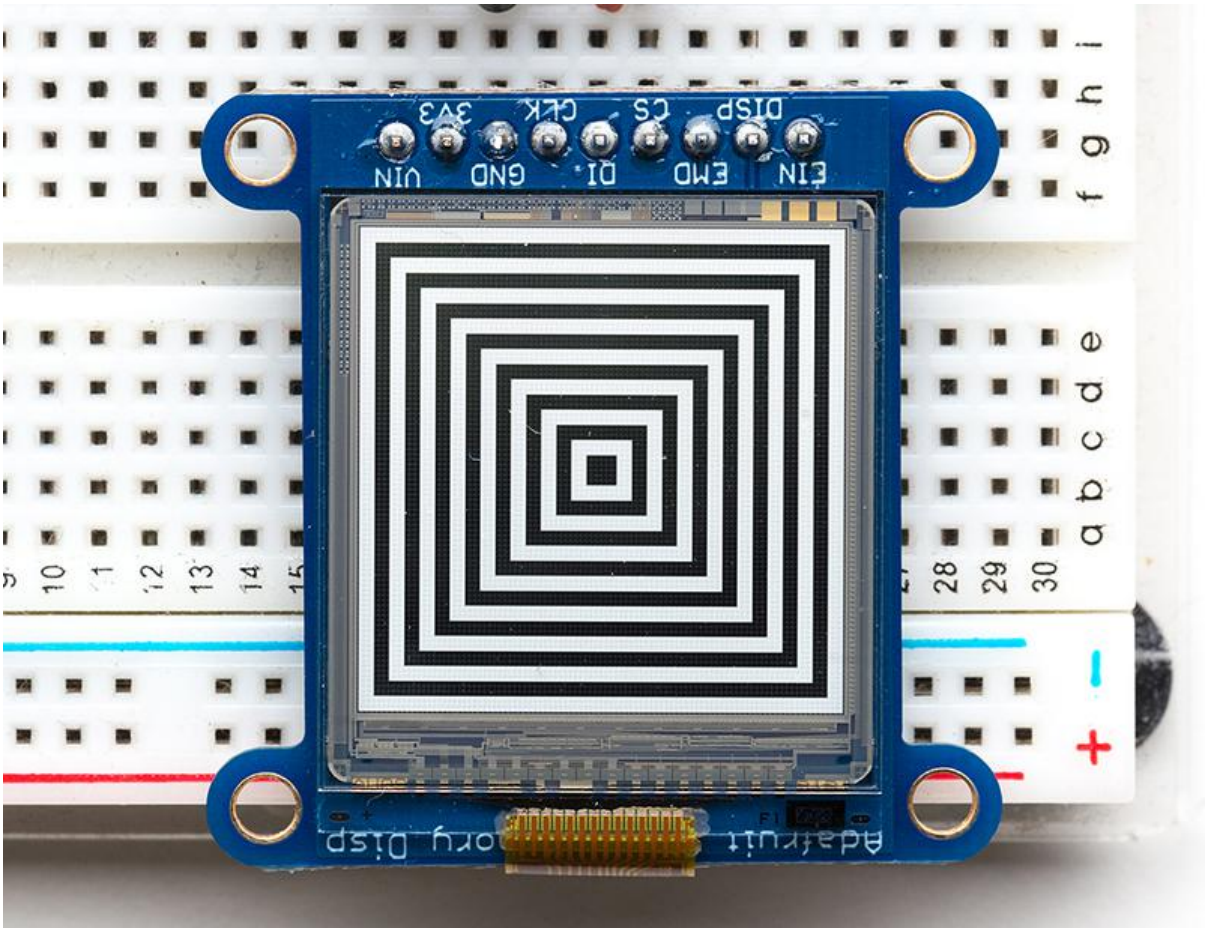
Adafruit Sharp Memory Display Breakout

Created by Bill Earl



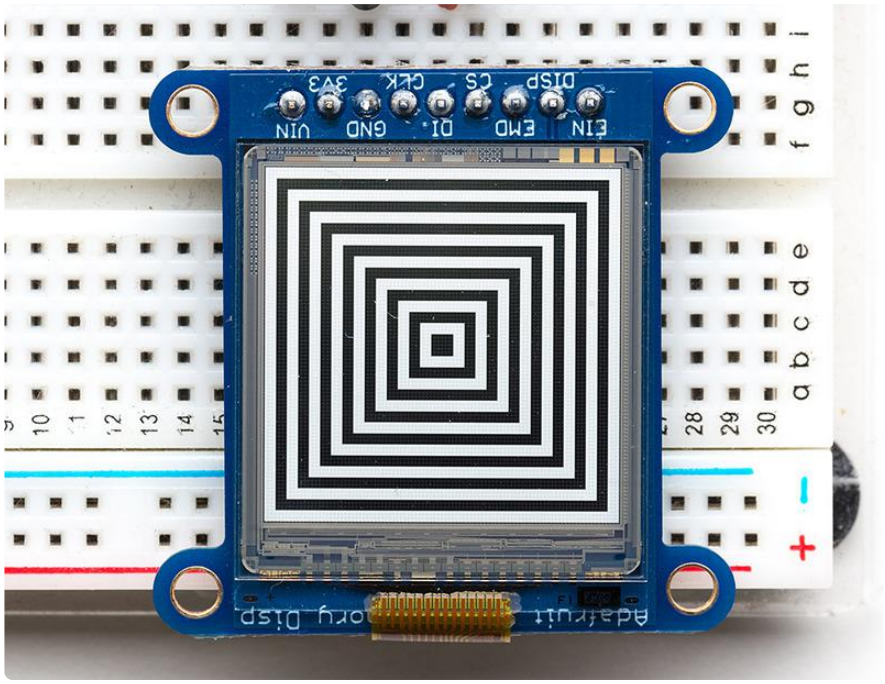
https://learn.adafruit.com/adafruit-sharp-memory-display-breakout

Last updated on 2022-12-01 02:00:23 PM EST

Table of Contents

- Schematic & Fabrication Print 1.3" Display

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.

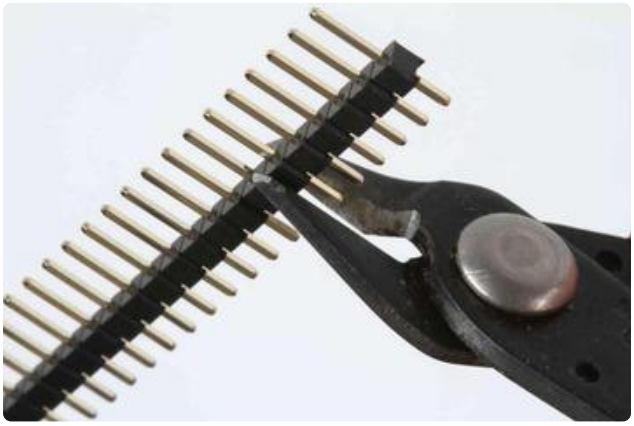
If you have one of the newer 168x144 pixel versions, then 168x144 bits = 3,024 bytes. That won't fit on an Arduino Uno or Leonardo! You must use a chip with more RAM like a Metro or Feather M0 or ESP8266.

