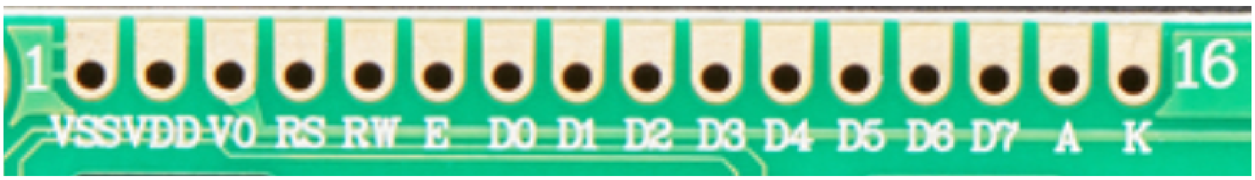
<u>How-To Guide for 16x2 Character LCD Through GPIO</u>

This guide will demonstrate how to wire a 16x2 Character LCD to a BeagleBone Green (BBG) and interface with it through GPIO. Similar online guides are written for Arduinos, and older student guides are either written for older BeagleBones or use a different interface from GPIO. See the references section for secondary resources.

Wiring [1]

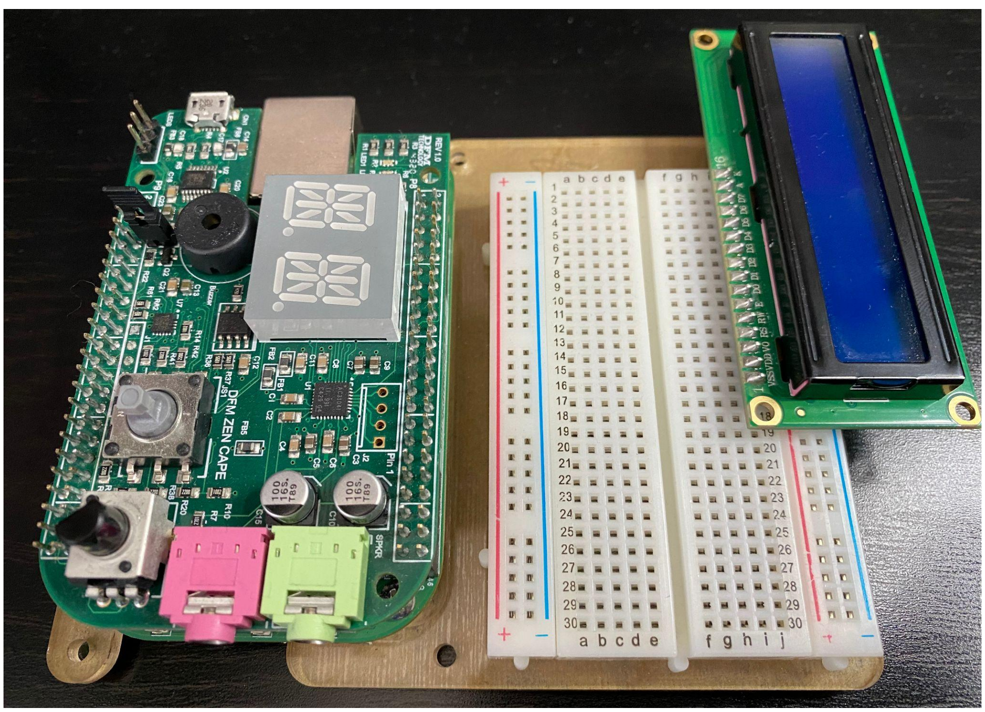
The LCD has 16 pins as shown in this diagram. In this guide, the pins will be referred to by name, as well as their pin number relative to the left and right labels (1, 16). The purpose of each pin will be explained as they are connected to the BBG when it is needed.



LCD Pin Labels (1 - 16, left to right) [1]

First, we will wire up the LCD backlight.

