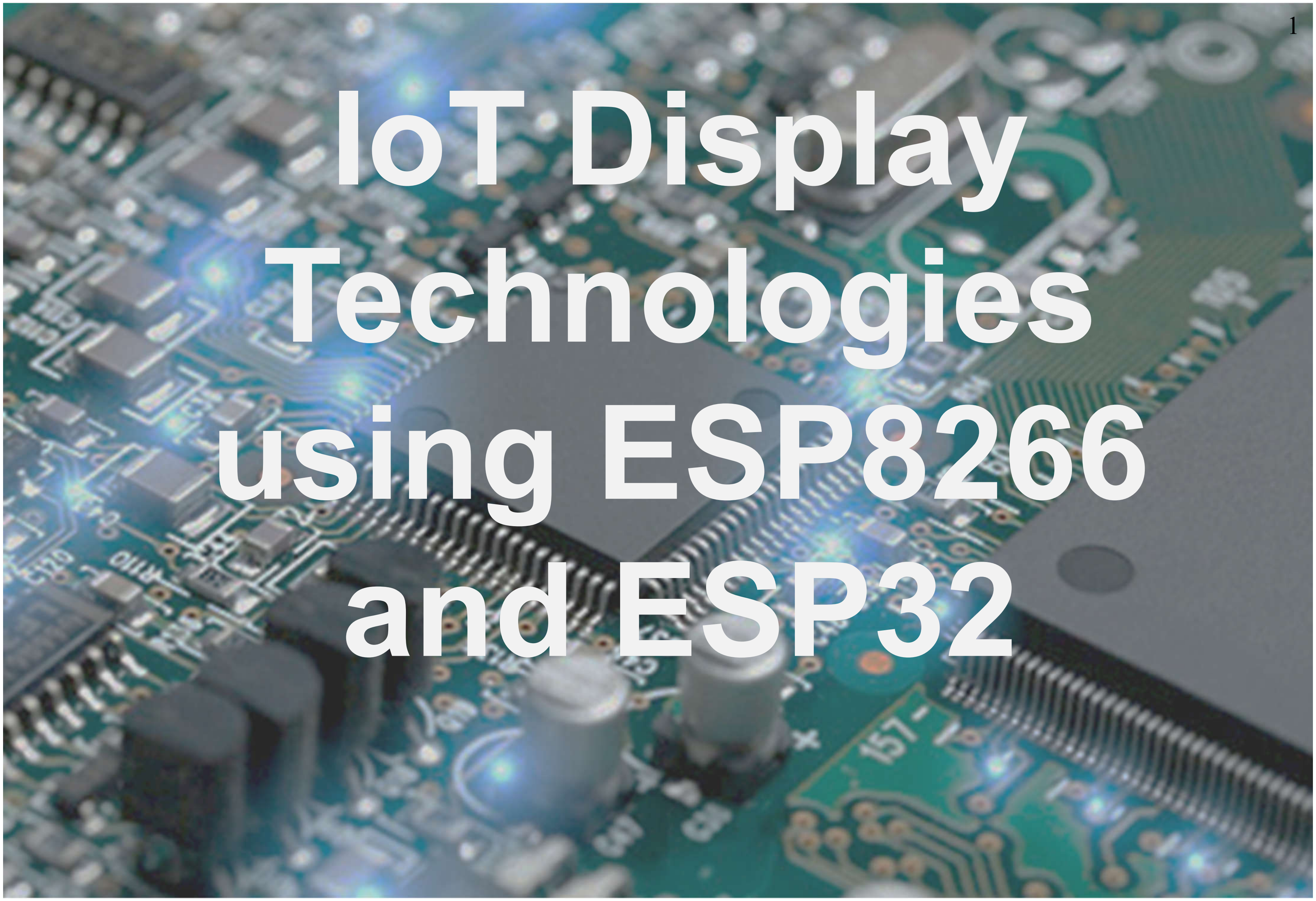


IoT Display Technologies using ESP8266 and ESP32

Display Technologies

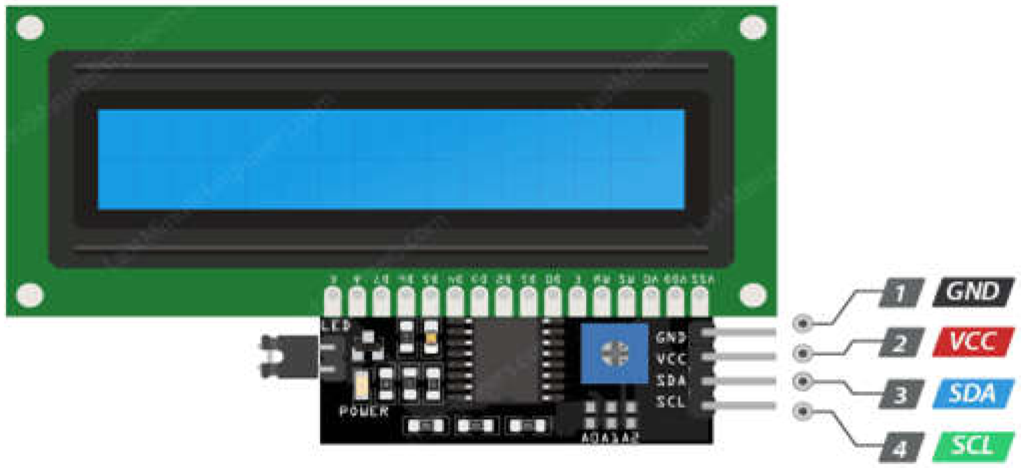
<u>Contents</u>

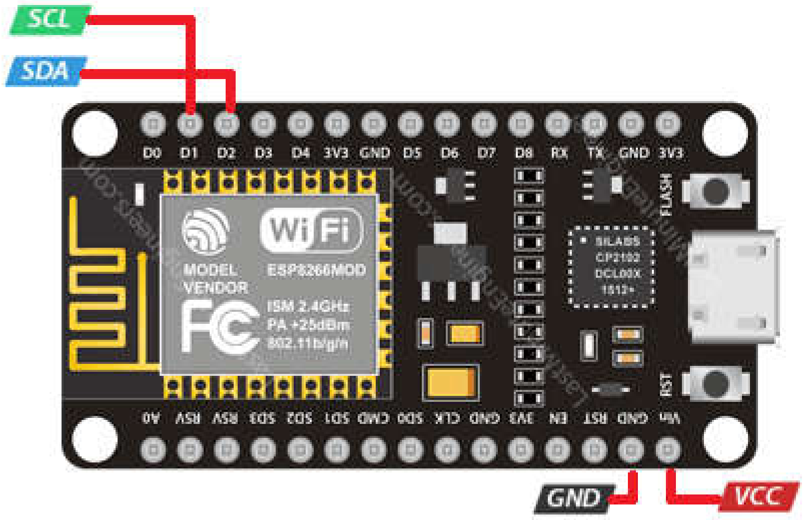
Objectives
Liquid Crystal Display 16X02 I2C using ESP8266
ST7735 Display using ESP8266
ST7789 Display using ESP32
OLED SSD1306 using ESP32

Objectives

- In this tutorial, you will learn:
  - To know how to make Liquid Crystal Display I2C projects.
  - To become familiar using ST7735R TFT Display.
  - To understand how to make ST7789 IPS Display projects.
  - To know how to use OLED SSD1306 Display.

