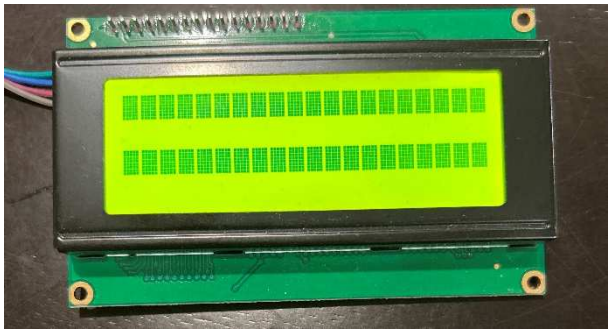
4x20 LCD Display with I2C I/O Expander Guide

Introduction:

This Guide will walk through the steps needed to connect the 4x20 LCD display to the Beagle Bone Green using I2C. We will go over how to communicate with the I/O expander, and then look at how the I/O expander communicates with the LCD screen.

Hardware:

1 – 4x20 LCD Display (2004A – V20):



4 – Female to Female jumper cables[1]:

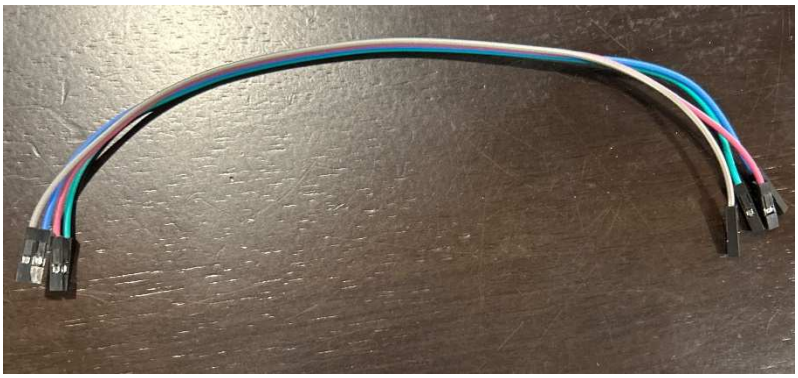


Table of Contents

1: You can also use 8 male to female jumper cables and connect them using the breadboard

1. Wiring

Since we are connecting to the screen via an I2C expander, we only need 4 jumpers. These 4 jumpers correspond to the channels needed for a basic I2C slave device to communicate with a master:

1) **GND**: The ground pin used to complete the circuit needed to power the LCD display and I/O expander.
2) **VCC**: The pin the supplies power to the LCD display and I/O Expander (3.3v)
3) **SDA**: The pin used to send data to the I/O expander.
4) **SCA**: The pin used to send clock pulses to the I/O expander.

****IMPORTANT****

It is recommended that the following steps are attempted when the beagle bone is powered off. This will help prevent potentially shorting the beagle bone when fidgeting with pins.

****

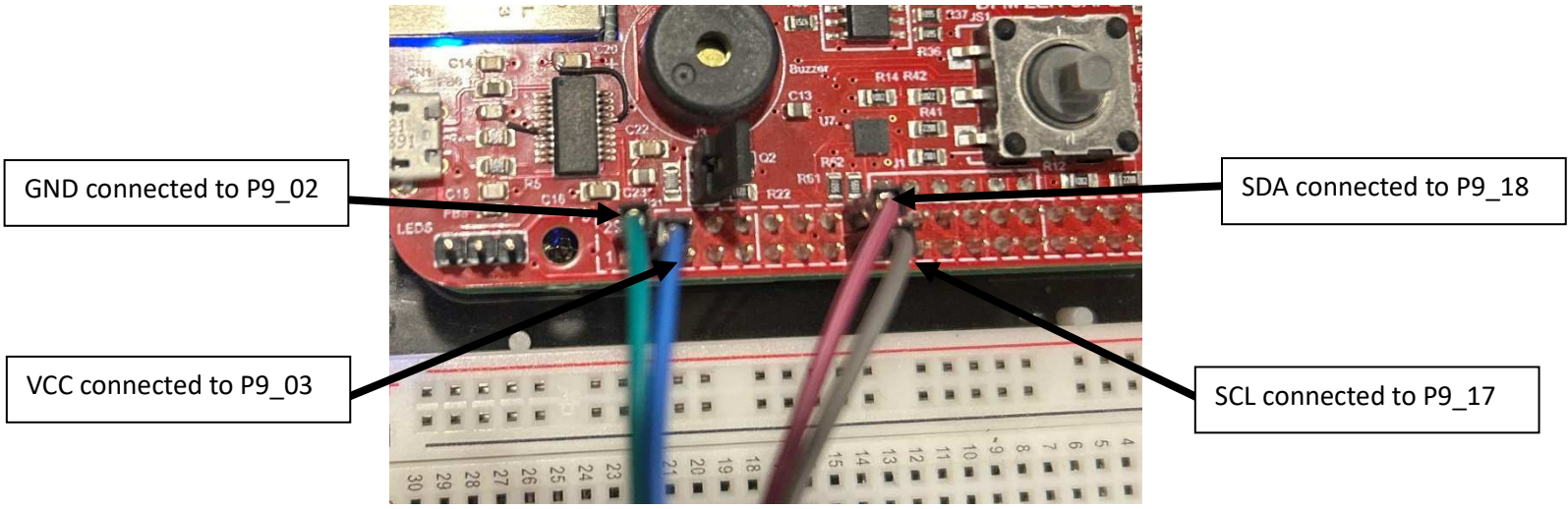
First connect the 4 female to female jumpers to the pins on the I/O Expander:

