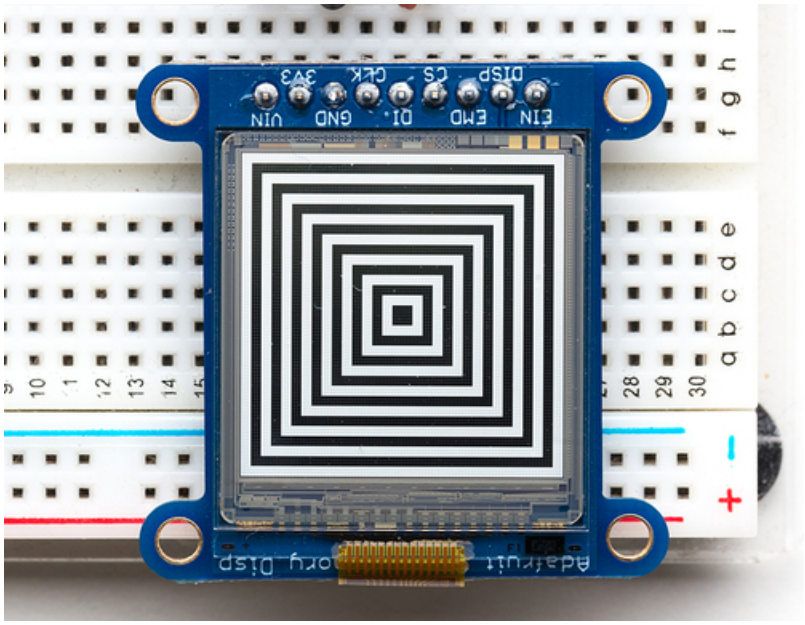
adafruit learning system

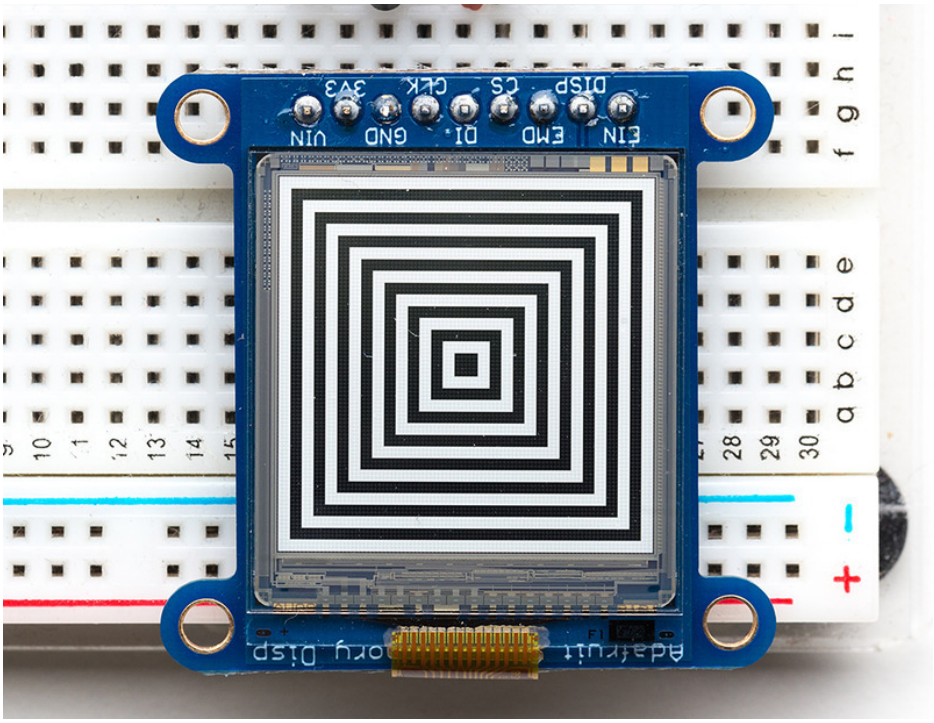
Adafruit Sharp Memory Display Breakout

Created by Bill Earl



Last updated on 2020-05-18 09:41:20 PM EDT

Overview



The 1.3" SHARP Memory LCD display is a cross between an eInk (e-paper) display and an LCD. It has the ultra-low power usage of eInk and the fast-refresh rates of an LCD. This model has a matt silver background, and pixels show up as little mirrors for a silver-reflective display, a really beautiful and unique look. It does not have a backlight, but it is daylight readable. For dark/night reading you may need to illuminate the LCD area with external LEDs.

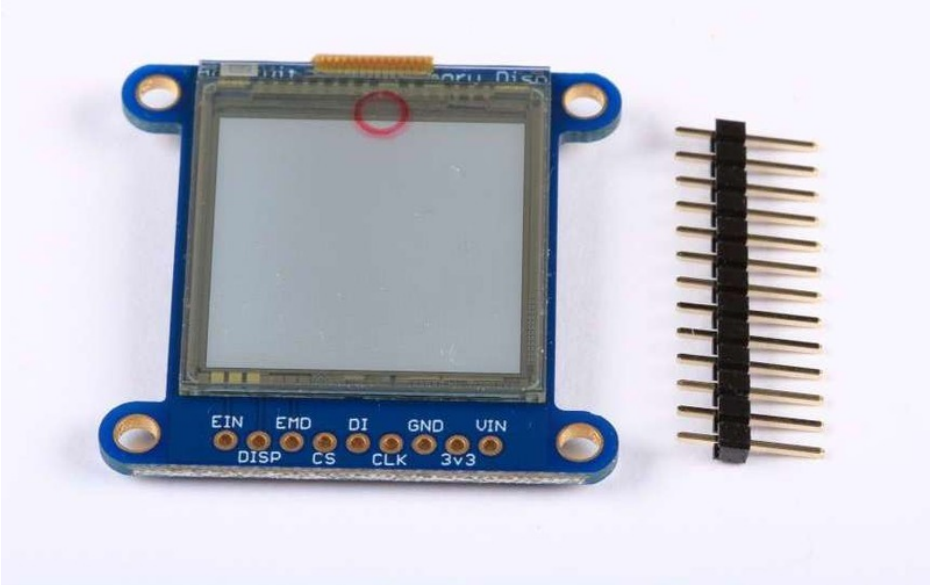
The display is 3V powered and 3V logic, so we placed it on a fully assembled & tested breakout board with a 3V regulator and level shifting circuitry. The display slots into a ZIF socket on board and we use a piece of double-sided tape to adhere it onto one side. There are four mounting holes so you can easily attach it to a box.

The display is 'write only' which means that it only needs 3 pins to send data. The downside of a write-only display is that the entire memory must be buffered by the microcontroller driver.

If you have one of the older 96x96 pixel versions, then 96x96 bits = 1,152 bytes. On an Arduino Uno/Leonardo that's half the RAM available and so it might not be possible to run this display with other RAM-heavy libraries like SD interfacing.

