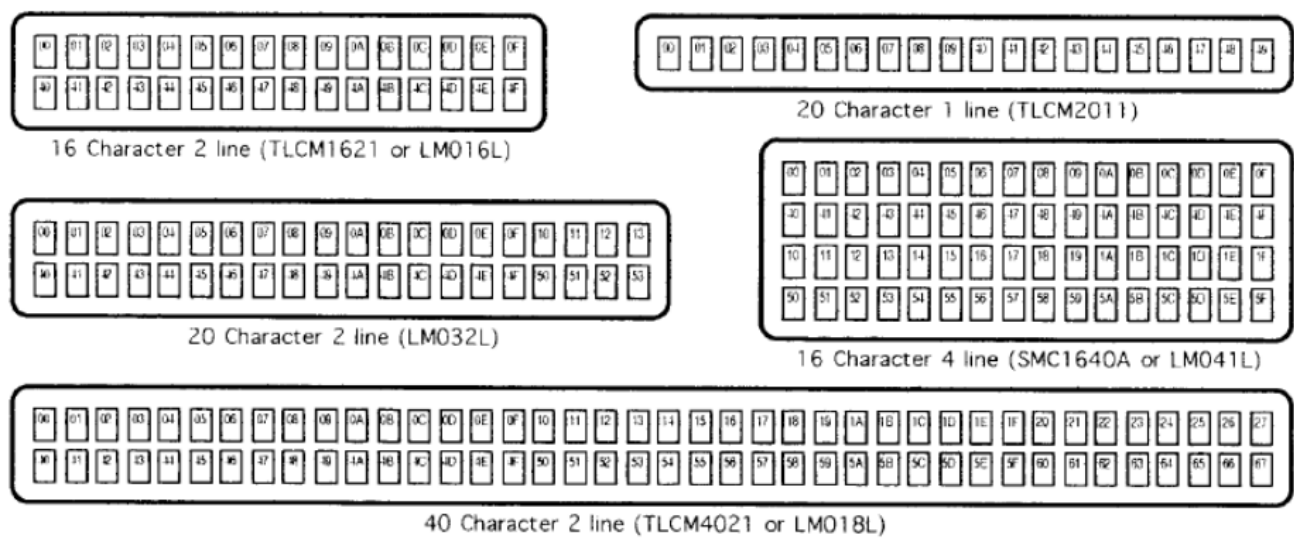


*A Generic Keypad and LCD Tutorial*  
*Ashraf Suyyagh*

LCD

This tutorial is specific to the LCDs based on the Hitachi HD44780 controller chipset. These LCDs come in different shapes and sizes with 8, 16, 20, 24, 32, and 40 characters per line as a standard in 1, 2 and 4-line versions. However, all LCD's regardless of their external shape are internally built and addressed as if they were in 40x2 format. See figure below



LCD I/O

Most LCD modules conform to a standard interface specification. A 14-pin access is provided having eight data lines, three control lines and three power lines as shown below. Some LCD modules have 16 pins where the two additional pins are typically used for backlight control. Pin 1 is marked on the LCD module to avoid confusion. Powering up the LCD requires connecting three lines: one for the positive power  $V_{dd}$  (usually +5V), one for ground  $V_{ss}$ . The  $V_{ee}$  pin is usually connected to a potentiometer which is used to vary the contrast of the LCD display. You can connect this pin to GND for maximum contrast. With 16-pin LCDs, you can use the L+ and L- pins to turn the backlight (BL) on/off.

To operate the LCD, there are three control lines: RS (Register Select), RW (read/write) and E (Enable), and eight data lines D7-D0.

You need to make the necessary GPIO configurations for each one of the above I/O lines.

Operating the LCD



When powered up, the LCD display should show a series of dark squares. These cells are actually in their off state. When power is applied, the LCD is reset; therefore we should issue a command to set it on. Moreover, you should issue some commands which configure the LCD. (See the table which lists all possible

PIN NO	NAME	FUNCTION
L+	Anode	Background Light
L-	Cathode	Background Light
1	Vcc	Ground
2	Vdd	+ve Supply
3	Vee	Contrast
4	RS	Register Select
5	R/W	Read/Write
6	E	Enable
7	D0	Data Bit 0
8	D1	Data Bit 1
9	D2	Data Bit 2
10	D3	Data Bit 3
11	D4	Data Bit 4
12	D5	Data Bit 5
13	D6	Data Bit 6
14	D7	Data Bit 7

configurations presented later in this tutorial and the explanation to each field).

