Adafruit Monochrome 1.12" 128x128 OLED Graphic Display

Created by Kallni Rembor

https://learn.adafruit.com/adafruit-monochrome-1-12-in-128x128-oled

Last updated on 2023-08-29 04:44:41 PM EDT
# Table of Contents

## Overview

## Pinouts

- Power Pins
- STEMMA QT Connectors
- I2C Pins
- SPI Pins
- I2C and SPI Jumpers
- LED Disable Pad
- RST Pin

## CircuitPython

- CircuitPython Microcontroller Wiring
- CircuitPython Usage
- Example Code

## Python Docs

## Arduino

- Wiring
- Library Installation
- Load Example

## Arduino Docs

## Downloads

- Files
- Schematic and Fab Print
Overview

This OLED goes out to all the fans who want more pixels in a smaller size! Normally our 128x64 OLEDs are the biggest ones we've stocked that can use I2C. This one is a whopping 128x128 pixels in crisp monochrome.

This display is a petite 1.12" diagonal, but very readable due to the high contrast of an OLED display. This display is made of 128x128 individual white OLED pixels, each one is turned on or off by the controller chip. Because the display makes its own light, no backlight is required. This reduces the power required to run the OLED and is why the
display has such high contrast; we really like these miniature displays for their crispness!

The driver chip, SSD1107 can communicate in two ways: I2C or SPI. The OLED itself requires a 3.3V and 12V power supply and 3.3V logic levels for communication. We include a 3.3V regulator and 12V boost converter, and all pins are fully level shifted so you can use with 3V or 5V devices!

If you are using I2C, we’ve included SparkFun qwiic () compatible STEMMA QT () connectors for the I2C bus so you don't even need to solder! Plug and play with any board that has a Qwiic or STEMMA QT connector for effortless prototyping and development. QT Cable is not included, but we have a variety in the shop ().
This display, being 128x128 pixels, requires 128 * 128 = 2KB of SRAM just to buffer the display. So you can't use it with a small chip such as the Arduino UNO (ATmega328 or 32u4). Pick a microcontroller or microcomputer with 16KB+ RAM - a SAMD21, SAMD51, ESP, nRF52, Teensy, etc. will do an excellent job. As long as you have I2C or SPI interface available, you're good to go - SPI will be much faster but I2C requires fewer pins.

We have both Arduino () and CircuitPython support () for this display chipset (SH1107).

Please note that OLED displays are made of hundreds of...OLEDs! That means each pixel is a little organic LED, and if it's kept on for over 1000 hours, it'll start to dim. If
you want to keep the display uniformly bright, please turn off the display (set the pixels off) when it isn't needed to keep them from dimming.

Pinouts
Power Pins

GND - this is the power and signal ground pin
Vin - this is the power pin, connect to 3-5VDC - it has reverse polarity protection but try to wire it right!
3Vo - this is the 3.3V output from the onboard regulator, you can 'borrow' about 100mA if you need to power some other 3.3V logic devices

STEMMA QT Connectors

The display comes with two STEMMA QT ports. These connectors allow you to connect to development boards with STEMMA QT connectors or to other things with various associated accessories. Since they are connected in parallel, either side can be used as an input or output. They can be used to daisychain multiple STEMMA QT breakouts.

I2C Pins

Clk - this is the Clock pin. Connect to I2C SCL
Data - this is the Data pin. Connect to I2C SDA
SPI Pins

CS - this is the Chip Select pin, it is used by SPI to select the device.
DC - this is the Data/Command selector pin used to differentiate between Data and Commands that are sent to the display.
Clk - this is the Clock pin. Connect to SPI SCLK
Data - this is the Data pin. Connect to SPI MOSI

I2C and SPI Jumpers

The STEMMA QT ports will only work when the display is in I2C Mode.

You'll notice there are two open jumpers on the back towards the center of the board.