Liquid Crystal Display 16X02 I2C




SCL
SDA
D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3
A0 RSV RSV SD3 SD2 SD1 CMD SD0 CLK GND 3V3 EN RST GND Vin
FLASH
RST
GND
VCC

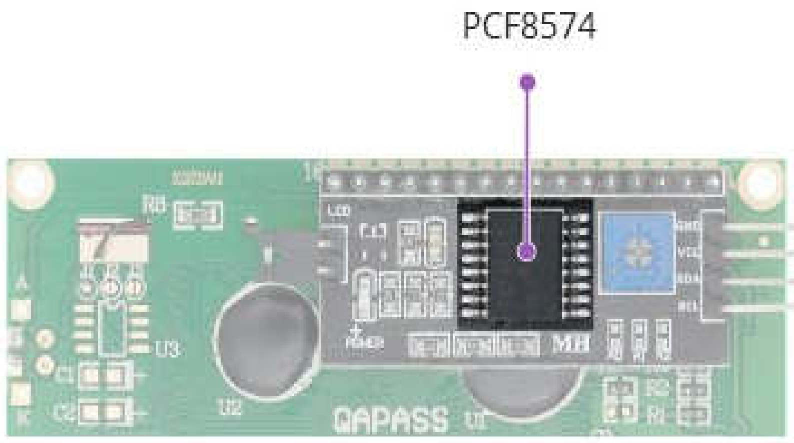
Liquid Crystal Display 16X02 I2C

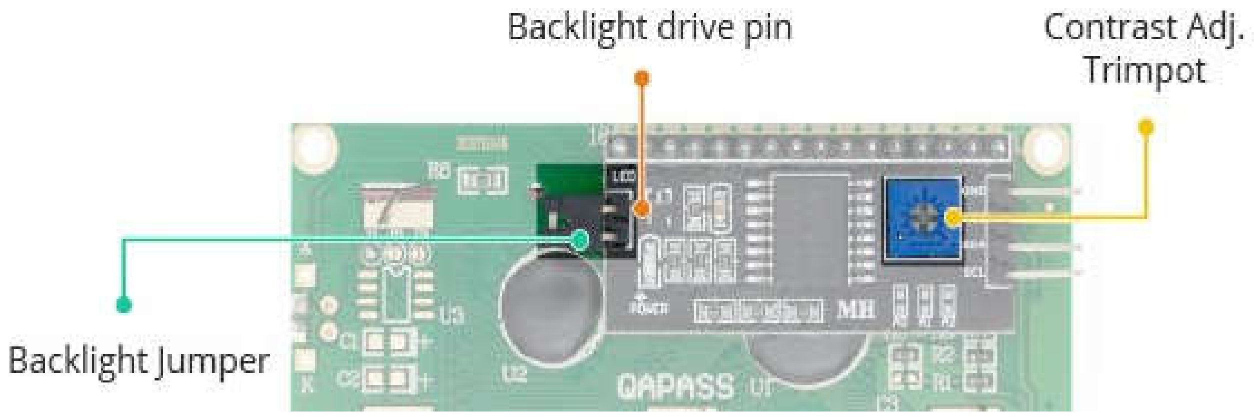
I2C LCD Overview

- to save I/O pins of Arduino
- it takes only 2 I/O's A4 and A5 (SDA and SCL)

Liquid Crystal Display 16X02 I2C

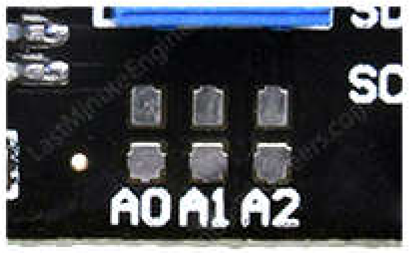
I2C LCD Adapter 8-bit I/O expander chip PCF8574



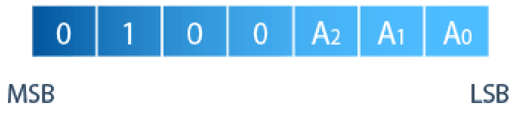


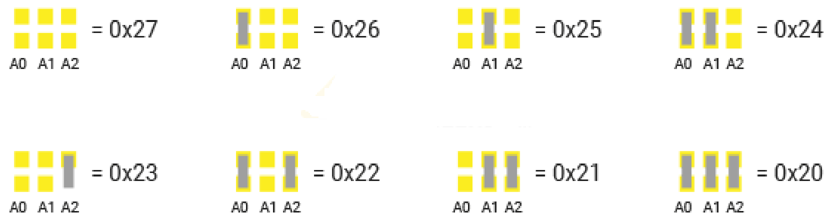
Liquid Crystal Display 16X02 I2C

I2C Address-depends on IC used

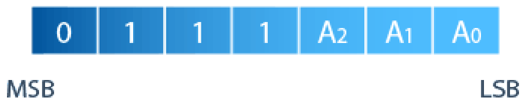


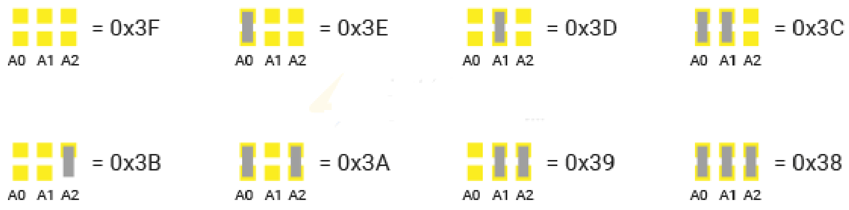
Texas Instrument:





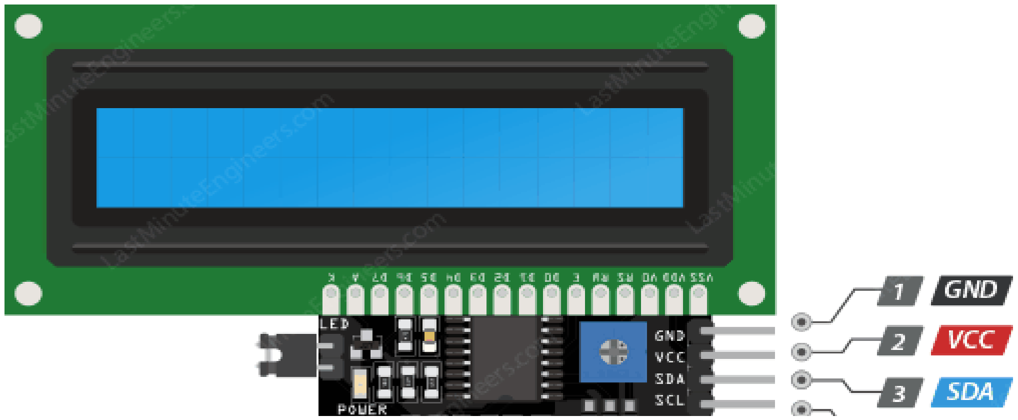
NXP:

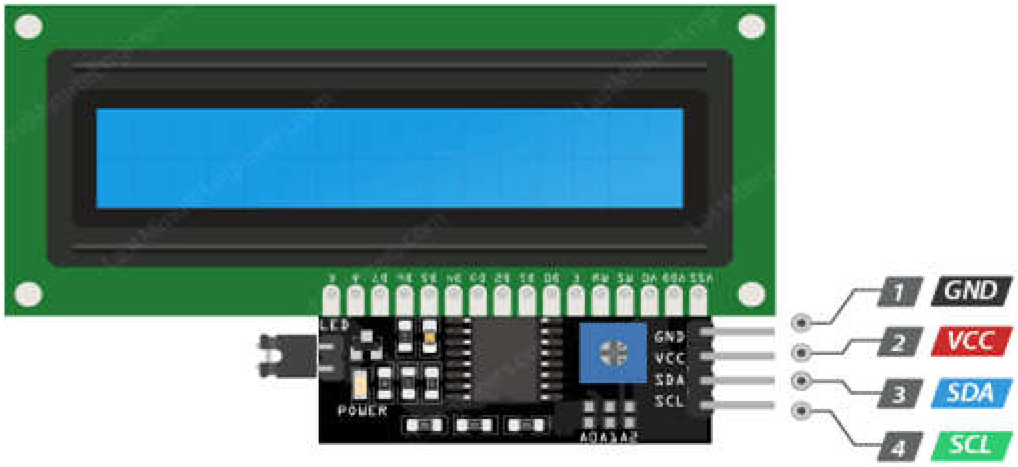


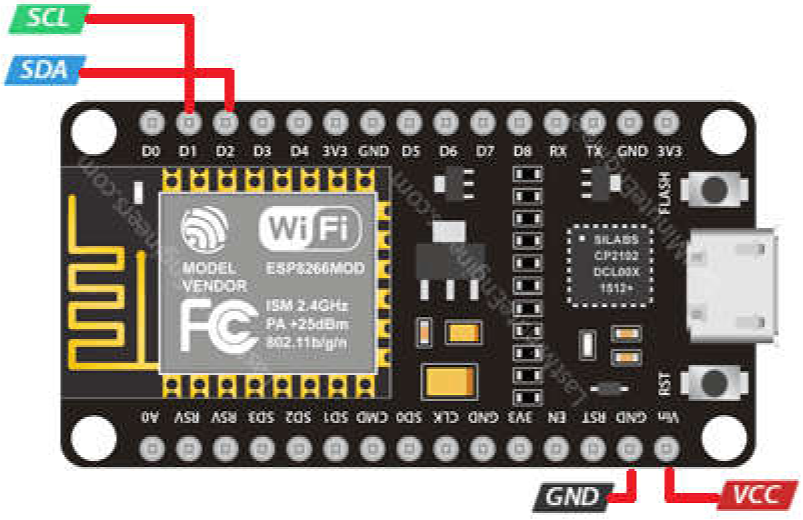


Liquid Crystal Display 16X02 I2C

I2C LCD Display Pinout



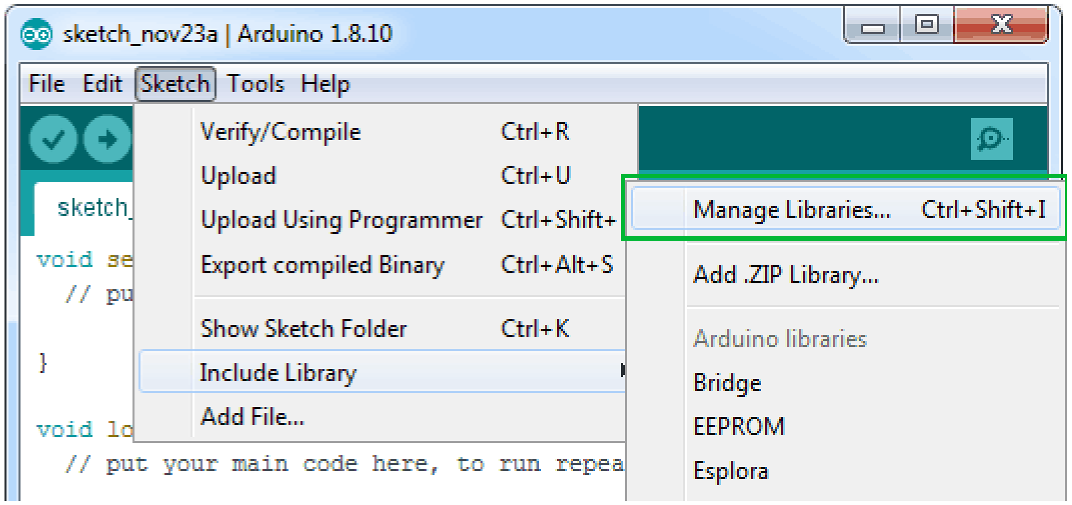



SCL
SDA
D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3
A0 RSV RSV SD3 SD2 SD1 CMD SD0 CLK GND 3V3 EN RST GND Vin
FLASH
RST
WiFi
MODEL ESP8266MOD
VENDOR
ISM 2.4GHz
PA +25dBm
802.11b/g/n
SILABS CP2102
GND
VCC

Liquid Crystal Display 16X02 I2C

Library Installation

Sketch > Include Library > Manage Libraries... Wait for Library Manager to download libraries index and update list of installed libraries.

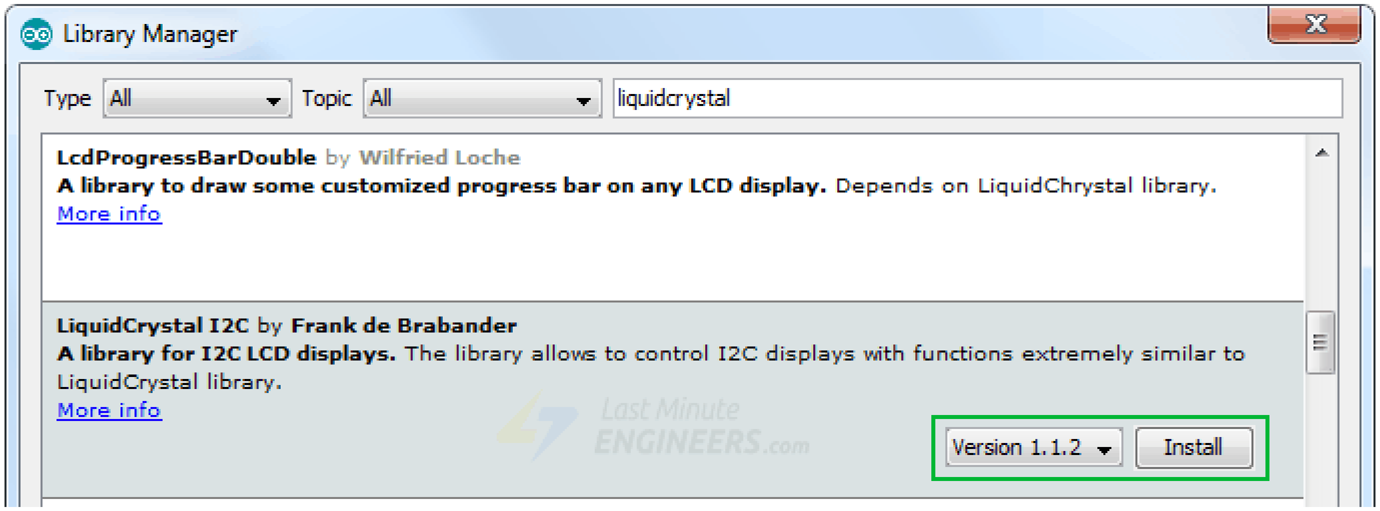

sketch_nov23a | Arduino 1.8.10
File Edit Sketch Tools Help
Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+
Export compiled Binary Ctrl+Alt+S
Show Sketch Folder Ctrl+K
Include Library
Add File...
Manage Libraries... Ctrl+Shift+I
Add .ZIP Library...
Arduino libraries
Bridge
EEPROM
Esplora

```
void se
  // pu

}

void lo
  // put your main code here, to run repea
```

Liquid Crystal Display 16X02 I2C

Library Installation

Typing ‘liquidcrystal’. There should be a couple entries. Look for LiquidCrystal I2C library by Frank de Brabander. Click on that entry, and then select Install.



LcdProgressBarDouble by Wilfried Loche
A library to draw some customized progress bar on any LCD display. Depends on LiquidChrystal library.
More info

LiquidCrystal I2C by **Frank de Brabander**
A library for I2C LCD displays. The library allows to control I2C displays with functions extremely similar to LiquidCrystal library.
More info

Liquid Crystal Display 16X02 I2C

I2C Address Checking

LCD probably has an I2C address 0x27Hex or 0x3FHex. Nevertheless it is recommended that you find out the actual I2C of the LCD before using. Luckily there is a simple way to do this, thanks to Nick Gammon‘s I2C code.