Using an LCD is a simple procedure. Simply send a value to the LCD lines D7-D0 (this value might be an ASCII value (character to be displayed), or another hexadecimal value corresponding to a certain command). Then follow with certain signals on the control lines (see procedure below). So how will the LCD differentiate if this value on D7-D0 is corresponding to data or command?

Observe the figure below, as you might see the only difference is in the RS signal (Register Select). This is the only way for the LCD controller to know whether it is dealing with a character or a command!

Command	Binary											
	RS	R/W	E	D7	D6	D5	D4	D3	D2	D1	D0	
Write Data to CG or DD RAM	1	0		ASCII Value								
Write Command	0	0		Refer to the Command Table below								

Procedure for sending commands/data to the LCD (If you only write to the LCD, just ground the R/W pin):

<p>Steps to send character to LCD</p> <ol style="list-style-type: none"><li>1.Place the ASCII character on the D0-D7 lines</li><li>2. Register Select (RS) = 1 to send characters</li><li>3. "Enable" Pulse (Set High – Delay – Set Low)</li><li>4. Delay to give LCD the time needed to display the character</li></ol>	<p>Steps to send a command to LCD</p> <ol style="list-style-type: none"><li>1.Place the command on the D0-D7 lines</li><li>2. Register Select (RS) = 0 to send commands</li><li>3. "Enable" Pulse (Set High – Delay – Set Low)</li><li>4. Delay to give LCD the time needed to carry out the command</li></ol>
--	--

Delay times are around 37µs for most commands and data writes. The clear display command takes 1.52 ms.

The Enable line is negative edge triggered. So a pulse width of few microseconds is sufficient

You can issue reads from the LCD by changing the control lines and sending an enable signal, you have two options:

- (RS/RW = 01): Reads busy flag (BF) (Which is the MSB of datelines) and returns CGRAM or DDRAM address counter contents (depending on previous instruction).
- (RS/RW = 11): Read the actual data content of the CGRAM or DDRAM address

## Displaying Characters

All English letters and numbers (as well as special characters, some Japanese and Greek letters) are built in the LCD module in such a way that it **conforms to the ASCII standard**. In order to display a character, you only need to send its ASCII code to the LCD which it uses to display the character and follow with control lines as described previously.

Notice that from column 1 to D, the character resolution is 5 pixels wide x 7 pixels high (5x7). Whereas the character resolution of columns E and F is 5 pixels wide x 10 pixels high (5x10). You should change the resolution if you wish to use characters from different resolution columns. This can be done using a command discussed later.

## Sending Commands

Sending commands is no different than displaying characters except for the logic level of the RS line. The available commands are summarized in the table below. You need to initialize the LCD before using it. Detailed explanation of the settings follows:

Library 8 Bits Control 4 Bits	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	CG RAM (1)															
0001	CG RAM (2)															
0010	CG RAM (3)															
0011	CG RAM (4)															
0100	CG RAM (5)															
0101	CG RAM (6)															
0110	CG RAM (7)															
0111	CG RAM (8)															
1000	CG RAM (1)															
1001	CG RAM (2)															
1010	CG RAM (3)															
1011	CG RAM (4)															
1100	CG RAM (5)															
1101	CG RAM (6)															
1110	CG RAM (7)															
1111	CG RAM (8)															

Command	Binary								Hex
	D7	D6	D5	D4	D3	D2	D1	D0	
Clear Display	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	1	x	02 or 03
Character Entry Mode	0	0	0	0	0	1	1/D	S	04 to 07
Display On/Off & Cursor	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	1	D/C	R/L	x	x	10 to 1F
Function Set	0	0	1	8/4	2/1	10/7	x	x	20 to 3F
Set CGRAM Address	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	1	A	A	A	A	A	A	A	80 to FF

1/D: 1=Increment\*, 0=Decrement  
S: 1=Display shift on, 0=Off\*  
D: 1=Display on, 0=Off\*  
U: 1=Cursor underline on, 0=Off\*  
B: 1=Cursor blink on, 0=Off\*  
D/C: 1=Display shift, 0=Cursor move

R/L: 1=Right shift, 0=Left shift  
8/4: 1=8-bit interface\*, 0=4-bit interface  
2/1: 1=2 line mode, 0=1 line mode\*  
10/7: 1=5x10 dot format, 0=5x7 dot format\*  
x = Don't care      \* = Initialization settings

### Clear Display

Clears the LCD display, however the cursor will remain at its last position. Any future character writes will start from the last location. To reset the cursor position use the Display and Cursor Home command.

