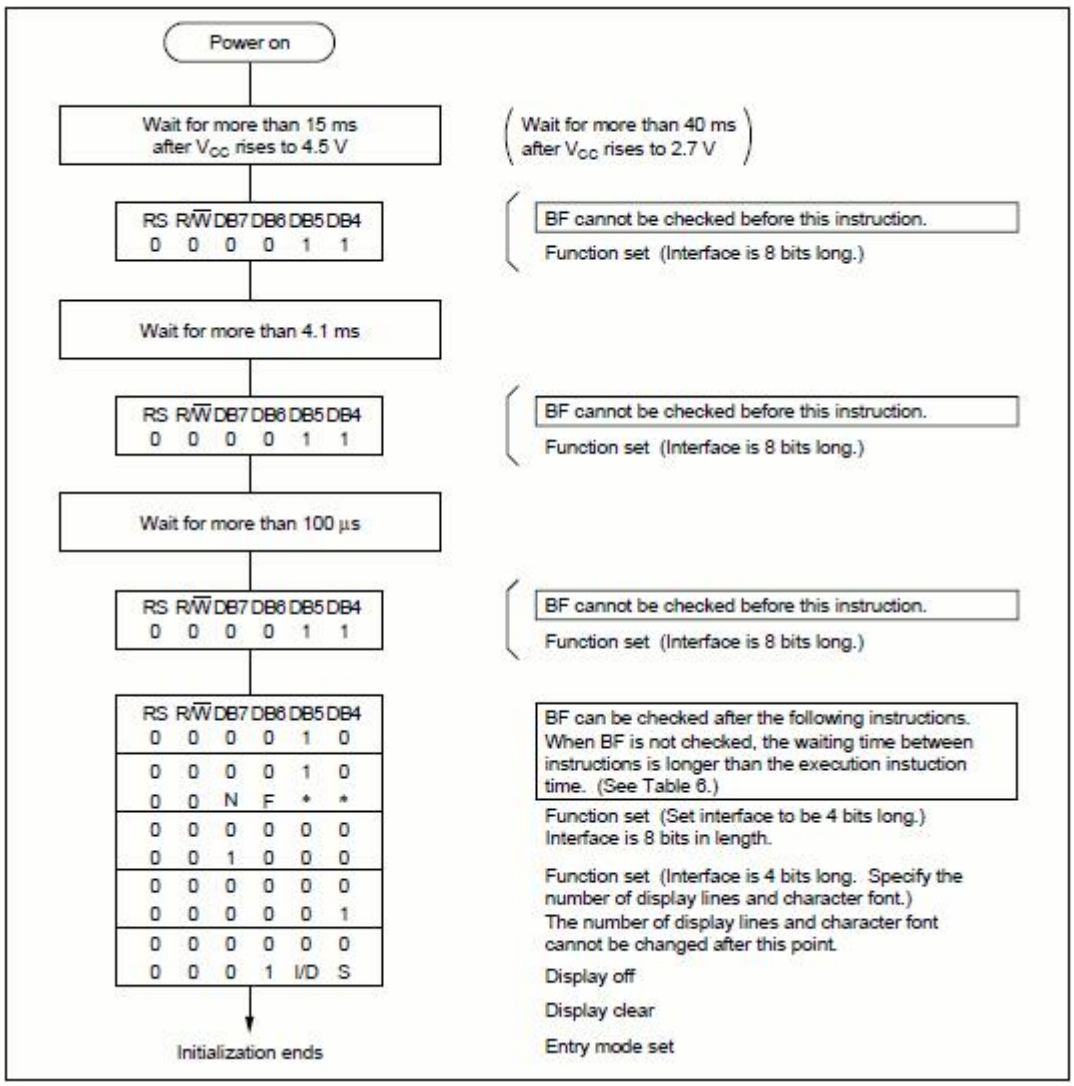


Figure 8

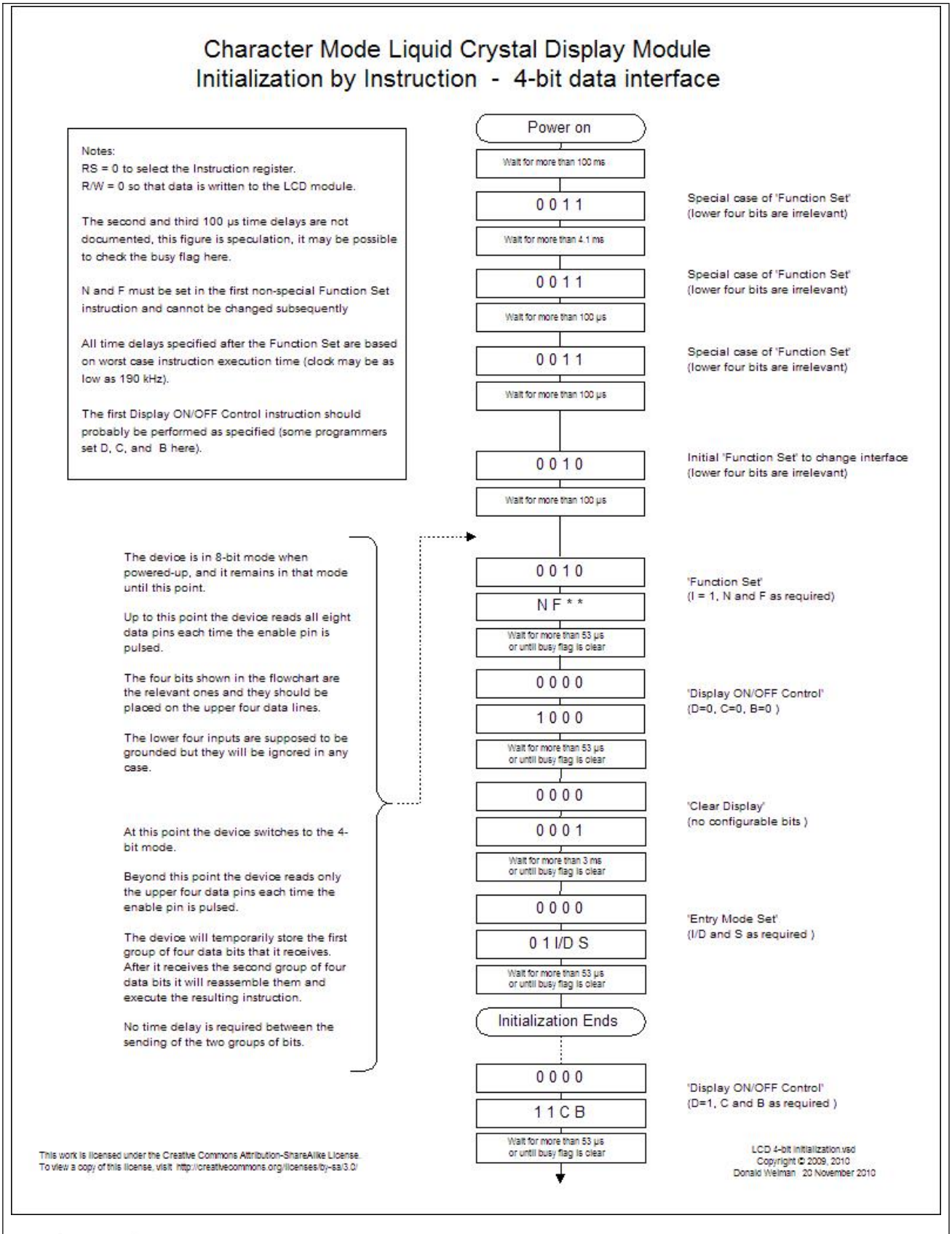


Figure 9

The first initial steps are identical because the LCD controller starts off in the 8-bit mode regardless of how many data lines are actually being used. Since this is an initialization sequence for the 4-bit mode there are only four data lines connected between the microcontroller and the LCD

module, however the LCD controller is currently in the 8-bit mode and it expects data on all eight data pins. Fortunately, for this special case of the **Function Set** instruction, the lower four bits are irrelevant so the fact that they are not connected to the microcontroller is also irrelevant. There is one extra step here. The 8-bit data is to be sent 4-bit at a time, higher nibble first and then lower nibble of the data by left shifting 4 times.