To use in I2C mode, you'll want to make sure the J1 jumper is open. To use in SPI mode, you'll want to bridge the J1 jumper. The board comes in I2C mode by default. The A0 jumper is used for setting the I2C address. When it is open, the I2C address is 0x3D. When it is bridged, the I2C Address is 0x3C. Alternatively, by tying the A0 pin to ground, you can set the I2C address to 0x3C.
LED Disable Pad

Above the "on" LED on the left side of the back of the board is a closed jumper. If you want to disable the power LED, you can cut the trace here. After cutting the trace, if you wish to enable it again, you can solder the jumper closed.

RST Pin

Rst - this is the Reset pin, you may be able to share this with your microcontroller reset pin but if you can, connect it to a digital pin.

CircuitPython

It's easy to use the Monochrome 1.12" 128x128 OLED with Python or CircuitPython, and the Adafruit CircuitPython DisplayIO SH1107 () module. This module allows you to easily write Python code that controls the Monochrome 1.12" 128x128 OLED graphic display.

CircuitPython Microcontroller Wiring

First wire up your OLED to your board exactly as shown below. Here's an example of wiring a Feather M4 to the display with I2C using one of the handy STEMMA QT () connectors:
Board 3V to display VIN (red wire)
Board GND to display GND (black wire)
Board SCL to display Clk (yellow wire)
Board SDA to display Data (blue wire)

You can also use the standard 0.100" pitch headers to wire it up via I2C on a breadboard:

To use the board in SPI mode, you must close the J1 jumper on the back.

Board 3V to display VIN (red wire)
Board GND to display GND (black wire)
Board SCK to display Clk (yellow wire)
Board MOSI to display Data (blue wire)
Board D5 to display CS (orange wire)
Board D6 to display DC (green wire)
Board D9 to display Rst (purple wire)
CircuitPython Usage

To use with CircuitPython, you need to first install the DisplayIO SH1107 library into the lib folder on your CIRCUITPY drive. This example also requires the Display Text library. 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, and copy the entire lib folder and the code.py file to your CIRCUITPY drive.

Your CIRCUITPY/lib folder should contain the following folder and file:

- adafruit_displayio_sh1107.mpy
- adafruit_display_text/

Example Code

This example defaults to using I2C. To use with SPI, comment out the I2C setup, and uncomment the SPI setup.

```python
# SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: Unlicense

""
Based on example by Mark Roberts (mdroberts1243).
This example writes text to the display, and draws a series of squares and a rectangle.
""

import board
import displayio
import terminalio
from adafruit_display_text import bitmap_label as label
from adafruit_displayio_sh1107 import SH1107, DISPLAY_OFFSET_ADAFRUIT_128x128_OLED_5297

displayio.release_displays()

# For I2C
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a microcontroller
display_bus = displayio.I2CDisplay(i2c, device_address=0x3D)

# For SPI:
```

Remember, to use the board in SPI mode, you must close the J1 jumper on the back.
# import busio
# spi_bus = busio.SPI(board.SCK, board.MOSI)
# display_bus = displayio.FourWire(spi_bus, command=board.D6, chip_select=board.D5, reset=board.D9)

# Width, height and rotation for Monochrome 1.12" 128x128 OLED
WIDTH = 128
HEIGHT = 128
ROTATION = 90

# Border width
BORDER = 2

display = SH1107(
    display_bus,
    width=WIDTH,
    height=HEIGHT,
    display_offset=DISPLAY_OFFSET_ADAFRUIT_128x128_OLED_5297,
    rotation=ROTATION,
)

# Make the display context
splash = displayio.Group()
display.show(splash)

color_bitmap = displayio.Bitmap(WIDTH, HEIGHT, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0xFFFFFF  # White

bg_sprite = displayio.TileGrid(color_bitmap, pixel_shader=color_palette, x=0, y=0)
splash.append(bg_sprite)