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

Assembly

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:

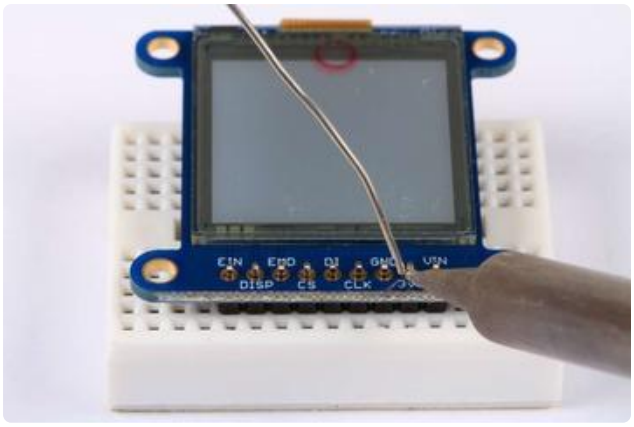


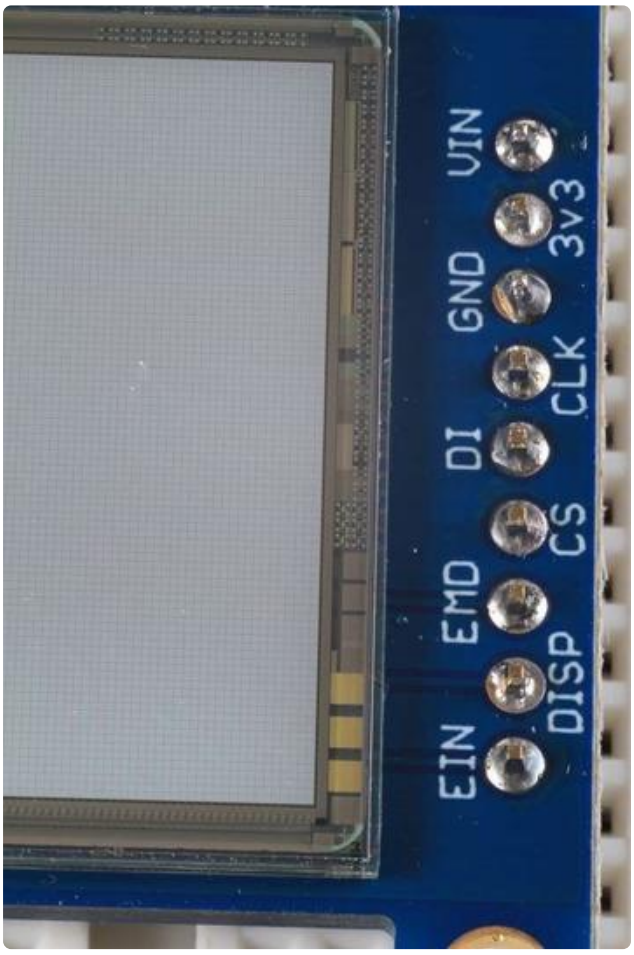
Position the header

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

Position the display

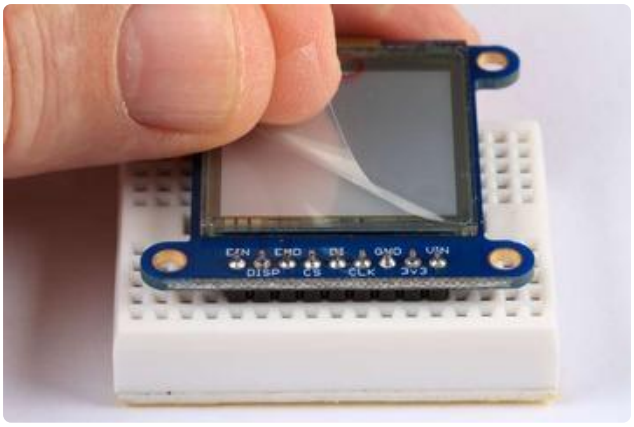
Place the Sharp Memory Display over the pins on the breadboard.





Solder!

Solder each pin to assure good electrical conductivity.



Remove the Protective Film

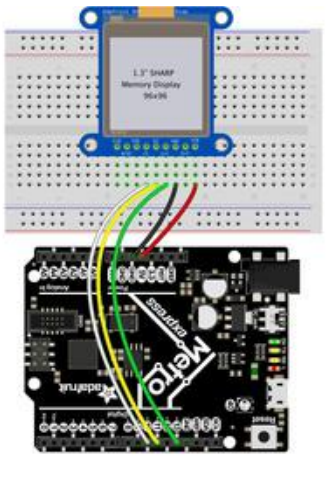
After soldering is complete. Gently peel the film from the display.

Arduino Wiring

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

Download Fritzing Diagram

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.

Arduino 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 ()
- Adafruit GFX Library ()
- Adafruit BusIO Library ()

For details on how to install libraries, see this guide: All About Arduino Libraries ().

Run the Example Code

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

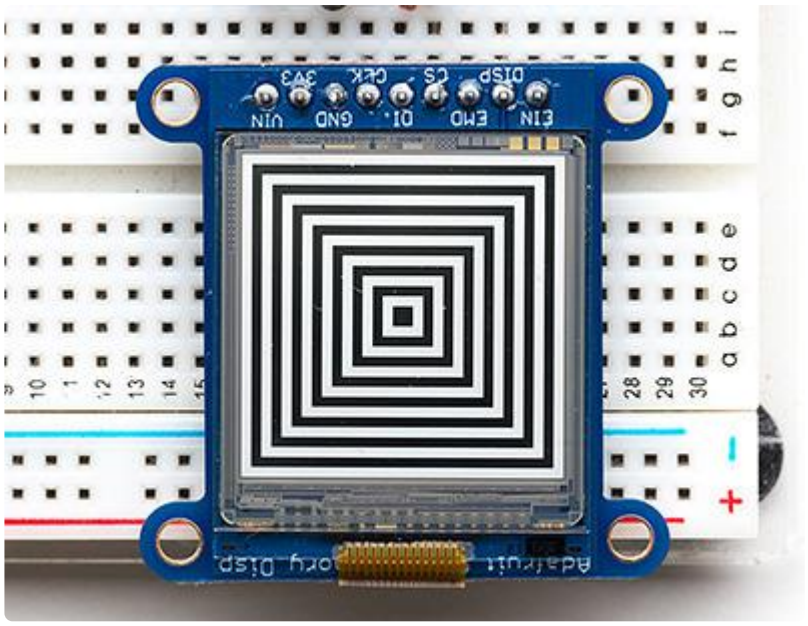
File->Examples->Adafruit_SHARP_Memory_Display->sharpmemtest

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 ().



