Group: *BCDJ*

4x20 LCD Custom Characters Guide

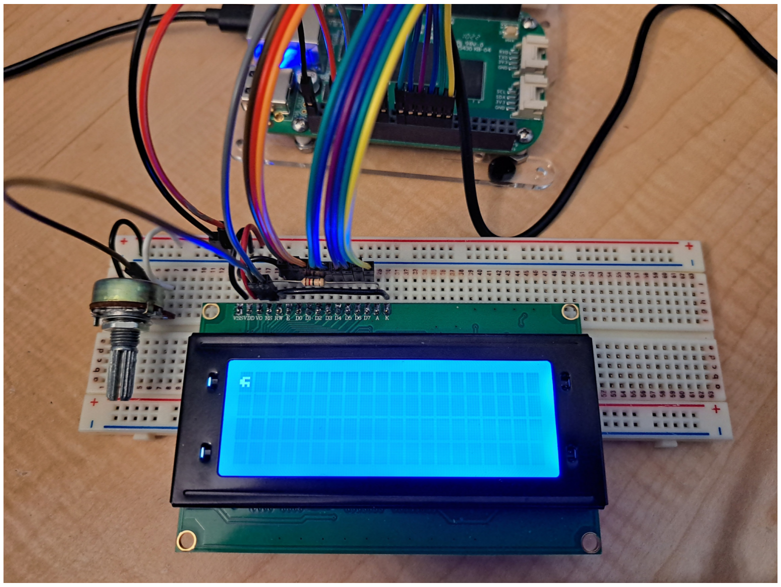
Jake Merkl
Bowie Gian
Chenting Mao

This guide will lead you through the steps of wiring and programming a 4x20 LCD and manipulating it to display something beyond the basic functionality of text and number output.

Hardware requirements

- 4x20 LCD
- 1 kΩ ± 5%
- Potentiometer (optional)

