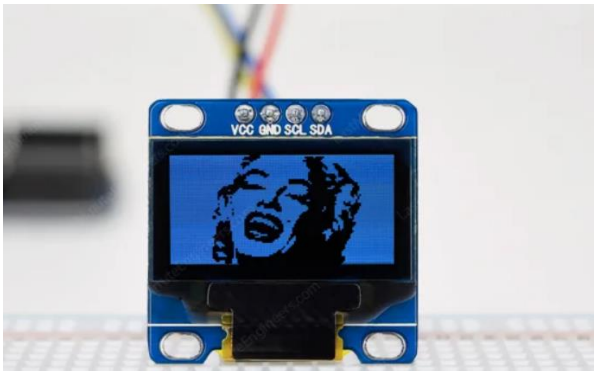
OLED Graphic Display Module



Hardware Overview

At the heart of the module is a powerful single-chip CMOS OLED driver controller – SSD1306. It can communicate with the microcontroller via I2C and SPI.

Because the SSD1306 controller is so versatile, the module comes in different sizes and colors, such as 128×64, 128×32, with white OLEDs, blue OLEDs, and dual-color OLEDs. The good news is that these displays are all interchangeable.

SPI Display -vs- I2C Display. Which one to choose?

SPI is faster than I2C in general, but it requires more I/O pins. I2C, on the other hand, requires only two pins and can be shared with other I2C peripherals. Because it is a trade-off between pins and speed, the choice is entirely yours.

Power

An OLED display, unlike a character LCD display, does not require a backlight because it generates its own light. This explains the display's high contrast, extremely wide viewing angle, and ability to display deep black levels. The absence of a backlight reduces power consumption significantly. The display uses about 20mA on average, though this varies depending on how much of the display is lit.

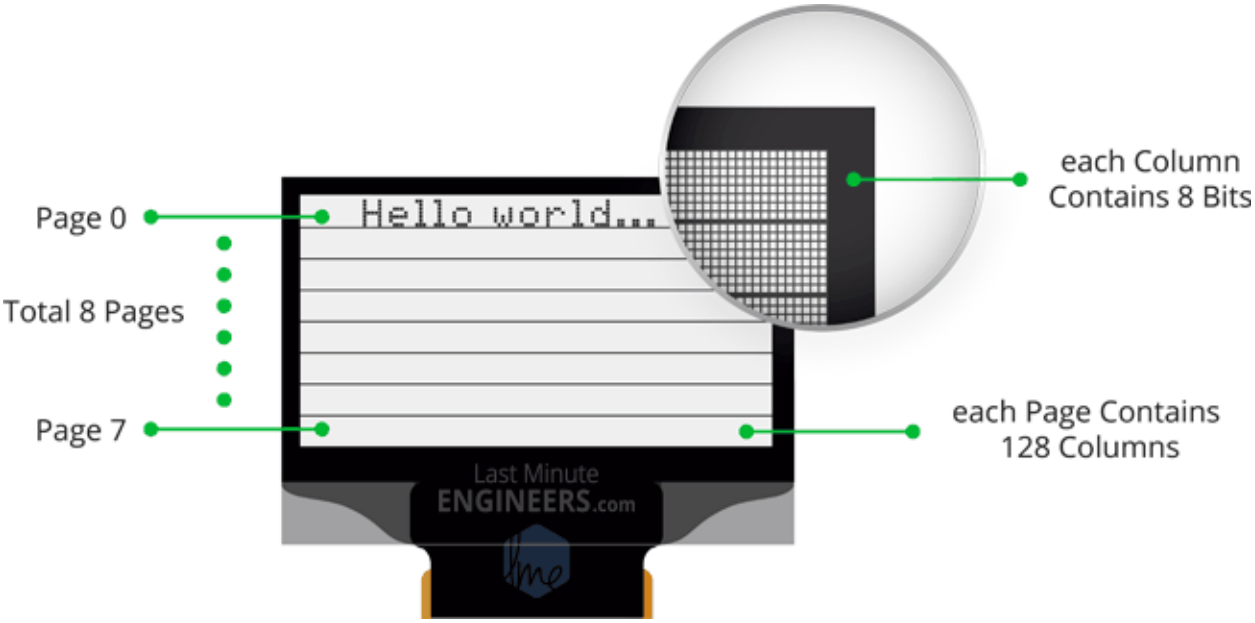
The SSD1306 controller operates at 1.65V to 3.3V, while the OLED panel requires a 7V to 15V supply voltage. All of these various power requirements are fulfilled by internal charge pump circuitry. This makes it possible to connect the display to an Arduino or any other 5V logic microcontroller without requiring a logic level converter.

OLED Memory Map

Regardless of the size of the OLED display, the SSD1306 driver includes a 1KB Graphic Display Data RAM (GDDRAM) that stores the bit pattern to be displayed on the screen. This 1 KB memory area is divided into 8 pages (from 0 to 7). Each page has 128 columns/segments (block 0 to 127). Furthermore, each column can store 8 bits of data (from 0 to 7). That certainly proves that we have:

8 pages x 128 segments x 8 bits of data = 8192 bits = 1024 bytes = 1KB memory

The entire 1K memory, including pages, segments, and data, is highlighted below.



Each bit represents a single OLED pixel on the screen that can be turned ON or OFF programmatically.

As previously stated, regardless of the size of the OLED module, each module contains 1KB of RAM. The 128×64 OLED module displays the entire contents of 1KB of RAM (all 8 pages), whereas the 128×32 OLED module displays only half of the RAM (the first 4 pages).

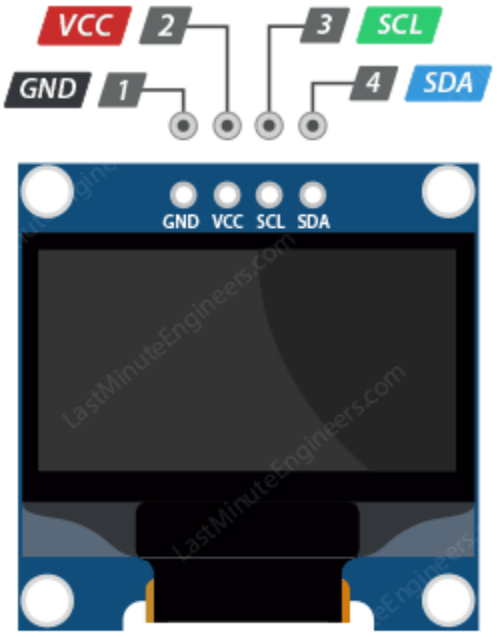
Technical Specifications

Here are the specifications:

Display Technology	OLED (Organic LED)
MCU Interface	I2C / SPI
Screen Size	0.96 Inch Across
Resolution	128×64 pixels
Operating Voltage	3.3V – 5V
Operating Current	20mA max
Viewing Angle	160°
Characters Per Row	21
Number of Character Rows	7

OLED Display Module Pinout

I2C OLED Display Module



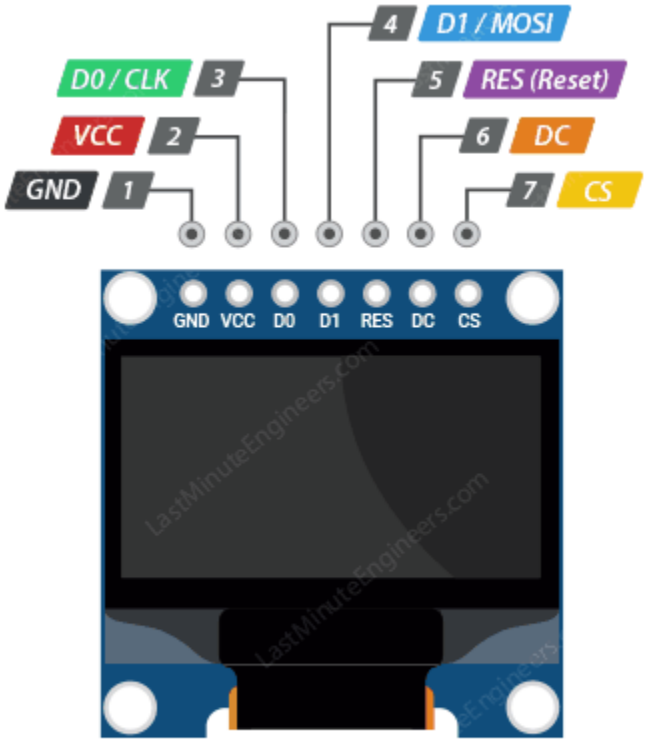
GND is the ground pin.

VCC is the power supply for the display, which we connect to the 5V pin on the Arduino.

SCL is a serial clock pin for the I2C interface.

SDA is a serial data pin for the I2C interface.

SPI OLED Display Module



GND is the ground pin.

VCC is the power supply for the display, which we connect to the 5V pin on the Arduino.

D0 / CLK is the SPI Clock pin. It's an input to the chip.

D1 / MOSI is the Serial Data In pin, for data sent from your microcontroller to the display.

RES (Reset) pin resets the internal buffer of the OLED driver.

DC (Data/Command) is used by the library to separate the commands (such as setting the cursor to a specific location, clearing the screen, etc.) from the data.

CS is the Chip Select pin.

Wiring an OLED display module to an Arduino

Before we get to uploading code and sending data to the display, let's hook the display up to the Arduino.