Liquid Crystal Display 16X02 I2C

I2C Address Checking

```
#include <Wire.h>

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

  // Leonardo: wait for serial port to connect
  while (!Serial)
    {
    }

  Serial.println ();
  Serial.println ("I2C scanner. Scanning ...");
  byte count = 0;

  Wire.begin();
  for (byte i = 8; i < 120; i++)
  {
    Wire.beginTransmission (i);
    if (Wire.endTransmission () == 0)
      {
      Serial.print ("Found address: ");
      Serial.print (i, DEC);
      Serial.print (" (0x");
      Serial.print (i, HEX);
      Serial.println (")");
      count++;
      delay (1);  // maybe unneeded?
      } // end of good response
  } // end of for loop
  Serial.println ("Done.");
  Serial.print ("Found ");
  Serial.print (count, DEC);
  Serial.println (" device(s).");
}  // end of setup

void loop() {}
```

Liquid Crystal Display 16X02 I2C

Hello World I2C LCD

```
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x3F,16,2);  // set the LCD address to 0x3F for a 16 chars and 2
line display

void setup() {
  lcd.init();
  lcd.clear();
  lcd.backlight();      // Make sure backlight is on

  // Print a message on both lines of the LCD.
  lcd.setCursor(2,0);   //Set cursor to character 2 on line 0
  lcd.print("Hello world!");

  lcd.setCursor(2,1);   //Move cursor to character 2 on line 1
  lcd.print("LCD Tutorial");
}

void loop() {
}
```

Liquid Crystal Display 16X02 I2C

Hello World I2C LCD

Liquid Crystal Display 16X02 I2C

Other useful functions of the library

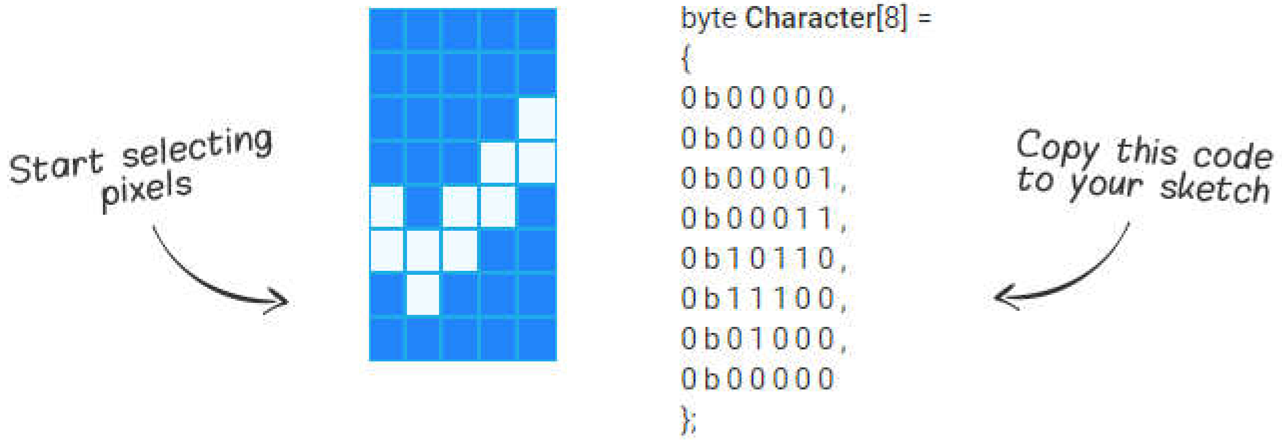
home() – positions the cursor in the top-left corner of the LCD without clearing the display.
cursor() – displays the LCD cursor, an underscore (line) at the position of the next character to be printed.
noCursor() – hides the LCD cursor.
blink() – creates a blinking block style LCD cursor: a blinking rectangle of 5×8 pixels at the position of the next character to be printed.
noBlink() – disables the blinking block style LCD cursor.
display() – turns on the LCD screen and displays the characters that were previously printed on the display.
noDisplay() – turns off the LCD screen. Simply turning off the LCD screen does not clear data from the LCD memory. This means that it will be shown again when the display() function is called.
scrollDisplayLeft() – scrolls the contents of the display one space to the left. If you want to scroll the text continuously, you need to use this function inside a loop.
scrollDisplayRight() – scrolls the contents of the display one space to the right.
autoscroll() – turns on automatic scrolling of the LCD. If the current text direction is left-to-right (default), the display scrolls to the left, if the current direction is right-to-left, the display scrolls to the right.
noAutoscroll() – turns off automatic scrolling.

Liquid Crystal Display 16X02 I2C

Create and Display Custom Characters

-to create your own custom characters (glyph) and symbols for your LCD.

-custom character the createChar() function is used. This function accepts an array of 8 bytes. Each byte (only 5 bits are considered) in the array defines one row of the character in the 5×8 matrix.



```
byte Character[8] =
{
0b00000,
0b00000,
0b00001,
0b00011,
0b10110,
0b11100,
0b01000,
0b00000
};
```

Liquid Crystal Display 16X02 I2C

Create and Display Custom Characters

```
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x3F, 16, 2);  // set the LCD address to 0x3F for a
16 chars and 2 line display

// make some custom characters:
byte Heart[8] = {
0b00000,
0b01010,
0b11111,
0b11111,
0b01110,
0b00100,
0b00000,
0b00000
};

byte Bell[8] = {
0b00100,
0b01110,
0b01110,
0b01110,
0b11111,
0b00000,
0b00100,
0b00000
};
```

Liquid Crystal Display 16X02 I2C

Create and Display Custom Characters

```
byte Alien[8] = {
0b11111,
0b10101,
0b11111,
0b11111,
0b01110,
0b01010,
0b11011,
0b00000
};

byte Check[8] = {
0b00000,
0b00001,
0b00011,
0b10110,
0b11100,
0b01000,
0b00000,
0b00000
};
```

Liquid Crystal Display 16X02 I2C

Create and Display Custom Characters

```
byte Speaker[8] = {
0b00001,
0b00011,
0b01111,
0b01111,
0b01111,
0b00011,
0b00001,
0b00000
};


byte Sound[8] = {
0b00001,
0b00011,
0b00101,
0b01001,
0b01001,
0b01011,
0b11011,
0b11000
};
```

Liquid Crystal Display 16X02 I2C

Create and Display Custom Characters

```
byte Skull[8] = {
0b00000,
0b01110,
0b10101,
0b11011,
0b01110,
0b01110,
0b00000,
0b00000
};

byte Lock[8] = {
0b01110,
0b10001,
0b10001,
0b11111,
0b11011,
0b11011,
0b11111,
0b00000
};
```

Liquid Crystal Display 16X02 I2C

Create and Display Custom Characters

```
void setup()
{
  lcd.init();
  // Make sure backlight is on
  lcd.backlight();

  // create a new characters
  lcd.createChar(0, Heart);
  lcd.createChar(1, Bell);
  lcd.createChar(2, Alien);
  lcd.createChar(3, Check);
  lcd.createChar(4, Speaker);
  lcd.createChar(5, Sound);
  lcd.createChar(6, Skull);
  lcd.createChar(7, Lock);

  // Clears the LCD screen
  lcd.clear();

  // Print a message to the lcd.
  lcd.print("Custom Character");
}

// Print All the custom characters
```

Liquid Crystal Display 16X02 I2C

Create and Display Custom Characters

```
void loop()
{
  lcd.setCursor(0, 1);
  lcd.write(0);

  lcd.setCursor(2, 1);
  lcd.write(1);

  lcd.setCursor(4, 1);
  lcd.write(2);

  lcd.setCursor(6, 1);
  lcd.write(3);

  lcd.setCursor(8, 1);
  lcd.write(4);

  lcd.setCursor(10, 1);
  lcd.write(5);

  lcd.setCursor(12, 1);
  lcd.write(6);

  lcd.setCursor(14, 1);
  lcd.write(7);
}
```