Next, we will connect these jumpers to P9 header pins on the zen cape attached to the beagle bone:

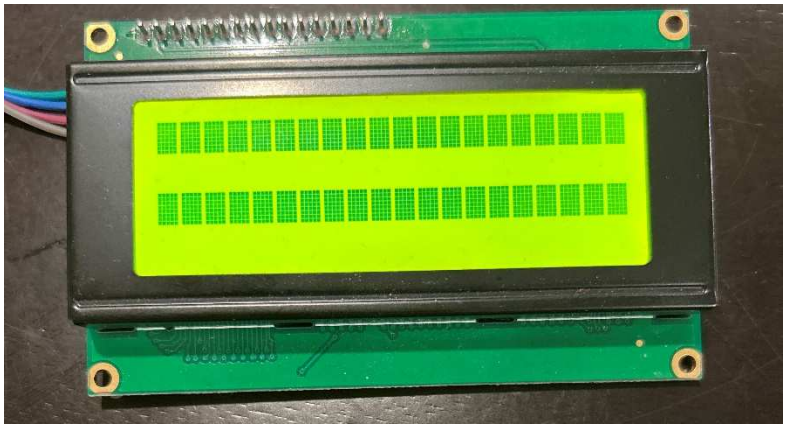
1) **GND**[2]: Connect this jumper to **P9_02**
2) **VCC**[2] : Connect this jumper to **P9_03**
3) **SDA**[3]**:** Connect this jumper to **P9_18**
4) **SCA**[3]**:** Connect this jumper to **P9_17**

2: The GND jumper can be connected to any pin marked as 'gnd' and the VCC jumper can be connected to any pin marked as '3.3v' in the header reference https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/guides/files/bbg_docs/BeagleboneBlackP9HeaderTable.pdf

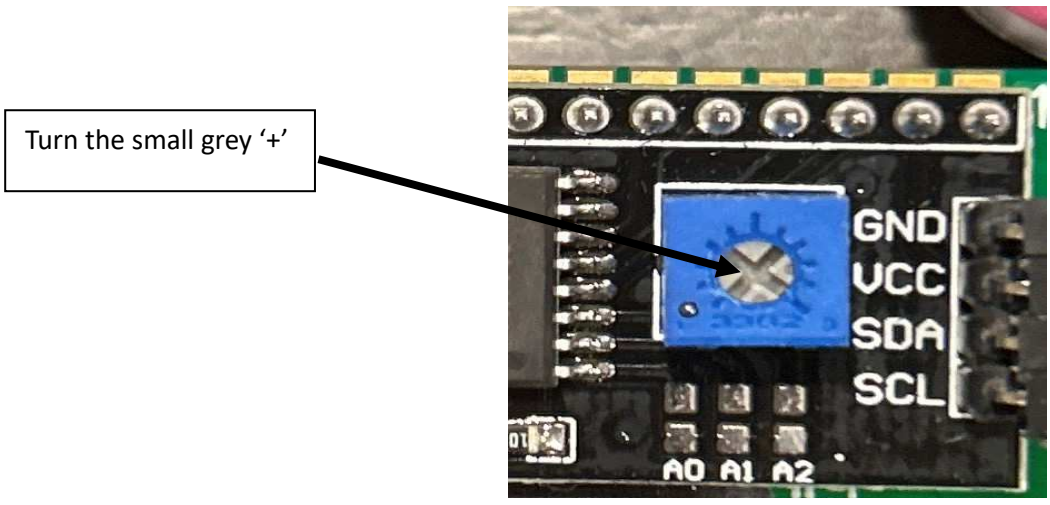
3: The SDA and SCL jumpers can be connected to either P9_18, P9_17 or P9_20, P9_19. This will then determine the I2C bus we will be working with. In this guide we will be using I2C bus 1 (P9_18, P9_17). More info regarding the I2C bus can be found at: https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/guides/files/I2CGuide.pdf