### Display and Cursor Home

Resets cursor location to position 00 of the LCD screen. Future writes will start at the first location of the first line.

### Character Entry Mode

This command has two parameters 1/D and S:

**1/D:** By default, the cursor is automatically set to move from location 00 to 01 and so on (Increment mode). Suppose now that you are to write from right to left, then you have to set the cursor to the Decrement mode.

**S:** Accompanies the D/C parameter, explained below

### Display On/OFF and Cursor

This command has three parameters:

**D:** Shows/hides all characters on the display.

**U:** This displays the cursor (in the form of a horizontal line at the bottom of the character) when it is high and turns the cursor off when it is low

**B:** If the underline cursor option is enabled, this will blink the cursor if set high.

### Display/Cursor Shift

All LCDs based on the HD44780 format - whatever their actual physical size is - are internally built as 40 characters x 2 lines. The upper row has the display addresses 0-27H (27H = 39D so 0-39 = 40 Characters). The lower row from 40 H - 67H. Now suppose you have an LCD with the physical size of 20x2 lines, when you start writing to the LCD and the cursor reaches locations 20D, 21 D, and 22 D ..., you will not see the characters. They are still being written in their respective locations but you cannot see them. You are limited by the maximum 20 **visible** characters (in our assumed LCD). All you have to do is shift the display as follows:

1. Determine the direction of the shift (R/L)
2. Issue the shift Command D/C

**R/L:** Determines the direction of the shift.

**D/C:** if this bit has a value of 0, then no shift occurs. If set to logic high, the display is shifted once. You might need to issue this command multiple times in order to shift the display by multiple locations.

### Function Set

This command has three parameters:

**8/4:** Eight/Four bits mode

8 – Bit interface: you send the whole command/character (8 bits) in one stage to the D0-D7 lines

4 – Bit interface: you send the command/character in two stages as nibbles to D4-D7 lines.

When to use the 4-bit mode?

1. Interfacing LCD with older devices which have 4-bit wide buses (Remember this LCD came out in the 1970's)
2. You don't have enough I/O pins remaining; you want to conserve the I/O pins for other HW

**2/1:** Line mode, determines whether you want to use the upper line of the LCD or both lines

**10/7:** Dot format, based on the LCD built-in characters table, note the following:

- \* 5x7 format (Default) is used whenever you use the characters found in columns 1 to D
- \* 5x7 format is also used whenever you use the built in characters in CG-RAM.
- \* 5x10 format is only used when displaying the characters found in columns E and F

### Set Display Address command

Syntax: 1AAAAAA

This command allows you to move the cursor to whichever location you want. Suppose you want to start writing in the middle of the display (assuming the **visible** width of the LCD screen is 20), then you will observe that location 06 approximately sets the word in the middle.

1AAAAAA → 10000110 → 0x86

Moreover, suppose you wish to move to the second line which starts at location 40, same as above  
1AAAAAA → 11000000 → 0xC0

The decimal address is filled in as BCD in the address field.

### Set CG-RAM Address command

Syntax: 01AAAAA

If you give a closer look at the ASCII table shown above, you will clearly see that the table only contains English and Japanese characters, numbers, symbols as well as special characters. Suppose now that you would like to display a character not found in the built-in table of the LCD. In this case we will have to use what is called the CG-RAM (Character Generation RAM), which is a reserved memory space in which you could draw your own characters and later display them.

Observe column one in the ASCII table, the locations inside this column are reserved for the CG-RAM. Even though you see 16 locations (0 to F), you only have the possibility to use the first 8 locations 0 to 7 because locations 8 to F are mirrors of locations 0 – 7.