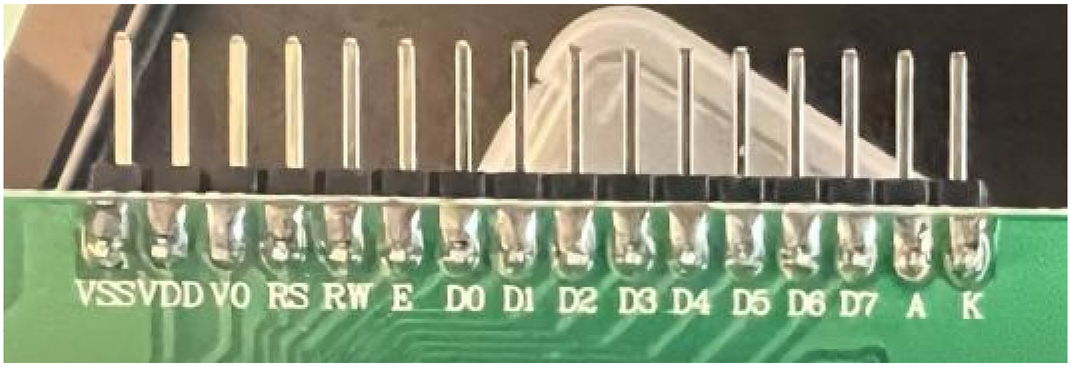
Table of Contents



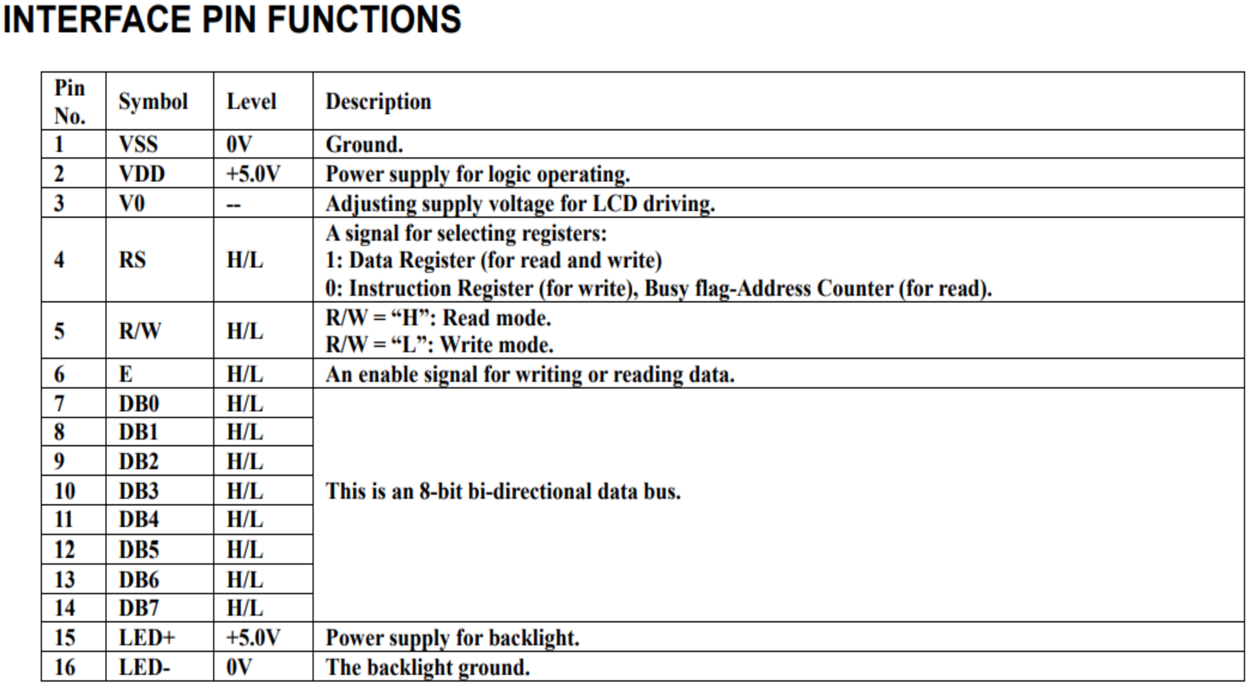
(fig.1 - Custom character output)

1. Component Wiring

The first step is getting your LCD onto the breadboard. There are a total of 16 pins on the LCD.



(fig.2 - LCD pins on screen)



INTERFACE PIN FUNCTIONS

Pin No.	Symbol	Level	Description
1	VSS	0V	Ground.
2	VDD	+5.0V	Power supply for logic operating.
3	V0	--	Adjusting supply voltage for LCD driving.
4	RS	H/L	A signal for selecting registers: 1: Data Register (for read and write) 0: Instruction Register (for write), Busy flag-Address Counter (for read).
5	R/W	H/L	R/W = "H": Read mode. R/W = "L": Write mode.
6	E	H/L	An enable signal for writing or reading data.
7	DB0	H/L	This is an 8-bit bi-directional data bus.
8	DB1	H/L	
9	DB2	H/L	
10	DB3	H/L	
11	DB4	H/L	
12	DB5	H/L	
13	DB6	H/L	
14	DB7	H/L	
15	LED+	+5.0V	Power supply for backlight.
16	LED-	0V	The backlight ground.