# Draw a smaller inner rectangle in black
inner_bitmap = displayio.Bitmap(WIDTH - BORDER * 2, HEIGHT - BORDER * 2, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0x000000  # Black
inner_sprite = displayio.TileGrid(
    inner_bitmap, pixel_shader=inner_palette, x=BORDER, y=BORDER
)
splash.append(inner_sprite)

# Draw some white squares
small_bitmap = displayio.Bitmap(8, 8, 1)
small_square = displayio.TileGrid(small_bitmap, pixel_shader=color_palette, x=58, y=17)
splash.append(small_square)

medium_bitmap = displayio.Bitmap(16, 16, 1)
medium_square = displayio.TileGrid(medium_bitmap, pixel_shader=color_palette, x=71, y=15)
splash.append(medium_square)

large_bitmap = displayio.Bitmap(32, 32, 1)
large_square = displayio.TileGrid(large_bitmap, pixel_shader=color_palette, x=91, y=28)
splash.append(large_square)

bottom_bitmap = displayio.Bitmap(110, 50, 1)
bottom_rectangle = displayio.TileGrid(
    bottom_bitmap, pixel_shader=color_palette, x=10, y=69
)
splash.append(bottom_rectangle)

# Draw some label text
name_text = "Monochrome 1.12in"
name_text_area = label.Label(terminalio.FONT, text=name_text, color=0xFFFFFF, x=8, y=8)
splash.append(name_text_area)
Once everything is saved to the CIRCUITPY drive, you should see the display change!

That's all there is to using the Monochrome 1.12" 128x128 OLED Graphic Display with CircuitPython!

Python Docs

Arduino

Using the Monochrome 1.12" 128x128 OLED Graphic Display with Arduino involves wiring up the display to your Arduino-compatible microcontroller, installing the Adafruit SH110x library and running the provided example code.

This display, being 128x128 pixels, requires $128 \times 128 = 2\text{KB}$ of SRAM just to buffer the display. So you can't use it with a small chip such as the Arduino UNO (ATmega328 or 32u4). Pick a microcontroller or microcomputer with 16KB+ RAM - a SAMD21, SAMD51,
ESP, nRF52, Teensy, etc. will do an excellent job. As long as you have I2C or SPI interface available, you're good to go - SPI will be much faster but I2C requires fewer pins.

**Wiring**

Here is how to wire up the display via I2C using one of the STEMMA QT () connectors. The examples show a Metro but wiring will work the same for an Arduino or other compatible board.

Connect Vin (red wire) to the power supply, 3-5V is fine. Use the same voltage that the microcontroller logic is based off of. For most Arduinos, that is 5V.

Connect GND (black wire) to common power/data ground.

Connect the Clk (yellow wire) pin to the I2C clock SCL pin on your Arduino.

Connect the Data (blue wire) pin to the I2C data SDA pin on your Arduino.

Here is how to wire the display via I2C to a board using a solderless breadboard:

Connect Vin (red wire) to the power supply, 3-5V is fine. Use the same voltage that the microcontroller logic is based off of. For most Arduinos, that is 5V.

Connect GND (black wire) to common power/data ground.

Connect the Clk (yellow wire) pin to the I2C clock SCL pin on your Arduino.

Connect the Data (blue wire) pin to the I2C data SDA pin on your Arduino.

Here is how to wire the display via SPI to a board using a solderless breadboard:

To use the board in SPI mode, you must close the J1 jumper on the back.
Connect Vin to the power supply, 3-5V is fine. Use the same voltage that the microcontroller logic is based off of. For most Arduinos, that is 5V
Connect GND to common power/data ground
Connect the Clk pin to the SPI clock SCK pin on your Arduino. On the Metro, you'll want to use the ICSP Header Pin 3.
Connect the Data pin to the SPI data MOSI pin on your Arduino. On the Metro, you'll want to use the ICSP Header Pin 4.
Connect the DC pin to D8 on the Arduino and similar shaped boards. If you don't have this pin on your Microcontroller, feel free to use a different one and update the example.
Connect the CS pin to D10 on the Arduino and similar shaped boards. If you don't have this pin on your Microcontroller, feel free to use a different one and update the example.