So, to organize things, in order to use our own characters we have to do the following:

1. Draw and store our own defined characters in CG-RAM
2. Display the characters on the LCD screen as if it were any of the other characters in the table

#### ***Drawing and storing our own defined characters in CG-RAM***

To choose where each one of our eight user-defined characters is stored, there is a simple rule

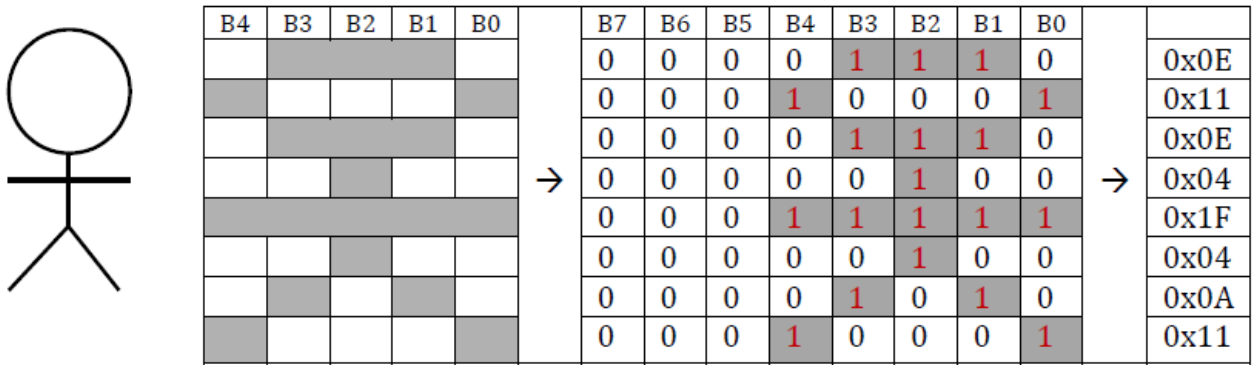
1. To specify you want to write/build/store a character in location 00, you send the CG-RAM address **command** as follows: 01AAAAA → 01000000 → 0x40
2. To specify you want to write to any location from 01 to 07, you have to skip eight locations for each. So the CG-RAM address **command** will send **0x48** (to store a character in location 1), **0x50** (to store a character in location 2) and so on.
3. After either 1 or 2, send the pattern you want to store as eight sequential characters (character mode)
4. Issue the clear display command

**(All steps above need to be done once, in initialization stage)**

5. To display, any of the user-defined characters, simply send 00, 01 – 07 in data mode.

**Drawing your pattern:**

This is the fun part, draw a 5x8 Grid and start drawing your character inside, then replace each shaded cell with one and the empty ones with zero. Append three zeros to the left. Finally encode the sequence into hexadecimal format. This is the sequence which you will fill in the CG-RAM SEQUENTIALLY once you have set the CG-RAM Address before.



**Animation**

You can do very simple animations on this LCD. For example, suppose you want to animate a dancing stickman. Draw and save the dance patterns of your stickman in the CGRAM. Once done, start displaying them as before. The only difference is that you need to change the curser location back to the same location. (Remember, the cursor auto-increments / auto-decrements after each data display). If you are using only the first location (00) for your animation, then a cursor home command will suffice. Else, you need to use the set cursor command.

## Keypad

The following presentation is quite generic. We assume the keypad with the alphanumeric layout shown in Figure 1. Many keypads have different key layouts.

### General Overview

For each column/row, there is a pin (hole or leg connector) on the keypad associated with it. So in this case, for a 4x4 Keypad, we have four for each of the rows and columns.

You can identify the order of pins (that is to which pin is each column/row connected) by either referring to the schematic (pretty much a standard for most typical keypads) or by making use of the multimeter to detect a closed circuit. Press a button and use the probes of the multimeter over the pins to see when it gives you a short circuit alert sound.

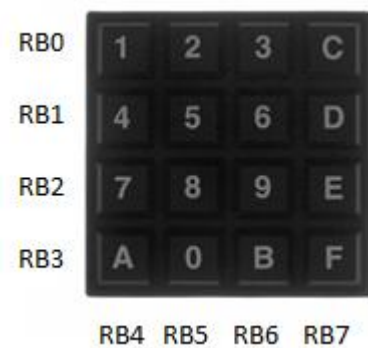


Figure 1 - a 4x4 Keypad

Once you identified the pin associations, keep record of that, you will need it later to write down the pattern table and interfacing the keypads.

### Pin assignment and interfacing

Choose a port of your choice (let's arbitrarily say PORTB) and connect the pins to it in the order shown in Figure 1:

- PORTB 0-3 to the rows (Configure initially as output).
- PORTB 4-7 to the columns (Configure initially as input, pull up mode enabled).

Actually, you can use more than one port and different pins (not necessarily sequential). But for sake of clarity and simplicity, we'll keep it this way.