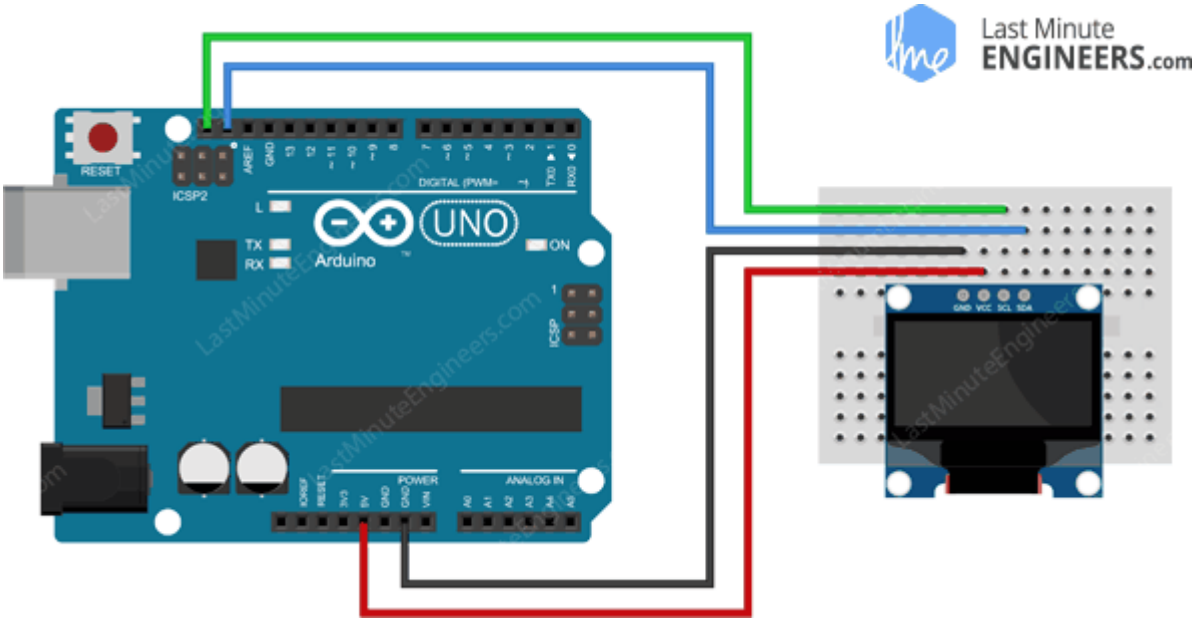
I2C OLED Display Wiring

If you're using an I2C OLED display, please refer to this wiring.

Connections are straightforward. Begin by connecting the VCC pin to the Arduino's 5V output and the GND pin to ground.

Connect the SCL pin to the I2C clock pin and the SDA pin to the I2C data pin on your Arduino. It is important to note that each Arduino board has different I2C pins. On Arduino boards with the R3 layout, the SDA (data line) and SCL (clock line) are on the pin headers close to the AREF pin. They are also known as A5 (SCL) and A4 (SDA).

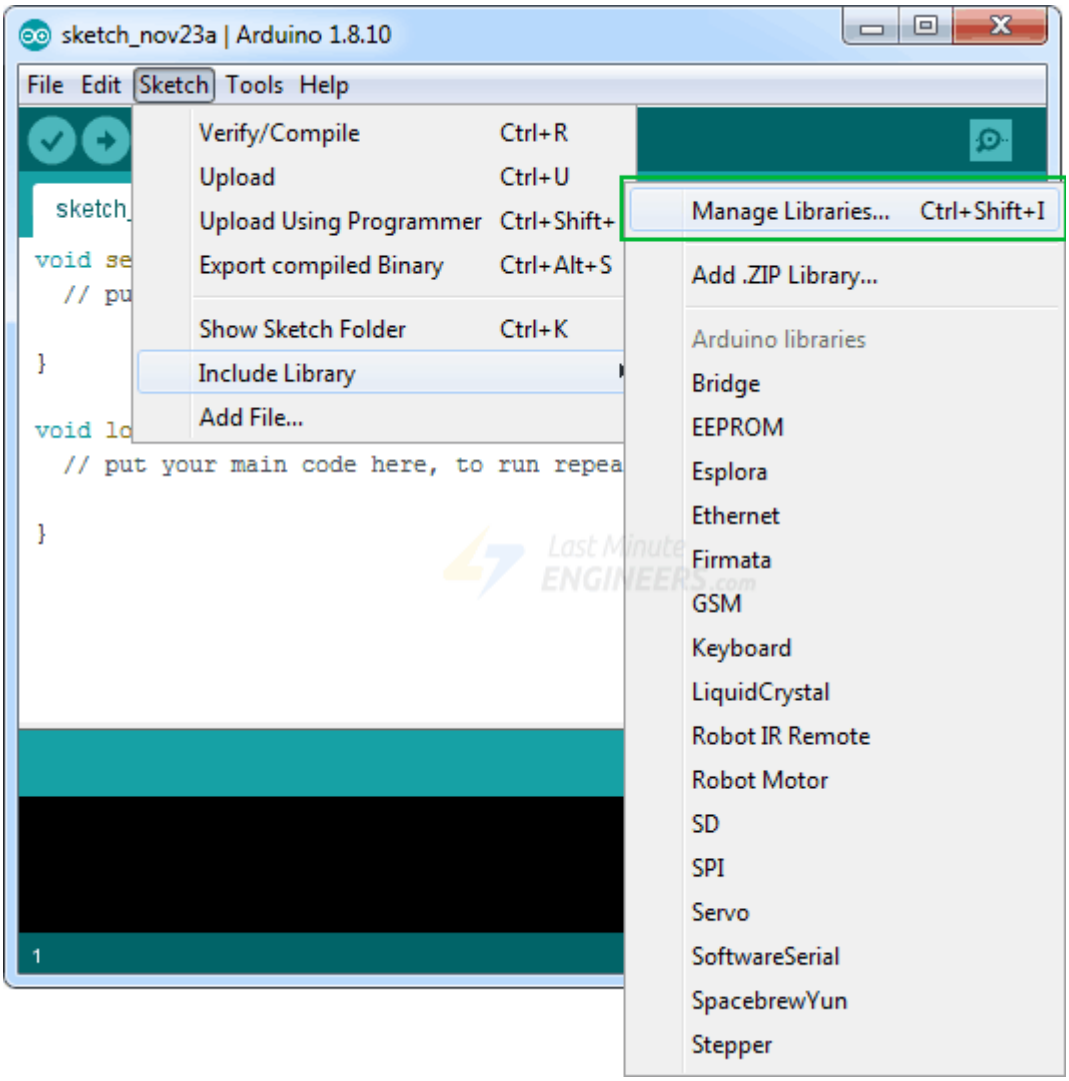
The diagram below shows how to connect everything.



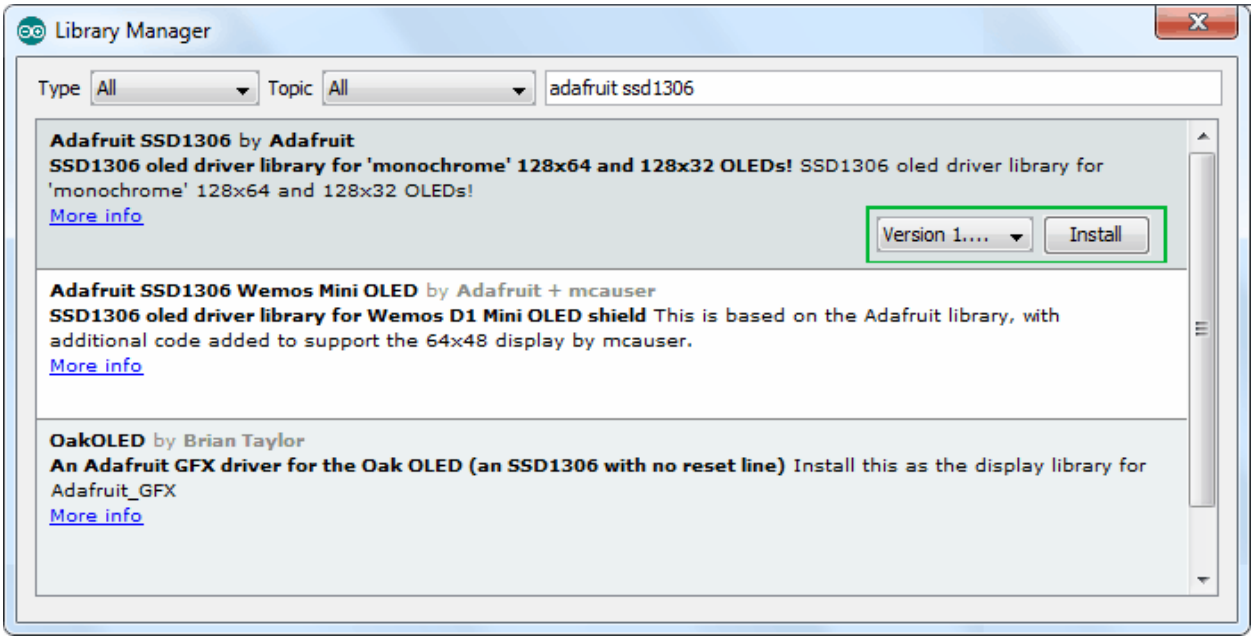
Installing an OLED Display Module Library

The SSD1306 controller in the OLED display has flexible but complex drivers. To use the SSD1306 controller, extensive knowledge of memory addressing is required. Fortunately, the Adafruit SSD1306 library was written to hide the complexities of the SSD1306 controller, allowing us to control the display with simple commands.