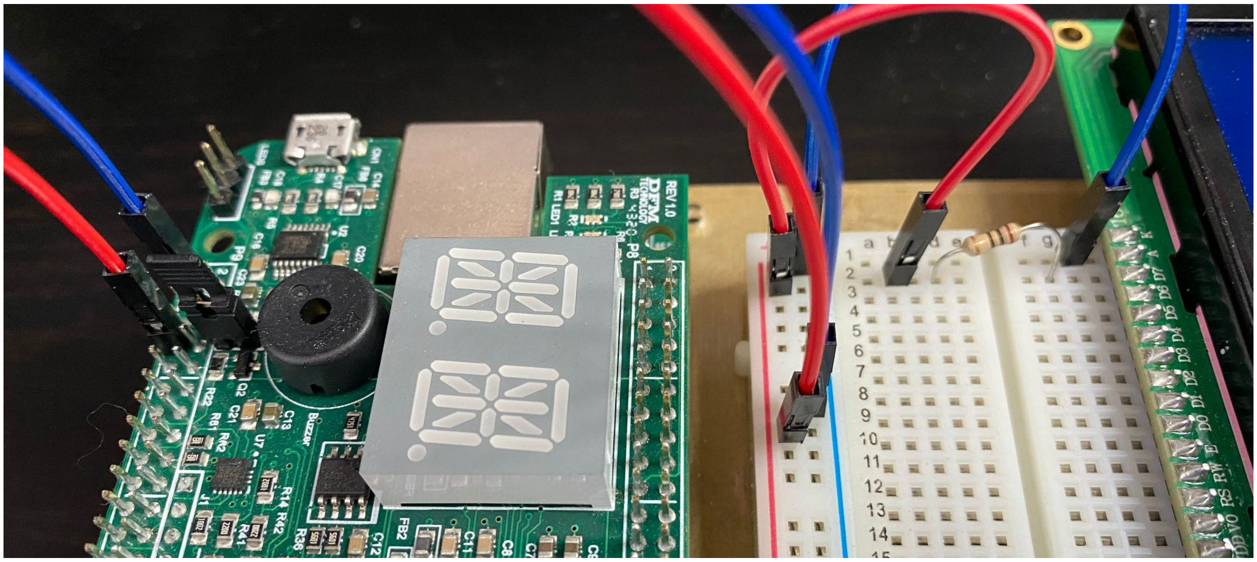
1. Attach the LCD screen to the breadboard.



The LCD attached to the breadboard

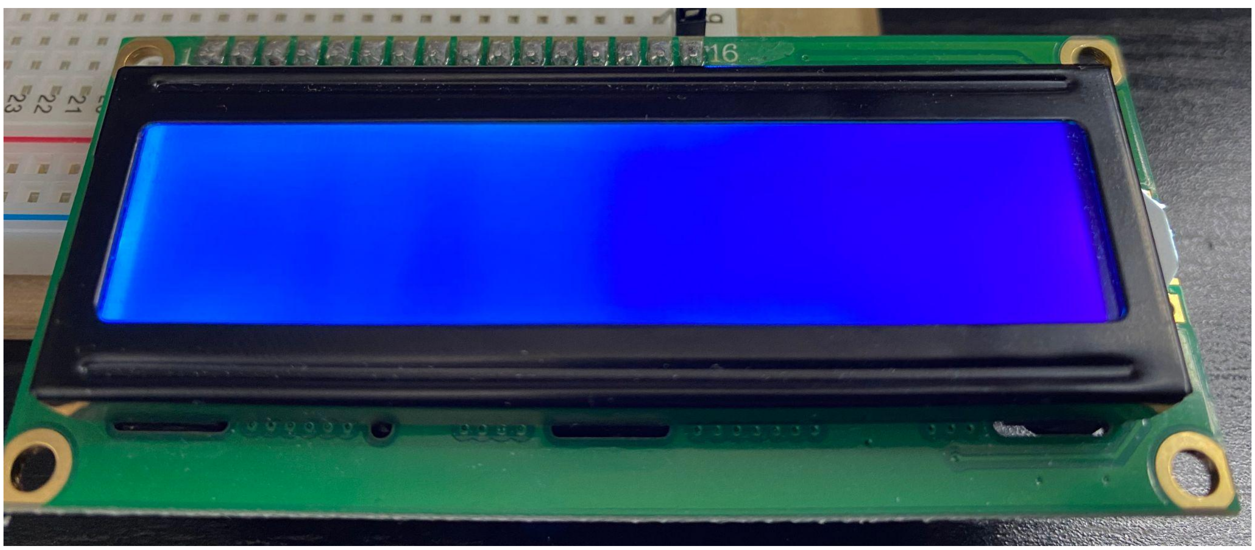
2. Connect a SYS_5V pin on the BBG to the breadboard's red rail, and a GND pin to the breadboard's blue rail.
3. Connect the LCD 'K' pin (16) to the blue rail. The K pin is the ground for the backlight.
4. To provide power to the backlight, connect the LCD 'A' pin (15) to the red rail **with a 10 ohm resistor**. Without this resistor, you may burn out the backlight [2]. If you don't have