Once the jumpers are connected you should be able to turn on the beagle bone green and the display's backlight should turn on, indicating the gnd and vcc pins are connected properly:



NOTE: If the backlight is on but the black squares are not visible, this is due to the backlight being too bright. You can toggle the backlight level by turning a small potentiometer on the I/O expander. It is recommended to use a small Philips screwdriver:



2. I2C Communication with I/O Expander

Now that the LCD display is properly connected to the beagle bone, we can interface with it. We have connected the display via I2C on pins P9_18 and P9_17, and to use the pins for I2C we need to first configure them with the following commands:

$(bbg) config-pin P9_17 i2c
$(bbg) config-pin P9_18 i2c

Next, to confirm the screen is connected via I2C run:

$(bbg) i2cdetect -y -r 1

You should see the LCD screen at 0x27:

```
debian@nhannay-beagle:~$ i2cdetect -y -r 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- 18 -- -- -- -- -- -- --
20: 20 -- -- -- -- -- -- 27 -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Now that we have confirmed the LCD screen is connected to the beagle bone via I2C, we will go over how exactly we communicate with the I/O expander.

We will communicate with the I/O expander by sending 1 byte at a time. Each byte XXXXYYYY has the following format:

XXXX – The data/instruction to be passed along to the LCD Display
YYYY – Toggles pins on the LCD display

The data/instruction bits depend on the data/instruction being sent; however, the toggle bits have specific functionality related to the LCD display:

$Y_3Y_2Y_1Y_0$: Y_3 – Unused
Y_2 – toggles E Pin
Y_1 – toggles R/W pin
Y_0 – toggles R/S pin

More specific information about the I/O expander can be found at:
https://www.mouser.ca/datasheet/2/302/PCF8574_PCF8574A-1127673.pdf

3. Communication with LCD Display via I/O expander

As we have seen, to communicate with the LCD screen via I2C, we must first communicate with the I/O expander. We have also seen that the lower 4 bits of the I2C message to the I/O expander are reserved for pin configurations, leaving us with only 4 bits to send data. Further, Looking at the data sheet for the LCD screen found at:
https://image.dfrobot.com/image/data/DFR0154/LCD2004%20hd44780%20Datasheet.pdf
We can see that the instructions consist of 8 bits:

		Code						
DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description
0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.
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.
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.
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).
0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.
0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).
0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.
1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.
BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.

Fig 1

This may seem like a problem, as we need to send 8 bit instructions using only 4 bits. However, the LCD screen has a built in mode for this, namely “4 bit mode”. This allows us to send some 8 bit data by splitting it up into two nibbles and sending them in two separate I2C messages, with the high order nibble being sent first:

For example, say we want to send the letter ‘H’ to the screen (ASCII = 01001000) ignoring the lower 4 bits related to pin toggling:

> We would first send ‘0100XXXX’ (where XXXX represents lower 4 pin toggling bits)
> And then ‘1000XXXX’ (where XXXX represents lower 4 pin toggling bits)

Further, the LCD screen operates on specific timing. This means that certain pins must be toggled at certain times in order for data to be read into the LCD's internal registers. In 4 bit mode the timing is:

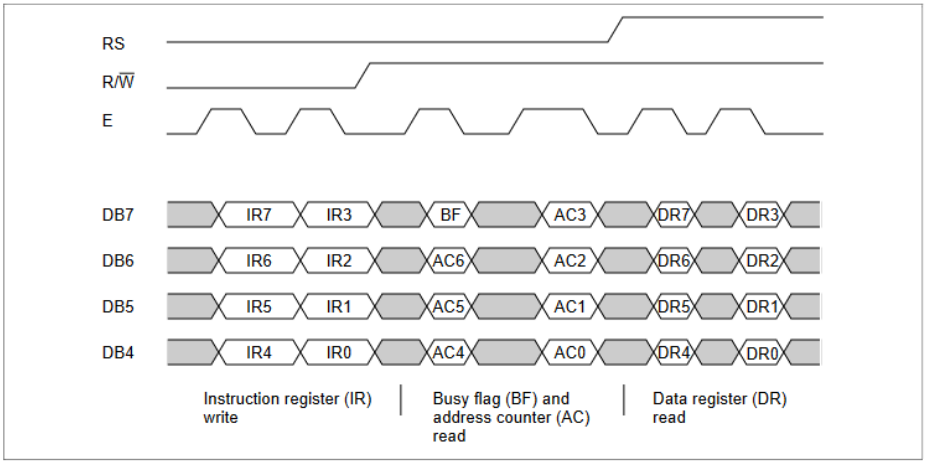


Figure 9 4-Bit Transfer Example

Now looking at exactly what RS, R/W and E represent:

RS	1	I	MPU	Selects registers. 0: Instruction register (for write) Busy flag: address counter (for read) 1: Data register (for write and read)
R/W	1	I	MPU	Selects read or write. 0: Write 1: Read
E	1	I	MPU	Starts data read/write.

We can see exactly how to clear the display and set the cursor to the first line (Fig 1)(00000001):

We first send the higher nibble (0000), set the E pin high (1), set R/W low (0) since we are writing, and set RS low (0) since this is an instruction.

00000100

Then we send a pulse low, signaling the LCD screen to read the 4 high order bits into internal registers:

00000000

Then we send the low order nibble along with a high pulse:

00010100

And finally, we send a low pulse to signal the LCD to read the 4 high order bits into internal registers (since we have enabled 4 bit mode, executing these steps back to back will result in the

LCD accepting the first data nibble as the high order bits and the second data nibble as being the low order bits)

00000000

This is the general format for displaying characters on the screen or executing instructions:

Find the ASCII code for the desired char or look to the LCD manual for the desired instruction. Once found, send the high order bits XXXX along with 010Z, where Z is 1 if it is an instruction, or 0 if it is char data. Followed by a low pulse (00000000), then the low order nibble (XXXX010Z) followed by a low pulse (00000000).

4. Sample C Code

Below is some code related to initializing the LCD display and displaying any ASCII char