2.7" Display Bad Apple Example

For this example, you'll want to preferably use a Metro M4 or Feather M4 so that it will be plenty fast.

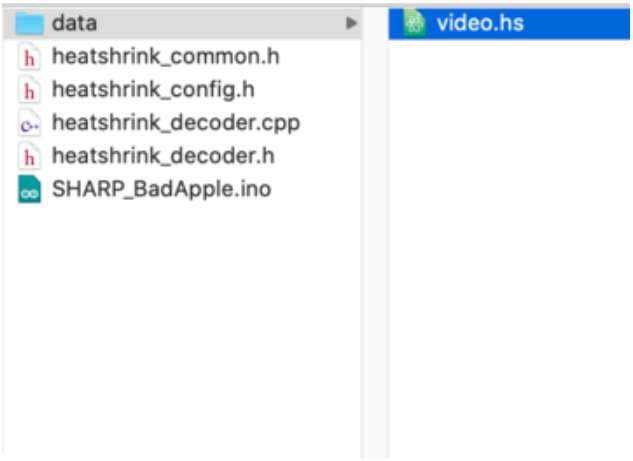
The Bad Apple video example was written for the 2.7" Sharp Display. This is a port of ESP32_BadApple () which as the name implies, was plays the Bad Apple video and was written to run on the ESP32 and OLED displays. You'll want to start by downloading and unzipping the example files first:

Download Example Files

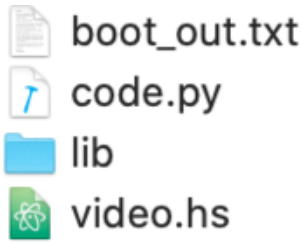
Uploading the Video

It's easiest to start with uploading the video file first. The video data is kept in flash memory, so the easiest way to put it in there is to install CircuitPython briefly just to copy the file over before uploading the Arduino code. Fortunately, Installing CircuitPython is really easy. If you are not sure how, you can check out our Welcome to CircuitPython guide ().

After it finished uploading, and with the Circuit Playground board connected over USB, it should appear on your computer as a flash drive called CIRCUITPY.



You can find the file inside the data folder named video.hs.



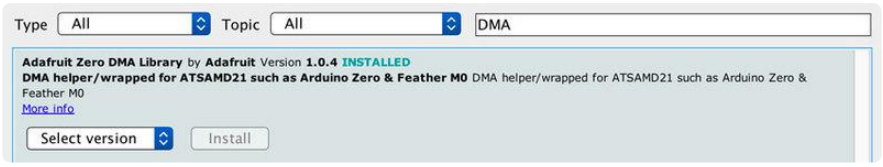
Go ahead and copy the video file over to the root of your CIRCUITPY drive. That's it, it's time to move over to arduino.

Additional Libraries

We'll assume you've already installed the libraries mentioned on the Arduino Programming page. There are a few additional libraries required to run the example:

Adafruit Zero DMA

This is used by the Graphics Library if you choose to use DMA

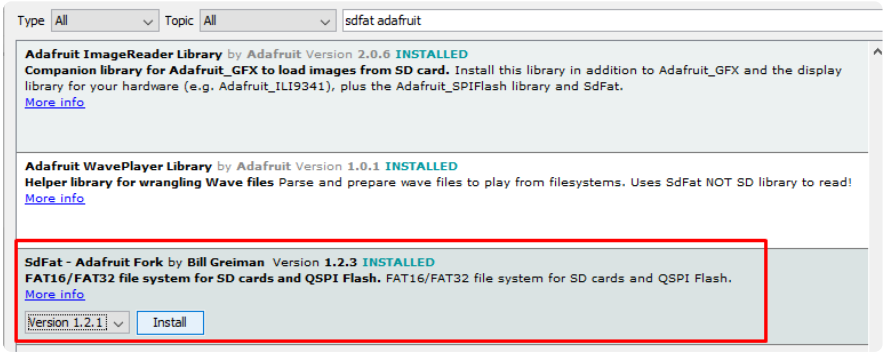


Adafruit SPIFlash

This will let you read/write to the onboard FLASH memory with super-fast QSPI support

SdFat (Adafruit Fork)

The Adafruit fork of the really excellent SD card library that gives a lot more capability than the default SD library

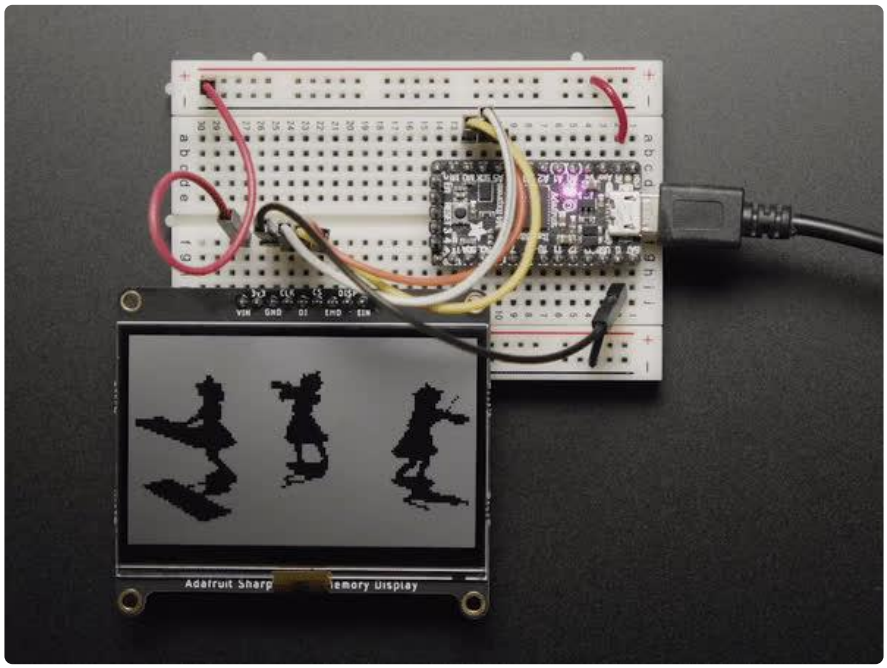


Open in Arduino

Go ahead and open up the SHARP_BadApple.ino file up in Arduino. There's only one small change we may need to make. Go ahead and locate the following line of code:

```
#define SHARP_SS   A5
```

If your Chip Select line is connected differently, you may to change the value of `SHARP_SS`. That's the only change. After that, go ahead and upload it to your board. You should see an animation similar to the following:



CircuitPython displayio Setup

CircuitPython Installation

First make sure you are running the latest version of Adafruit CircuitPython () for your board, and that `sharpdisplay` is in its supported modules list.