If you have one of the newer 168x144 pixel versions, then 168x144 bits = 3,024 bytes. **That won't fit on an Arduino Uno or Leonardo!** You must use a chip with more RAM like a Metro or Feather M0 or ESP8266.

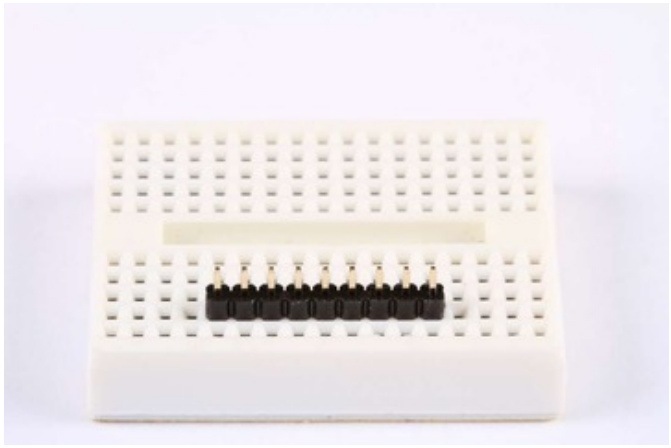
The Sharp Memory Display breakout board ships with optional headers for use in a breadboard.


EIN
EMD
DI
GND
VIN
DISP
CS
CLK
3v3

Assembly and Wiring

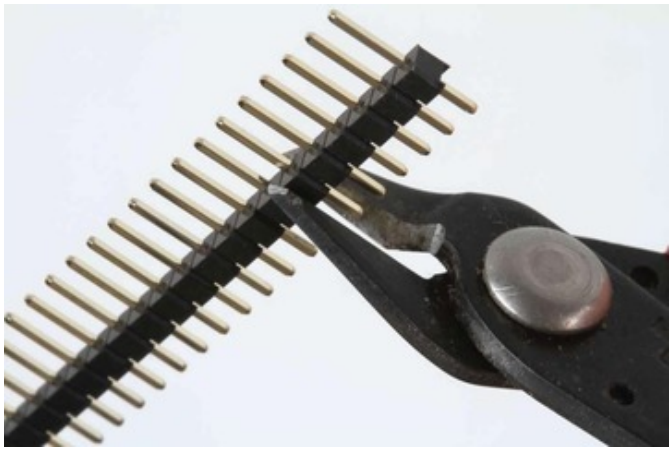
The display and support circuitry come pre-assembled and fully tested on a handy breakout board. For use in a breadboard, you will want to install the included 0.1" header strip:

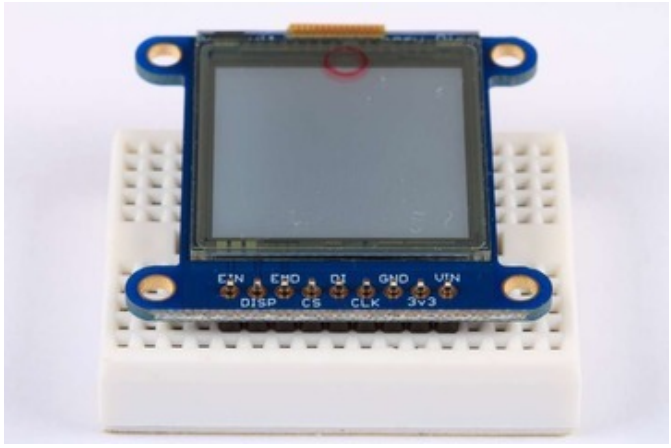
Installing the Header:



Position the header

Trim the header to length if necessary and place it long pins down in your breadboard.

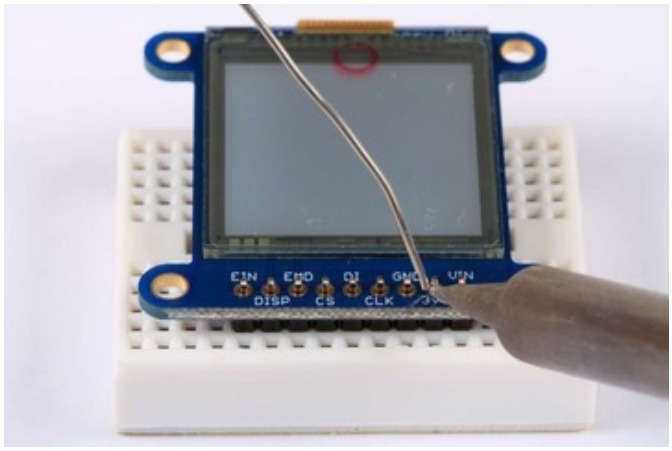




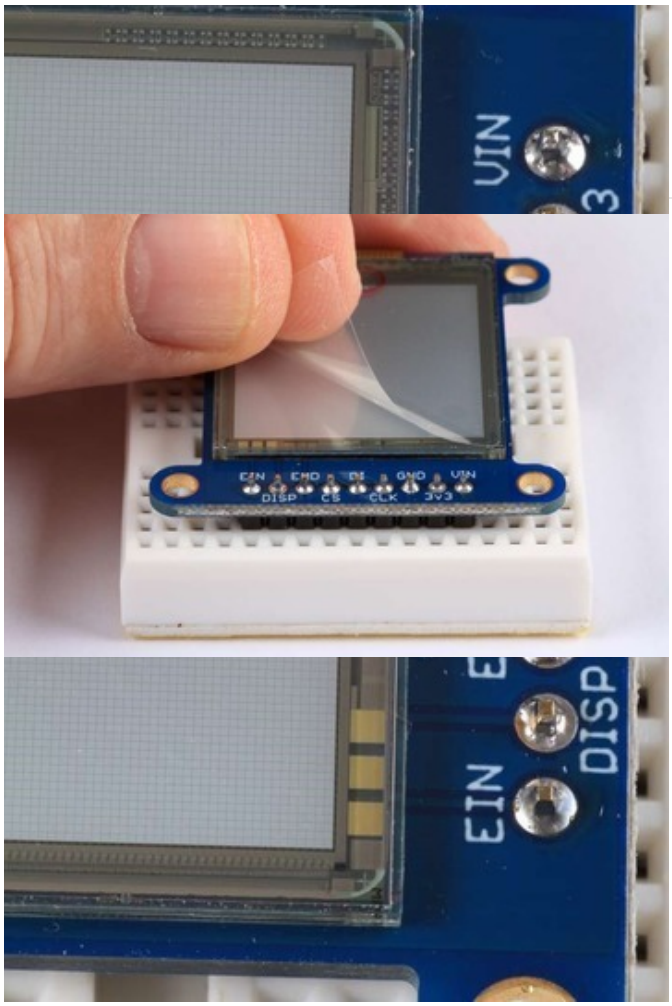
Position the display

Place the Sharp Memory Display over the pins on the breadboard.

Solder!



Solder each pin to assure good electrical conductivity.