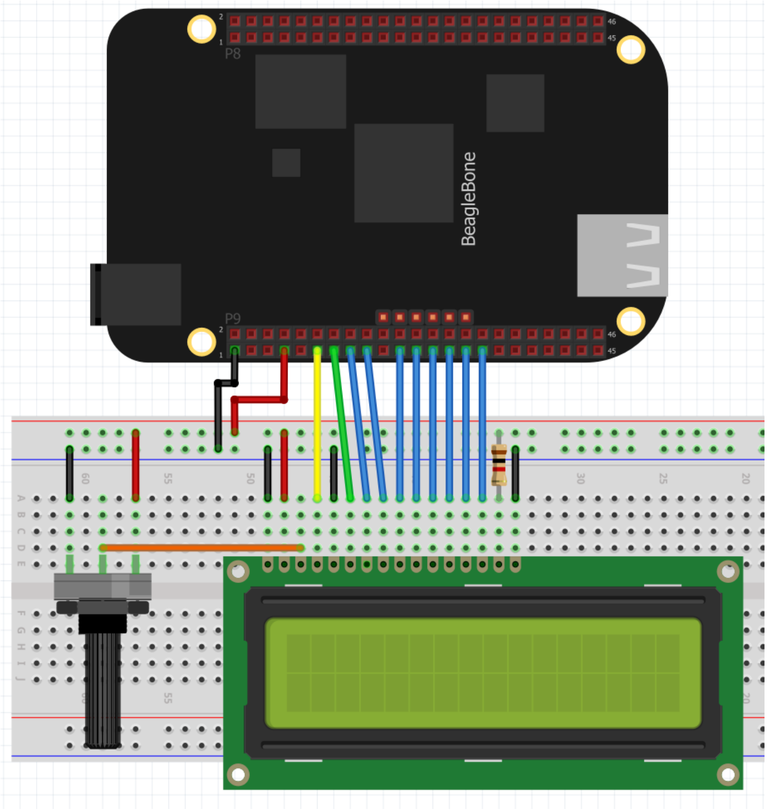
(fig. 3 - LCD pinout from Adafruit TC1602A-01T datasheet [1])

***Note that in the table above LED+/- are used to represent A (LED+) and K (LED-)*

Connect the VSS to ground and the VDD to +5V. The V0 pin will be your screen brightness pin where you would connect the potentiometer. If you decide you don't want to use the potentiometer or don't have one on hand, a simple voltage divider with resistor values to fit your brightness requirements to this pin works just as well.

The pins RS to D7 are all GPIO pins that can go to any pin. We ground the RW pin so that it reads a 0 and is stuck in write mode since we always want to be writing to the screen in this case. It is good practice to place a resistor between the +5V and the A pin so as to not run too much current through it and burn the backlight.

The sample code uses P9 - 11, 13, 15, 17, 21, 23, 25, 27, 29, 31
organized in order : [RS, EN, D0, D1, D2, D3, D4, D5, D6, D7]



(fig.4 - wiring for sample code)

Troubleshooting:

- Remember to set all GPIO pins before attempting to write to them
- If your screen wont turn on check the voltage input pins and make sure you didn't burn the backlight

2. Initialization

The LCD has on board functions to deal with the type of job you are looking to apply it to. First we set what mode we want to write in. The screen lets us choose between an 8-bit mode using all registers or a 4-bit mode using only 4. The sample code uses 8-bit mode so we will list commands as such within this guide.

***If you are interested in 4-bit mode you can toggle it on with the boolean variable isNibbleMode within the sample code.*

After selecting the write mode, we then clear the screen, turn off the cursor, and set the address of the 5x8 character spaces to auto-increment.