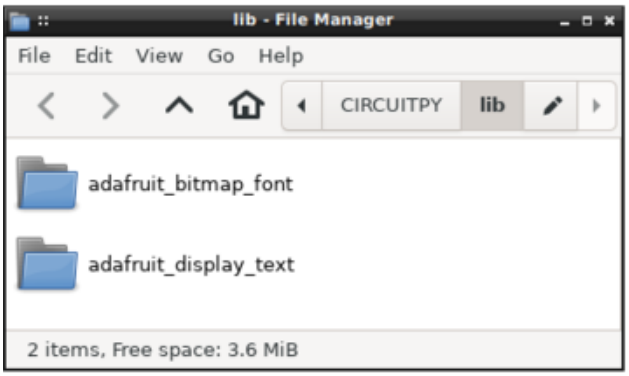
This feature is very new and not currently available in the latest public build, so you will need to go to https://circuitpython.org/, select your board, and find the "Absolute Newest" image.

Libraries

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 (). Our CircuitPython starter guide has a great page on how to install the library bundle ().

Download latest Library Bundle

There are many libraries in the bundle that enhance displayio, but the two we need for the example code are



adafruit_display_text - to show text and labels
adafruit_bitmap_font - to load fonts from CIRCUITPY for better typography than the "built in" font
Before continuing make sure your board's lib folder or root filesystem has the adafruit_display_text and adafruit_bitmap_font folders copied over.

CircuitPython displayio Usage

This feature is very new and not currently available in the latest public build, so you will need to go to https://circuitpython.org/, select your board, and find the "Absolute Newest" image.

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

Check the support matrix () for your board to see whether it supports the `sharpdisplay` module.

To demonstrate the usage, we'll initialize the library and the Python REPL will be displayed on it. You can type these lines in directly or put them in code.py.

Initialization

First, import required modules and release the existing display (if any).

```
import board
import displayio
import framebufferio
import sharpdisplay

# Release the existing display, if any
displayio.release_displays()
```

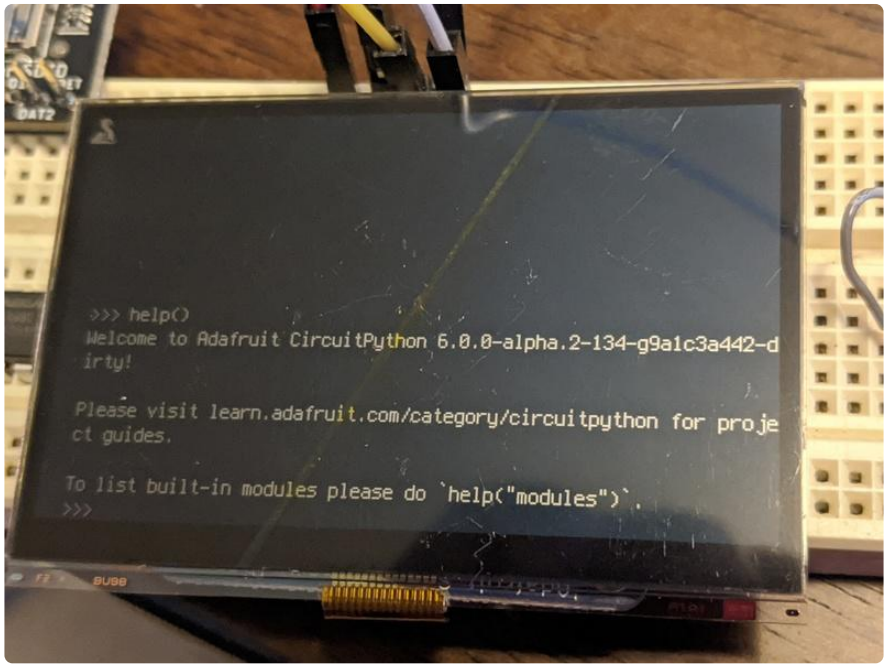
Next, create the display using the appropriate SPI bus, Chip Select (CS) Pin, width, and height. The baudrate can also be set, though the default value of 2MHz should work with all Sharp Memory Displays. Make sure to use the right pin names as you have wired up to your board! If you use a nonstandard SPI bus, construct it with `busio.SPI` instead of using `board.SPI()`.

```
bus = board.SPI()
chip_select_pin = board.D6
# Select JUST ONE of the following lines:
# For the 400x240 display (can only be operated at 2MHz)
framebuffer = sharpdisplay.SharpMemoryFramebuffer(bus, chip_select_pin, 400, 240)
# For the 144x168 display (can be operated at up to 8MHz)
#framebuffer = sharpdisplay.SharpMemoryFramebuffer(bus, chip_select_pin, width=144,
height=168, baudrate=8000000)
```

The last thing to do before you can use displayio routines is to connect the framebuffer as a display:

```
display = framebufferio.FramebufferDisplay(framebuffer)
```

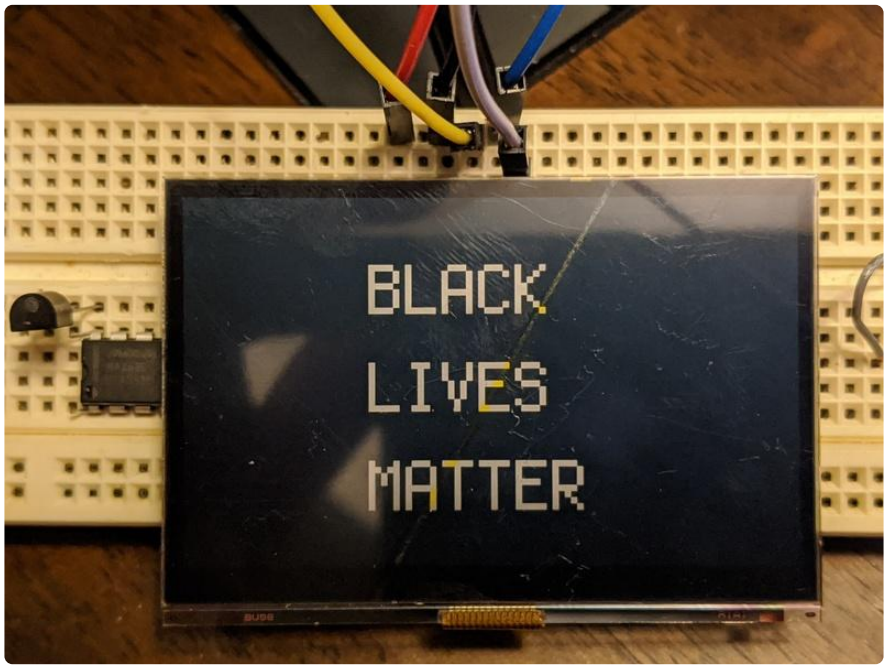
If you are doing this interactively at the Python REPL, you will now see the REPL mirrored onto the Sharp Memory Display.