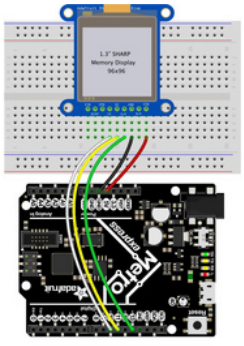
Remove the Protective Film

After soldering is complete. Gently peel the film from the display.

For the 144x168 Sharp Memory Display, you will need a microprocessor with more memory than the Uno such as the Arduino Mega, Metro M0 or Metro M4

Wiring to the Arduino:

This display can be driven with only 3 pins. Any pins can be used. The wiring we show here uses pins 10, 11 and 13 for compatibility with the library example code.



- **Microcontroller GND** to **LCD Gnd**
- **Microcontroller 5V** to **LCD Vin**
- **Microcontroller D13** to **LCD Clk**
- **Microcontroller D11** to **LCD DI**
- **Microcontroller D10** to **LCD CS**

https://adafru.it/GBd

https://adafru.it/GBd

The other wires are optional, and connect directly to the Memory Display for more advanced uses. Check the raw display datasheet (in the downloads area) for details.

Programming

Download the Libraries

To use the Sharp Memory Display with your Arduino, you will need to download and install 2 libraries:

- Sharp Memory Display Library (https://adafru.it/cgJ)
- Adafruit GFX Library (https://adafru.it/aJa)
- Adafruit BusIO Library (https://adafru.it/GxD)

For details on how to install libraries, see this guide: All About Arduino Libraries (https://adafru.it/aYM).

Run the Example Code

Once your libraries are installed, open the Arduino IDE and select:

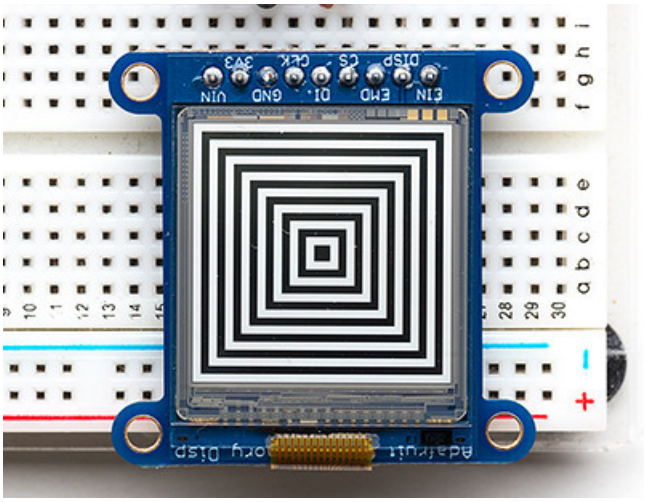
File->Examples->Adafruit_SHARP_Memory_Display->sharpmemtest

Upload the example code to your Arduino and you should see the test graphics drawn on the screen.

Programming GFX Graphics

The Sharp Memory Display is part of the growing family of Adafruit graphical displays that use the Adafruit GFX Library. This library lets you use a common set of graphical drawing functions on a whole variety of displays including.LED matrices, OLEDs, TFT LCDs, eInk and the Sharp Memory Display!

For more details about programming with GFX, see our Adafruit GFX Graphics Library Guide (https://adafru.it/aPx).



CircuitPython Wiring

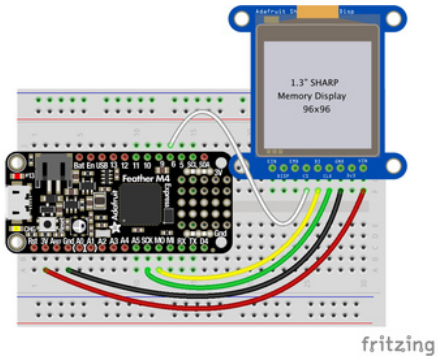
It's easy to use the Sharp Memory Display with CircuitPython and the Adafruit CircuitPython SharpMemoryDisplay (https://adafru.it/GBn) module. This module allows you to easily write Python code to control the display.

You can use this display with any CircuitPython microcontroller board or with a computer that has GPIO and Python thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library (https://adafru.it/BSN).

We'll cover how to wire the display to your CircuitPython microcontroller board. First assemble your Sharp Display.

We're going to show you how to wire it up to a Feather M4 Express. Because the logic levels on a Feather M4 Express are already 3.3v, you don't need a logic level shifter.

Connect the display to your microcontroller board as shown below.



- **Microcontroller GND** to **LCD Gnd**
- **Microcontroller 3V** to **LCD Vin**
- **Microcontroller SCK** to **LCD Clk**
- **Microcontroller MOSI** to **LCD DI**
- **Microcontroller D6** to **LCD CS**

https://adafru.it/GBo

https://adafru.it/GBo

CircuitPython Setup

CircuitPython Installation of SharpMemoryDisplay Library

To use the Sharp Memory Display with your Adafruit CircuitPython board you'll need to install the Adafruit CircuitPython SharpMemoryDisplay (https://adafru.it/GBn) module on your board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/uap). Our CircuitPython starter guide has a great page on how to install the library bundle (https://adafru.it/ABU).

If you choose, you can manually install the libraries individually on your board:

- **adafruit_sharpmemorydisplay**
- **adafruit_framebuf**

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_sharpmemorydisplay.mpy** and **adafruit_framebuf.mpy** files copied over.

Next connect to the board's serial REPL (https://adafru.it/Awz) so you are at the CircuitPython >>> prompt.

CircuitPython Usage

It's easy to use the Sharp Memory Display with CircuitPython and the Adafruit CircuitPython SharpMemoryDisplay (https://adafru.it/GBn) module. This module allows you to easily write Python code to control the display.

You can use this display with any CircuitPython microcontroller board.

To demonstrate the usage, we'll initialize the library and use Python code to control the LCD from the board's Python REPL.