### General Keypad Operation

A keypad is interfaced in a way such that initially if no key is pressed you will read a certain logic level, and when you press a button a signal with the opposite of the original logic level will be read. So you have two cases:

1. Fix the initial button state to be read as logic 1 (through the use of pull-up resistors). When you press a button you will read logic 0.
  2. Fix the initial button state to be read as logic 0 (through the use of pull-down resistors). When you press a button you will read logic 1.
- *Pull-up and pull-down resistors are used to limit the amount of current and protect the circuit. (not to read a floating state)*
  - *Pull-up and pull-down resistors could be connected externally, but you can make use of the internal pull-up resistors found internally in the GPIO modules in many microcontrollers/microprocessors (Push/Pull mode in GPIO configuration).*

Whether you use internal or external pull-up resistors the concept of operating a keypad is the same.

### Technique (Scanning Algorithm)

The procedure to identify which button of a keypad has been pressed is fairly easy. It is done through a scanning algorithm. We will explain this procedure based on a typical 4x4 keypad. The procedure could be applied to keypads of any size.

First the row bits are set to output, with the column bits as input. The output rows are set to logic 0. If no button is pressed, all column line inputs will be read as logic 1 due to the action of the weak pull-up resistors.



If, however, a button is pressed then its corresponding switch will connect column and row lines, and the corresponding column line will be read as low.

To detect this logic transition from high to low (that is to know whether a key has been pressed or not), we have to either:

1. Keep pulling the inputs (columns) continuously until 0 is detected on one of the lines.
2. Make use of interrupts

So far, we have identified the column in which the key was pressed but not the button itself. So what we do now is save the column pattern we read and repeat the same procedure above with the following minor modification:

*The column bits are set to output, and the row bits as input. Then the output columns are set to logic 0.*

Since the button is still pressed (this switching is assumed to happen in a few cycles), then the pressed button is still connecting column and row lines. If you read the row pins, then the corresponding row line will be read as low. If, however, the button is released all row line inputs will be read as logic 1 due to the action of the weak pull-up resistors. Now we have a pattern for the row.

*\*It does not matter whether you start scanning rows or columns first. But if you are using interrupts, make sure that you enable them for the ones you read first.*

Example

Suppose we connect the columns to PORTB 4-7 as input and the rows to PORTB 0-3 as output (with the value of 0). If we continuously read the inputs they will always be read as 1 because of the internal pull-up resistors on PORTB. If one presses number “7”, this will make us read logic 0 on RB4 (means that we have pressed a button in the first column).

Now let us reverse the roles; that is set the rows (PORTB 0-3) as input and the columns (PORTB 4-7) as output (with the value of 0). If we read the input we will find that RB2 is 0. Now we have identified the row of the pressed button. Using both patterns, we are ready to identify which button was pressed

So what comes next?

The above scanning technique helps us determine the position of the pressed button (in terms of its row/column intersection) by providing us with unique patterns for each button. Given that, you can use that unique value to do anything you want. Case/Switch statements can be handy in this case. Table one lists the unique patterns you will get based on the above discussion and the keypad pattern shown in Figure 1.

Table 1 - The values which will be read when a key is pressed

Key pressed	Column RB7, RB6, RB5, RB4	Row RB3, RB2, RB1, RB0
1	1110	1110
4	1110	1101
7	1110	1011
A	1110	0111
2	1101	1110
5	1101	1101
8	1101	1011
0	1101	0111
3	1011	1110
6	1011	1101
9	1011	1011
B	1011	0111
C	0111	1110
D	0111	1101
E	0111	1011
F	0111	0111



### Important notes and common mistakes

1. You can run this algorithm in its own thread such that it could run periodically to scan for button presses or better, you can use interrupts to detect a change from 1 to 0 on the lines and inside the ISR signal the thread to start scanning. But be sure you won't miss the button press event or read the keypad after it has been depressed.
2. Of course there are other things to take into consideration:
  - De-bouncing
  - Duration of the press
  - Do you want a certain functionality to be ON once a button is pressed or DURING the whole time the button is pressed
  - Multi-key presses at the same time
  - Composite keys (shift key)

You will have to find your own solutions and implementations

3. The order you connect the rows and columns pins to PORTB pins determines the overall final pattern for each button. (Switching the order, switches the patterns, your own design choice).
4. The code will not work without enabling pull up resistors, the algorithm is based on detecting changes from 1 to zero and recording the patterns.
5. You can decide to read the columns or rows first, it does not matter. It depends on which you configure as input or output first.
6. After you read the keypad for the second run, don't forget to switch back the input/output configuration to the initial state you started with.