Drawing

The SharpMemoryDisplay module supports all the methods for drawing that DisplayIO supports: Text, bitmaps, shapes, etc. For instance, if you wanted to display a label using the built-in terminal font, you would use something like the following:

```
from adafruit_display_text.label import Label
from terminalio import FONT
label = Label(font=FONT, text="BLACK\nLIVES\nMATTER", x=120, y=120, scale=4,
line_spacing=1.2)
display.show(label)
```



We cover CircuitPython displayio more in depth in its own guide, so now that you've got the display going, learn more about CircuitPython Display Support Using displayio () to get the most out of it.

Circuitpython displayio Example

This example displays the names of just some of the Black people injured or killed by police brutality in the United States. For more information about Adafruit's response to racial injustice, visit our dedicated Black Lives Matter () page.

This particular example is designed to use the Adafruit 2.7" SHARP Memory Display.

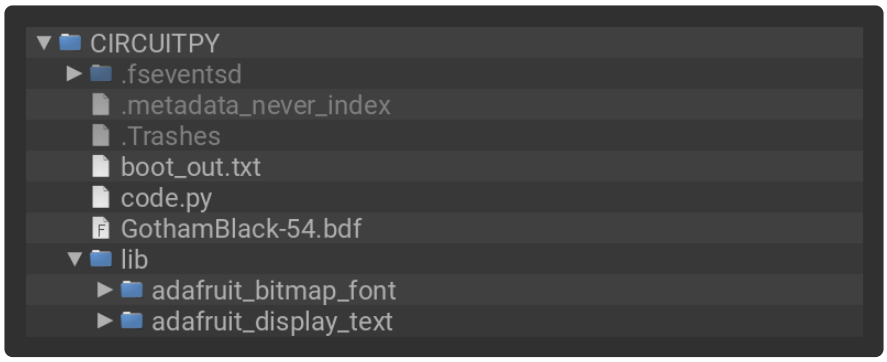
Installing Project Code

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your CIRCUITPY drive. Then you need to update code.py with the example script.

Thankfully, we can do this in one go. In the example below, click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, open the directory CircuitPython_Shar

pDisplay_Displayio/ and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your CIRCUITPY drive.