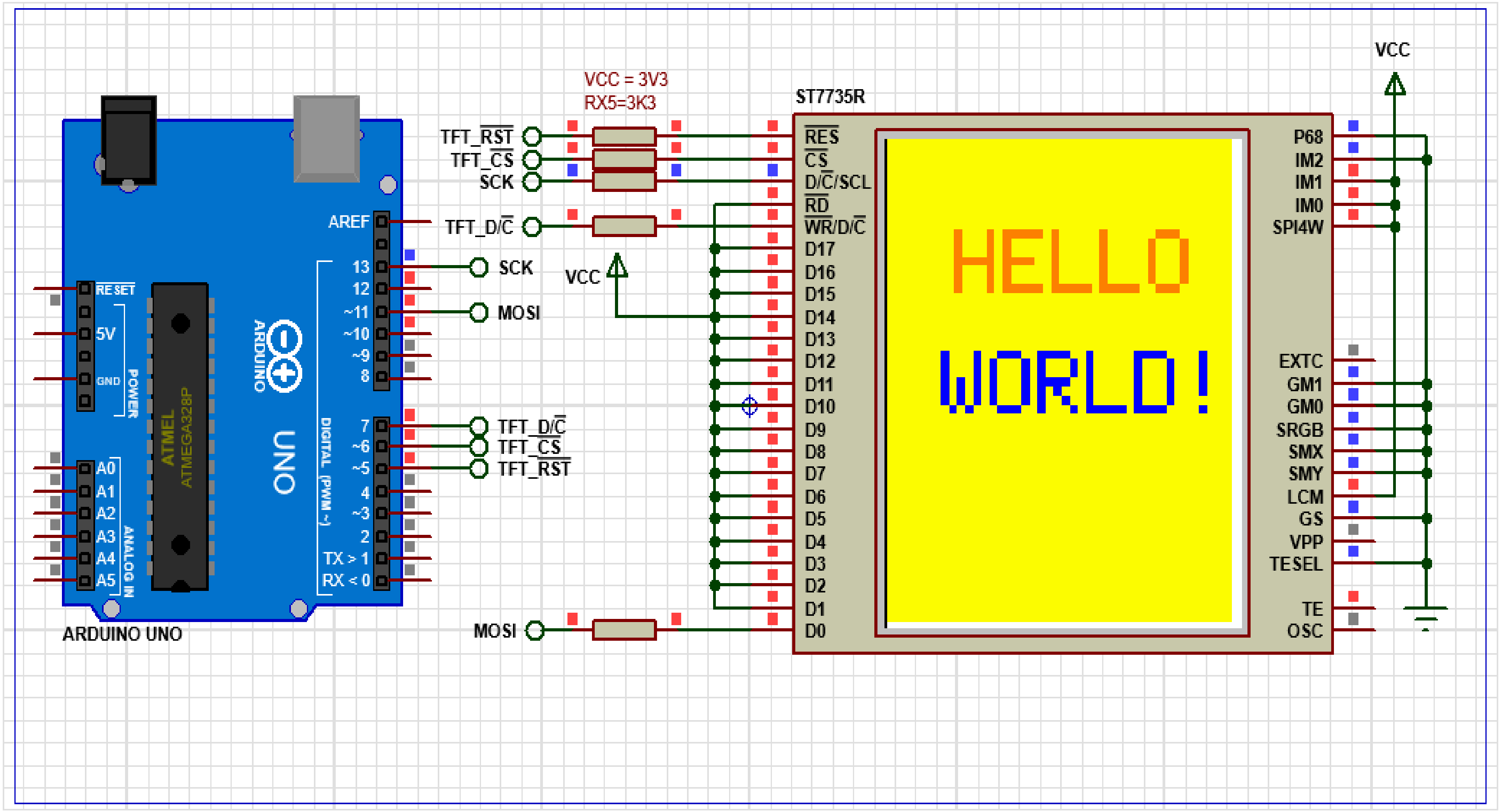
Liquid Crystal Display 16X02 I2C

```
/*
  Example LCD1602_I2C_8266.ino
*/
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// LCD address and geometry and library initialization
const byte lcdAddr = 0x27;  // Address of I2C backpack
const byte lcdCols = 16;    // Number of character in a row
const byte lcdRows = 2;     // Number of lines
//const byte lcdAddr = 0x3F;  // Address of I2C backpack
//const byte lcdCols = 20;    // Number of character in a row
//const byte lcdRows = 4;     // Number of lines

LiquidCrystal_I2C lcd(lcdAddr, lcdCols, lcdRows);

// Demo parameters
const char demoText[]= “#OneIECEP!";
const unsigned int scrollDelay = 500;   // Miliseconds before scrolling next char
const unsigned int demoDelay = 2000;    // Miliseconds between demo loops
byte textLen;                           // Number of visible characters in the text

void setup() {
  textLen = sizeof(demoText) - 1;
  lcd.init();
  lcd.backlight();
  lcd.print(demoText);
  delay(demoDelay);
}
```

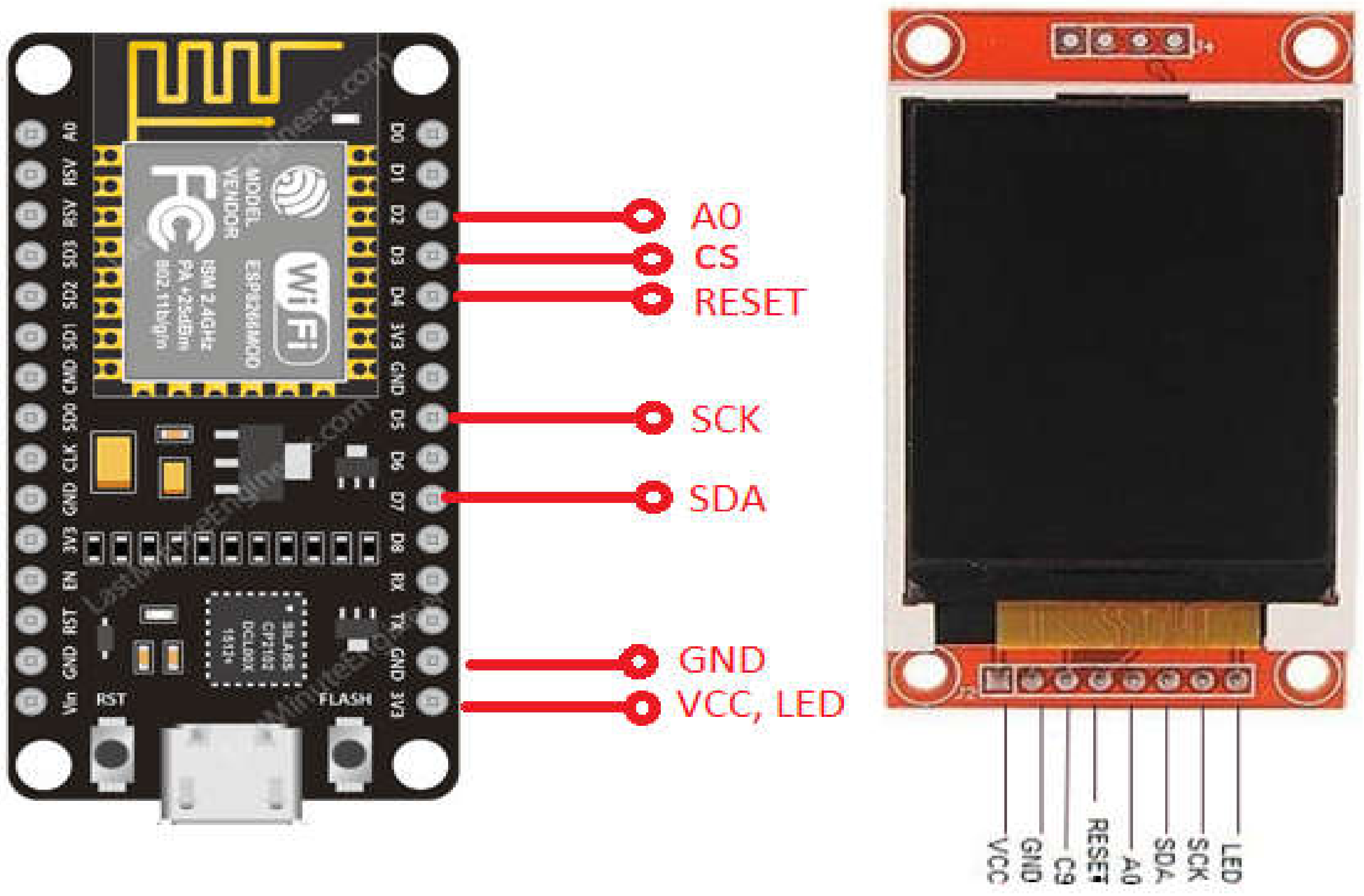
Liquid Crystal Display 16X02 I2C

```
void loop()
{
  // Scroll entire text in a row to the left outside the screen
  for (byte positionCounter = 0; positionCounter < textLen;
positionCounter++) {
    lcd.scrollDisplayLeft();
    delay(scrollDelay);
  }
  // Scroll hidden text through entire row to the right outside the screen
  for (byte positionCounter = 0; positionCounter < textLen + lcdCols;
positionCounter++)
  {
    lcd.scrollDisplayRight();
    delay(scrollDelay);
  }
  // Scroll text to the right back to original position
  for (byte positionCounter = 0; positionCounter < lcdCols;
positionCounter++)
  {
    lcd.scrollDisplayLeft();
    delay(scrollDelay);
  }
  delay(demoDelay);
}
```