After sending **instruction 0010b(2H)**, then **delay > 100 us,** the LCD controller is expecting the 'real' **Function Set** instruction which, in the 8-bit mode, would start with 0011. Instead, it gets a **Function Set** instruction starting with 0010. This is it's signal to again ignore the lower four bits, switch to the four-bit mode, and expect another 'real' **Function Set** instruction.

The LCD controller is now in the 4-bit mode. This means that the LCD controller reads only the four high order data pins each time the Enable pin is pulsed. To accommodate this, the host microcontroller must put the four high bits on the data lines and pulse the Enable pin. There is no need for a delay between these two sequence because the LCD controller isn't processing the instruction yet. After the second group of data bits is received the LCD controller reconstructs and executes the instruction and this is when the delay is required.

Instruction 0010b(2H), then 1000b(8H), then delay >53 us or check BF

This is the real **Function Set** instruction. This is where the interface, the number of lines, and the font are specified. Since we are implementing the 4-bit interface we make D=0. The no of lines being specified here is the no of lines used of the DDRAM which has maximum of 2 but not of the actual display. This is 2 in most cases except for some type of 16*1 LCDs so N=1. There are very few displays capable of displaying a 5*10 font so the 5*7 choice is almost always correct and we set F=0.

Instruction 0000b(0h), then 1000b(8h) then delay > 53 us or check BF

This is the **Display on/off Control instruction.** This instruction is used to control several aspects of the display but now is not the time to set the display or cursor or blinking as shown in flowchart.

Instruction 0000b(0h), then 0001b(1h) then delay > 3 ms or check BF

This is the **Clear Display** instruction which, since it has to write information to all 80 DDRAM addresses, takes more time to execute than most of the other instructions.

Instruction 0000b(0h), then 0110b(6h), then delay > 53 us or check BF

This is the **Entry Mode Set** instruction. This instruction determines which way the cursor and/or the display moves when we enter a string of characters. We normally want the cursor to increment(move from left to right) and the display to not shift so we set I/D=1 and S=0. If your application requires a different configuration you could change this instruction alone and just add another **Entry Mode Set** instruction where appropriate in your program.

Initialization ends

This is the end of the actual initialization sequence, but note that the display is off according to the above instruction.

Instruction 0000b(0h), then 1100b(0Ch), then delay > 53 us or check BF

This is another **Display on/off Control** instruction where the display is turned on, cursor made visible and/or the cursor location blink. This example shows the display on and the other two options off, D=1, C=0, and B=0.

Note that if you have seen the data sheet you may have seen the execution time of 37us with a clock frequency of 270 KHz. But if 190 KHz, which increases delay time by 40%. The delay time considered here is for worst case and should work fine with all.

Interfacing LCD Module with AVR in 4-Bit Mode

Although 8-bit mode is faster but most of the time we require 4-bit mode as it requires less no of pins of microcontroller and the saved 4 bits can be used with other I/O devices.

Connection:

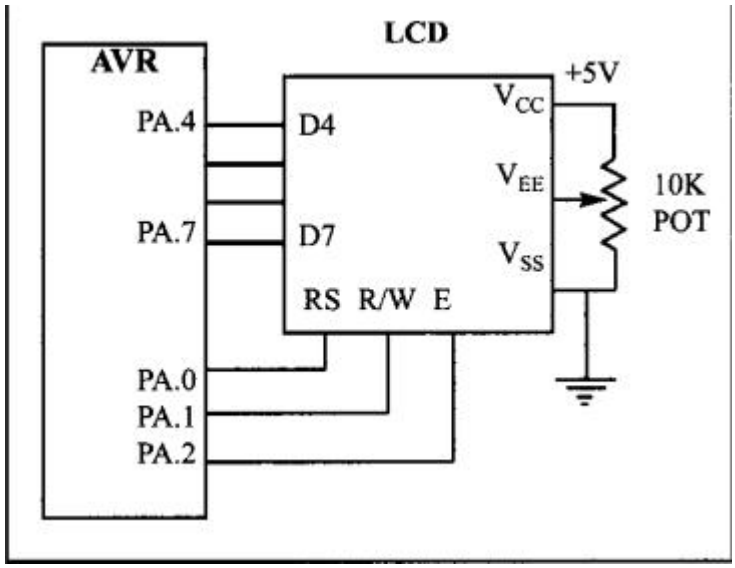


Figure 10

No more explanation for connection believing you are clear from the figure and with the help of above explanations.

Coding(Method One: Software Time Delay):

```
/*-- Description: Using LCD in 4-bit mode using single port only.
 *     This uses time delay instead of Busy Flag(BF).
 *     Note that the delay required is given in data sheet and delay used here is
 *     considering the worst condition like using clock frequency 190 KHz
 --*/
/*-- Date: 19 April 2020 --*/


#include <avr/io.h>
#include <util/delay.h>
#define F_CPU 1000000UL // CPU frequency set to internal 1MHz


/*-- Definition Section --*/
#define LCD_DDR DDRA     // Both command and data DDR (D4-D7 pins of LCD to PORTA.4-PORTA7)
#define LCD_PRT PORTA    // Both command and data PORT
#define LCD_RS   0     // Connected to PA0
#define LCD_RW   1     // Connected to PA1
#define LCD_EN   2     // Connected to PA2


/*-- LCD Instructions --*/
```