Your CIRCUITPY drive should now look similar to the following image:




```
# SPDX-FileCopyrightText: 2020 Limor Fried for Adafruit Industries
# SPDX-FileCopyrightText: 2020 Jeff Epler for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import random
import time

import adafruit_display_text.label
from adafruit_bitmap_font import bitmap_font
import board
import displayio
import framebufferio
import sharpdisplay

## When making several changes, this ensures they aren't shown partially
## completed (except for the time to actually update the display)
class BatchDisplayUpdate:
    def __init__(self, the_display):
        self.the_display = the_display
        self.auto_refresh = the_display.auto_refresh

    def __enter__(self):
        self.the_display.auto_refresh = False

    def __exit__(self, unused1, unused2, unused3):
        self.the_display.refresh()
        self.the_display.auto_refresh = self.auto_refresh

# https://saytheirnames.com/
# real people, not just #hashtags
names = [
    "Rodney King",
    "Abner Louima",
    "Amadou Diallo",
    "Sean Bell",
    "Oscar Grant",
    "Eric Garner",
    "Michael Brown",
    "Laquan McDonald",
    "Freddie Gray",
    "Antwon Rose Jr",
    "Ahmaud Arbery",
    "Breonna Taylor",
    "John Crawford III",
```

```
    "Ezell Ford",
    "Dante Parker",
    "Michelle Cusseaux",
    "Laquan Mcdonald",
    "George Mann",
    "Tanisha Anderson",
    "Akai Gurley",
    "Tamir Rice",
    "Rumain Brisbon",
    "Jerame Reid",
    "Matthew Ajibade",
    "Frank Smart",
    "Nastasha McKenna",
    "Tony Robinson",
    "Anthony Hill",
    "Mya Hall",
    "Phillip White",
    "Eric Harris",
    "Walter Scott",
    "William Chapman II",
    "Alexia Christian",
    "Brendon Glenn",
    "Victor Maunel Larosa",
    "Jonathan Sanders",
    "Freddie Blue",
    "Joseph Mann",
    "Salvado Ellswood",
    "Sanda Bland",
    "Albert Joseph Davis",
    "Darrius Stewart",
    "Billy Ray Davis",
    "Samuel Dubose",
    "Michael Sabbie",
    "Brian Keith Day",
    "Christian Taylor",
    "Troy Robinson",
    "Asshams Pharoah Manley",
    "Felix Kumi",
    "Keith Harrison Mcleod",
    "Junior Prosper",
    "Lamontez Jones",
    "Paterson Brown",
    "Dominic Hutchinson",
    "Anthony Ashford",
    "Alonzo Smith",
    "Tyree Crawford",
    "India Kager",
    "La'vante Biggs",
    "Michael Lee Marshall",
    "Jamar Clark",
    "Richard Perkins",
    "Nathaniel Harris Pickett",
    "Benni Lee Tignor",
    "Miguel Espinal",
    "Michael Noel",
    "Kevin Matthews",
    "Bettie Jones",
    "Quintonio Legrier",
    "Keith Childress Jr",
    "Janet Wilson",
    "Randy Nelson",
    "Antronie Scott",
    "Wendell Celestine",
    "David Joseph",
    "Calin Roquemore",
    "Dyzhawn Perkins",
    "Christoper Davis",
    "Marco Loud",
    "Peter Gaines",
```

```
    "Torry Robison",
    "Darius Robinson",
    "Kevin Hicks",
    "Mary Truxillo",
    "Demarcus Semer",
    "Willie Tillman",
    "Terrill Thomas",
    "Sylville Smith",
    "Sean Reed",
    "Alton Streling",
    "Philando Castile",
    "Terence Crutcher",
    "Paul O'Neal",
    "Alteria Woods",
    "Jordan Edwards",
    "Aaron Bailey",
    "Ronell Foster",
    "Stephon Clark",
    "Antwon Rose II",
    "Botham Jean",
    "Pamela Turner",
    "Dominique Clayton",
    "Atatiana Jefferson",
    "Christopher Whitfield",
    "Christopher Mccovey",
    "Eric Reason",
    "Michael Lorenzo Dean",
    "Tony McDade",
    "David McAtee",
    "George Floyd",
]

# A function to choose "k" different items from the "population" list
# We'll use it to select the names to display
def sample(population, k):
    population = population[:]
    for _ in range(k):
        j = random.randint(0, len(population)-1)
        yield population[j]
        population[j] = population[-1]
        population.pop()

# Initialize the display, cleaning up after a display from the previous run
# if necessary
displayio.release_displays()
bus = board.SPI()
framebuffer = sharpdisplay.SharpMemoryFramebuffer(bus, board.D6, 400, 240)
display = framebufferio.FramebufferDisplay(framebuffer, auto_refresh = True)

# Load our font
font = bitmap_font.load_font("/GothamBlack-54.bdf")

# Create a Group for the BLM text
blm_group = displayio.Group()
display.show(blm_group)

# Create a 3 line set of text for BLM
blm_font = [None, None, None]
for line in range(3):
    label = adafruit_display_text.label.Label(font, color=0xFFFFFF)
    label.anchor_point = (0, 0)
    label.anchored_position = (8, line*84+8)
    blm_font[line] = label
    blm_group.append(label)

# Get something on the display as soon as possible by loading
# specific glyphs.
font.load_glyphs(b"BLACK")
blm_font[0].text = "BLACK"
```

```
font.load_glyphs(b"ISEV")
blm_font[1].text = "LIVES"
font.load_glyphs(b"RMT")
blm_font[2].text = "MATTER"
font.load_glyphs(b"' DFGHJNOPQUWXYZabcdefghijklmnopqrstuvwxyz")


# Create a 2 line set of font text for names
names_font = [None, None]
for line in range(2):
    label = adafruit_display_text.label.Label(font, color=0xFFFFFF)
    # Center each line horizontally, position vertically
    label.anchor_point = (0.5, 0)
    label.anchored_position = (200, line*84+42)
    names_font[line] = label

# Create a Group for the name text
name_group = displayio.Group()
for line in names_font:
    name_group.append(line)

# Repeatedly show the BLM slogan and then 5 names.
while True:
    display.show(blm_group)

    # Show the BLM slogan
    with BatchDisplayUpdate(display):
        blm_font[1].color = blm_font[2].color = 0  # hide lines 2&3
    time.sleep(1)

    with BatchDisplayUpdate(display):
        blm_font[1].color = 0xFFFFFF  # show middle line
        blm_font[0].color = blm_font[2].color = 0  # hide lines 1&3
    time.sleep(1)

    with BatchDisplayUpdate(display):
        blm_font[2].color = 0xFFFFFF  # show last line
        blm_font[0].color = blm_font[1].color = 0  # hide lines 1&2
    time.sleep(1)

    with BatchDisplayUpdate(display):
        for line in blm_font:
            line.color = 0xFFFFFF
    time.sleep(2)

    # Show 5 names
    display.show(name_group)
    for name in sample(names, 5):
        print(name)
        lines = name.split(" ")
        with BatchDisplayUpdate(display):
            for i in range(2):
                names_font[i].text = lines[i]

                # Due to a bug in adafruit_display_text, we need to reestablish
                # the position of the labels when updating them.
                # Once https://github.com/adafruit/
Adafruit_CircuitPython_Display_Text/issues/82
                # has been resolved, this code will no longer be necessary (but
                # will not be harmful either)
                names_font[i].anchor_point = (0.5, 0)
                names_font[i].anchored_position = (200, i*84+42)
        time.sleep(5)
    names_font[0].text = names_font[1].text = ""
```

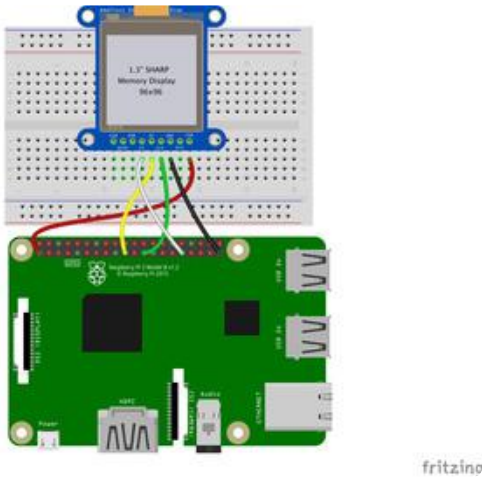
Python Wiring

It's easy to use the Sharp Memory Display with Python and the Adafruit CircuitPython SharpMemoryDisplay () 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 ().

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

Download Fritzing Diagram

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 ()!

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 fonts-dejavu`

This package was previously calls ttf-dejavu, so if you are running an older version of Raspberry Pi OS, it may be called that.

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 () 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 ().

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 ().

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

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""
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.