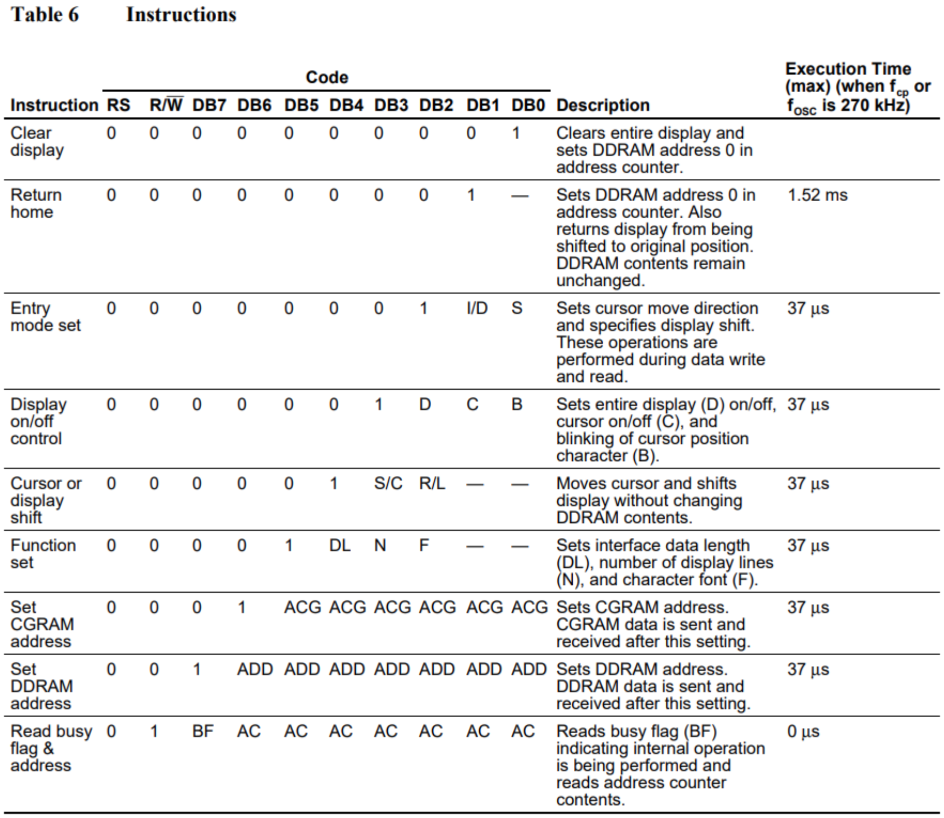


Table 6 Instructions

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

(fig.5 - Instructions table from HD44780U HITACHI manual [2])

Troubleshooting -

- If somethings going wrong, you're probably not waiting for the suggested execution time.

3. Sample Output

3.1 GPIO manipulation -

The HD44780U has a library of readily available characters to print within its memory that we can read from. For this section we will use letters since they follow the C char value and are easiest to deal with.