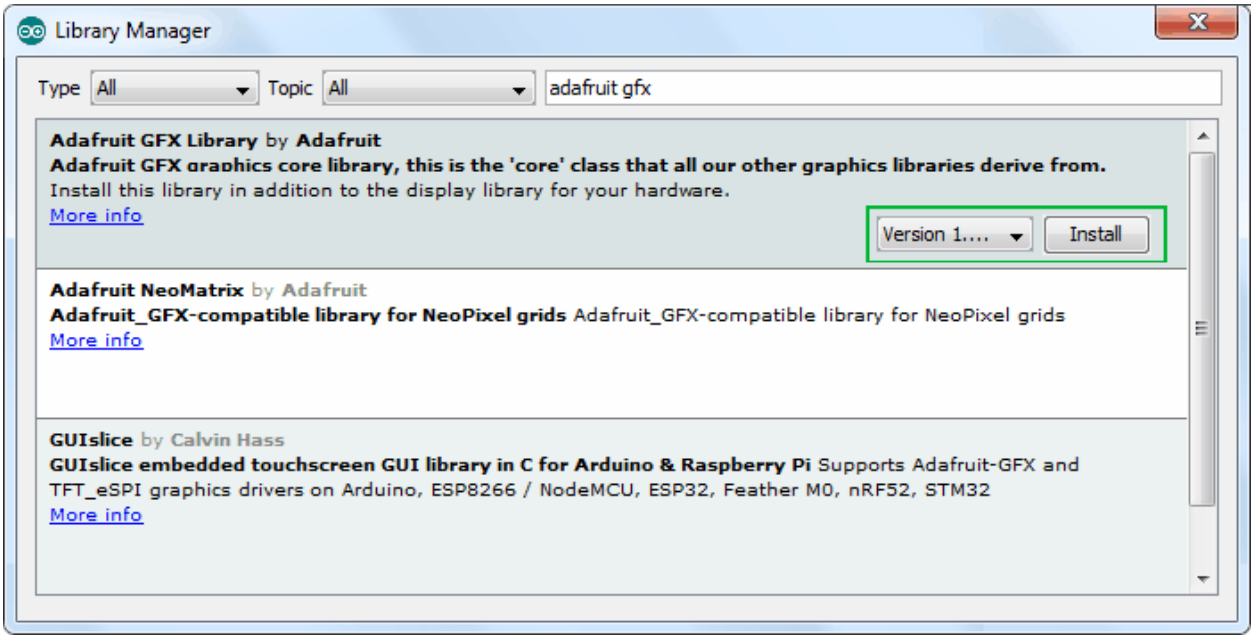
To install the library, navigate to Sketch > Include Library > Manage Libraries… Wait for the Library Manager to download the library index and update the list of installed libraries.



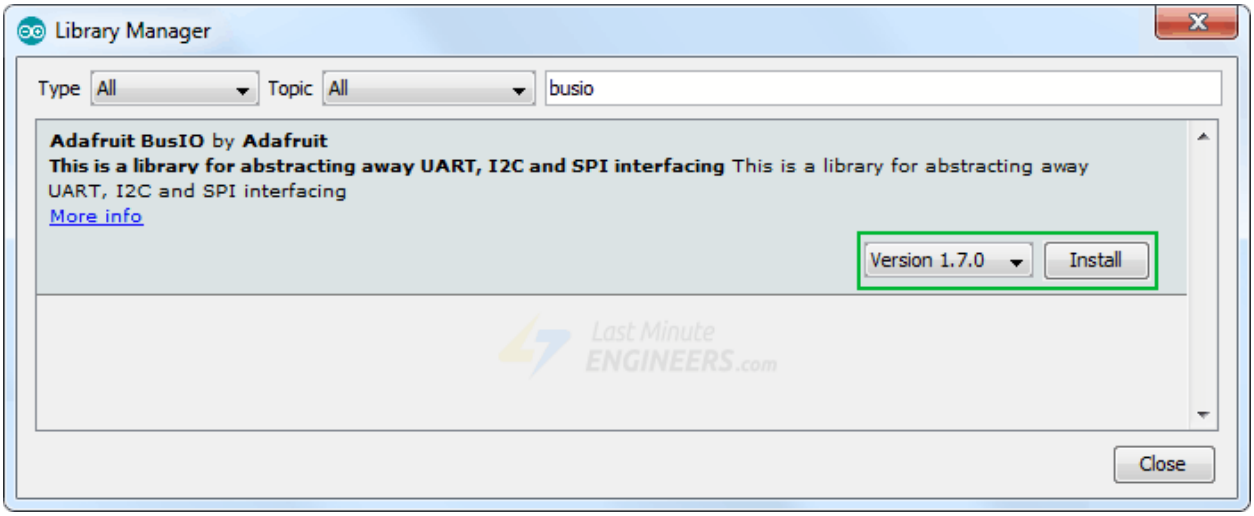
Filter your search by typing ‘adafruit ssd1306’. There should be a few entries. Look for Adafruit SSD1306 by Adafruit. Click on that entry and then choose Install.



This Adafruit SSD1306 library is a hardware-specific library for low-level functions. To display graphics primitives such as points, lines, circles, and rectangles, it must be paired with the Adafruit GFX Library. Install this library as well.



Internally, the Adafruit SSD1306 library makes use of the Adafruit Bus IO Library. So, look for Adafruit BusIO in the library manager and install it as well.


Library Manager
Type All Topic All busio
Adafruit BusIO by Adafruit
This is a library for abstracting away UART, I2C and SPI interfacing This is a library for abstracting away UART, I2C and SPI interfacing
More info
Version 1.7.0 Install
Last Minute ENGINEERS.com
Close

Arduino Example 1 – Displaying Text

Here's a simple sketch that will do the following:

- Display simple text.
- Display inverted text.
- Display numbers.
- Display numbers with base (Hex, Dec).
- Display ASCII symbols.
- Scroll text horizontally and vertically.
- Scroll part of the display.

This sketch will provide you with a thorough understanding of how to operate the OLED display and can serve as the foundation for more practical experiments and projects. Try out the sketch, and then we'll go over it in detail.

```
#include <SPI.h>

#include <Wire.h>

#include <Adafruit_GFX.h>

#include <Adafruit_SSD1306.h>


#define SCREEN_WIDTH 128 // OLED display width, in pixels

#define SCREEN_HEIGHT 64 // OLED display height, in pixels


// Declaration for SSD1306 display connected using I2C

#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)

#define SCREEN_ADDRESS 0x3C

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);


// Declaration for SSD1306 display connected using software SPI:

//#define OLED_MOSI   9

//#define OLED_CLK   10

//#define OLED_DC    11
```

```
//#define OLED_CS    12
//#define OLED_RESET 13
//Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, OLED_MOSI, OLED_CLK,
OLED_DC, OLED_RESET, OLED_CS);

void setup()
{
  Serial.begin(9600);

  // initialize the OLED object
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }

  // Uncomment this if you are using SPI
  //if(!display.begin(SSD1306_SWITCHCAPVCC)) {
  //  Serial.println(F("SSD1306 allocation failed"));
  //  for(;;); // Don't proceed, loop forever
  //}

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

  // Display Text
  display.setTextSize(1);
  display.setTextColor(WHITE);
```

```
display.setCursor(0,28);
display.println("Hello world!");
display.display();
delay(2000);
display.clearDisplay();

// Display Inverted Text
display.setTextColor(BLACK, WHITE); // 'inverted' text
display.setCursor(0,28);
display.println("Hello world!");
display.display();
delay(2000);
display.clearDisplay();

// Changing Font Size
display.setTextColor(WHITE);
display.setCursor(0,24);
display.setTextSize(2);
display.println("Hello!");
display.display();
delay(2000);
display.clearDisplay();

// Display Numbers
display.setTextSize(1);
display.setCursor(0,28);
display.println(123456789);
```

```
display.display();
delay(2000);
display.clearDisplay();

// Specifying Base For Numbers
display.setCursor(0,28);
display.print("0x"); display.print(0xFF, HEX);
display.print("(HEX) = ");
display.print(0xFF, DEC);
display.println("(DEC)");
display.display();
delay(2000);
display.clearDisplay();

// Display ASCII Characters
display.setCursor(0,24);
display.setTextSize(2);
display.write(3);
display.display();
delay(2000);
display.clearDisplay();

// Scroll full screen
display.setCursor(0,0);
display.setTextSize(1);
display.println("Full");
display.println("screen");
```

```
  display.println("scrolling!");
  display.display();
  display.startscrollright(0x00, 0x07);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrollleft(0x00, 0x07);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrolldiagright(0x00, 0x07);
  delay(2000);
  display.startscrolldiagleft(0x00, 0x07);
  delay(2000);
  display.stopscroll();
  display.clearDisplay();

  // Scroll part of the screen
  display.setCursor(0,0);
  display.setTextSize(1);
  display.println("Scroll");
  display.println("some part");
  display.println("of the screen.");
  display.display();
  display.startscrollright(0x00, 0x00);
}
```

```
void loop() {

}
```

This is what the output looks like.

Code Explanation:

The sketch begins with the inclusion of four libraries: SPI.h, Wire.h, Adafruit GFX.h, and Adafruit SSD1306.h. Although the SPI.h library is not required for I2C OLED displays, we must include it to compile our program.

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

The next step is to create an object of Adafruit_SSD1306.h. The Adafruit_SSD1306 constructor accepts 3 arguments: screen width, screen height, and the Arduino pin number to which the display's reset pin is connected.

So, a couple of constants are defined to be passed to the constructor. Also, since the OLED display we are using doesn't have a RESET pin, we will send -1 to the constructor to indicate that none of the Arduino pins are used to reset the display.

```
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for SSD1306 display connected using I2C
```

```
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS 0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
```

This sketch uses the I2C protocol for communicating with the display. However, the sketch is ready if you wish to use SPI. Simply uncomment the following lines of code.

```
// Declaration for SSD1306 display connected using software SPI:
//#define OLED_MOSI   9
//#define OLED_CLK   10
//#define OLED_DC    11
//#define OLED_CS    12
//#define OLED_RESET 13
//Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, OLED_MOSI, OLED_CLK,
OLED_DC, OLED_RESET, OLED_CS);
```

In the setup function, we need to initialize the OLED object using the `begin()` function. This function accepts two parameters. The first parameter, SSD1306_SWITCHCAPVCC, turns on the internal charge pump circuitry, and the second parameter sets the OLED display's I2C address. Most OLED display modules of this type have an I2C address of 0x3C, but some have 0x3D.