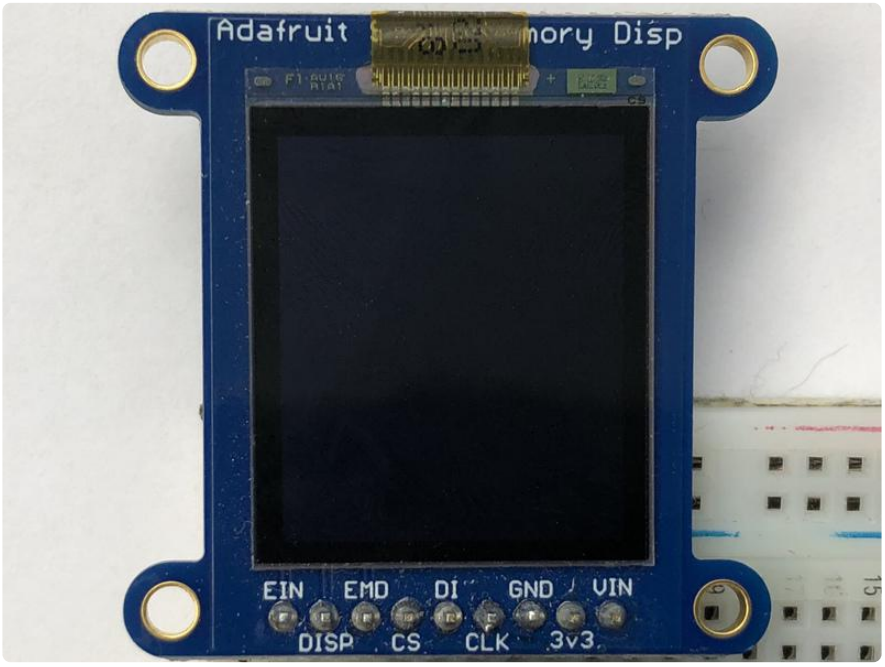
```
# Create blank image for drawing.
# Make sure to create image with mode '1' for 1-bit color.
image = Image.new('1', (display.width, display.height))

# 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.

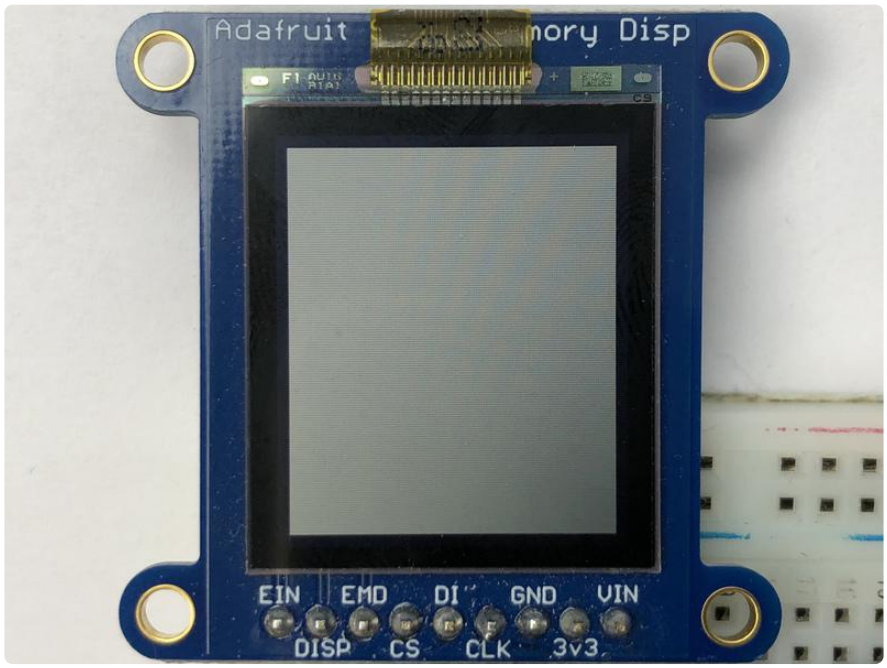


Next we will create a smaller white rectangle. The easiest way to do this is to draw another rectangle a little smaller than the full screen with no fill or outline and place it in a specific location. In this case, we will create a rectangle that is 5 pixels smaller on each side. This is where the BORDER 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.

CircuitPython Docs

CircuitPython Docs ()