ST7735 Display



ST7735 Display



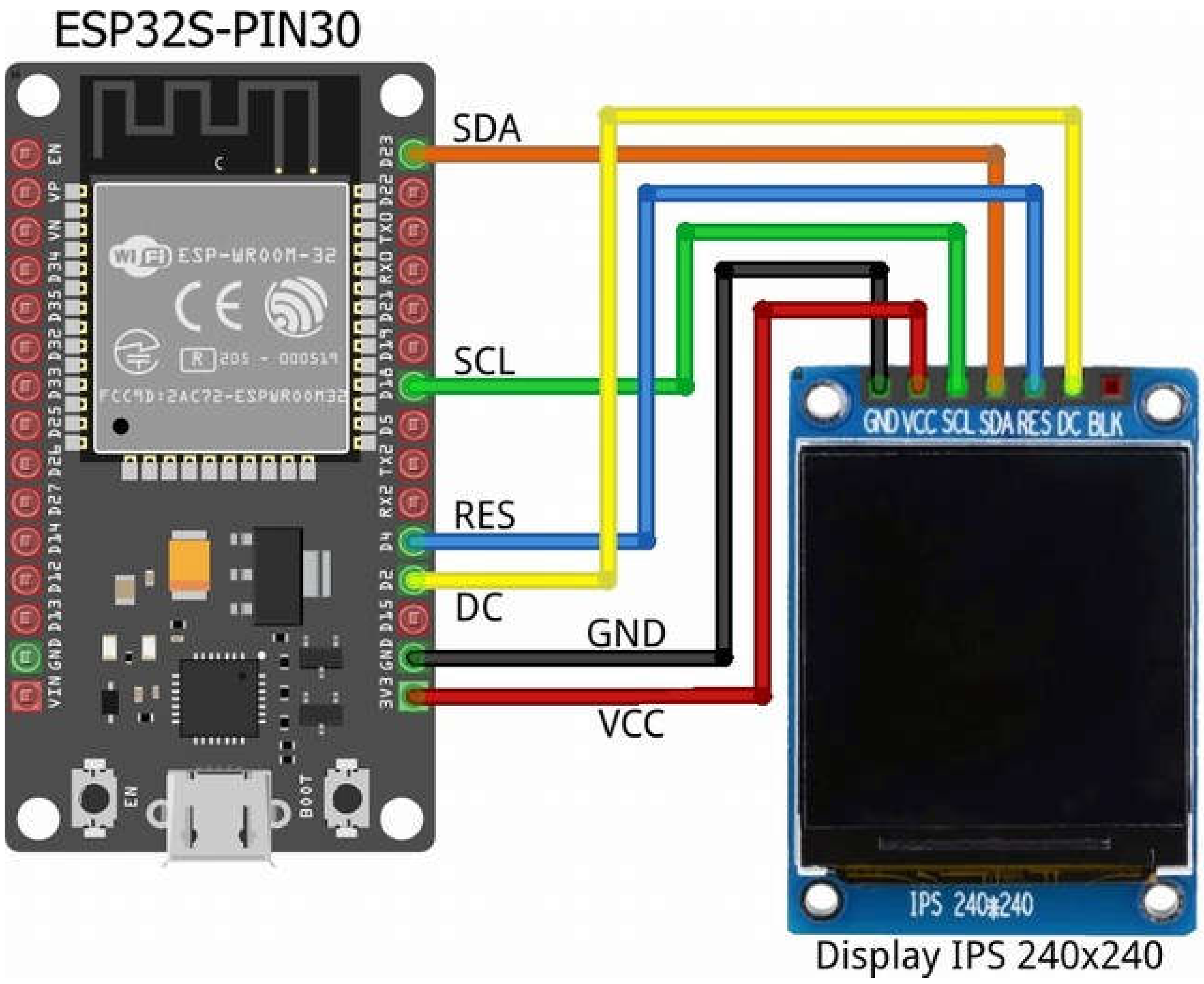
ST7735 Display

```
// Example ST7735Test1_8266.ino
#include <SPI.h>
#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_ST7735.h> // Hardware-specific library

// ST7735 TFT module connections
#define TFT_RST   D4     // TFT RST pin is connected to NodeMCU pin D4 (GPIO2)
#define TFT_CS    D3     // TFT CS  pin is connected to NodeMCU pin D4 (GPIO0)
#define TFT_DC    D2     // TFT DC  pin is connected to NodeMCU pin D4 (GPIO4)
#define TFT_MOSI  D7     //SDA
#define TFT_CLK   D5     //SCK

Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_MOSI, TFT_CLK, TFT_RST);
void setup(void) {
  //tft.initR(INITR_144GREENTAB); // Init ST7735R chip, green tab
  tft.initR(INITR_BLACKTAB);
  //tft.fillScreen(ST77XX_BLACK);
  tft.setRotation(0); // set display orientation
}
void loop()
{
  tft.fillScreen(ST77XX_YELLOW);
  print_text(25,30,"HELLO",3,ST77XX_ORANGE);
  print_text(20,70,"WORLD!",3,ST77XX_BLUE);
  delay(5000);

  tft.fillScreen(ST77XX_BLACK);
  tft.fillRoundRect(25, 10, 78, 60, 8, ST77XX_WHITE);
  tft.fillTriangle(42, 20, 42, 60, 90, 40, ST77XX_RED);
  delay(5000);

  tft.fillScreen(ST77XX_GREEN);
  tft.drawRect(5,5,120,120,ST77XX_RED);
  tft.drawFastHLine(5,60,120,ST77XX_RED);
  tft.drawFastVLine(60,5,120,ST77XX_RED);
  delay(5000);
}
```

ST7735 Display

```
void print_text(byte x_pos, byte y_pos, char *text, byte text_size, uint16_t color)
{
  tft.setCursor(x_pos, y_pos);
  tft.setTextSize(text_size);
  tft.setTextColor(color);
  tft.setTextWrap(true);
  tft.print(text);
}
```