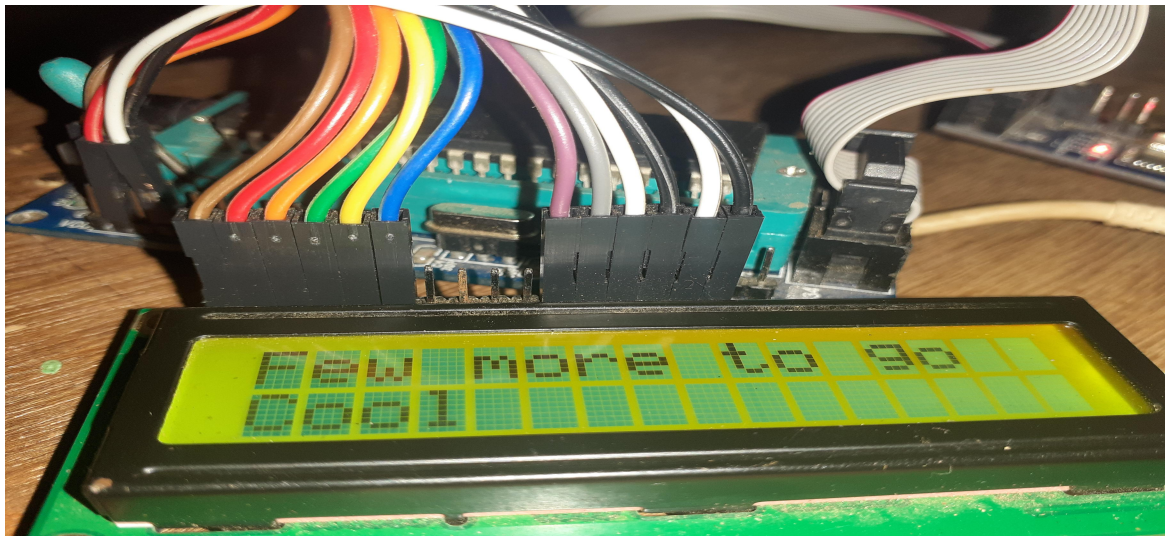
```
#define LCD_CLEAR          0b00000001    // Replace all characteres with ASCII 'space'
#define LCD_HOME           0b00000010    // Return cursor to first position on first line
#define LCD_ENTRY_MODE             0b00000110    // Shift cursor from left to right on read/write
#define LCD_DISPLAY_OFF      0b00001000    // Turn display off
#define LCD_DISPLAY_ON       0b00001100    // Display on, cursor off, don't blink character
#define LCD_FUNCTION_RESET 0b00110000  // Reset the LCD
#define LCD_FUNCTION_4BIT    0b00101000  // 4-bit data, 2-line display, 5*7 font


/*-- User-defined Functions Declaration Section --*/
void lcd_command(unsigned char cmnd);
void lcd_data(unsigned char data);
void lcd_gotoxy(unsigned char x, unsigned char y);
void lcd_init();
void lcd_print(char *str);
void lcd_write_4(unsigned char byte_value);
void delay_ms(unsigned int delay);   // Why made my own delay function?
      // For this see the comment of the code for 8-Bit mode.
void delay_us(unsigned int delay);


/*-- Main Function Section --*/
int main (void) {
      lcd_init();
      while(1) {
            lcd_gotoxy(1,1);
            lcd_print("Wow");
            lcd_gotoxy(5,2);
            lcd_print("Level up");
            delay_ms(2000);
            lcd_gotoxy(1,1);
            lcd_print("Few more to go");
            delay_ms(2000);
            // When you observe LCD display after sometime you only see change in "Few"
            // and "Wow". It is because only the first three location of DDRAM is changed
            // and other remains same. You can use LCD_CLEAR command to see your desired
            // result.
      }
      return 0;
}


/*-- Function Definition Section --*/
void lcd_command(unsigned char cmnd) {
      LCD_PRT &= ~(1 << LCD_RS);      // RS = 0 for command
      LCD_PRT &= ~(1 << LCD_RW);          // RW = 0 for write mode
      LCD_PRT &= ~(1 << LCD_EN);      // Make sure E is initially low
      lcd_write_4(cmnd);              // Write the upper 4 bits of command
      lcd_write_4(cmnd << 4);             // Write the lower 4 bits of command
}
```

```
void lcd_data(unsigned char data) {
	LCD_PRT |= (1 << LCD_RS);
	LCD_PRT &= ~(1 << LCD_RW);
	lcd_write_4(data);			// Write the upper 4 bits of data
	lcd_write_4(data << 4);			// Write the lower 4 bits of data
}

void lcd_write_4(unsigned char byte_value) {
	// In 4-bit mode we have to sent the 4 bit of data only from initial although the LCD
	// is initially at 8-bit mode. If we use lcd_command function initially then it will
	// breaks the 8-bit into two 4-bit and send to the LCD. But	the reset
	// function's lower four bits are irrelevant
	LCD_PRT &= 0x0F;// Data in each data pin of LCD be 0
	LCD_PRT |= (byte_value & 0xf0);			// Set data pins according to cmnd's
		// upper nibble
// write the data
	LCD_PRT |= (1<<LCD_EN);
	delay_us(1);				// Implement 'Data set-up time' (80
		// ns) and 'Enable pulse'
	LCD_PRT &= ~(1<<LCD_EN);
	delay_us(1);				// Implement 'Data hold time' (10 ns)
		// and 'Enable cycle'
}

void lcd_init() {
	delay_ms(100);				// Wait for stable power
	LCD_DDR = 0xFF;				// LCD port as output
	LCD_PRT &= ~(1 << LCD_EN);			// Make sue E is initially low
	lcd_write_4(LCD_FUNCTION_RESET);		// First part of reset sequence
	delay_ms(10);				// 4.1 ms delay(min);
	lcd_write_4(LCD_FUNCTION_RESET);		// Second part of reset sequence
	delay_us(200);				// 100us delay(min)
	lcd_write_4(LCD_FUNCTION_RESET);		// Third part of reset sequence
	delay_us(200);				// This delay is omitted in the data sheet
	lcd_write_4(LCD_FUNCTION_4BIT);		// Set 4-bit mode
	delay_us(80);				// 40 us delay (min)

	lcd_command(LCD_FUNCTION_4BIT);			// Set mode, lines, and font
	delay_us(80);				// 40 us delay (min)
// The next three instruction are specified in the data sheet as part of initialization routine,
// so it is a good idea (but not necessary) to do them just as specified and then redo them later
// if the application requires	a different configuration.
	lcd_command(LCD_DISPLAY_OFF);		// Turn display OFF
	delay_us(80);
	lcd_command(LCD_CLEAR);			// Clear Display Data RAM
	delay_us(80);
	lcd_command(LCD_ENTRY_MODE);		// Set the desired shift characteristics
	delay_us(80);
	lcd_command(LCD_DISPLAY_ON);		// Turn the display on
	delay_us(80);
}
```