a 10 ohm resistor, you can try using stronger resistors instead, but the backlight may become too dim.



The LCD backlight wiring

Once you have double checked that your wiring is correct, plug in your BBG to see the backlight power on.



The backlight power on.

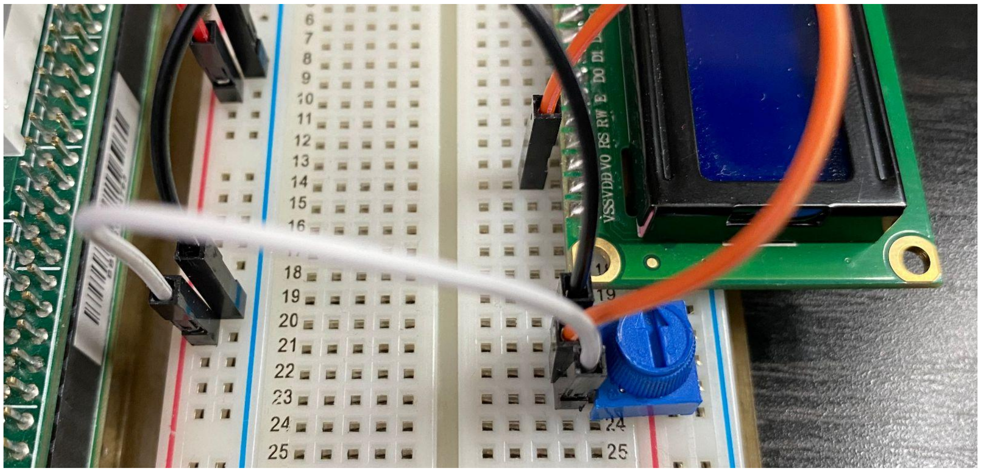
Backlight not powering on?

- Double check that you have wired the BBG power (SYS_5V) and ground correctly.
- Double check that you have connected the resistor.
- Try SYS_3V3 in case it still does not work: the backlight will be dimmer however.

Next, we will attach the contrast potentiometer (pot).

5. Attach the contrast pot to the breadboard.

6. The contrast pot has 3 pins. Connect the left and right pins to the red and blue rail (which side doesn't matter).
7. Connect the middle pin to the LCD 'V0' pin (3). The V0 pin handles the contrast of the character display against the backlight. This connection allows the pot to control the voltage.