Library Installation

You can install the Adafruit SH110x library for Arduino using the Library Manager in the Arduino IDE.

Click the Manage Libraries ... menu item, search for SH1107, and select the Adafruit SSH110x library, and install the latest version.
When asked to install the dependencies, click Install all.

Load Example

Open up File -> Examples -> Adafruit SH110x -> SH1107_128x128

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SH110X.h>

#define SCREEN_WIDTH 128  // OLED display width, in pixels
#define SCREEN_HEIGHT 128 // OLED display height, in pixels
#define OLED_RESET -1     // can set an oled reset pin if desired
Adafruit_SH1107 display = Adafruit_SH1107(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
                                          OLED_RESET, 1000000, 100000);

#define NUMFLAKES 10
```
#define XPOS 0
#define YPOS 1
#define DELTAY 2

#define LOGO_HEIGHT 16
#define LOGO_WIDTH 16
static const unsigned char PROGMEM logo_bmp[] =
{ 0b00000000, 0b11000000,
  0b00000001, 0b11000000,
  0b00000001, 0b11000000,
  0b00000011, 0b11100000,
  0b11110011, 0b11100000,
  0b11111110, 0b01111111,
  0b01111110, 0b11111111,
  0b00110011, 0b10011111,
  0b00011111, 0b11111111,
  0b00011111, 0b11111100,
  0b00001101, 0b01110000,
  0b00011011, 0b10100000,
  0b00111111, 0b11100000,
  0b00111111, 0b11110000,
  0b01111100, 0b11110000,
  0b01110000, 0b01110000,
  0b00000000, 0b00110000 };

void setup() {
    Serial.begin(9600);
    delay(250); // wait for the OLED to power up

    // Show image buffer on the display hardware.
    // Since the buffer is intialized with an Adafruit splashscreen
    // internally, this will display the splashscreen.
    display.begin(0x3D, true); // Address 0x3D default
    //display.setContrast (0); // dim display

    display.display();
    delay(2000);

    // Clear the buffer.
    display.clearDisplay();

    // draw a single pixel
    display.drawPixel(10, 10, SH110X_WHITE);
    // Show the display buffer on the hardware.
    // NOTE: You _must_ call display after making any drawing commands
    // to make them visible on the display hardware!
    display.display();
    delay(2000);
    display.clearDisplay();
    testdrawline();  // Draw many lines
    testdrawrect();   // Draw rectangles (outlines)
testfillrect();  // Draw rectangles (filled)
testdrawcircle(); // Draw circles (outlines)
testfillcircle();  // Draw circles (filled)
testdrawroundrect(); // Draw rounded rectangles (outlines)
testfillroundrect(); // Draw rounded rectangles (filled)
testdrawtriangle(); // Draw triangles (outlines)
testfilltriangle();  // Draw triangles (filled)
testdrawchar();    // Draw characters of the default font
testdrawstyles();  // Draw 'stylized' characters
testdrawbitmap();  // Draw a small bitmap image
testanimate(logo_bmp, LOGO_WIDTH, LOGO_HEIGHT); // Animate bitmaps
}
void loop() {
}
void testdrawline() {
int16_t i;

display.clearDisplay(); // Clear display buffer

for(i=0; i<display.width(); i+=4) {
display.drawLine(0, 0, i, display.height()-1, SH110X_WHITE);
display.display();  // Update screen with each newly-drawn line
delay(1);
}
for(i=0; i<display.height(); i+=4) {
display.drawLine(0, 0, display.width()-1, i, SH110X_WHITE);
display.display();
delay(1);
} 
delay(250);
display.clearDisplay();

for(i=0; i<display.width(); i+=4) {