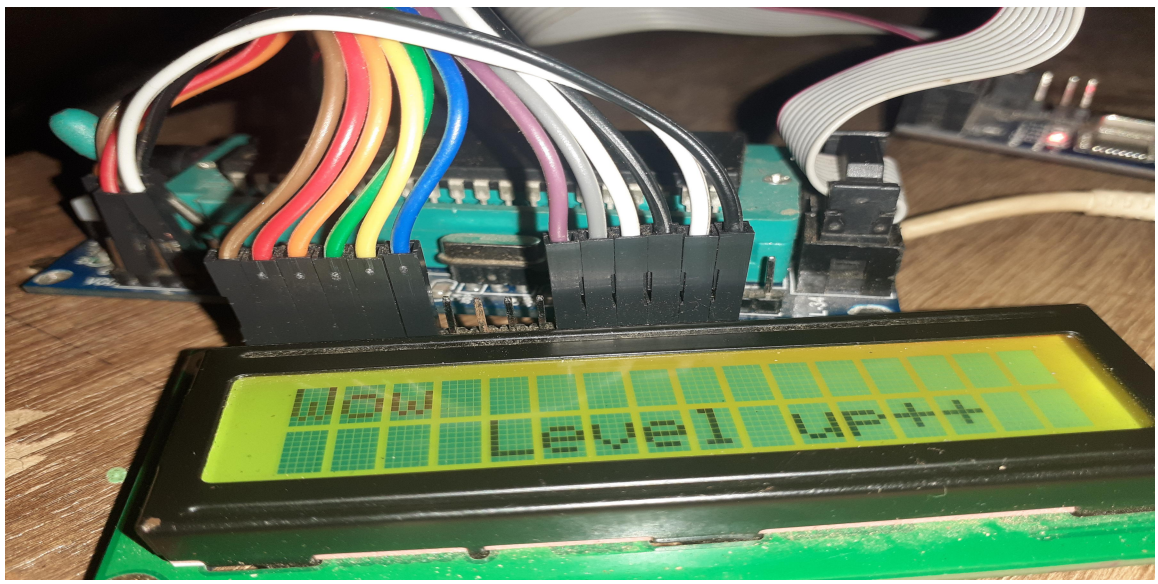
```
void lcd_gotoxy(unsigned char x, unsigned char y) {      // First column then row
      unsigned char first_char_adr[] = {0x80, 0xC0, 0x94, 0xD4};
            // These are the starting address of each row in LCD as can be
            // seen from data sheet
      lcd_command(first_char_adr[y-1] + x-1);          // To move the cursor to
            // the defined location the address of that location should be
            // send as command for eg. the command for 2nd column 1st row is
            // 0x81.
      delay_us(100);
}

void lcd_print(char *str) {
      unsigned char i = 0;
      while(str[i] != 0) {
            // 0 is the ACII value for null character
            lcd_data(str[i]);
            i++;
            delay_us(80);                    // 40 us delay (min)
      }
}

void delay_ms(unsigned int delay) {
      //_delay_ms(d);
      //*-- This line gives us error because we cannot use the delay function
      //       with the variable value as given in the user manual --* //
      while(delay --) {
            _delay_ms(1);
      }
}

void delay_us(unsigned int delay) {
      while(delay --)
            _delay_us(1);
}
```

==========

Output



Output 2



Output 3

Coding(Method Two: Busy Flag):

```
/*-- Description: Using LCD in 4-bit mode using single port only.
 *     This uses Busy Flag(BF) and is faster method as it uses execution time only.
 *     Before we have put extreme delays.
 *     --*/
/*-- Date: 19 April 2020 --*/


#include <avr/io.h>
#include <util/delay.h>
#define F_CPU 1000000UL // CPU frequency set to internal 1MHz


/*-- Definition Section --*/
#define LCD_DDR DDRA     // Both command and data DDR (D4-D7 pins of LCD to PORTA.4-PORTA7)
#define LCD_PRT PORTA    // Both command and data PORT
#define LCD_PIN  PINA      // Both command and data PIN
#define LCD_RS   0     // Connected to PA0
#define LCD_RW   1     // Connected to PA1
#define LCD_EN   2     // Connected to PA2


/*-- LCD Instructions --*/
#define LCD_CLEAR          0b00000001    // Replace all characteres with ASCII 'space'
#define LCD_HOME           0b00000010    // Return cursor to first position on first line
#define LCD_ENTRY_MODE             0b00000110    // Shift cursor from left to right on read/write
#define LCD_DISPLAY_OFF      0b00001000    // Turn display off
#define LCD_DISPLAY_ON       0b00001100    // Display on, cursor off, don't blink character
#define LCD_FUNCTION_RESET 0b00110000  // Reset the LCD
#define LCD_FUNCTION_4BIT    0b00101000  // 4-bit data, 2-line display, 5*7 font


/*-- User-defined Functions Declaration Section --*/
void lcd_command(unsigned char cmnd);
void lcd_data(unsigned char data);
void lcd_gotoxy(unsigned char x, unsigned char y);
void lcd_init(void);
void lcd_print(char *str);
void lcd_write_4(unsigned char byte_value);
void lcd_check_BF_4(void);
void delay_ms(unsigned int delay);   // Why made my own delay function?
      // For this see the comment of the code for 8-Bit mode.
void delay_us(unsigned int delay);


/*-- Main Function Section --*/
int main (void) {
      lcd_init();
      while(1) {
```