Initialization

First need to initialize the SPI bus. To do that, run the following commands:

```
import board
import busio
import digitalio
import adafruit_sharpmemorydisplay

spi = busio.SPI(board.SCK, MOSI=board.MOSI)
scs = digitalio.DigitalInOut(board.D6) # inverted chip select

display = adafruit_sharpmemorydisplay.SharpMemoryDisplay(spi, scs, 144, 168)
```

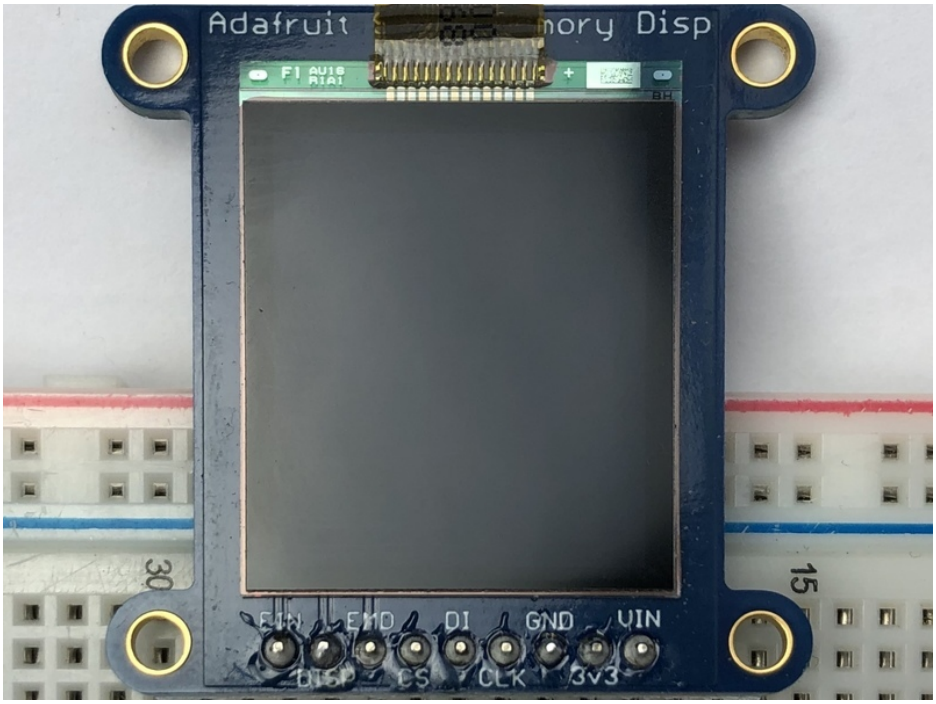
The last three parameters to the initializer are the pins connected to the display's **CS** line, **width** and **height** in that order. Again make sure to use the right pin names as you have wired up to your board!

Drawing

The SharpMemoryDisplay module currently supports a basic set of commands to draw on the display. You can set individual pixels, fill the screen, and write lines of text.

To fill or clear the entire screen use the `fill` function. This function takes a parameter which specifies the color to fill with, either `0` for black or `1` for white. For example to fill the screen black:

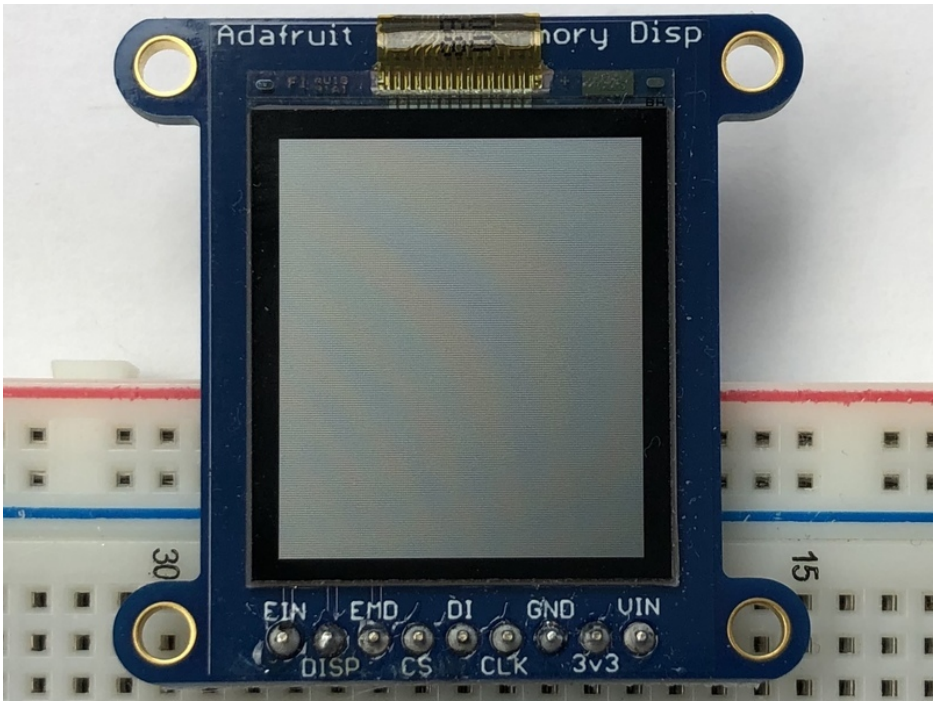
```
display.fill(0)
display.show()
```



Notice the `fill` function doesn't actually change the display. You must call `show` after making drawing commands to send the updated pixel data to the display!

To clear the screen to white just call `fill` again but with the color `1`:

```
display.fill(1)
display.show()
```

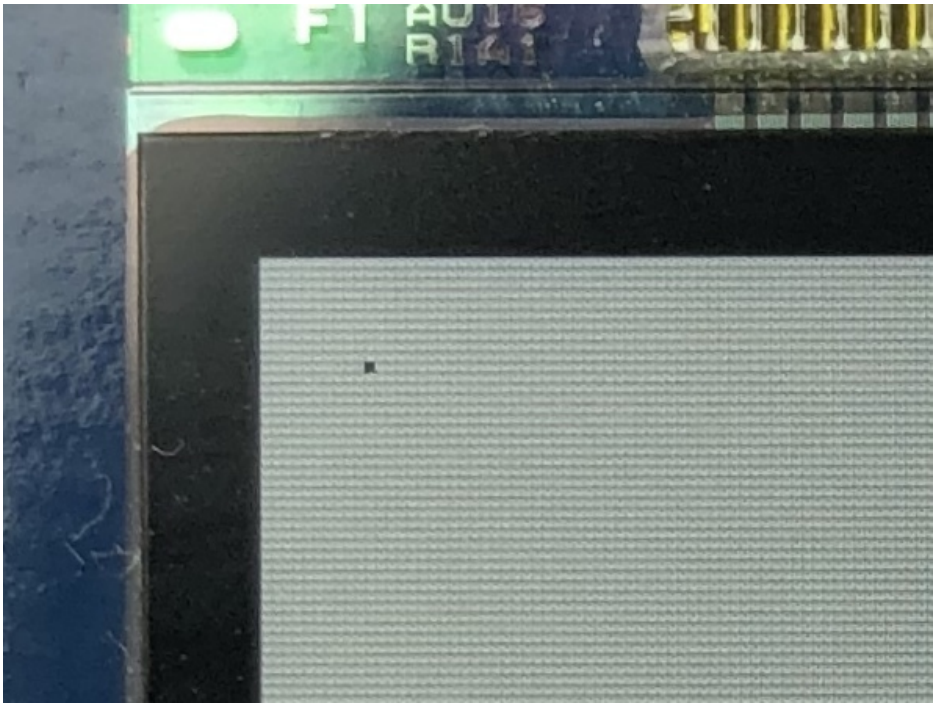


To set a pixel use the `pixel` function. This function takes the following parameters:

- **Pixel X position**
- **Pixel Y position**
- **Pixel color** (`0` = black, `1` = white)

For example to set the pixel at `(10, 10)` black:

```
display.pixel(10, 10, 0)
display.show()
```



Try setting other pixels white by changing the X and Y position. Remember you have to call `show` (https://adafru.it/Dvz) after setting pixels to see them appear!