After that, we clear the buffer before printing our first message to the screen.

```
// initialize the OLED object
if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
  Serial.println(F("SSD1306 allocation failed"));
  for(;;); // Don't proceed, loop forever
}

// Uncomment this if you are using SPI
//if(!display.begin(SSD1306_SWITCHCAPVCC)) {
//  Serial.println(F("SSD1306 allocation failed"));
//  for(;;); // Don't proceed, loop forever
//}

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

Displaying simple Text (Hello World)


```
// Display Text
display.clearDisplay();
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,28);
display.println("Hello world!");
display.display();
delay(2000);
```

To display text on the screen, we must first set the font size. This can be accomplished by calling `setTextSize()` and passing a font size (starting from 1) as a parameter.

Next, we need to set the font color by calling the function `setTextColor()`.
Pass `WHITE` for a dark background and `BLACK` for a bright one.

Before printing the message, we must first set the cursor position by calling the `setCursor(X,Y)` function. Pixels on the screen are referenced by their horizontal (X) and vertical (Y) coordinates. The origin (0,0) is located in the upper left corner, with positive X increasing to the right and positive Y increasing downward.

To print the message on the screen, we can use the `print(" ")` or `println(" ")` functions, similar to how we print data on the serial monitor. Keep in mind that `println()` will move the cursor to the next line.

The final step is to use the `display()` command to instruct the library to bulk transfer the screen buffer to the SSD1306 controller's internal memory and display the contents on the OLED screen.

Displaying Inverted Text


```
// Display Inverted Text
display.clearDisplay();
display.setTextColor(BLACK, WHITE); // 'inverted' text
display.setCursor(0,28);
display.println("Hello world!");
display.display();
delay(2000);
```

To display inverted text, we use the `setTextColor(FontColor,BackgroundColor)` function once more. If you've been paying attention, you'll notice that we previously passed only one parameter to this function, but now we're passing two. This is possible due to function overloading.

In this case, using `setTextColor(BLACK, WHITE)` results in black text on a filled background.

Scaling Font Size


```
// Changing Font Size
display.clearDisplay();
display.setTextColor(WHITE);
display.setCursor(0,24);
display.setTextSize(2);
display.println("Hello!");
display.display();
delay(2000);
```

Earlier in this tutorial, we used the `setTextSize()` function to change the font size, passing 1 as a parameter. You can scale the font by passing any non-negative integer to this function.

Characters are rendered in a 7:10 ratio. In other words, passing font size 1 renders the text at 7×10 pixels per character, passing font size 2 renders the text at 14×20 pixels per character, and so on.

Displaying Numbers


```
// Display Numbers
display.clearDisplay();
display.setTextSize(1);
display.setCursor(0,28);
display.println(123456789);
display.display();
delay(2000);
```

The `print()` or `println()` functions can be used to display numbers on the OLED display. Because an overloaded implementation of these functions accepts 32-bit unsigned int values, you can only display numbers ranging from 0 to 4,294,967,295.

Specifying Base For Numbers


```
// Specifying Base For Numbers
display.clearDisplay();
display.setCursor(0,28);
display.print("0x"); display.print(0xFF, HEX);
display.print("(HEX) = ");
display.print(0xFF, DEC);
display.println("(DEC)");
display.display();
delay(2000);
```

The `print()` and `println()` functions have an optional second parameter that specifies the base (format) to use; valid values are `BIN` (binary, or base 2), `OCT` (octal, or base 8), `DEC` (decimal, or base 10) and `HEX` (hexadecimal, or base 16). For floating-point numbers, this parameter specifies the number of decimal places to use. For instance:

- print(78, BIN) prints “1001110”
- print(78, OCT) prints “116”
- print(78, DEC) prints “78”
- print(78, HEX) prints “4E”
- println(1.23456, 0) prints “1”
- println(1.23456, 2) prints “1.23”
- println(1.23456, 4) prints “1.2346”

Displaying ASCII Symbols


```
// Display ASCII Characters
display.clearDisplay();
display.setCursor(0,24);
display.setTextSize(2);
display.write(3);
display.display();
delay(2000);
```

The `print()` and `println()` functions send data to the display as human-readable ASCII text, whereas the `write()` function sends binary data to the display. This function can thus be used to display ASCII symbols. For example, sending 3 displays a heart symbol.

Full Screen Scrolling

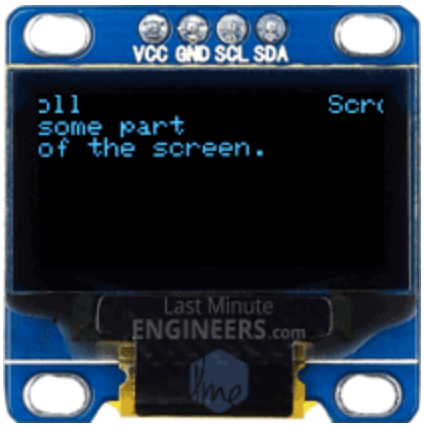

```
// Scroll full screen
```

```
display.clearDisplay();
display.setCursor(0,0);
display.setTextSize(1);
display.println("Full");
display.println("screen");
display.println("scrolling!");
display.display();
display.startscrollright(0x00, 0x07);
delay(2000);
display.stopscroll();
delay(1000);
display.startscrollleft(0x00, 0x07);
delay(2000);
display.stopscroll();
delay(1000);
display.startscrolldiagright(0x00, 0x07);
delay(2000);
display.startscrolldiagleft(0x00, 0x07);
delay(2000);
display.stopscroll();
```

You can scroll the display horizontally by calling the functions `startscrollright()` and `startscrollleft()`, and diagonally by calling the functions `startscrolldiagright()` and `startscrolldiagleft()`. All of these functions take two parameters: start page and stop page. Refer to the OLED Memory Map section for an explanation of the pages. Because the display has eight pages from 0 to 7, you can scroll the entire screen by scrolling all the pages, i.e. passing parameters 0x00 and 0x07.

The `stopscroll()` function can be used to stop the display from scrolling.

Scrolling Part of the Screen




```
// Scroll part of the screen
display.setCursor(0,0);
display.setTextSize(1);
display.println("Scroll");
display.println("some part");
display.println("of the screen.");
display.display();
display.startscrollright(0x00, 0x00);
```

Sometimes, we don't want to scroll the whole display, but just a part of it. You can accomplish this by passing the appropriate start and stop page information to the scrolling functions.

Passing 0x00 for both parameters will only scroll the first page of the display (the first 8 rows).

Arduino Example 2 – Basic Drawings

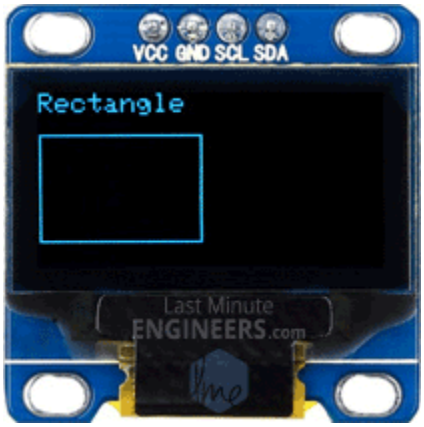
```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for SSD1306 display connected using I2C
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS 0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Declaration for SSD1306 display connected using software SPI:
//#define OLED_MOSI   9
//#define OLED_CLK   10
//#define OLED_DC    11
//#define OLED_CS    12
//#define OLED_RESET 13
//Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, OLED_MOSI, OLED_CLK,
OLED_DC, OLED_RESET, OLED_CS);

void setup()
{
  Serial.begin(9600);

  // initialize the OLED object
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }

  // Uncomment this if you are using SPI
  //if(!display.begin(SSD1306_SWITCHCAPVCC)) {
  //  Serial.println(F("SSD1306 allocation failed"));
  //  for(;;); // Don't proceed, loop forever
```

```
//}

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

// Draw Rectangle
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("Rectangle");
display.drawRect(0, 15, 60, 40, WHITE);
display.display();
delay(2000);
display.clearDisplay();

// Draw Filled Rectangle
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("Filled Rectangle");
display.fillRect(0, 15, 60, 40, WHITE);
display.display();
delay(2000);
display.clearDisplay();

// Draw Round Rectangle
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("Round Rectangle");
display.drawRoundRect(0, 15, 60, 40, 8, WHITE);
display.display();
delay(2000);
display.clearDisplay();

// Draw Filled Round Rectangle
display.setTextSize(1);
```

```
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("Filled Round Rectangl");
display.fillRoundRect(0, 15, 60, 40, 8, WHITE);
display.display();
delay(2000);
display.clearDisplay();

// Draw Circle
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("Circle");
display.drawCircle(20, 35, 20, WHITE);
display.display();
delay(2000);
display.clearDisplay();

// Draw Filled Circle
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("Filled Circle");
display.fillCircle(20, 35, 20, WHITE);
display.display();
delay(2000);
display.clearDisplay();

// Draw Triangle
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("Triangle");
display.drawTriangle(30, 15, 0, 60, 60, 60, WHITE);
display.display();
delay(2000);
display.clearDisplay();
```