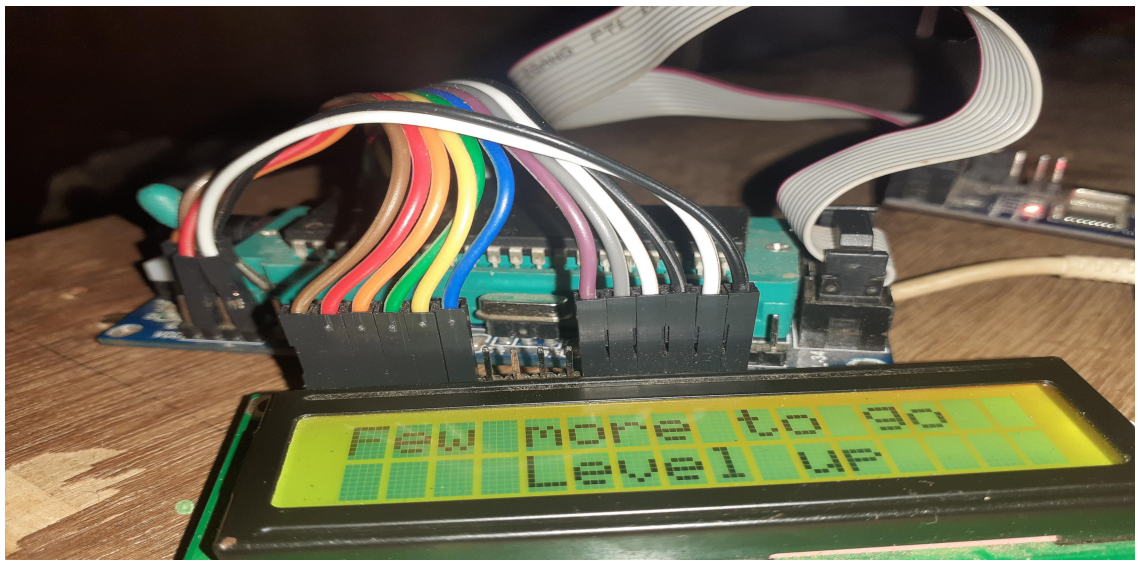
```
            lcd_check_BF_4();
            lcd_gotoxy(1,1);
            lcd_print("Wow");
            lcd_gotoxy(5,2);
            lcd_print("Level up++");
            delay_ms(1000);          // To hold the screen so that we can see the output.
            lcd_check_BF_4();  // This line is no necessary here because we've already
                  // used delay of 2 sec before it.
            lcd_command(LCD_CLEAR);
            lcd_check_BF_4();
            lcd_gotoxy(1,1);
            lcd_print("Few more to go");
            lcd_check_BF_4();
            lcd_gotoxy(1,2);
            lcd_check_BF_4();
            lcd_print("Cool");
            delay_ms(1000);          // To keep the output within out persistence of vision.
            lcd_command(LCD_CLEAR);
            // Solution of the previous problem of changing only "Few" and "Wow"
      }
      return 0;
}


/*-- Function Definition Section --*/
void lcd_command(unsigned char cmnd) {
      LCD_PRT &= ~(1 << LCD_RS);      // RS = 0 for command
      LCD_PRT &= ~(1 << LCD_RW);          // RW = 0 for write mode
      LCD_PRT &= ~(1 << LCD_EN);     // Make sure E is initially low
      lcd_write_4(cmnd);              // Write the upper 4 bits of command
      lcd_write_4(cmnd << 4);             // Write the lower 4 bits of command
}

void lcd_data(unsigned char data) {
      LCD_PRT |= (1 << LCD_RS);
      LCD_PRT &= ~(1 << LCD_RW);
      lcd_write_4(data);                  // Write the upper 4 bits of data
      lcd_write_4(data << 4);                   // Write the lower 4 bits of data
}

void lcd_write_4(unsigned char byte_value) {
      // In 4-bit mode we have to sent the 4 bit of data only from initial although the LCD
      // is initially at 8-bit mode. If we use lcd_command function initially then it will
      // breaks the 8-bit into two 4-bit and send to the LCD. But    the reset
      // function's lower four bits are irrelevant
      LCD_PRT &= 0x0F;// Data in each data pin of LCD be 0
      LCD_PRT |= (byte_value & 0xf0);                   // Set data pins according to cmnd's
            // upper nibble
// write the data
      LCD_PRT |= (1<<LCD_EN);
      delay_us(1);                        // Implement 'Data set-up time' (80
```

```
        // ns) and 'Enable pulse'
    LCD_PRT &= ~(1<<LCD_EN);
    delay_us(1);                    // Implement 'Data hold time' (10 ns)
        // and 'Enable cycle'
}

void lcd_init() {
    delay_ms(100);                  // Wait for stable power
    LCD_DDR = 0xFF;                 // LCD port as output
    LCD_PRT &= ~(1 << LCD_EN);          // Make sue E is initially low
    lcd_write_4(LCD_FUNCTION_RESET);        // First part of reset sequence
    delay_ms(10);                   // 4.1 ms delay(min);
    lcd_write_4(LCD_FUNCTION_RESET);        // Second part of reset sequence
    delay_us(200);                  // 100us delay(min)
    lcd_write_4(LCD_FUNCTION_RESET);        // Third part of reset sequence
    delay_us(200);                  // This delay is omitted in the data sheet
    lcd_write_4(LCD_FUNCTION_4BIT);         // Set 4-bit mode
    delay_us(80);                   // 40 us delay (min)

    lcd_check_BF_4();               // Make sure LCD controller is ready
    lcd_command(LCD_FUNCTION_4BIT);             // Set mode, lines, and font
    //delay_us(80);
    //------- From this point on the busy flag is available ----------//
// The next three instruction are specified in the data sheet as part of initialization routine,
// so it is a good idea (but not necessary) to do them just as specified and then redo them later
// if the application requires    a different configuration.
    lcd_check_BF_4();               // Make sure LCD controller is ready
    lcd_command(LCD_DISPLAY_OFF);           // Turn display OFF
    lcd_check_BF_4();               // Make sure LCD controller is ready
    lcd_command(LCD_CLEAR);                 // Clear Display Data RAM
    lcd_check_BF_4();               // Make sure LCD controller is ready
    lcd_command(LCD_ENTRY_MODE);            // Set the desired shift characteristics
    lcd_check_BF_4();               // Make sure LCD controller is ready
    lcd_command(LCD_DISPLAY_ON);            // Turn the display on
}

void lcd_gotoxy(unsigned char x, unsigned char y) {     // First column then row
    unsigned char first_char_adr[] = {0x80, 0xC0, 0x94, 0xD4};
        // These are the starting address of each row in LCD as can be
        // seen from data sheet

    lcd_check_BF_4();               // Make sure LCD controller is ready
    lcd_command(first_char_adr[y-1] + x-1);         // To move the cursor to
        // the defined location the address of that location should be
        // send as command for eg. the command for 2nd column 1st row is
        // 0x81.

}

void lcd_print(char *str) {
    unsigned char i = 0;
```

```
	while(str[i] != 0) {
		// 0 is the ACII value for null character
		lcd_check_BF_4();			// Make sure LCD controller is ready
		lcd_data(str[i]);
		i++;
	}
}

void lcd_check_BF_4(void) {
	uint8_t busy_flag;			// Busy flag 'mirror'
	LCD_DDR &= ~(1<<7);			// Set PA7 to input
	LCD_PRT &= ~(1<<LCD_RS);
	LCD_PRT |= (1<<LCD_RW);			// Read from LCD module

	do {
		busy_flag = 0;			// Initialize busy flag 'mirror'
		LCD_PRT |= (1<<LCD_EN);
		delay_us(1);			// Implement 'Delay data time' (160 ns)
			// and 'Enable pulse width' (230 ns)
		busy_flag |= (LCD_PIN & (1<<7));	// Consider only 7th bit and mask remaining
		LCD_PRT &= ~(1<<LCD_EN);
		delay_us(1);			// Implement 'Address hold time (10 ns),
			// 'Data hold time' (10 ns), and 'Enable cycle time' (500 ns)

	// Read and discard alternate nibbles (D3 information)
		LCD_PRT |= (1<<LCD_EN);
		delay_us(1);			// Implement 'Delay data time' (160 ns)
			// and 'Enable pulse width' (230 ns)
		LCD_PRT &= ~(1<<LCD_EN);
		delay_us(1);			// Implement 'Address hold time (10 ns),
			// 'Data hold time' (10 ns), and 'Enable cycle time' (500 ns)
	} while(busy_flag);

	LCD_PRT &= ~(1<<LCD_RW);			// Write to LCD module
	LCD_DDR |= (1<<7);			// Reset D7 data direction to output
}

void delay_ms(unsigned int delay) {
	//_delay_ms(d);
	//*-- This line gives ++us error because we cannot use the delay function
	//	with the variable value as given in the user manual --* //
	while(delay --) {
		_delay_ms(1);
	}
}

void delay_us(unsigned int delay) {
	while(delay --)
		_delay_us(1);
}
```

Output

Output 4



Output 5

Custom character

Most of the creative minds may fell sorry to see just alphabets, numbers or some symbols in the widely used LCD i.e. ASCII characters. Not much but well it's time to show your some inbuilt talent.

As you are known now, a LCD display is made up of blocks, which are fundamentally 5 dots in a row and 8 dots in a column, which are lighted up according to the code or predefined characters inside the LCD memory, there are 32 such blocks in a 16*2 LCD display.To display the character 'S' or '4', some of the dots among them are lighted and it appears as that character for us. What character we see depends on which dots are lighted.

For letters, numbers and some characters, LCD knows which dots to be lighted. This is because the data is stored inside its memory. As the topic of memory has entered you should know the three types of memories present inside the HD44780 LCD controller IC.