Text

To write text to your display, you must download a font file and copy it to your CIRCUITPY drive. Click the button below to download the file, and then copy **font5x8.bin** to your **CIRCUITPY** drive.

https://adafru.it/DvA

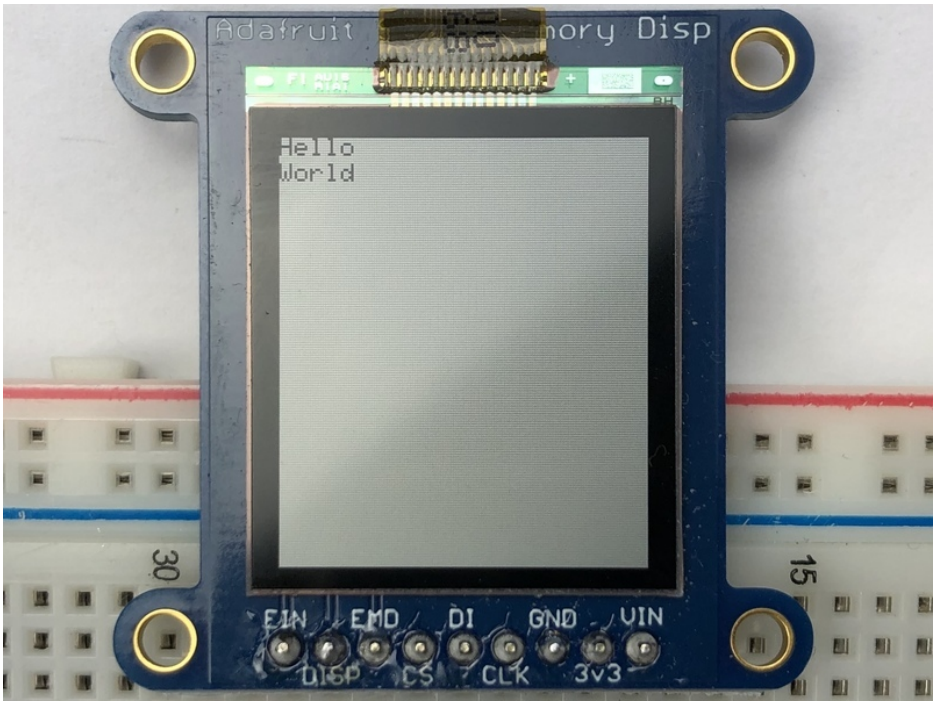
https://adafru.it/DvA

You can write a line of text with the `text` function. This function takes the following parameters:

- **String of text**
- **Text X position**
- **Text Y position**
- **Text color** (`0` = black, `1` = white)

For example to clear the screen and then write two lines of text:

```
display.fill(1)
display.text('Hello', 0, 0, 0)
display.text('World', 0, 10, 0)
display.show()
```



Notice the second line of text starts at Y position `10`, this moves it down the display 10 pixels so it's below the first line of text. The font used by the text function is 8 pixels tall so a size of `10` gives a bit of room between the lines.

That's all there is to drawing on the Sharp Memory Display with CircuitPython! The drawing functions are basic but provide building blocks for more advanced usage. For example you can display text with sensor readings or other state, or even program a simple game like pong!

Full Example Code

```
import time
import board
import busio
import digitalio

import adafruit_sharpmemorydisplay

# Initialize SPI bus and control pins
spi = busio.SPI(board.SCK, MOSI=board.MOSI)
scs = digitalio.DigitalInOut(board.D6)  # inverted chip select

# pass in the display size, width and height, as well
# display = adafruit_sharpmemorydisplay.SharpMemoryDisplay(spi, scs, 96, 96)
display = adafruit_sharpmemorydisplay.SharpMemoryDisplay(spi, scs, 144, 168)

print("Pixel test")

# Clear the display.  Always call show after changing pixels to make the display
```

```
# update visible!
display.fill(1)
display.show()

# Set a pixel in the origin 0,0 position.
display.pixel(0, 0, 0)
# Set a pixel in the middle position.
display.pixel(display.width // 2, display.width // 2, 0)
# Set a pixel in the opposite corner position.
display.pixel(display.width - 1, display.height - 1, 0)
display.show()
time.sleep(2)

print("Lines test")
# we'll draw from corner to corner, lets define all the pair coordinates here
corners = (
    (0, 0),
    (0, display.height - 1),
    (display.width - 1, 0),
    (display.width - 1, display.height - 1),
)

display.fill(1)
for corner_from in corners:
    for corner_to in corners:
        display.line(corner_from[0], corner_from[1], corner_to[0], corner_to[1], 0)
display.show()
time.sleep(2)

print("Rectangle test")
display.fill(1)
w_delta = display.width / 10
h_delta = display.height / 10
for i in range(11):
    display.rect(0, 0, int(w_delta * i), int(h_delta * i), 0)
display.show()
time.sleep(2)

print("Text test")
display.fill(1)
display.text(" hello world!", 0, 0, 0)
display.text(" This is the", 0, 8, 0)
display.text(" CircuitPython", 0, 16, 0)
display.text("adafruit library", 0, 24, 0)
display.text(" for the SHARP", 0, 32, 0)
display.text(" Memory Display :) ", 0, 40, 0)
display.show()
```

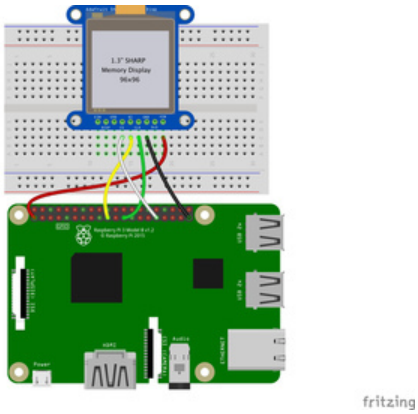
Python Wiring

It's easy to use the Sharp Memory Display with Python and the Adafruit CircuitPython SharpMemoryDisplay (https://adafru.it/GBn) module. This module allows you to easily write Python code to control the display.

We'll cover how to wire the display to your Raspberry Pi. First assemble your Sharp Display.

Since there's *dozens* of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported (https://adafru.it/BSN).

Connect the display as shown below to your Raspberry Pi.



- **Raspberry Pi GND** to **LCD Gnd**
- **Raspberry Pi 3.3V** to **LCD Vin**
- **Raspberry Pi SCK (GPIO 11)** to **LCD Clk**
- **Raspberry Pi MOSI (GPIO 10)** to **LCD DI**
- **Raspberry Pi GPIO 6** to **LCD CS**

https://adafru.it/GBp

https://adafru.it/GBp

Python Setup

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling SPI on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready (https://adafru.it/BSN)!