ST7789 Display


ESP32S-PIN30
SDA
SCL
RES
DC
GND
VCC
EN VP VN D34 D35 D32 D33 D25 D26 D27 D14 D12 D13 GND VIN
D23 D22 TXD RXD D21 D19 D18 D5 TX2 RX2 D4 D2 D15 GND 3V3
ESP-WROOM-32
EN
BOOT
GND VCC SCL SDA RES DC BLK
IPS 240*240
Display IPS 240x240

ST7789 Display

```
//Example: ST7789_ESP32_test1.ino
#include <Adafruit_GFX.h>    // Core graphics library by Adafruit
#include <Arduino_ST7789.h> // Library for ST7789
#include <SPI.h>

#define TFT_DC    2     // Data/Command
#define TFT_RST   4     // ST7789 Reset
#define TFT_MOSI  23    // SPI data pin
#define TFT_SCLK  18    // SPI sclk pin

Arduino_ST7789 tft = Arduino_ST7789(TFT_DC, TFT_RST);

float p = 3.1415926;
//==================================================================================
//  Initialization
//==================================================================================
void setup() {

  tft.init(240, 240);   // initialize ST7789 chip at 240x240 pixels

  // Paint red/green/blue rectangles
  tft.fillRect(0, 0 , 240, 80, RED);
  tft.fillRect(0, 80 , 240, 160, GREEN);
  tft.fillRect(0, 160 , 240, 240, BLUE);
  delay (1000);

  // large block of text
  tft.fillScreen(BLACK);
  tft.setTextSize(3);
  tft.setTextColor(WHITE);
  tft.setCursor(0, 0);
  tft.println(" IECEP RIZAL");
  tft.setTextColor(RED);
  tft.println("ST7789 IPS");
  tft.setTextColor(YELLOW);
  tft.println("240X240 IPS");
  delay(3000);
```

ST7789 Display

```
// a single pixel
  tft.drawPixel(tft.width() / 2, tft.height() / 2, GREEN);
  delay(1000);

  // line draw test
  testlines(YELLOW);
  delay(1000);

  // optimized lines
  testfastlines(RED, BLUE);
  delay(1000);

  testdrawrects(GREEN);
  delay(1000);

  testfillrects(YELLOW, MAGENTA);
  delay(1000);

  tft.fillScreen(BLACK);
  testfillcircles(10, BLUE);
  testdrawcircles(10, WHITE);
  delay(1000);

  testroundrects();
  delay(1000);

  testtriangles();
  delay(1000);

  mediabuttons();
  delay(1000);
}
```

ST7789 Display

```
//================================================================================
//  Main
//================================================================================
void loop() {

  tft.invertDisplay(true);
  delay(500);
  tft.invertDisplay(false);
  delay(500);
}
//================================================================================
//  Subroutines
//================================================================================
void testlines(uint16_t color) {
  tft.fillScreen(BLACK);
  for (int16_t x = 0; x < tft.width(); x += 6) {
    tft.drawLine(0, 0, x, tft.height() - 1, color);
  }
  for (int16_t y = 0; y < tft.height(); y += 6) {
    tft.drawLine(0, 0, tft.width() - 1, y, color);
  }
}

void testdrawtext(char *text, uint16_t color) {
  tft.setCursor(0, 0);
  tft.setTextColor(color);
  tft.setTextWrap(true);
  tft.print(text);
}
```

ST7789 Display

```
void testfastlines(uint16_t color1, uint16_t color2) {
  tft.fillScreen(BLACK);
  for (int16_t y = 0; y < tft.height(); y += 5) {
    tft.drawFastHLine(0, y, tft.width(), color1);
  }
  for (int16_t x = 0; x < tft.width(); x += 5) {
    tft.drawFastVLine(x, 0, tft.height(), color2);
  }
}

void testdrawrects(uint16_t color) {
  tft.fillScreen(BLACK);
  for (int16_t x = 0; x < tft.width(); x += 6) {
    tft.drawRect(tft.width() / 2 - x / 2, tft.height() / 2 - x / 2 , x, x, color);
  }
}

void testfillrects(uint16_t color1, uint16_t color2) {
  tft.fillScreen(BLACK);
  for (int16_t x = tft.width() - 1; x > 6; x -= 6) {
    tft.fillRect(tft.width() / 2 - x / 2, tft.height() / 2 - x / 2 , x, x, color1);
    tft.drawRect(tft.width() / 2 - x / 2, tft.height() / 2 - x / 2 , x, x, color2);
  }
}

void testfillcircles(uint8_t radius, uint16_t color) {
  for (int16_t x = radius; x < tft.width(); x += radius * 2) {
    for (int16_t y = radius; y < tft.height(); y += radius * 2) {
      tft.fillCircle(x, y, radius, color);
    }
  }
}
```

ST7789 Display

```
void testdrawcircles(uint8_t radius, uint16_t color) {
  for (int16_t x = 0; x < tft.width() + radius; x += radius * 2) {
    for (int16_t y = 0; y < tft.height() + radius; y += radius * 2) {
      tft.drawCircle(x, y, radius, color);
    }
  }
}

void testtriangles() {
  tft.fillScreen(BLACK);
  int color = 0xF800;
  int t;
  int w = tft.width() / 2;
  int x = tft.height() - 1;
  int y = 0;
  int z = tft.width();
  for (t = 0 ; t <= 15; t++) {
    tft.drawTriangle(w, y, y, x, z, x, color);
    x -= 4;
    y += 4;
    z -= 4;
    color += 100;
  }
}
```

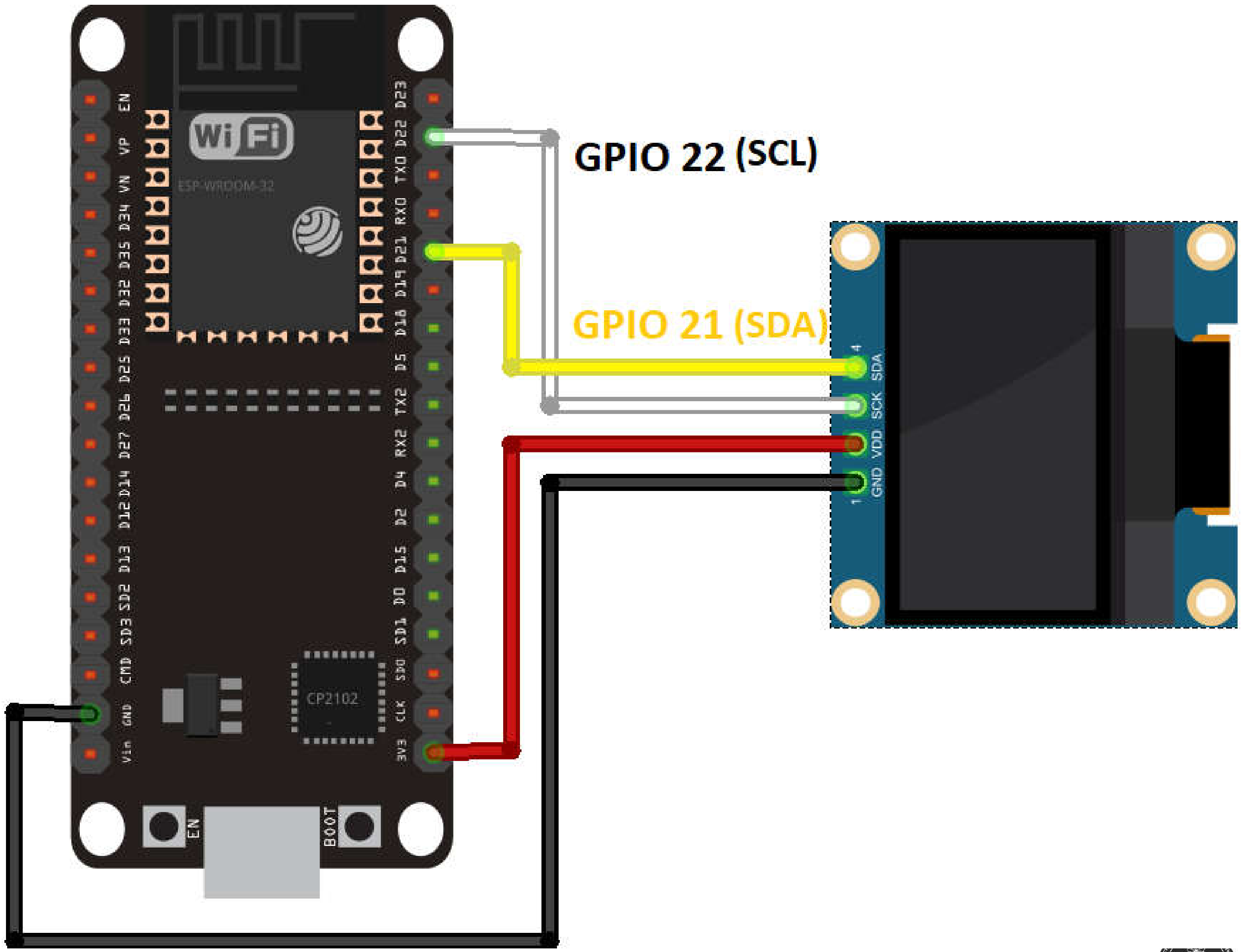
ST7789 Display

```
void testroundrects() {
  tft.fillScreen(BLACK);
  int color = 100;
  int i;
  int t;
  for (t = 0 ; t <= 4; t += 1) {
    int x = 0;
    int y = 0;
    int w = tft.width() - 2;
    int h = tft.height() - 2;
    for (i = 0 ; i <= 16; i += 1) {
      tft.drawRoundRect(x, y, w, h, 5, color);
      x += 2;
      y += 3;
      w -= 4;
      h -= 6;
      color += 1100;
    }
    color += 100;
  }
}
```