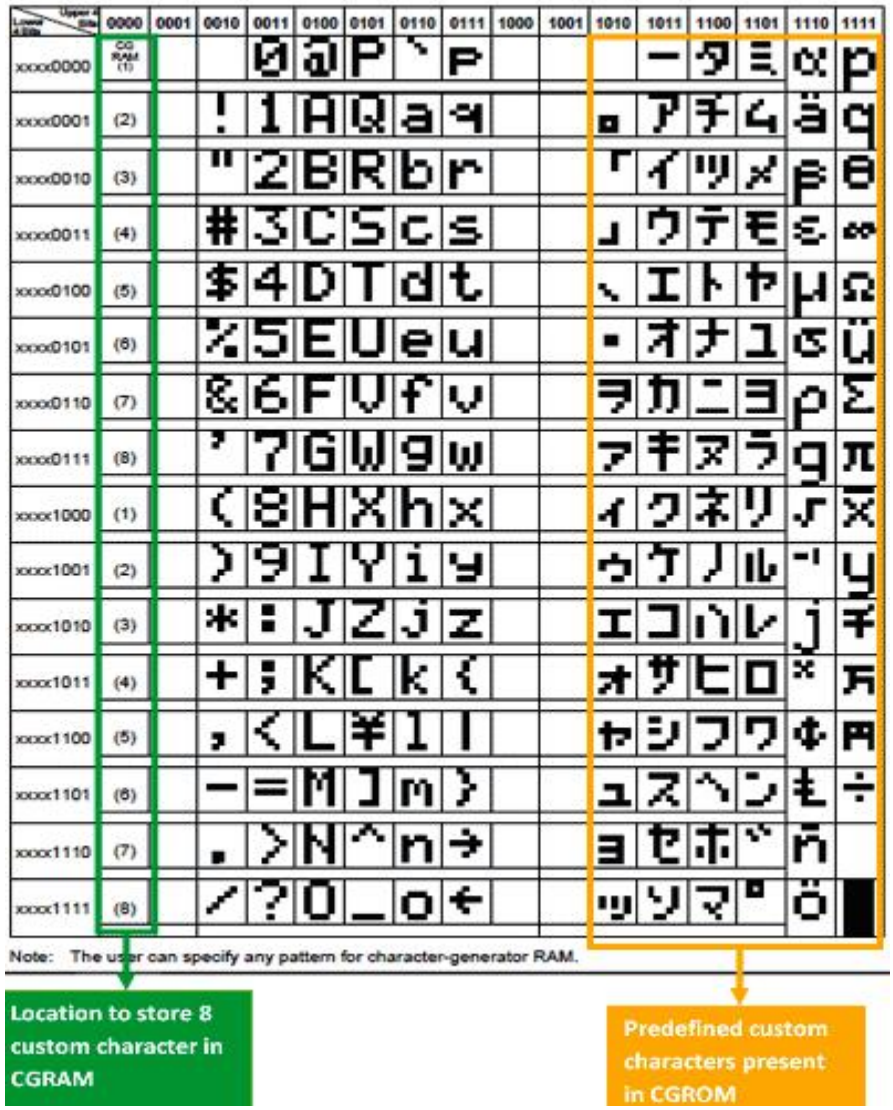
Character Generator ROM (CGROM): It is the read only memory which contains all the patterns of the characters pre-defined inside it. This ROM will vary from each type of Interface IC, and some might have some pre-defined custom character with them.

Display Data RAM (DDRAM): This is a random access memory. Each time you display a character its pattern will be fetched from the CGROM and transferred to the DDRAM and then will be placed on the screen. DDRAM has the patterns of all characters that are currently being displayed on the LCD Screen. This way for each cycle the IC need not fetch data from CGROM, and helps in getting a short update frequency.

Character Generator RAM (CGRAM): This is also a Random access memory, so you can write and read data from it. This memory is used to generate custom character. You have to form a pattern for the character and write it in the CGRAM, this pattern can be read and displayed on the screen when required.

Correspondence between Character Codes and Character Patterns(ROM Code:A00)

Table 4 Correspondence between Character Codes and Character Patterns (ROM Code: A00)



Note: The user can specify any pattern for character-generator RAM.

Table 4

Correspondence between Character Codes and Character Patterns(ROM Code:A02)

Upper 4 Bits / Lower 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)															
xxxx0001	(2)															
xxxx0010	(3)															
xxxx0011	(4)															
xxxx0100	(5)															
xxxx0101	(6)															
xxxx0110	(7)															
xxxx0111	(8)															
xxxx1000	(1)															
xxxx1001	(2)															
xxxx1010	(3)															
xxxx1011	(4)															
xxxx1100	(5)															
xxxx1101	(6)															
xxxx1110	(7)															
xxxx1111	(8)															

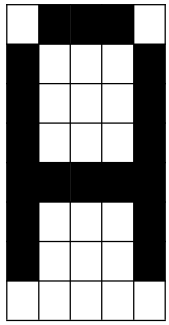
Table 5

If you see the data sheet you will find both the table. But note that your CGROM contains only one of them.

When you write lcd_data(0x41), character ‘A; is stored in the specified DDRAM address and displayed at specified position of LCD. From the table you can see the character code for the pattern ‘A’ is 0x41, which is its ASCII value as well. From the same table now you may know

that lcd_data(0x00) should display custom character. But which custom character? Let's slowly be clear about it.

CGRAM has 64 byte space. In 5*8 font we need 8 bytes to form a character. So we can have maximum of 8 custom characters. But only 4 if chosen 5*10.



Figure

The character 'A' has following binary patterns:

0	1	1	1	0
1	0	0	0	1
1	0	0	0	1
1	0	0	0	1
1	1	1	1	1
1	0	0	0	1
1	0	0	0	1
Cursor P.				

Figure

In hex: 0x0E
0x11
0x11
0x11
0x1F
0x11
0x11
0x00(If no cursor)

Out of 8 bits only lower 5 bits are present and remaining upper 3 bits are considered 0. This 8-byte data of character 'A' is stored already in CGROM. Now you may be thinking that each dot is like individual LED and you can control each individual dots and. Passing 1 to that dot position turns ON that dot and 0 turns OFF. So passing 1 or 0 to individual dots can form different characters. This could be like playing with with LEGO.

Like 8-byte data of 'A' being present in CGROM, you should store the 8-byte data of your custom character in CGRAM. The CGRAM address starts from 0x40 like of DDRAM 0x80. Each character is of 8 bytes so starting address of the 8 custom characters are 0x40, 0x48, 0x50, 0x58, 0x60, 0x68, 0x70 and 0x78. Note that the CGROM initially contains some custom character i.e. characters except than ACII characters.

You may think how to store those 8 bytes in CGRAM. Trust me you have already done this. What?? When?? If this is your expression then after reading further your expression changes to "Oh! why don't I fancy of that!".

Just pass the starting address of the CGRAM as command and then 8 bytes sequentially as data like as shown below for 'A":

```
lcd_command(0x40);
lcd_data(0x0E);
lcd_data(0x11);
.
.
.
.
lcd_data(0x00);
```

Like this you have already done to store characters in DDRAM or to display character in LCD screen. The only difference here is that these data being stored in CGRAM are not displayed at screen. Note that after each data write operation the CGRAM address increments(normally but decrements also possible) by 1. To display the custom character at the screen first you need to change the address counter to point to DDRAM and then pass 0, 1, 2, ... 7 as data like 0x41 as data for 'A".

```
lcd_command(0x80);  // Or lcd_gotoxy(1, 1);
lcd_data(0);                    // To print the custom character being stored at CGRAM location
                                // 0x40, similarly 1 for 0x48 ..... 7 for the address 0x68 as can      //
                            be seen in table 4 and 5.
```

Some might get confuse that instead of lcd_command(0x80), we've used lcd_gotoxy(1,1), but if you look closer into that function we are doing same.