display.drawLine(0, display.height()-1, i, 0, SH110X_WHITE);
display.display();
delay(1);
}
for(i=display.height()-1; i>=0; i-=4) {
display.drawLine(0, display.height()-1, display.width()-1, i, SH110X_WHITE);
display.display();
delay(1);
}
delay(250);
display.clearDisplay();
for(i=display.width()-1; i>=0; i-=4) {
  display.drawLine(display.width()-1, display.height()-1, i, 0, SH110X_WHITE);
display.display();
delay(1);
}
for(i=display.height()-1; i>=0; i-=4) {
  display.drawLine(display.width()-1, display.height()-1, 0, i, SH110X_WHITE);
display.display();
delay(1);
}
delay(250);
display.clearDisplay();
for(i=0; i<display.height(); i+=4) {
  display.drawLine(display.width()-1, 0, 0, i, SH110X_WHITE);
display.display();
delay(1);
}
for(i=0; i<display.width(); i+=4) {
  display.drawLine(display.width()-1, 0, i, display.height()-1, SH110X_WHITE);
display.display();
delay(1);
}
delay(2000); // Pause for 2 seconds
}
void testdrawrect(void) {
  display.clearDisplay();
  for(int16_t i=0; i<display.height()/2; i+=2) {
    display.drawRect(i, i, display.width()-2*i, display.height()-2*i, SH110X_WHITE);
display.display(); // Update screen with each newly-drawn rectangle
    delay(1);
  }
delay(2000);
}
void testfillrect(void) {
  display.clearDisplay();
  for(int16_t i=0; i<display.height()/2; i+=3) {
    // The INVERSE color is used so rectangles alternate white/black
    display.fillRect(i, i, display.width()-i*2, display.height()-i*2,
    SH110X_INVERSE);
display.display(); // Update screen with each newly-drawn rectangle
    delay(1);
  }
delay(2000);
}
void testdrawcircle(void) {
  display.clearDisplay();
  for(int16_t i=0; i<max(display.width(),display.height())/2; i+=2) {
    display.drawCircle(display.width()/2, display.height()/2, i, SH110X_WHITE);
display.display();
  }  
}
delay(1);
}
delay(2000);
}

void testfillcircle(void) {
    display.clearDisplay();
    for(int16_t i=max(display.width(),display.height())/2; i>0; i-=3) {
        // The INVERSE color is used so circles alternate white/black
        display.fillCircle(display.width() / 2, display.height() / 2, i, SH110X_INVERSE);
        display.display(); // Update screen with each newly-drawn circle
        delay(1);
    }
delay(2000);
}

void testdrawroundrect(void) {
    display.clearDisplay();
    for(int16_t i=0; i<display.height()/2-2; i+=2) {
        display.drawRoundRect(i, i, display.width()-2*i, display.height()-2*i, display.height()/4, SH110X_WHITE);
        display.display();
        delay(1);
    }
delay(2000);
}

void testfillroundrect(void) {
    display.clearDisplay();
    for(int16_t i=0; i<display.height()/2-2; i+=2) {
        // The INVERSE color is used so round-rects alternate white/black
        display.fillRoundRect(i, i, display.width()-2*i, display.height()-2*i, display.height()/4, SH110X_INVERSE);
        display.display();
        delay(1);
    }
delay(2000);
}

void testdrawtriangle(void) {
    display.clearDisplay();
    for(int16_t i=0; i<max(display.width(),display.height())/2; i+=5) {
        display.drawTriangle(
            display.width()/2, display.height()/2-i, display.width()/2-i, display.height()/2+i, display.width()/2+i, display.height()/2+i, SH110X_WHITE);
        display.display();
        delay(1);
    }
delay(2000);
}

void testfilltriangle(void) {
    display.clearDisplay();