```
  // Draw Filled Triangle
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.println("Filled Triangle");
  display.fillTriangle(30, 15, 0, 60, 60, 60, WHITE);
  display.display();
  delay(2000);
  display.clearDisplay();
}

void loop() {
}
```

This is what the output looks like.

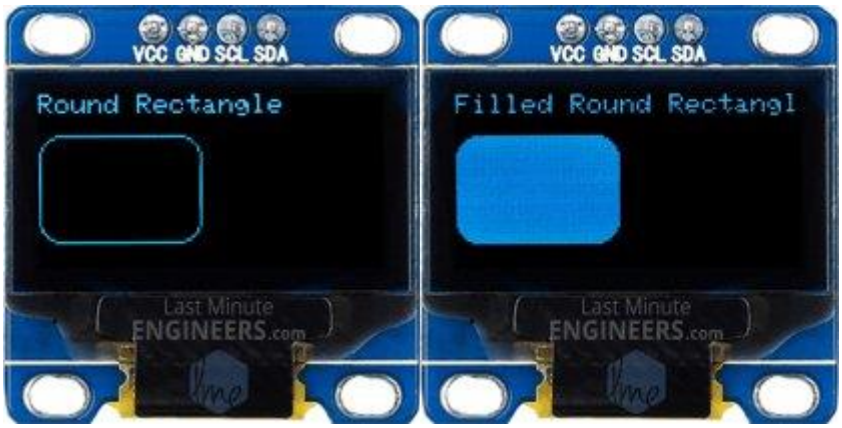


Drawing Rectangle


```
display.clearDisplay();
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("Rectangle");
display.drawRect(0, 15, 60, 40, WHITE);
display.display();
delay(2000);

display.clearDisplay();
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("Filled Rectangle");
display.fillRect(0, 15, 60, 40, WHITE);
display.display();
delay(2000);
```

The `drawRect()` function can be used to draw a rectangle on the screen. This function accepts five parameters: X and Y coordinates, width, height, and color. This function actually draws a hollow rectangle with a 1 pixel border. The `fillRect()` function can be used to draw a filled rectangle.

Drawing Round Rectangle

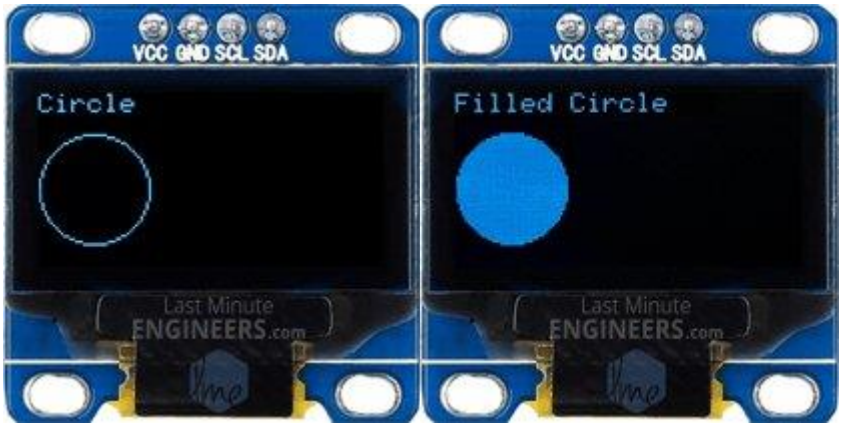



```
display.clearDisplay();
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("Round Rectangle");
display.drawRoundRect(0, 15, 60, 40, 8, WHITE);
display.display();
delay(2000);

display.clearDisplay();
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("Filled Round Rectangl");
display.fillRoundRect(0, 15, 60, 40, 8, WHITE);
display.display();
delay(2000);
```

The `drawRoundRect()` function can be used to draw a round rectangle on the screen. This function accepts the same parameters as `drawRect()`, with the exception of one additional parameter – the radius of corner rounding. This function actually draws a

hollow round rectangle with a 1 pixel border. The `fillRoundRect()` function can be used to draw a filled round rectangle.

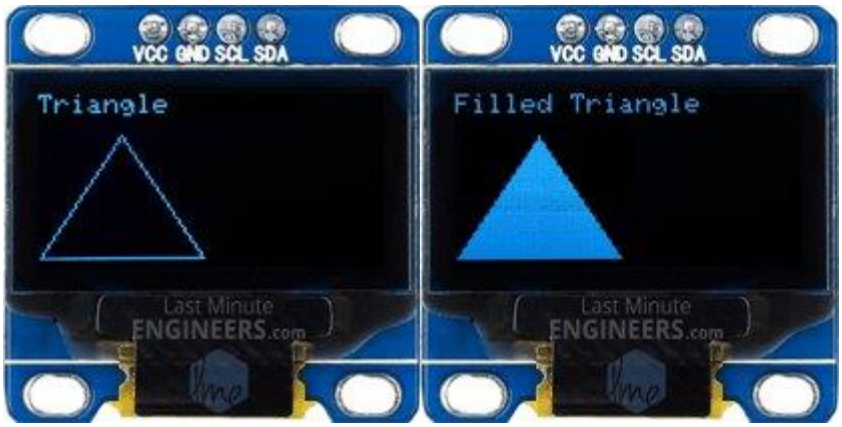
Drawing Circle




```
display.clearDisplay();
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("Circle");
display.drawCircle(20, 35, 20, WHITE);
display.display();
delay(2000);

display.clearDisplay();
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("Filled Circle");
display.fillCircle(20, 35, 20, WHITE);
display.display();
delay(2000);
```

The `drawCircle()` function can be used to draw a circle on the screen. This function accepts four parameters: the X coordinate of the center, the Y coordinate of the center, the radius, and the color. This function draws a hollow circle with a 1 pixel border. The `fillCircle()` function can be used to draw a filled circle.

Drawing Triangle




```
display.clearDisplay();
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("Triangle");
display.drawTriangle(30, 15, 0, 60, 60, 60, WHITE);
display.display();
delay(2000);