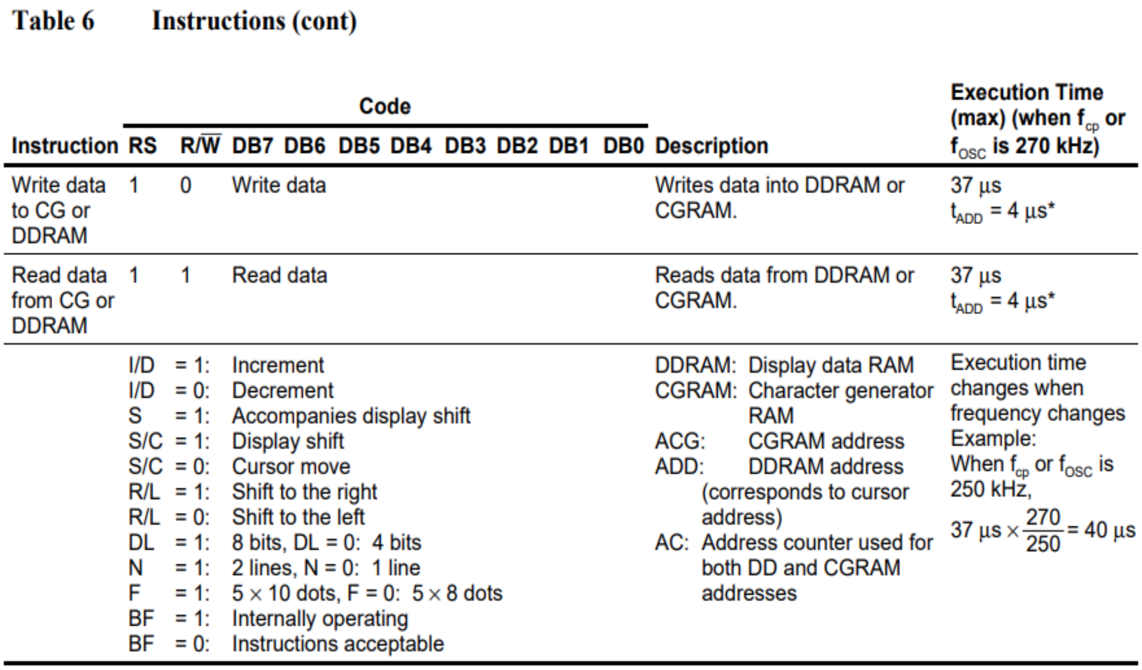


Table 6 Instructions (cont)

	Code											Execution Time (max) (when f_{cp} or
Instruction	RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	f_{OSC} is 270 kHz)
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.	37 μs t_{ADD} = 4 μs*
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.	37 μs t_{ADD} = 4 μs*
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8 bits, DL = 0: 4 bits N = 1: 2 lines, N = 0: 1 line F = 1: 5 × 10 dots, F = 0: 5 × 8 dots BF = 1: Internally operating BF = 0: Instructions acceptable										DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (corresponds to cursor address) AC: Address counter used for both DD and CGRAM addresses	Execution time changes when frequency changes Example: When f_{cp} or f_{OSC} is 250 kHz, $37\ \mu s \times \frac{270}{250} = 40\ \mu s$

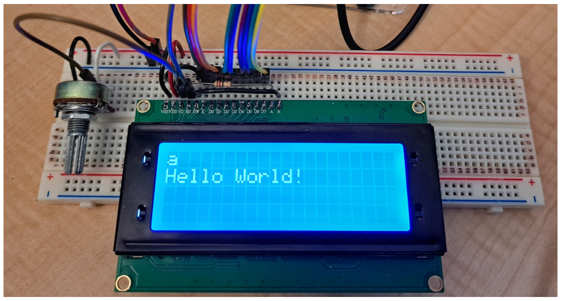
(fig.6 - Read/Write bit table from HD44780U HITACHI manual [2])

To start outputting to your screen use the sample code supplied and #include the .h file into a main.c file and use the commands shown below. Then run the code on your BBG.

```
LcdScreen_setup(false); // 8-bit mode, cursor is reset to (0,0)
LcdScreen_sendData('a');
LcdScreen_moveCursor(1,0);
LcdScreen_writeString("Hello World!        ");
```

***Note you must add spaces after the string to fill the remainder of the row due to 0 being a special memory location (see section 5)*

(fig.7 - sample text output)



Troubleshooting-
- Make sure to include all files in the makefile
- Make sure the shared folder is set up and mounted

3.2 Important functions -

Within the sample code we have included multiple functions that could be useful in use of the screen for text output such as:

```
LcdScreen_moveCursor();
LcdScreen_showCursor();
LcdScreen_hideCursor();
LcdScreen_sendData();
LcdScreen_writeString();
```

These functions allow the user to have more freedom within the screen and make outputting to it much more simplified.

4. Creating a custom character

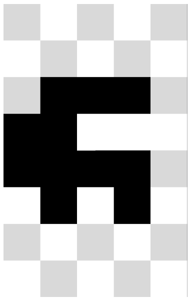
Now that the main file is set up and the screen has an output, it's time to make that output into something custom rather than text. This output can be whatever you can draw in a 5x8 pixel box.

***Note the below steps are optional and are simply for designing your custom character conceptually*

1. First go to https://www.pixilart.com/draw
2. Click skip at the top for the tutorial and wait for the popup
3. Edit the width to be 5 and height to be 8
4. Now create your own drawing in the canvas
5. Refer to lcdScreen.h and create a customChar_t for that character.