    for(int16_t i=max(display.width(),display.height())/2; i>0; i-=5) {
// The INVERSE color is used so triangles alternate white/black

display.fillTriangle
(display.width()/2 , display.height()/2-i,
 display.width()/2-i, display.height()/2+i,
 display.width()/2+i, display.height()/2+i, SH110X_INVERSE);
display.display();
delay(1);
}
delay(2000);
}
void testdrawchar(void) {
  display.clearDisplay();
  display.setTextSize(1);      // Normal 1:1 pixel scale
  display.setTextColor(SH110X_WHITE); // Draw white text
  display.setCursor(0, 0);     // Start at top-left corner
  display.cp437(true);         // Use full 256 char 'Code Page 437' font

  // Not all the characters will fit on the display. This is normal.
  // Library will draw what it can and the rest will be clipped.
  for(int16_t i=0; i<256; i++) {
    if(i == '\n') display.write(' ');
    else          display.write(i);
  }
display.display();
delay(2000);
}

void testdrawstyles(void) {
  display.clearDisplay();
  display.setTextSize(1);             // Normal 1:1 pixel scale
  display.setTextColor(SH110X_WHITE);        // Draw white text
  display.setCursor(0,0);             // Start at top-left corner
  display.println(F("Hello, world!"));
  display.setTextColor(SH110X_BLACK, SH110X_WHITE); // Draw 'inverse' text
  display.println(3.141592);
  display.setTextSize(2);             // Draw 2X-scale text
  display.setTextColor(SH110X_WHITE);
  display.print(F("0x")); display.println(0xDEADBEEF, HEX);    
  display.display();
delay(2000);
}
void testdrawbitmap(void) {
  display.clearDisplay();
  display.drawBitmap( 
(display.width() - LOGO_WIDTH ) / 2,
(display.height() - LOGO_HEIGHT) / 2,
logo_bmp, LOGO_WIDTH, LOGO_HEIGHT, 1);
display.display();
  delay(1000);
}

#define XPOS   0 // Indexes into the 'icons' array in function below
#define YPOS   1
#define DELTAY 2

void testanimate(const uint8_t *bitmap, uint8_t w, uint8_t h) {
  int8_t f, icons[NUMFLAKES][3];
  // Initialize 'snowflake' positions
  for(f=0; f< NUMFLAKES; f++) {
    icons[f][XPOS]   = random(1 - LOGO_WIDTH, display.width());
    icons[f][YPOS]   = -LOGO_HEIGHT;
    icons[f][DELTAY] = random(1, 6);
    Serial.print(F("x: ");)
    Serial.print(icons[f][XPOS], DEC);
    Serial.print(F(" y: ");)
    Serial.print(icons[f][YPOS], DEC);
    Serial.print(F(" dy: ");)
    Serial.println(icons[f][DELTAY], DEC);
  }
  for(;;) { // Loop forever...
    display.clearDisplay(); // Clear the display buffer
    // Draw each snowflake:
    for(f=0; f< NUMFLAKES; f++) {
      display.drawBitmap(icons[f][XPOS], icons[f][YPOS], bitmap, w, h,
SH110X_WHITE);
    }
    display.display(); // Show the display buffer on the screen
    delay(200); // Pause for 1/10 second
    // Then update coordinates of each flake...
    for(f=0; f< NUMFLAKES; f++) {
      icons[f][YPOS] += icons[f][DELTAY];
      // If snowflake is off the bottom of the screen...
      if (icons[f][YPOS] >= display.height()) {
        // Reinitialize to a random position, just off the top
        icons[f][XPOS]   = random(1 - LOGO_WIDTH, display.width());
        icons[f][YPOS]   = -LOGO_HEIGHT;
        icons[f][DELTAY] = random(1, 6);
      }
    }
  }
}
Downloads

Files

- [SH1107 datasheet](#)
- [EagleCAD PCB files on GitHub](#)
- [Fritzing object in the Adafruit Fritzing Library](#)
Schematic and Fab Print