display.clearDisplay();
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("Filled Triangle");
display.fillTriangle(30, 15, 0, 60, 60, 60, WHITE);
display.display();
delay(2000);
```

The `drawTriangle()` function can be used to draw a triangle on the screen. This function accepts seven parameters: three X and Y coordinates (x0, y0, x1, y1, x2 & y2) of triangle vertices and a color. (X0,y0) is the top vertex, (x1,y1) is the left vertex, and (x2,y2) is the right vertex.

This function draws a hollow triangle with a 1 pixel border.
The `fillTriangle()` function can be used to draw a filled triangle.

Arduino Example 3 – Displaying Bitmap

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for SSD1306 display connected using I2C
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS 0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Declaration for SSD1306 display connected using software SPI:
//#define OLED_MOSI   9
//#define OLED_CLK   10
//#define OLED_DC    11
//#define OLED_CS    12
//#define OLED_RESET 13
//Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, OLED_MOSI, OLED_CLK,
OLED_DC, OLED_RESET, OLED_CS);

// Bitmap of MarilynMonroe Image
const unsigned char MarilynMonroe [] PROGMEM = {
  0xff, 0xff, 0xff, 0xff, 0xff, 0xf8, 0x1f, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0xc0, 0x1f, 0xff, 0xff, 0xf0, 0x41, 0xff,
0xff, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0x7f, 0xff, 0xff, 0xf8, 0x03, 0xff,
0xff, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0xf9, 0xff, 0xff, 0xff, 0xe0, 0x07, 0xff,
0xff, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0x87, 0xff, 0xff, 0xff, 0xf8, 0x03, 0xff,
0xff, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0x07, 0xff, 0xff, 0xff, 0xf8, 0x01, 0xf1,
0xff, 0xff, 0xff, 0xff,
```

```
  0xff, 0xff, 0xff, 0xff, 0xff, 0x9f, 0xff, 0xff, 0xff, 0xf8, 0x00, 0xf8,
0xff, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0xbf, 0xff, 0xff, 0xff, 0xfc, 0x02, 0x78,
0x7f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0xfc, 0x3f, 0xff, 0xff, 0xfe, 0x03, 0x7c,
0x1f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0x07, 0xff, 0xff, 0xfe, 0x01, 0xfe,
0x1f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xfd, 0xe0, 0x03, 0xff, 0xff, 0xfc, 0x00, 0xfe,
0x0f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xfe, 0x87, 0xe0, 0xff, 0xff, 0xfc, 0x00, 0x06,
0x07, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xfc, 0x1f, 0xf9, 0xff, 0xff, 0xfc, 0x00, 0x02,
0x07, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xf8, 0x1f, 0xff, 0xff, 0xff, 0xfc, 0x00, 0xc3,
0xc3, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xf0, 0x3f, 0xff, 0xff, 0xe0, 0x0c, 0x00, 0xe7,
0x81, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xf0, 0x0f, 0xff, 0xff, 0xe0, 0x02, 0x00, 0x02,
0x00, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xf0, 0x0f, 0xff, 0xff, 0xe0, 0x01, 0x00, 0x00,
0x00, 0x3f, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0x80, 0x00, 0x3f, 0xff, 0xff, 0xe0, 0x00, 0x00,
0x1e, 0x3f, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xfc, 0x00, 0x00, 0x0f, 0xff, 0x3f, 0xf8, 0x00, 0x18,
0x7f, 0x1f, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xf8, 0x01, 0x80, 0x03, 0xfc, 0x3f, 0xfc, 0x00, 0x70,
0xfe, 0x1f, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xf0, 0x43, 0xff, 0xff, 0xf8, 0x7f, 0xf8, 0x00, 0x00,
0x7e, 0x1f, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xe0, 0x07, 0xff, 0xff, 0xf0, 0xff, 0xfc, 0x00, 0x00,
0x7c, 0x3f, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xe0, 0x0f, 0xff, 0xff, 0xf1, 0xef, 0xf8, 0x00, 0x01,
0xfc, 0x3f, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xe4, 0xff, 0xff, 0xff, 0xf3, 0x80, 0xa0, 0x00, 0x07,
0xfc, 0xaf, 0xff, 0xff,
```

```
  0xff, 0xff, 0xff, 0xec, 0x5f, 0xff, 0xff, 0xe7, 0xf0, 0x00, 0x00, 0x03,
0xfe, 0xdf, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xee, 0x7f, 0xff, 0xff, 0xc7, 0xf8, 0x00, 0x00, 0x03,
0xff, 0xdf, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xfe, 0x7f, 0xff, 0xf7, 0xc7, 0xff, 0x06, 0x00, 0x03,
0xff, 0xbf, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xfe, 0x5f, 0xff, 0xc7, 0x07, 0xff, 0x80, 0x00, 0x07,
0xdb, 0xbf, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xee, 0xff, 0xff, 0x80, 0x03, 0xff, 0xc0, 0x00, 0x03,
0xc3, 0x0f, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xfe, 0xff, 0xff, 0x98, 0x03, 0xff, 0xf8, 0x00, 0x07,
0xe0, 0x0f, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xef, 0xff, 0xff, 0xf8, 0x01, 0xff, 0xfc, 0x01, 0x07,
0xfc, 0x1f, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xcf, 0xef, 0xff, 0xff, 0xe1, 0xff, 0xfc, 0x01, 0x07,
0xf8, 0x1f, 0xff, 0xff,
  0xff, 0xff, 0xff, 0x9f, 0xff, 0xff, 0x7f, 0xf1, 0xff, 0xf8, 0x02, 0x07,
0x88, 0x3f, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xcf, 0xef, 0xf8, 0x0f, 0xff, 0xff, 0xe0, 0x00, 0x07,
0x84, 0x3f, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xe7, 0xef, 0xf0, 0x04, 0x7f, 0xff, 0xc0, 0x00, 0x07,
0x84, 0x7f, 0xff, 0xff,
  0xff, 0xff, 0xff, 0x3f, 0xff, 0xe0, 0x00, 0x1f, 0xff, 0x80, 0x00, 0x06,
0x04, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0x3f, 0x7f, 0xe1, 0xf0, 0x07, 0xff, 0x80, 0x00, 0x07,
0x06, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0xc3, 0xfe, 0x03, 0xff, 0x00, 0x00, 0x03,
0x80, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xf2, 0x3f, 0xc6, 0x7f, 0x81, 0xce, 0x00, 0x00, 0x01,
0xc1, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xe0, 0x3f, 0xc0, 0x07, 0xc1, 0xfe, 0x00, 0x00, 0x0d,
0xc0, 0x7f, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xe0, 0x3f, 0xc0, 0x01, 0xe0, 0xfc, 0x00, 0x00, 0x0f,
0xc0, 0x7f, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xc0, 0x3f, 0xc0, 0x00, 0x50, 0xfc, 0x00, 0x00, 0x0e,
0xc0, 0xff, 0xff, 0xff,
```

```
  0xff, 0xff, 0xff, 0xc0, 0x3f, 0xc0, 0x00, 0x18, 0xf8, 0x00, 0x00, 0x0e,
0xc1, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xc0, 0x3f, 0xc0, 0x00, 0x00, 0xf8, 0x00, 0x00, 0x66,
0x81, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xc0, 0x1f, 0xc7, 0x80, 0x00, 0xf8, 0x00, 0x01, 0xe0,
0x00, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xc0, 0x1f, 0xc1, 0xe0, 0x01, 0xf8, 0x00, 0x03, 0xf0,
0x01, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0x80, 0x1f, 0xc0, 0x3e, 0x03, 0xf0, 0x00, 0x00, 0xe0,
0x03, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0x00, 0x1f, 0xe0, 0xe0, 0x03, 0xf2, 0x00, 0x00, 0xc0,
0x03, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0x80, 0x1f, 0xf0, 0x00, 0x07, 0xe6, 0x00, 0x00, 0xc0,
0x03, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0x80, 0x1f, 0xff, 0x00, 0x1f, 0xee, 0x00, 0x00, 0x80,
0x07, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xb8, 0x0f, 0xff, 0xf0, 0x3f, 0xdc, 0x00, 0x00, 0x00,
0x0f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xbc, 0x0f, 0xff, 0xff, 0xff, 0xdc, 0x00, 0x00, 0x00,
0x0f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0x9e, 0x0f, 0xff, 0xff, 0xff, 0xf8, 0x00, 0x00, 0x00,
0x1f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0x08, 0x0f, 0xff, 0xff, 0xff, 0x70, 0x00, 0x00, 0x00,
0x1f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0x00, 0x0b, 0xff, 0xff, 0xfe, 0xe0, 0x00, 0x00, 0x00,
0x1f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0x00, 0x0b, 0xff, 0xff, 0xf9, 0xc0, 0x00, 0x00, 0x00,
0x3f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0x3c, 0x09, 0xff, 0xff, 0xf1, 0x80, 0x00, 0x00, 0x00,
0x7f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0x1e, 0x08, 0x3f, 0xff, 0xc0, 0x00, 0x00, 0x00, 0x00,
0x7f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0x1f, 0x08, 0x03, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00,
0x7f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0x00, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x1f, 0xff, 0xff, 0xff,
```

```
  0xff, 0xff, 0xff, 0x80, 0x1c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x1f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xce, 0x1c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x1f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xfe, 0x1c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x3f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0x7e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x7f, 0xff, 0xff, 0xff
};

void setup()
{
  Serial.begin(9600);

  // initialize the OLED object
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }

  // Uncomment this if you are using SPI
  //if(!display.begin(SSD1306_SWITCHCAPVCC)) {
  //  Serial.println(F("SSD1306 allocation failed"));
  //  for(;;); // Don't proceed, loop forever
  //}

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

  // Display bitmap
  display.drawBitmap(0, 0,  MarilynMonroe, 128, 64, WHITE);
  display.display();

  // Invert Display
  //display.invertDisplay(1);
}
```

```
void loop() {
```

Code Explanation:

The `drawBitmap()` function is used to display a bitmap image on an OLED display. This function accepts six parameters: top left corner X coordinate, top left corner Y coordinate, monochrome bitmap byte array, bitmap width in pixels, bitmap height in pixels, and color.

The bitmap image in our example is 128×64 bytes in size. So, the X and Y coordinates are set to 0, while the width and height are set to 128 and 64.

```
// Display bitmap
display.drawBitmap(0, 0,  MarilynMonroe, 128, 64, WHITE);
display.display();
```

But, before we can use the `drawBitmap()` function, we need an image to draw. Remember that the OLED display's screen resolution is 128×64 pixels, so images larger than that will not display properly. To get a properly sized image, open your favorite drawing program, such as Inkscape, Photoshop, or MS Paint, and set the canvas size to 128×64 pixels.

We used a picture of Marilyn Monroe as an example. We changed it to 128×64 pixels in MS Paint and saved it as a .bmp file.

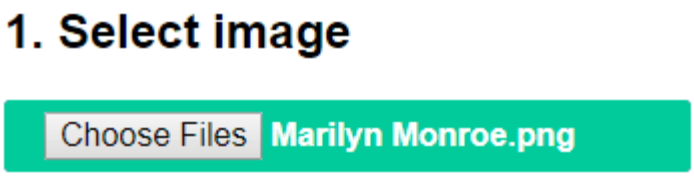
Once you have a bitmap, you must convert it into an array that the SSD1306 OLED controller can understand. This can be accomplished in two ways: online with image2cpp and offline with LCD Assistant.

Online Bitmap Array Generator – image2cpp

There's an online tool called image2cpp that can turn your image into an array. This tool allows you to:

- Convert multiple images simultaneously.
- Scale your image file – Stretch/Scale to fit/Original
- Adjust the Brightness threshold between black and white.
- Re-center the image vertically and / or horizontally.
- Reverse image colors

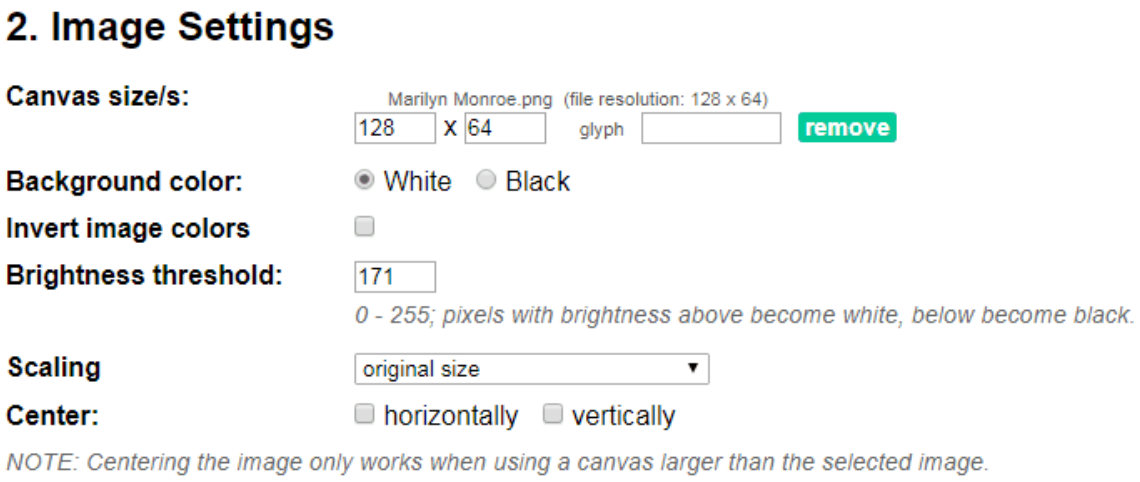
To begin with, open image2cpp in your browser and select an image to display on the OLED screen.