Python Installation of SharpMemoryDisplay Library

Once that's done, from your command line run the following command:

- `pip3 install adafruit-circuitpython-sharpmemorydisplay`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

If that complains about pip3 not being installed, then run this first to install it:

- `sudo apt-get install python3-pip`

DejaVu TTF Font

Raspberry Pi usually comes with the DejaVu font already installed, but in case it didn't, you can run the following to install it:

- `sudo apt-get install ttf-dejavu`

Pillow Library

We also need PIL, the Python Imaging Library, to allow using text with custom fonts. There are several system libraries that PIL relies on, so installing via a package manager is the easiest way to bring in everything:

- `sudo apt-get install python3-pil`

That's it. You should be ready to go.

Python Usage

It's easy to use the Sharp Memory Display with CircuitPython and the Adafruit CircuitPython SharpMemoryDisplay (https://adafru.it/GBn) module. This module allows you to easily write Python code to control the display.

You can use this display with a computer that has GPIO and Python thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library (https://adafru.it/BSN).

To demonstrate the usage, we'll initialize the library and use Python code to control the display from the board's Python REPL.

Since we are running full CPython on our Linux/computer, we can take advantage of the powerful Pillow image drawing library to handle text, shapes, graphics, etc. Pillow is a gold standard in image and graphics handling, you can read about all it can do here (https://adafru.it/FU7).

Initialization

First need to initialize the SPI bus. To do that, run the following commands:

```
import board
import busio
import digitalio
import adafruit_sharpmemorydisplay

spi = busio.SPI(board.SCK, MOSI=board.MOSI)
scs = digitalio.DigitalInOut(board.D6) # inverted chip select

display = adafruit_sharpmemorydisplay.SharpMemoryDisplay(spi, scs, 144, 168)
```

The last three parameters to the initializer are the pins connected to the display's **CS** line, **width** and **height** in that order. Again make sure to use the right pin names as you have wired up to your board!

Example Code

```
"""
This demo will fill the screen with white, draw a black box on top
and then print Hello World! in the center of the display

This example is for use on (Linux) computers that are using CPython with
Adafruit Blinka to support CircuitPython libraries. CircuitPython does
not support PIL/pillow (python imaging library)!
"""

import board
import busio
import digitalio
from PIL import Image, ImageDraw, ImageFont
import adafruit_sharpmemorydisplay

# Colors
BLACK = 0
WHITE = 255
```

```
# Parameters to Change
BORDER = 5
FONTSIZE = 10

spi = busio.SPI(board.SCK, MOSI=board.MOSI)
scs = digitalio.DigitalInOut(board.D6)  # inverted chip select

# display = adafruit_sharpmemorydisplay.SharpMemoryDisplay(spi, scs, 96, 96)
display = adafruit_sharpmemorydisplay.SharpMemoryDisplay(spi, scs, 144, 168)

# Clear display.
display.fill(1)
display.show()

# Create blank image for drawing.
# Make sure to create image with mode '1' for 1-bit color.
image = Image.new("1", (display.width, display.height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)

# Draw a black background
draw.rectangle((0, 0, display.width, display.height), outline=BLACK, fill=BLACK)

# Draw a smaller inner rectangle
draw.rectangle(
    (BORDER, BORDER, display.width - BORDER - 1, display.height - BORDER - 1),
    outline=WHITE,
    fill=WHITE,
)

# Load a TTF font.
font = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf", FONTSIZE)

# Draw Some Text
text = "Hello World!"
(font_width, font_height) = font.getsize(text)
draw.text(
    (display.width // 2 - font_width // 2, display.height // 2 - font_height // 2),
    text,
    font=font,
    fill=BLACK,
)

# Display image
display.image(image)
display.show()
```

Let's take a look at the sections of code one by one. We start by importing the `board` so that we can access the pin definitions, `busio` so we can initialize SPI, `digitalio`, several `PIL` modules for Image Drawing, and the `adafruit_sharpmemorydisplay` driver.

```
import board
import busio
import digitalio
from PIL import Image, ImageDraw, ImageFont
import adafruit_sharpmemorydisplay
```

To make it easy to keep track of which numbers represent which colors, we define some colors near the top.

```
# Colors
BLACK = 0
WHITE = 255
```

In order to make it easy to change display sizes, we'll define a few variables in one spot here. We have the border size and font size, which we will explain a little further below.

```
BORDER = 5
FONTSIZE = 10
```

Next we set the SPI object to the board's SPI with `busio.SPI()`. We also define some Pins that will be used for the display and initialize the display. See the initialization section above for more details. By default, the initializer for the 144x168 display is uncommented because that's what we currently have in the store. If you had the 96x96 pixel version of the screen, you could use the other initializer instead.

```
spi = busio.SPI(board.SCK, MOSI=board.MOSI)
dc = digitalio.DigitalInOut(board.D6) # data/command
cs = digitalio.DigitalInOut(board.CE0) # Chip select
reset = digitalio.DigitalInOut(board.D5) # reset

#display = adafruit_sharpmemorydisplay.SharpMemoryDisplay(spi, scs, 96, 96)
display = adafruit_sharpmemorydisplay.SharpMemoryDisplay(spi, scs, 144, 168)
```

Next we clear the display in case it was initialized with any random artifact data.

```
# Clear display.
display.fill(0)
display.show()
```

Next, we need to initialize PIL to create a blank image to draw on. Think of it as a virtual canvas. Since this is a monochrome display, we set it up for 1-bit color, meaning a pixel is either white or black. We can make use of the display's width and height properties as well.

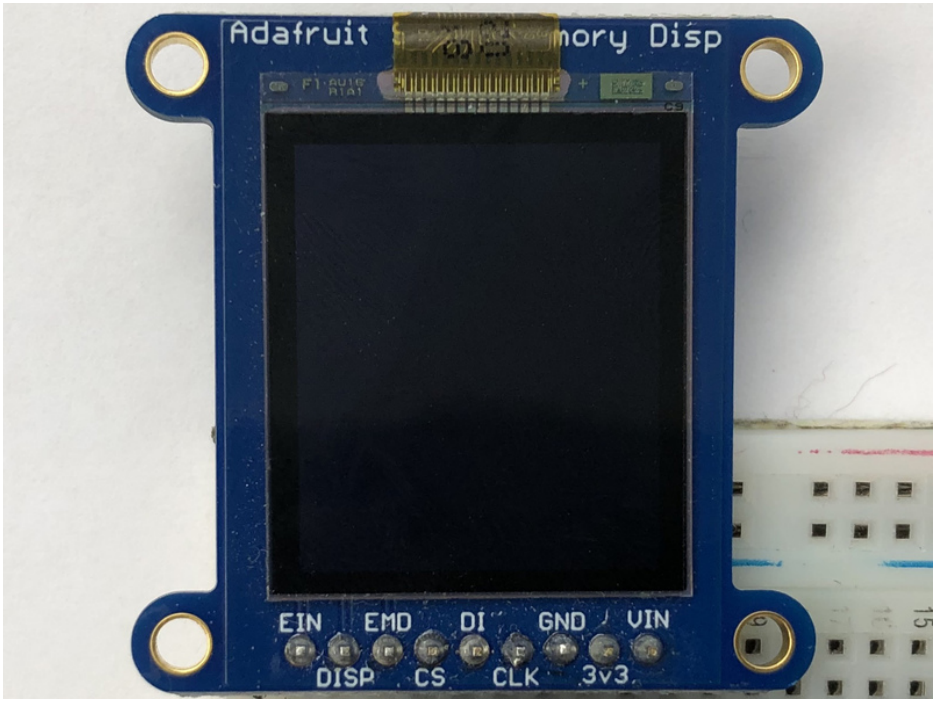
```
# Create blank image for drawing.
# Make sure to create image with mode '1' for 1-bit color.
image = Image.new('1', (display.width, display.height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)
```