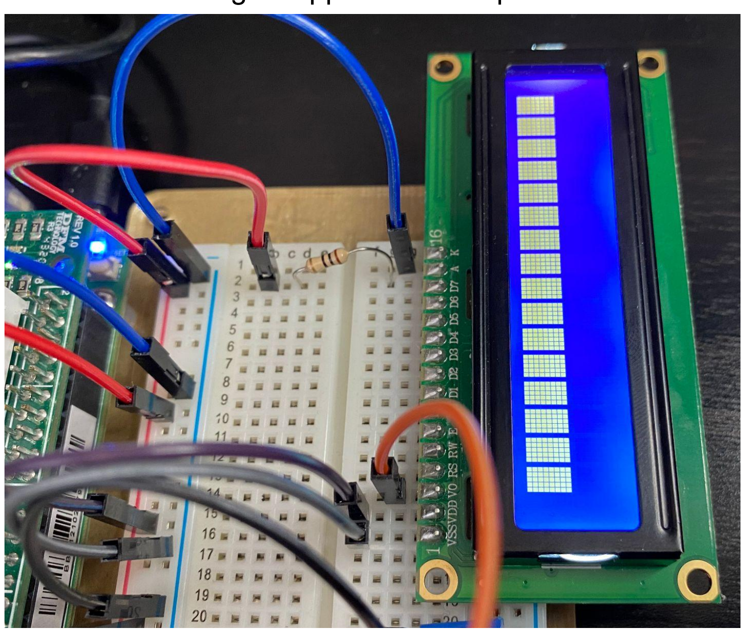


The LCD contrast pot

Now, we will wire up the LCD logic.

8. Connect the LCD 'VSS' pin (1) to the blue rail (ground).
9. Connect the LCD 'VDD' pin (2) to the red rail (voltage).

The contrast and logic can now be tested. Plug in your BBG again and try twisting the contrast pot. You should be able to see rectangles appear in the top line.



These rectangles should appear when you twist the pot

The rectangles not appearing?

- Double check that you have wired the LCD logic correctly.

- Try SYS_3V3 in case it still does not work.

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETN
GPIO_30	11	12	GPIO_60
GPIO_31	13	14	GPIO_50
GPIO_48	15	16	GPIO_51
GPIO_5	17	18	GPIO_4
I2C2_SCL	19	20	I2C2_SDA
GPIO_3	21	22	GPIO_2
GPIO_49	23	24	GPIO_15
GPIO_117	25	26	GPIO_14
GPIO_115	27	28	GPIO_113
GPIO_111	29	30	GPIO_112
GPIO_110	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	GPIO_7
DGND	43	44	DGND
DGND	45	46	DGND

P8

DGND	1	2	DGND
GPIO_38	3	4	GPIO_39
GPIO_34	5	6	GPIO_35
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
GPIO_23	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
GPIO_22	19	20	GPIO_63
GPIO_62	21	22	GPIO_37
GPIO_36	23	24	GPIO_33
GPIO_32	25	26	GPIO_61
GPIO_86	27	28	GPIO_88
GPIO_87	29	30	GPIO_89
GPIO_10	31	32	GPIO_11
GPIO_9	33	34	GPIO_81
GPIO_8	35	36	GPIO_80
GPIO_78	37	38	GPIO_79
GPIO_76	39	40	GPIO_77
GPIO_74	41	42	GPIO_75
GPIO_72	43	44	GPIO_73
GPIO_70	45	46	GPIO_71

GPIO Pin Table [3]

Finally, we will connect the LCD databus pins. The LCD has 10 databus pins, but we will only need to connect 6 pins (more on this later). For our project, we were able to use these pins:

Pin Number	GPIO Number	LCD Pin
P9_15	GPIO 48	D7
P9_27	GPIO 115	D6
P8_7	GPIO 66	D4
P8_8	GPIO 67	E
P8_9	GPIO 69	D5
P8_10	GPIO 68	RS

The guide will use these pins for instructions.

If some of these pins are unavailable to you, refer to the BBG header tables and the ZenCape schematic documents to find pins that are not reserved by the BBG or used by the ZenCape.