This variable is an array of size 8 with each index in binary that correlates to the 5 columns of the character space. As an example we will use this character:

(fig. 8 - custom character arranged from pixelart.com)



This character would then be represented within the customChar_t as such:

```
    customChar_t character = {0b00000, 0b00000, 0b01110,
0b11000, 0b11110, 0b01010, 0b00000, 0b00000};
```

***It is worth noting that the LCD only has enough space for 8 custom characters*

5. Outputting the Custom Character

At this point your custom character will be within your code to write to the screen. All that's left to do is simple:

1. Run command

   ```
   Lcdscreen_loadCstmChar(0, character);
   ```

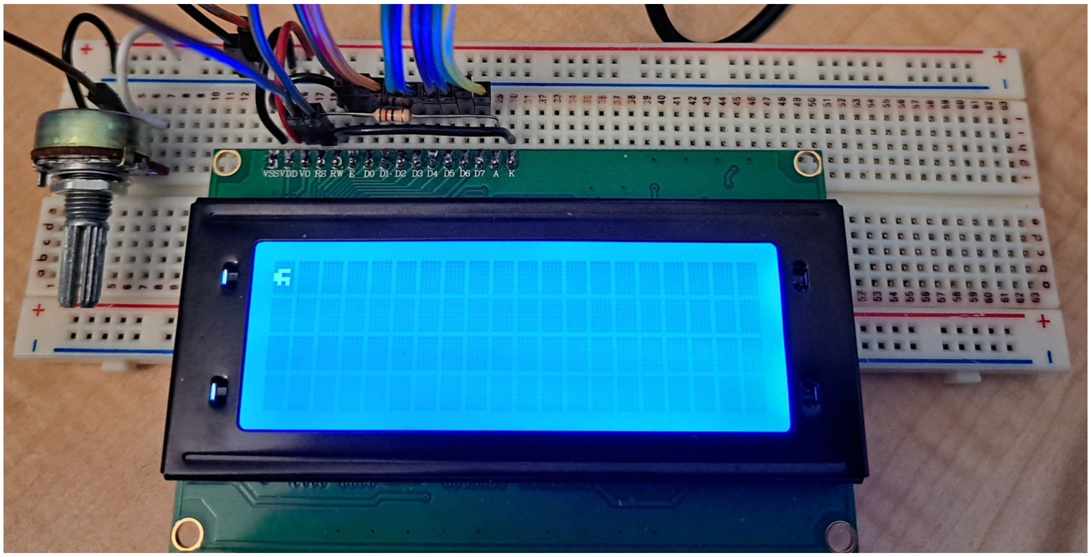
   after initialization.

This will load your character to the screen's memory at position 0 so it can read and write your character to the screen.

2. Run the command

   ```
   Lcdscreen_placeChar(int row, int col, 0);
   ```

This will then write your character to the screen at the position [row, col] based on the 4x20 character boxes.



You now have displayed your own custom character onto the screen.

Additional notes:

- Connecting a joystick and linking it to the Lcdscreen_placeChar(); function makes for easy controlling of the character movement on the screen.
- Using the on board functions of the LCD combined with your custom character you can make them scroll across the screen, jump down a row, animate them etc.
- You can alter how a character looks by editing the custom char file in between outputs to the screen rather than taking up another space in memory to animate it. Note this will affect all currently displayed instances of the custom character.
- Code can be easily adapted for a 2x16 LCD

References

[1] - Tinsharp industrial co.
https://cdn-shop.adafruit.com/datasheets/TC1602A-01T.pdf
Used pin description table (pg. 5)

[2] - Hitachi HD44780U User Manual
https://cdn-shop.adafruit.com/datasheets/HD44780.pdf
Various photos and tables used (pg. 24, 25)

[3] - GPIO guide by Brian Fraser
https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/ensc351/guides/files/GPIOGuide.pdf
Formatting adapted from this guide