Now we start the actual drawing. Here we are telling it we want to draw a rectangle from `(0,0)`, which is the upper left,

to the full width and height of the display. We want it both filled in and having an outline of black, so we pass `BLACK` for both values.

```
# Draw a black background
draw.rectangle((0, 0, display.width, display.height), outline=BLACK, fill=BLACK)
```

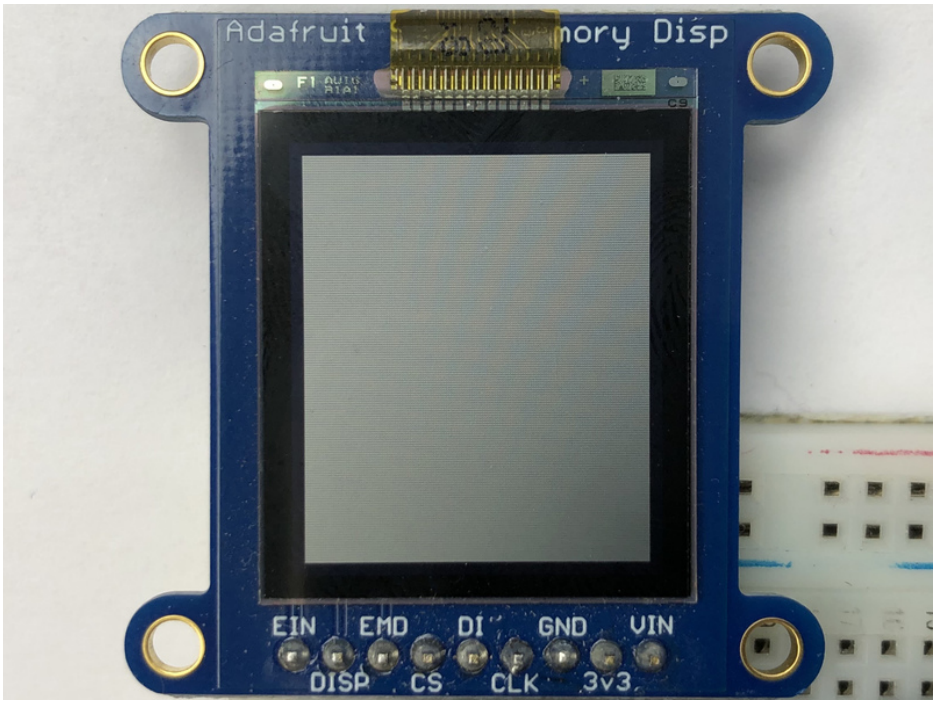
If we ran the code now, it would still show a blank display because we haven't told python to use our virtual canvas yet. You can skip to the end if you would like to see how to do that. This is what our canvas currently looks like in memory.



Next we will create a smaller white rectangle. The easiest way to do this is to draw another rectangle a little smaller than the full screen with no fill or outline and place it in a specific location. In this case, we will create a rectangle that is 5 pixels smaller on each side. This is where the **BORDER** variable comes into use. It makes calculating the size of the second rectangle much easier. We want the starting coordinate, which consists of the first two parameters, to be our BORDER value. Then for the next two parameters, which are our ending coordinates, we want to subtract our border value from the width and height. Also, because this is a zero-based coordinate system, we also need to subtract 1 from each number. Again, we set the `fill` and `outline` to `WHITE`.

```
# Draw a smaller inner rectangle
draw.rectangle((BORDER, BORDER, display.width - BORDER - 1, display.height - BORDER - 1),
               outline=WHITE, fill=WHITE)
```

Here's what our virtual canvas looks like in memory.



Now drawing text with PIL is pretty straightforward. First we start by setting the font to the default system text. After that we define our text and get the size of the text. We're grabbing the size that it would render at so that we can calculate the center position. Finally, we take the font size and screen size to calculate the position we want to draw the text at and it appears in the center of the screen.

```
# Load a TTF font.
font = ImageFont.truetype('/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf', FONTSIZE)

# Draw Some Text
text = "Hello World!"
(font_width, font_height) = font.getsize(text)
draw.text((display.width//2 - font_width//2, display.height//2 - font_height//2),
          text, font=font, fill=BLACK)
```

Finally, we need to display our virtual canvas to the display and we do that with 2 commands. First we set the image to the screen, then we tell it to show the image.

```
# Display image
display.image(image)
display.show()
```

Don't forget you MUST call display.image(image) and display.show() to actually display the graphics. The display takes a while to draw so cluster all your drawing functions into the buffer (fast) and then display them once to the display (slow)

Here's what the final output should look like.


Hello World!
EIN
EMD
DI
GND
VIN
DISP
CS
CLK
3v3

Downloads and Links

Libraries:

- Sharp Memory Display Library (https://adafru.it/cgJ)
- Adafruit GFX Library (https://adafru.it/aJa)