10. Connect the LCD ‘RW (5)’ pin to the blue rail.
    a. *The RW pin is used to read values from the LCD. This is not necessary to operate the LCD effectively, so we avoid using it and pull it to ground.*
11. Connect the LCD ‘RS’ pin (4) to P8_10.
    a. *The RS pin is used to set what message type is sent to the LCD microcontroller:* ***0 = data (ASCII char), 1 = command (clear display, move cursor)***
12. Connect the LCD ‘E’ (6) pin to P8_8.
    a. *The E pin (enable) is used to inform the LCD microcontroller that there is data in D4 - D7 to be read.*

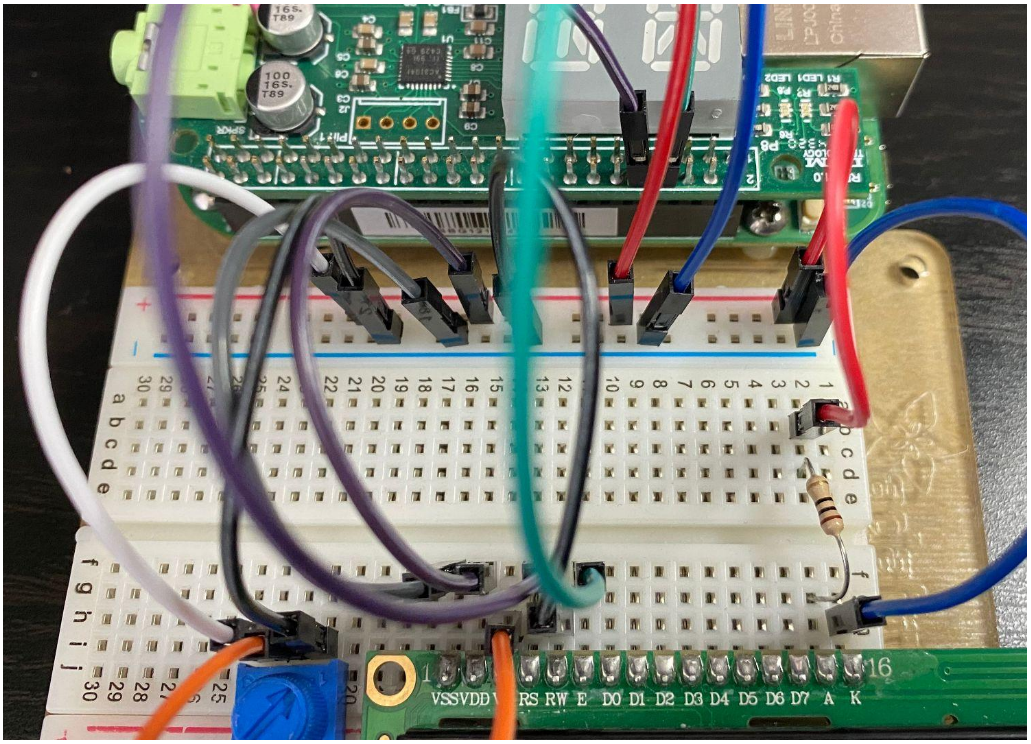
Next we connect the pins used to communicate the actual data to the LCD.

13. Connect the LCD ‘D4’ (11) pin to P8_7.
14. Connect the LCD ‘D5’ (12) pin to P8_9.
15. Connect the LCD ‘D6’ (13) pin to P9_27.
16. Connect the LCD ‘D7’ (14) pin to P9_15.