Note passing 8 as data also displays the custom character at the address 0x40, 9 displays the custom character at the address 0x48 ..... which can be seen in table 4 and 5 as well.

Coding:

```
/*-- Description: This is a simple example of custom character display which
  * shows an IC connected to LED and a button and animation of LED being ON when
  * button pressed and OFF when released is tried to be shown although it doesn't
  * looks like that.
  *    4-bit, delay using busy flag is used here.
  *     --*/
/*-- Date: 21 April 2020 --*/


#include <avr/io.h>
#include <util/delay.h>
#define F_CPU 1000000UL      // CPU frequency set to internal 1MHz


/*-- Definition Section --*/
```

```
#define LCD_DDR DDRA   // Both command and data DDR (D4-D7 pins of LCD to
PORTA.4-PORTA7)
#define LCD_PRT PORTA  // Both command and data PORT
#define LCD_PIN PINA      // Both command and data PIN
#define LCD_RS    0    // Connected to PA0
#define LCD_RW    1   // Connected to PA1
#define LCD_EN    2   // Connected to PA2


/*-- LCD Instructions --*/
#define LCD_CLEAR        0b00000001   // Replace all characteres with ASCII 'space'
#define LCD_HOME         0b00000010   // Return cursor to first position on first line
#define LCD_ENTRY_MODE          0b00000110   // Shift cursor from left to right on read/write
#define LCD_DISPLAY_OFF         0b00001000   // Turn display off
#define LCD_DISPLAY_ON          0b00001100   // Display on, cursor off, don't blink character
#define LCD_FUNCTION_RESET 0b00110000     // Reset the LCD
#define LCD_FUNCTION_4BIT    0b00101000     // 4-bit data, 2-line display, 5*7 font
#define CGRAM_ADDRESS           0b01000000   // Starting address of CGRAM
#define DDRAM_ADDRESS           0b10000000   // Starting address of DDRAM


/*-- Global Variable Declaration Section --*/
unsigned char ic_first[8] = {0x1F, 0x10, 0x1F, 0x09, 0x09, 0x09, 0x09, 0x09};
unsigned char ic_middle[8] = {0x1F, 0x00, 0x1F, 0x09, 0x09, 0x09, 0x00, 0x00};
unsigned char ic_last[8] = {0x1F, 0x01, 0x1F, 0x09, 0x09, 0x09, 0x00, 0x00};
unsigned char LED[8] = {0x0C, 0x0A, 0x19, 0x09, 0x19, 0x0A, 0x0C, 0x00};
unsigned char bright[8] = {0x04, 0x08, 0x10, 0x1E, 0x10, 0x08, 0x04, 0x00};
unsigned char button_on[8] = {0x00, 0x04, 0x0E, 0x1F, 0x11, 0x1F, 0x11, 0x00};
unsigned char button_off[8] = {0x04, 0x04, 0x0A, 0x1F, 0x11, 0x1F, 0x11, 0x00};
unsigned char led_connection[8] = {0x09, 0x09, 0x09, 0x08, 0x0F, 0x00, 0x00, 0x00};

/*-- User-defined Functions Declaration Section --*/
void lcd_command(unsigned char cmnd);
void lcd_data(unsigned char data);
void lcd_gotoxy(unsigned char x, unsigned char y);
void lcd_init(void);
void lcd_print(char *str);
void lcd_write_4(unsigned char byte_value);
void lcd_check_BF_4(void);
void delay_ms(unsigned int delay);// Why made my own delay function?
      // For this see the comment of the code for 8-Bit mode.
void delay_us(unsigned int delay);
void store_single_char(unsigned char loc, unsigned char *binary_data);    // To
      // store single custom character in CGRAM.
void built_char(void);        // This function stores all the custom characters
      // in CGRAM

/*-- Main Function Section --*/
```

```
int main (void) {
	unsigned char i = 0, col = 11;		// These variables are made to
			// change the position of the custom characters
			// relatively changing just this value.
	unsigned	col_copy = col;		// To save the col value as it its.
	lcd_init();
	built_char();

	lcd_gotoxy(1,1);
	lcd_print("Blink LED");
	lcd_gotoxy(1,2);
	lcd_print("using IC");

	lcd_gotoxy(col,1);

	for(; i < 7; i++) {
		if(i >= 3)
			lcd_gotoxy(col_copy++,2);
		lcd_data(i);
	}
	lcd_gotoxy(col+2,1);
	lcd_data(1);
	lcd_data(1);
	lcd_data(1);
	lcd_data(2);

	while(1) {
		lcd_gotoxy(col+2,2);
		lcd_data(' ');
		lcd_data(7);
		delay_ms(1000);

		lcd_gotoxy(col+2,2);
		lcd_data(5);
		lcd_data(6);
		delay_ms(1000);
	}
	return 0;
}


/*-- Function Definition Section --*/
void store_single_char(unsigned char loc, unsigned char *binary_data) {
	unsigned char j;
	if(loc<8) {
		lcd_check_BF_4();
		lcd_command(CGRAM_ADDRESS + (loc*8)); // Each character for 5*8 font is
			// of 8 bytes and starts from location 0x40
```