Files

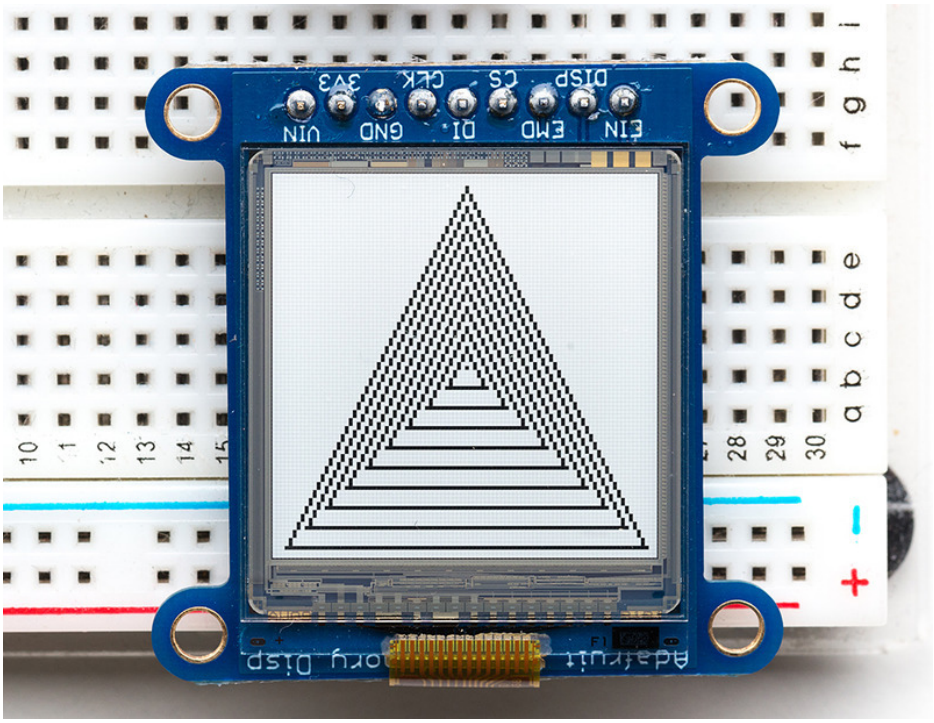
- Datasheet for the LS013B4DN04 LCD Module (https://adafru.it/cgK)
- Fritzing object in Adafruit Fritzing Library (https://adafru.it/aP3)
- EagleCAD PCB files on GitHub (https://adafru.it/rHE)

Library Reference

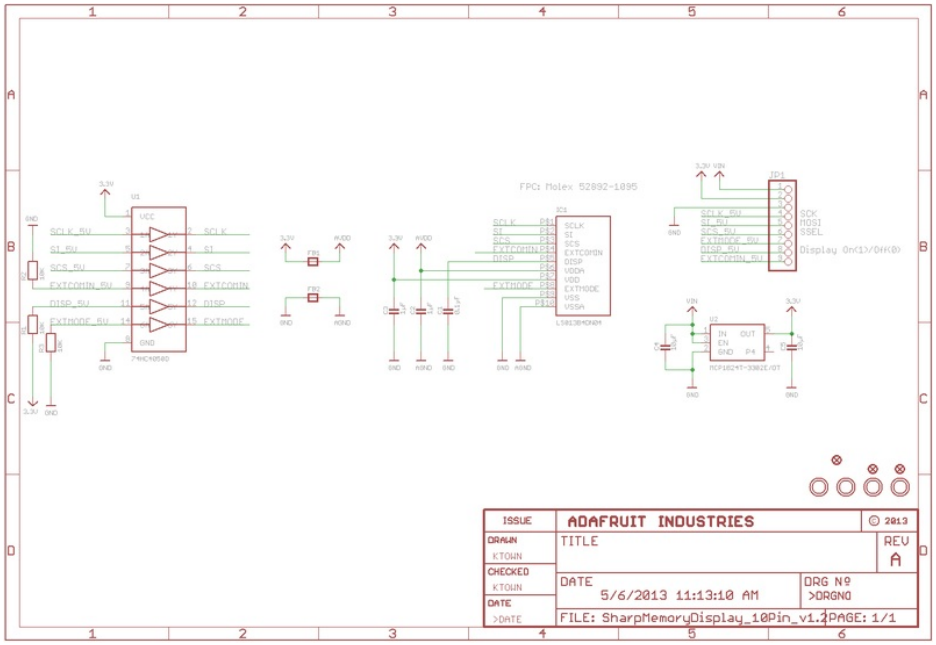
- Adafruit GFX Library (https://adafru.it/aPe)

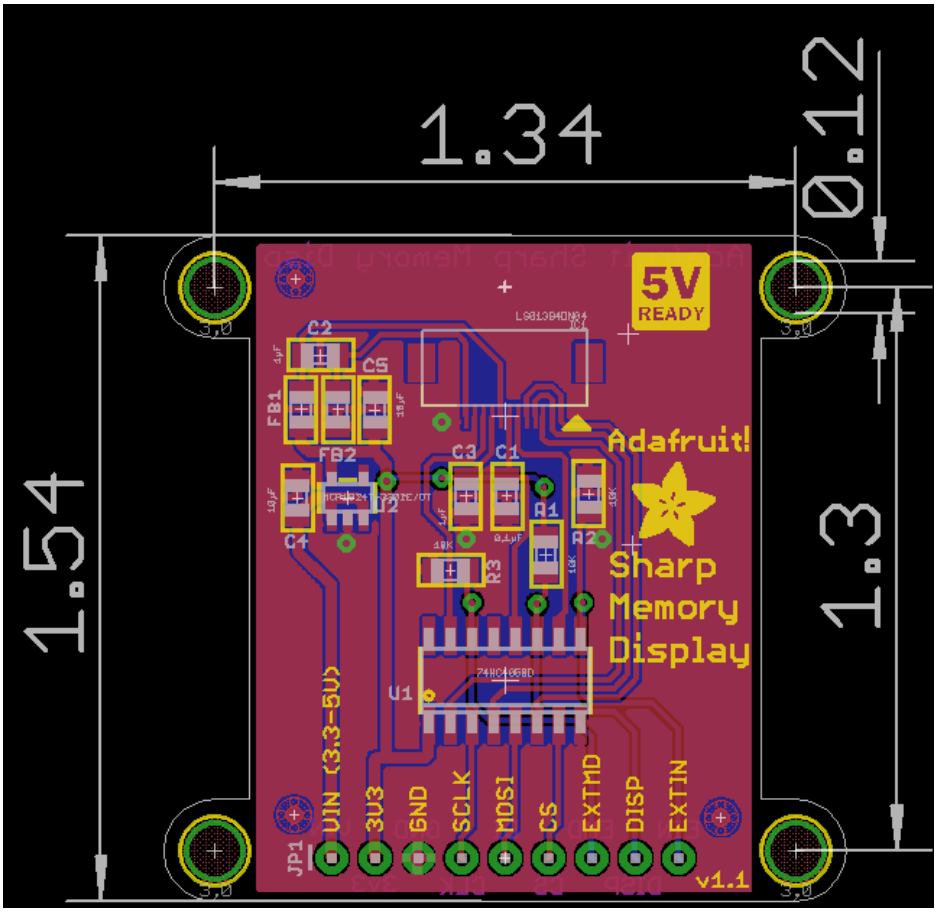
https://adafru.it/DwW

https://adafru.it/DwW



Schematic & Fabrication Print


ADAFRUIT INDUSTRIES
TITLE
DATE
5/6/2013 11:13:10 AM
FILE: SharpMemoryDisplay_10Pin_v1.2
PAGE: 1/1


1.34
0.12
1.54
1.3
5V
READY
Adafruit!
Sharp
Memory
Display
VIN (3.3-5V)
3V3
GND
SCLK
MOSI
CS
EXTMD
DISP
EXTIN
JP1
v1.1