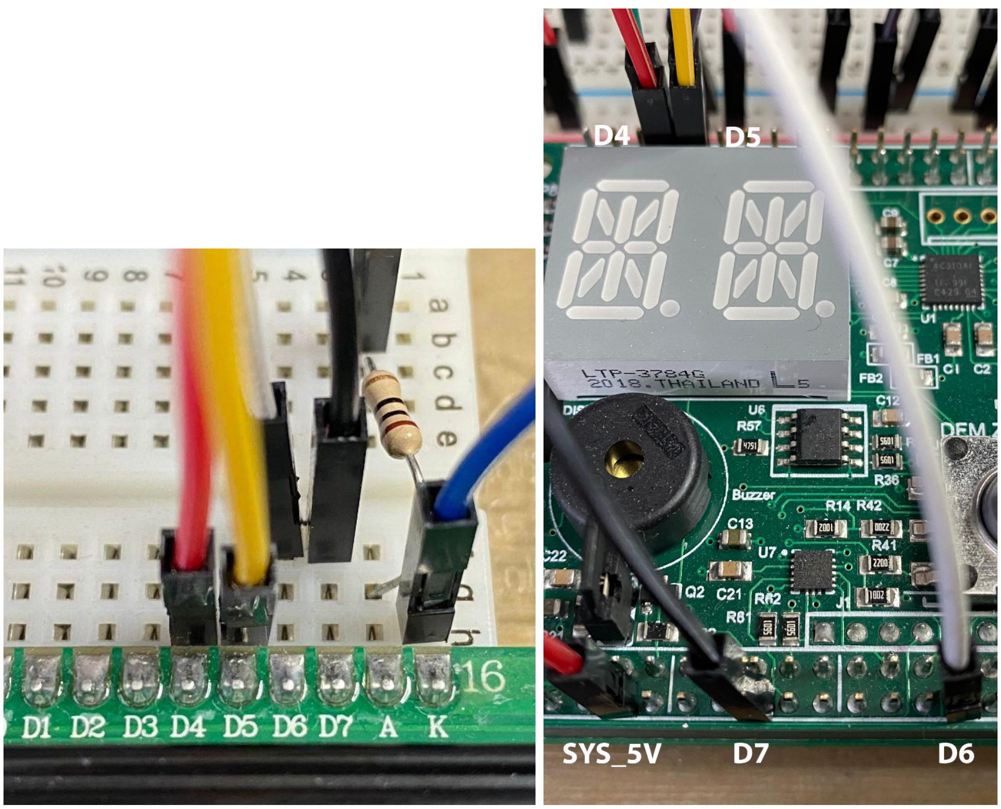
Images are shown on the next page…

The LCD D0, D1, D2, and D3 pins are not used. When using all 8 data pins, the LCD is in 8-bit mode, meaning it reads 8 bits of data by reading a bit from each pin. The LCD can also operate in 4-bit mode: it reads the upper and lower 4 bits of 8-bit data from D4, D5, D6, and D7 only.

The wiring is now complete. Next, we’ll try testing it by running some sample code.



The RW, RS, and E pins connected



The D4, D5, D6, D7 pins (left image) connected to the BBG (right image)

Sample Code [4]
To test the LCD, we will run the provided C sample code, which will initialize the LCD to 4-bit operation and display a message. Once you have the code available to the host machine, build it - it will automatically copy the executable to your public folder, as well as a shell script to export the necessary pins. In the target, cd to your /mnt/remote/myApps directory, and run the ./export_pins.sh first to export all the GPIO pins needed, then, run ./lcd to test the LCD. You should see “Hello world!” appear on your LCD:

Inspect the code and the following sections in the guide to understand what’s going on to initialize the code and write data to the LCD.

Sample code is not working:

- The pins may not have exported correctly: export manually.
- Double check your wiring again - ensure you wired the RW pin to ground.
    - D0, D1, D2, and D3 do not need to be wired at all.
- Experiment with the timing delays between commands - your LCD microcontroller may need longer delays on certain commands.
- Try other GPIO pins: make sure to update the variables in the code as well.

LCD 4-Bit Operation

The LCD can operate using 4 bits by reading 8-bit data in two halves. First, when the upper bits of data are present in D4-D7, the LCD will read these bits when the E pin flashes from low to high to low. The LCD will store these upper bits. Once the lower bits are in, and E is flashed from low to high to low again, then the LCD will read those lower bits, combine with the upper bits, and handle the combined 8-bit data accordingly.