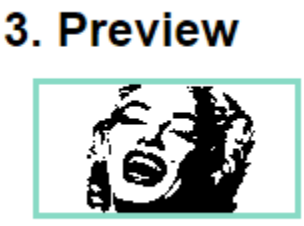
The dimensions of your image will be displayed in the Canvas size option under Image Settings. If your image is larger than 128×64, change it to 128×64 by selecting the appropriate scaling option. You can see the result in the Preview section.

If necessary, you can change the background color or invert the image colors.

Lastly, change the most important setting—the brightness threshold—so that it fits your needs. When you set a threshold, pixels above this level will be white and pixels below it will be black. In our case, we set it to 171 to get some nice details.

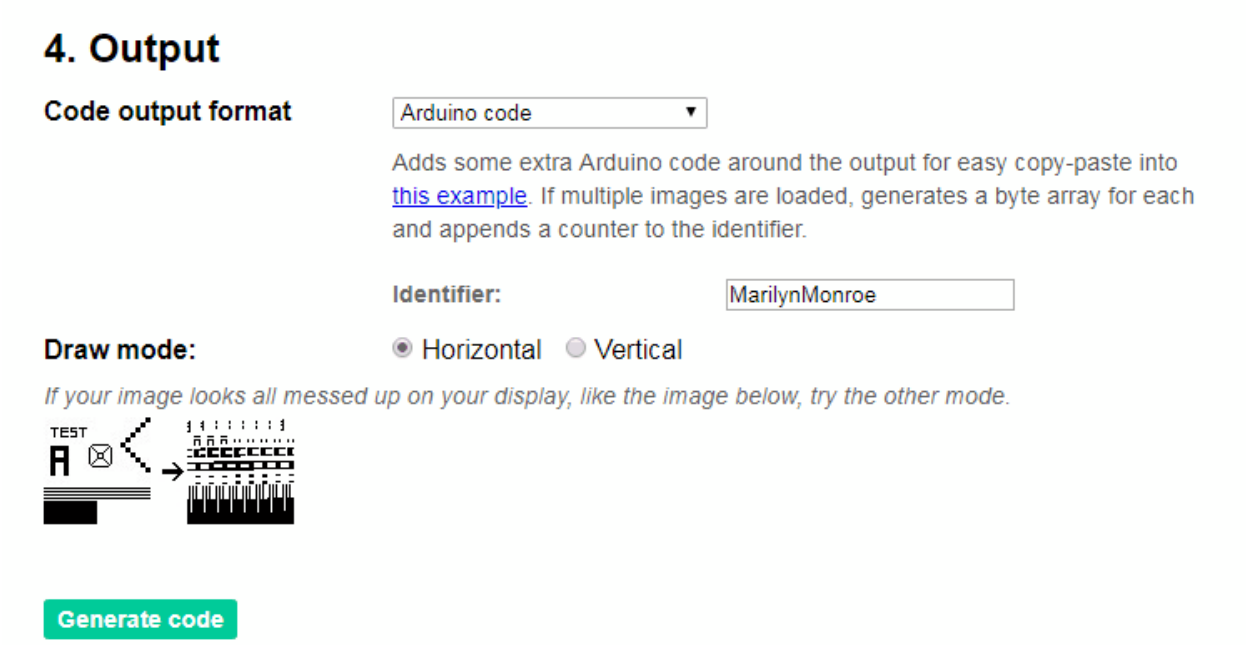


This preview reflects any changes you make to your settings. You can make changes while keeping an eye on it.



When you are satisfied with the results, you can proceed to generate the data array. Simply select Arduino Code as the code output format and press the Generate code button.

For your information, there is a setting called “Draw Mode”. It actually generates images based on the scanning pattern of the display. If your image appears distorted on your screen, try changing the mode.



That’s all. Your bitmap’s byte array will be created. You can use the output directly with our example code. Just make sure to give it a proper name. Then, within the `drawBitmap()` function, use your array.

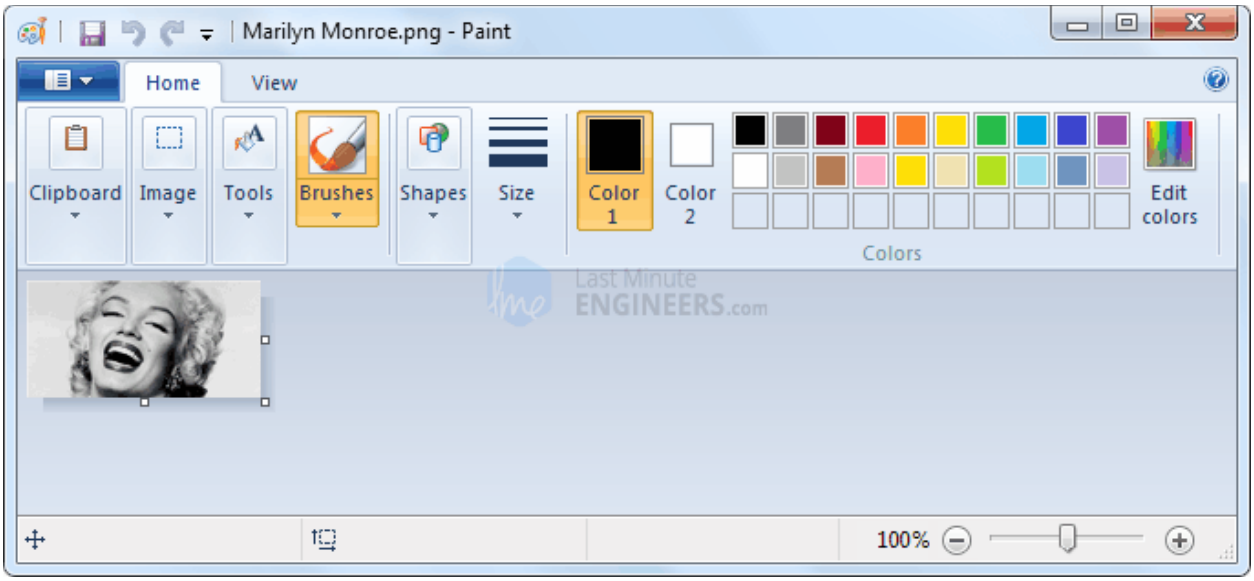
```
// 'Marilyn Monroe', 128x64px
const unsigned char MarilynMonroe [] PROGMEM = {
        0xff, 0xff, 0xff, 0xff, 0xff, 0xf8, 0x1f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xc0, 0x1f, 0xff, 0xff, 0xd0, 0x41, 0xff, 0xff, 0xff,
0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0x00, 0x7f, 0xff, 0xff, 0xf8, 0x03, 0xff, 0xff, 0xff,
0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0x79, 0xff, 0xff, 0xff, 0xe0, 0x07, 0xff, 0xff, 0xff,
0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0x87, 0xff, 0xff, 0xff, 0xf8, 0x03, 0xf7, 0xff, 0xff,
0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0x07, 0xff, 0xff, 0xff, 0xf8, 0x01, 0xf1, 0xff, 0xff,
```

Offline Bitmap Array Generator – LCD Assistant

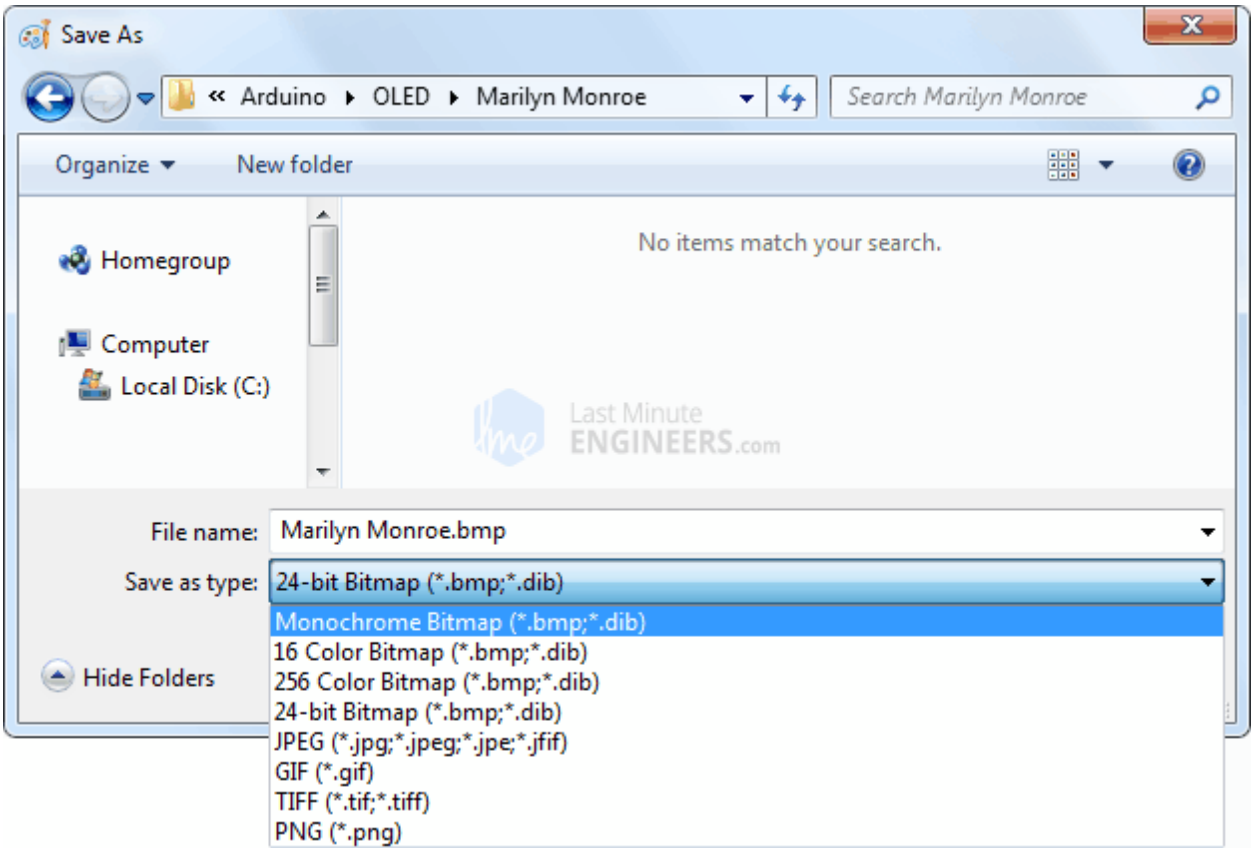
There's also a Windows application called LCD assistant that can turn your bitmap image into a data array. It is not as powerful as image2cpp, but it is still widely used by hobbyists.