ST7789 Display

```
void mediabuttons() {
  // play
  tft.fillScreen(BLACK);
  tft.fillRoundRect(25, 10, 78, 60, 8, WHITE);
  tft.fillTriangle(42, 20, 42, 60, 90, 40, RED);
  delay(500);
  // pause
  tft.fillRoundRect(25, 90, 78, 60, 8, WHITE);
  tft.fillRoundRect(39, 98, 20, 45, 5, GREEN);
  tft.fillRoundRect(69, 98, 20, 45, 5, GREEN);
  delay(500);
  // play color
  tft.fillTriangle(42, 20, 42, 60, 90, 40, BLUE);
  delay(50);
  // pause color
  tft.fillRoundRect(39, 98, 20, 45, 5, RED);
  tft.fillRoundRect(69, 98, 20, 45, 5, RED);
  // play color
  tft.fillTriangle(42, 20, 42, 60, 90, 40, GREEN);
}
```

OLED SSD1306 I2C



OLED SSD1306 I2C

```
/****************************************************************************
 *This is an example for our Monochrome OLEDs based on SSD1306 drivers
 *This example is for a 128x64 pixel display using I2C to communicate
 *3 pins are required to interface (two I2C and one reset).
 *Ex. ssd1306_12x64_i2c_test1.ino
****************************************************************************/
#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 an SSD1306 display connected to I2C (SDA, SCL pins)
//#define OLED_RESET     4 // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

#define NUMFLAKES     10 // Number of snowflakes in the animation example

#define LOGO_HEIGHT   16
#define LOGO_WIDTH    16
static const unsigned char PROGMEM logo_bmp[] =
{ B00000000, B11000000,
  B00000001, B11000000,
  B00000001, B11000000,
  B00000011, B11100000,
  B11110011, B11100000,
  B11111110, B11111000,
  B01111110, B11111111,
  B00110011, B10011111,
  B00011111, B11111100,
  B00001101, B01110000,
  B00011011, B10100000,
  B00111111, B11100000,
```

OLED SSD1306 I2C

```
  B00111111, B11110000,
  B01111100, B11110000,
  B01110000, B01110000,
  B00000000, B00110000 };

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

  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3D)) { // Address 0x3D for 128x64
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }

  // Show initial display buffer contents on the screen --
  // the library initializes this with an Adafruit splash screen.
  display.display();
  delay(2000); // Pause for 2 seconds

  // Clear the buffer
  display.clearDisplay();

  // Draw a single pixel in white
  display.drawPixel(10, 10, SSD1306_WHITE);

  // Show the display buffer on the screen. You MUST call display() after
  // drawing commands to make them visible on screen!
  display.display();
  delay(2000);
  // display.display() is NOT necessary after every single drawing command,
  // unless that's what you want...rather, you can batch up a bunch of
  // drawing operations and then update the screen all at once by calling
  // display.display(). These examples demonstrate both approaches...
```

OLED SSD1306 I2C

```
 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

  testscrolltext();    // Draw scrolling text

  testdrawbitmap();    // Draw a small bitmap image

  // Invert and restore display, pausing in-between
  display.invertDisplay(true);
  delay(1000);
  display.invertDisplay(false);
  delay(1000);

  testanimate(logo_bmp, LOGO_WIDTH, LOGO_HEIGHT); // Animate bitmaps
}
```

OLED SSD1306 I2C

```
 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, SSD1306_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, SSD1306_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, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  for(i=display.height()-1; i>=0; i-=4) {
    display.drawLine(0, display.height()-1, display.width()-1, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  delay(250);
```

OLED SSD1306 I2C

```
 display.clearDisplay();

  for(i=display.width()-1; i>=0; i-=4) {
    display.drawLine(display.width()-1, display.height()-1, i, 0, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  for(i=display.height()-1; i>=0; i-=4) {
    display.drawLine(display.width()-1, display.height()-1, 0, i, SSD1306_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, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  for(i=0; i<display.width(); i+=4) {
    display.drawLine(display.width()-1, 0, i, display.height()-1, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000); // Pause for 2 seconds
}
```

OLED SSD1306 I2C

```
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, SSD1306_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, SSD1306_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, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000);
}
```

OLED SSD1306 I2C

```
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, SSD1306_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, SSD1306_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, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000);
}
```

OLED SSD1306 I2C

```
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, SSD1306_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, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000);
}
```

OLED SSD1306 I2C

```
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, SSD1306_INVERSE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testdrawchar(void) {
  display.clearDisplay();

  display.setTextSize(1);      // Normal 1:1 pixel scale
  display.setTextColor(SSD1306_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);
}
```

OLED SSD1306 I2C

```
void testdrawstyles(void) {
  display.clearDisplay();

  display.setTextSize(1);             // Normal 1:1 pixel scale
  display.setTextColor(SSD1306_WHITE);        // Draw white text
  display.setCursor(0,0);             // Start at top-left corner
  display.println(F("Hello, world!"));

  display.setTextColor(SSD1306_BLACK, SSD1306_WHITE); // Draw 'inverse' text
  display.println(3.141592);

  display.setTextSize(2);             // Draw 2X-scale text
  display.setTextColor(SSD1306_WHITE);
  display.print(F("0x")); display.println(0xDEADBEEF, HEX);

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

OLED SSD1306 I2C

```
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);
  }
```

OLED SSD1306 I2C

```
void testscrolltext(void) {
  display.clearDisplay();

  display.setTextSize(2); // Draw 2X-scale text
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(10, 0);
  display.println(F("scroll"));
  display.display();      // Show initial text
  delay(100);

  // Scroll in various directions, pausing in-between:
  display.startscrollright(0x00, 0x0F);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrollleft(0x00, 0x0F);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrolldiagright(0x00, 0x07);
  delay(2000);
  display.startscrolldiagleft(0x00, 0x07);
  delay(2000);
  display.stopscroll();
  delay(1000);
}
```

OLED SSD1306 I2C

```
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, SSD1306_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);
      }
    }
  }
}
```

THANK YOU!