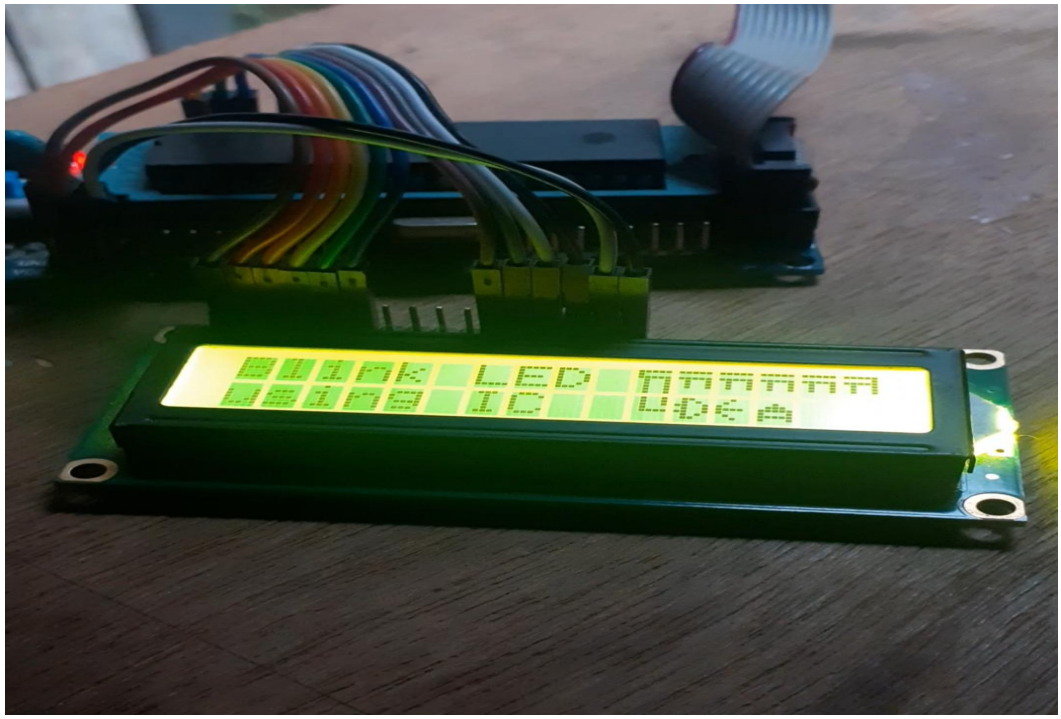
```
            for(j = 0; j < 8; j++) {
                  lcd_check_BF_4();
                  lcd_data(binary_data[j]);   // Store each byte of a character
                        // in the CGRAM
            }
      }
}

void built_char(void) {
      store_single_char(0, ic_first);
      store_single_char(1, ic_middle);
      store_single_char(2, ic_last);
      store_single_char(3, led_connection);
      store_single_char(4, LED);
      store_single_char(5, bright);
      store_single_char(6, button_on);
      store_single_char(7, button_off);
}

void lcd_command(unsigned char cmnd) {
      LCD_PRT &= ~(1 << LCD_RS);   // RS = 0 for command
      LCD_PRT &= ~(1 << LCD_RW);          // RW = 0 for write mode
      LCD_PRT &= ~(1 << LCD_EN);   // Make sure E is initially low
      lcd_write_4(cmnd);           // Write the upper 4 bits of command
      lcd_write_4(cmnd << 4);          // Write the lower 4 bits of command
}

void lcd_data(unsigned char data) {
      LCD_PRT |= (1 << LCD_RS);
      LCD_PRT &= ~(1 << LCD_RW);
      lcd_write_4(data);                        // Write the upper 4 bits of data
      lcd_write_4(data << 4);                         // Write the lower 4 bits of data
}

void lcd_write_4(unsigned char byte_value) {
      // In 4-bit mode we have to sent the 4 bit of data only from initial although the LCD
      // is initially at 8-bit mode. If we use lcd_command function initially then it will
      // breaks the 8-bit into two 4-bit and send to the LCD. But    the reset
      // function's lower four bits are irrelevant
      LCD_PRT &= 0x0F;// Data in each data pin of LCD be 0
      LCD_PRT |= (byte_value & 0xf0);                 // Set data pins according to cmnd's
            // upper nibble
// write the data
      LCD_PRT |= (1<<LCD_EN);
      delay_us(1);                        // Implement 'Data set-up time' (80
            // ns) and 'Enable pulse'
      LCD_PRT &= ~(1<<LCD_EN);
      delay_us(1);                        // Implement 'Data hold time' (10 ns)
```

```
        // and 'Enable cycle'
}

void lcd_init() {
    delay_ms(100);                  // Wait for stable power
    LCD_DDR = 0xFF;                 // LCD port as output
    LCD_PRT &= ~(1 << LCD_EN);          // Make sue E is initially low
    lcd_write_4(LCD_FUNCTION_RESET);        // First part of reset sequence
    delay_ms(10);                   // 4.1 ms delay(min);
    lcd_write_4(LCD_FUNCTION_RESET);        // Second part of reset sequence
    delay_us(200);                  // 100us delay(min)
    lcd_write_4(LCD_FUNCTION_RESET);        // Third part of reset sequence
    delay_us(200);                  // This delay is omitted in the data sheet
    lcd_write_4(LCD_FUNCTION_4BIT);         // Set 4-bit mode
    delay_us(80);                   // 40 us delay (min)

    lcd_check_BF_4();               // Make sure LCD controller is ready
    lcd_command(LCD_FUNCTION_4BIT);         // Set mode, lines, and font
    //delay_us(80);
    //------- From this point on the busy flag is available ----------//
// The next three instruction are specified in the data sheet as part of initialization routine,
// so it is a good idea (but not necessary) to do them just as specified and then redo them later
// if the application requires    a different configuration.
    lcd_check_BF_4();               // Make sure LCD controller is ready
    lcd_command(LCD_DISPLAY_OFF);           // Turn display OFF
    lcd_check_BF_4();               // Make sure LCD controller is ready
    lcd_command(LCD_CLEAR);                 // Clear Display Data RAM
    lcd_check_BF_4();               // Make sure LCD controller is ready
    lcd_command(LCD_ENTRY_MODE);            // Set the desired shift characteristics
    lcd_check_BF_4();               // Make sure LCD controller is ready
    lcd_command(LCD_DISPLAY_ON);            // Turn the display on
}

void lcd_gotoxy(unsigned char x, unsigned char y) { // First column then row
    unsigned char first_char_adr[] = {0x80, 0xC0, 0x94, 0xD4};
        // These are the starting address of each row in LCD as can be
        // seen from data sheet

    lcd_check_BF_4();               // Make sure LCD controller is ready
    lcd_command(first_char_adr[y-1] + x-1);         // To move the cursor to
        // the defined location the address of that location should be
        // send as command for eg. the command for 2nd column 1st row is
        // 0x81.

}

void lcd_print(char *str) {
    unsigned char i = 0;
```

```
	while(str[i] != 0) {
		// 0 is the ACII value for null character
		lcd_check_BF_4();		// Make sure LCD controller is ready
		lcd_data(str[i]);
		i++;
	}
}

void lcd_check_BF_4(void) {
	uint8_t busy_flag;			// Busy flag 'mirror'
	LCD_DDR &= ~(1<<7);			// Set PA7 to input
	LCD_PRT &= ~(1<<LCD_RS);
	LCD_PRT |= (1<<LCD_RW);				// Read from LCD module

	do {
		busy_flag = 0;			// Initialize busy flag 'mirror'
		LCD_PRT |= (1<<LCD_EN);
		delay_us(1);			// Implement 'Delay data time' (160 ns)
			// and 'Enable pulse width' (230 ns)
		busy_flag |= (LCD_PIN & (1<<7));	// Consider only 7th bit and mask remaining
		LCD_PRT &= ~(1<<LCD_EN);
		delay_us(1);			// Implement 'Address hold time (10 ns),
			// 'Data hold time' (10 ns), and 'Enable cycle time' (500 ns)

	// Read and discard alternate nibbles (D3 information)
		LCD_PRT |= (1<<LCD_EN);
		delay_us(1);			// Implement 'Delay data time' (160 ns)
			// and 'Enable pulse width' (230 ns)
		LCD_PRT &= ~(1<<LCD_EN);
		delay_us(1);			// Implement 'Address hold time (10 ns),
			// 'Data hold time' (10 ns), and 'Enable cycle time' (500 ns)
	} while(busy_flag);

	LCD_PRT &= ~(1<<LCD_RW);		// Write to LCD module
	LCD_DDR |= (1<<7);			// Reset D7 data direction to output
}

void delay_ms(unsigned int delay) {
	//_delay_ms(d);
	//*-- This line gives us error because we cannot use the delay function
	//	with the variable value as given in the user manual --* //
	while(delay --) {
			_delay_ms(1);
	}
}

void delay_us(unsigned int delay) {
	while(delay --)
```

```
                _delay_us(1);
}
```

==========



Output 6

Output 7

This is just a demo. I know you are more creative than me and can display much more interesting custom characters. Feel free to share your creativity among others and to me as well. You can mail your projects as well as confusion to me. I will help you as far as I can. Also help me to correct me. And one more thing if you know something more than this then don't forget to inform me as I'm also learner and love to learn from you.

Last modified: 24th April 2020
roshanshrestha04@gmail.com