To begin, convert your image into a 128×64 1-bit monochrome bitmap. You can do it in your favorite drawing program, such as Inkscape, Photoshop, or MS Paint, just like we did in MS Paint.

Open MS Paint and resize your file to 128×64 pixels.

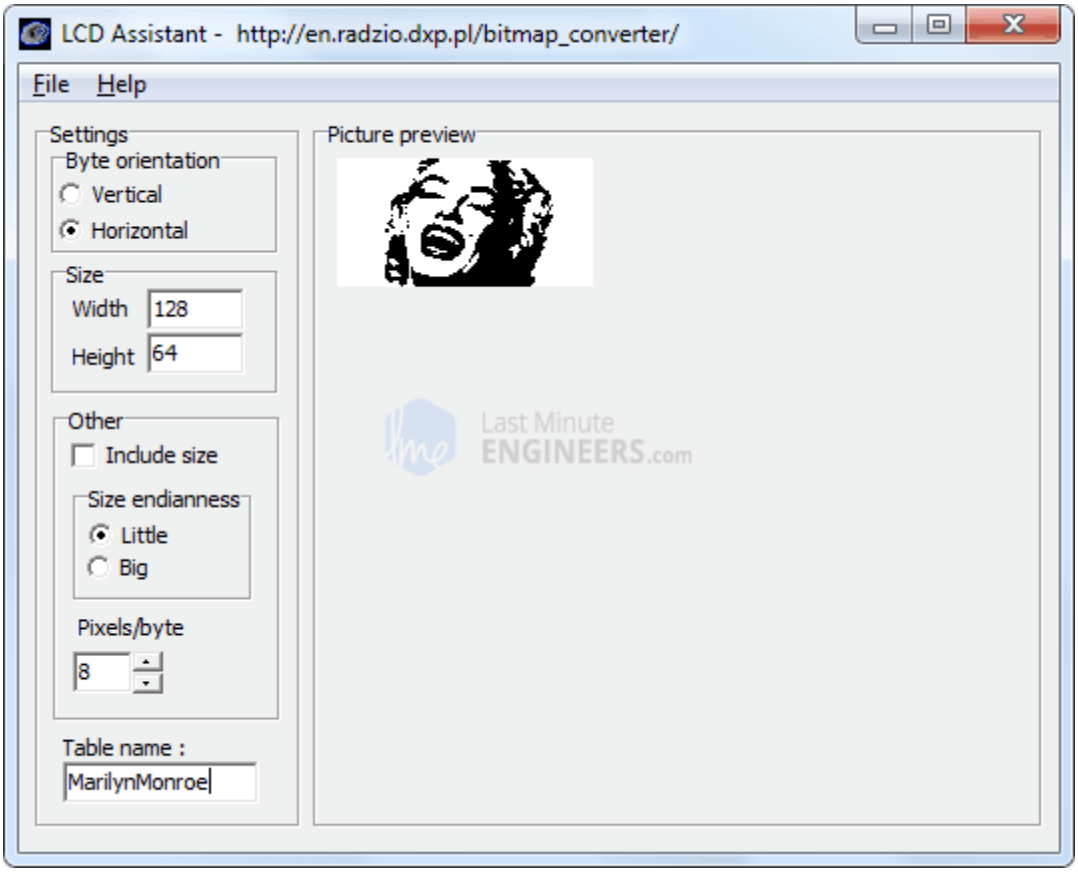


Now, save your file as a bitmap. When saving the file, select Monochrome Bitmap (*.bmp;*.dib). This will produce a 1-bit/binary bitmap image with only two possible values for each pixel: 0 (black) or 1 (white).



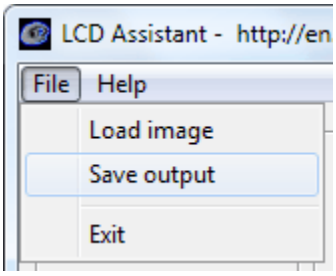
The only drawback is that you cannot set the brightness threshold level. It is by default set to 50% and cannot be changed.

Now, download the LCD Assistant program. Open the executable and select your bitmap from the File menu.

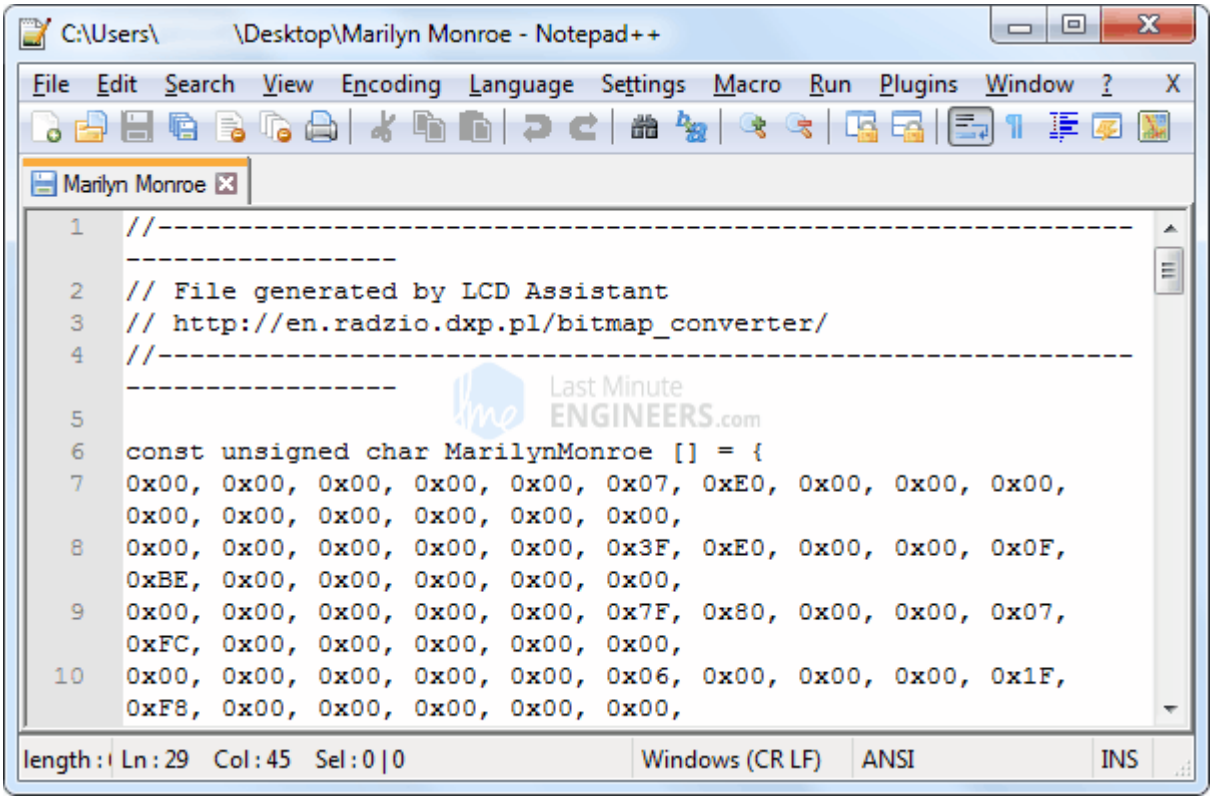


Simply go to the File menu and select the Save output option. Save the file as a text document.

For your information, there is a setting called Byte Orientation. It actually generates images based on the scanning pattern of the display. If your image appears distorted on your screen, try changing the mode.



That's all. After you've created your array, copy and paste it into your code.


C:\Users\ \Desktop\Marilyn Monroe - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ? X
Marilyn Monroe
1 //--
2 // File generated by LCD Assistant
3 // http://en.radzio.dxp.pl/bitmap_converter/
4 //--
5
6 const unsigned char MarilynMonroe [] = {
7 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0xE0, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
8 0x00, 0x00, 0x00, 0x00, 0x00, 0x3F, 0xE0, 0x00, 0x00, 0x0F,
0xBE, 0x00, 0x00, 0x00, 0x00, 0x00,
9 0x00, 0x00, 0x00, 0x00, 0x00, 0x7F, 0x80, 0x00, 0x00, 0x07,
0xFC, 0x00, 0x00, 0x00, 0x00, 0x00,
10 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00, 0x1F,
0xF8, 0x00, 0x00, 0x00, 0x00, 0x00,
Last Minute ENGINEERS.com
length : Ln : 29 Col : 45 Sel : 0 | 0
Windows (CR LF)
ANSI
INS