Downloads and Links

Libraries:

- Sharp Memory Display Library ()
- Adafruit GFX Library ()

Files

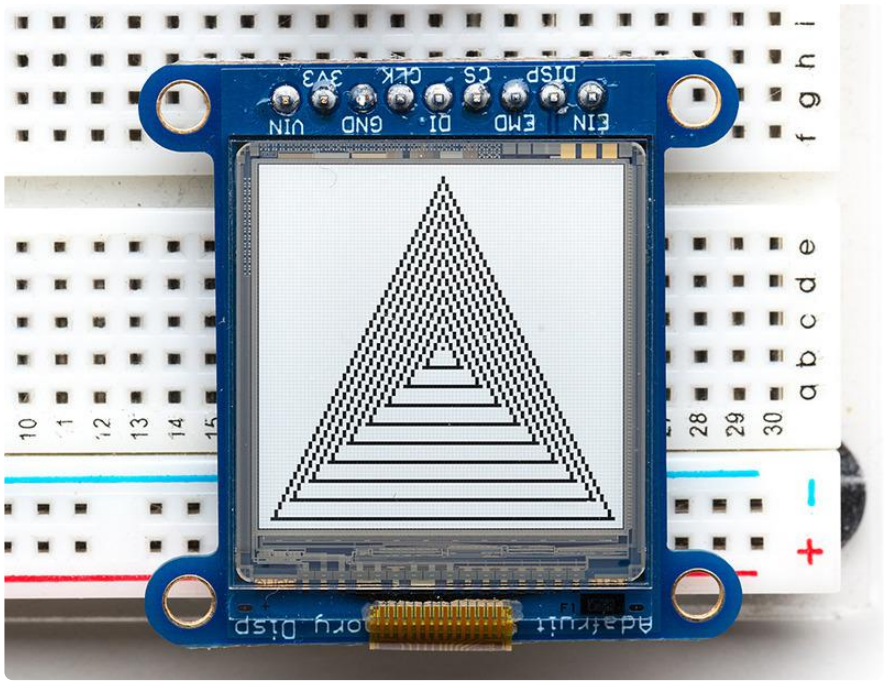
- Datasheet for the LS013B4DN04 LCD Module ()
- Datasheet for the LS027B7DH01 LCD Module ()

- Fritzing object in Adafruit Fritzing Library ()
- EagleCAD PCB files on GitHub ()

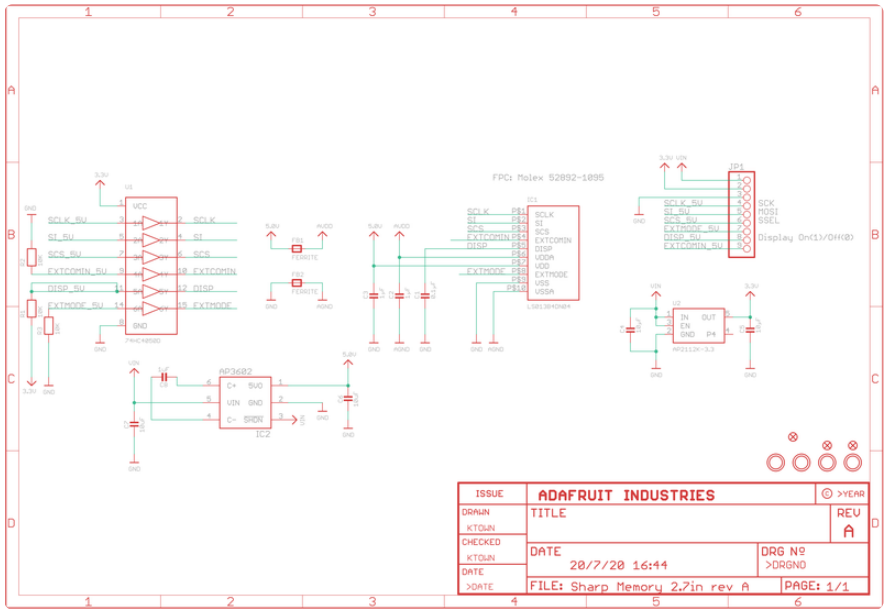
Library Reference

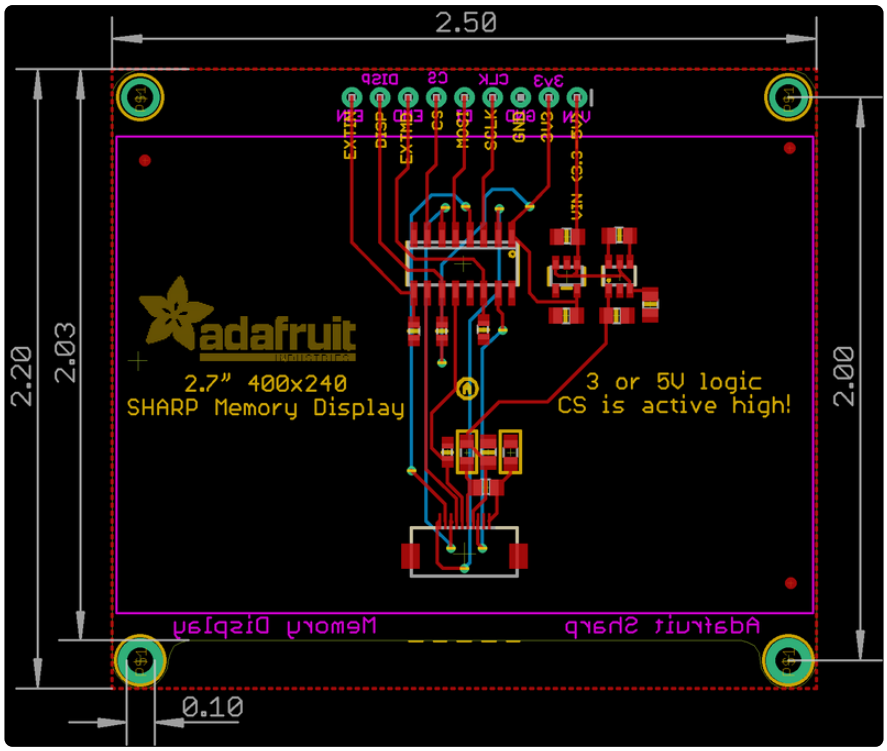
- Adafruit GFX Library ()

LS013B7DH05 Datasheet

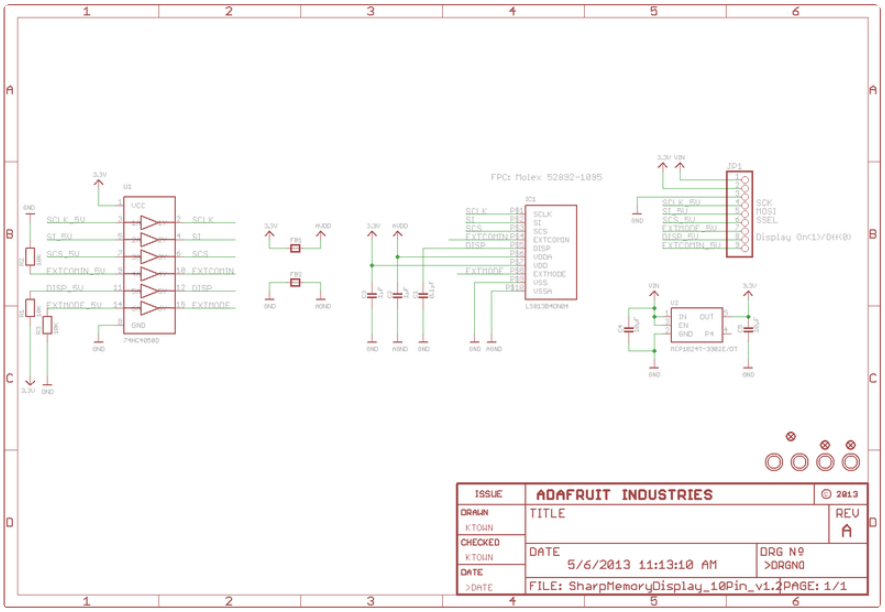


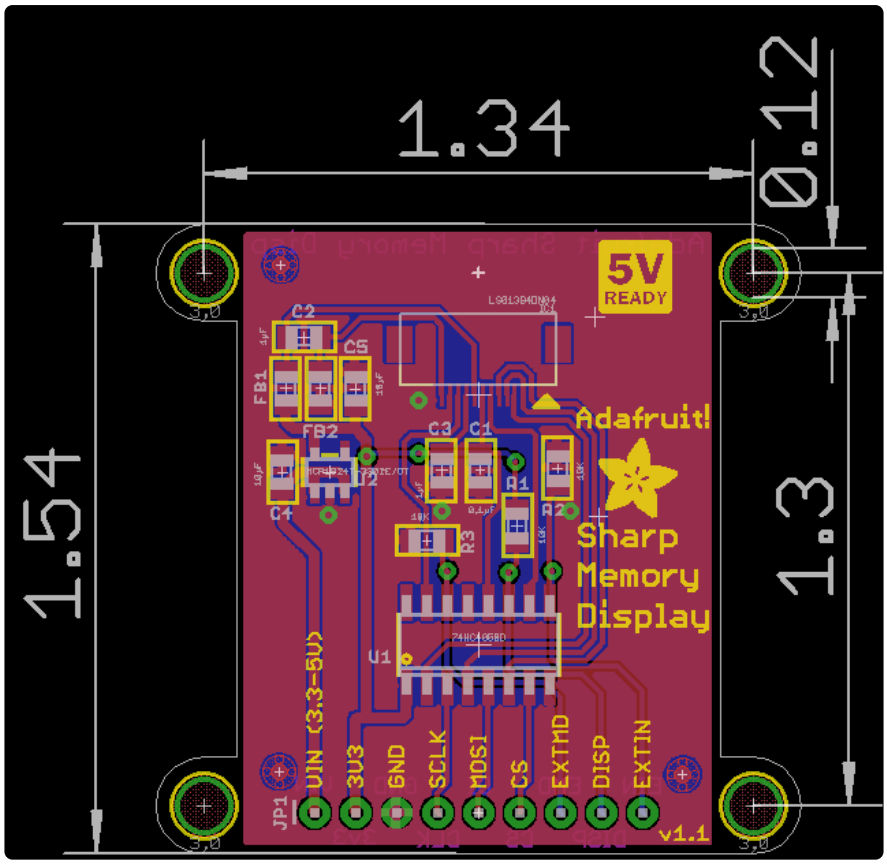
Schematic & Fabrication Print 2.7" Display





Schematic & Fabrication Print 1.3" Display




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