```
/* ------------------------------------------------------------------ *
 * Initialize the display, using the 4-bit mode initialization sequence *
 * ------------------------------------------------------------------ */
I2C_sendByte(0b00110100);
I2C_sendByte(0b00110000);
Sleep_ns(0 , 4100000);     // wait 4.1msec
I2C_sendByte(0b00110100);
I2C_sendByte(0b00110000);
Sleep_ns(0 , 100000);      // wait 100usec
I2C_sendByte(0b00110100); //
I2C_sendByte(0b00110000);
Sleep_ns(0 , 4100000);     // wait 4.1msec
I2C_sendByte(0b00100100);
I2C_sendByte(0b00100000); // 4 bit mode

/* ------------------------------------------------------------------ *
 * Set 4-bt, 2 Line, 5x8 char mode                                    *
 * ------------------------------------------------------------------ */
Sleep_ns(0, 40000);       // wait 40usec
I2C_sendByte(0b00100100);
I2C_sendByte(0b00100000); // keep 4-bit mode
I2C_sendByte(0b10000100);
I2C_sendByte(0b10000000); // D3=2 lines, D2=char5x8
```

After the screen has been initialized, we can then send an ASCII letter using the function:

```
void LCD_writeChar(unsigned char character)
{
    unsigned char full = 0x00;
    unsigned char high = character & 0xF0;
    unsigned char low = (character & 0x0F) << 4;
    full = full | high;
    full = full | 0x05;

    I2C_sendByte(full);
    //printf("sent: %0x\n", full);
    I2C_sendByte(0b00000000);

    full = 0x00;
    full = full | low;
    full = full | 0x05;

    I2C_sendByte(full);
    I2C_sendByte(0b00000000);
}
```

This function takes the ASCII value of the character and sends the high data bits XXXX along with the required pin configuration to write data (0x05) 0101 (high E pulse, write, data), followed by the low data bits.