But before any real commands or data can be written to the LCD like this however, it needs to be properly initialized.

Initialization [5]

To start interfacing with the LCD microcontroller, we must initialize it by sending a series of commands. These commands are sent as a hex number, where each of its 4 bits in binary correspond to the data pins D4, D5, D6, and D7. **D4 holds the least significant bit**, and thus **D7 holds the most significant bit**, e.g.,

$0x03_{(16)} = 0011_{(2)}$:

D7	D6	D5	D4
0	0	1	1

When the LCD first starts, it is in 8-bit operation mode, meaning it expects data from pins D0-D7. In the commands we are about to send however, the lower bits (represented by D0-D3) are irrelevant, meaning we can initialize to 4-bit operation mode without those pins.

One more important detail to operation: the LCD takes a certain amount of time (usually very short) to handle each command. If a command is sent while the LCD is still handling a previous command, then that command will be lost. To deal with this, we simply delay by using a sleep. The sleep length depends on the specific command. See the LCD microcontroller datasheet [6] for minimum durations, but we found these to not be exactly accurate, there is room for experimentation. We provide a recommended timing in the following table:

Initialization Commands:
[5][6]

Command	Value	Note:	Delay
Special Function Set 1	0x03	The first of three special function sets to reset the LCD.	5 ms
Special Function Set 2	0x03	The second of three special function sets to reset the LCD.	128 us
Special Function Set 3	0x03	The final of three special function sets to reset the LCD. After 3 consecutive commands, the **LCD recognizes this as a reset.**	128 us
Initial Function Set	0x02	**Convert the LCD to 4-bit operation.**	1 ms
'Real' Function Set	0x02 0x08	Set to two-line display and use the 5x8 character font. See the datasheet [6] for how to configure these values.	128 us
Display ON/OFF Control	0x00 0x08	This command must be called here as part of initialization: do **not configure display yet**.	128 us
Clear Display	0x00 0x01	This clears the LCD's DDRAM addresses that store data.	64 us
Entry Mode Set	0x00 0x06	Set the mode of character entry: automatic cursor increment and shift right as display is written.	128 us
Display ON/OFF Control	0x00 0x0F	Turn on the display, displaying the cursor and configuring the cursor to blink. See the datasheet [6] for how to configure these values.	64 us

Initialization is now complete. The LCD is fully operational.

References

[1] Adafruit Wiring Guide.
https://learn.adafruit.com/character-lcds/wiring-a-character-lcd
We followed this guide to learn how to wire the LCD, and have adapted parts of this guide for the BeagleBone instead of an Arduino.

[2] Backlight resistor.
https://electronics.stackexchange.com/questions/82092/just-burned-my-first-lcd-but-why
https://cdn-shop.adafruit.com/datasheets/TC1602A-01T.pdf
The stackexhange discussion provides a computed resistance that we verified and used, which is based off the operating voltage provided in the datasheet.

[3] GPIO Guide by Brian Fraser.
See course page - URL may not be accessible at the time you access this guide.
The GPIO pin to GPIO number table is used.

[4] Arduino LiquidCrystal library.
https://github.com/arduino-libraries/LiquidCrystal/blob/master/src/LiquidCrystal.cpp
Loosely referenced when writing the sample code.

[5] LCD Initialization.
https://web.alfredstate.edu/faculty/weimandn/lcd/lcd_initialization/lcd_initialization_index.html
Provides a in-depth description for initializing the LCD to both 8-bit and 4-bit modes. See this for useful command flowcharts for said initialization.

[6] HD44780 Microcontroller Datasheet.
https://cdn-shop.adafruit.com/datasheets/HD44780.pdf
Referenced for finding commands, studying functionalities, and timing delays.