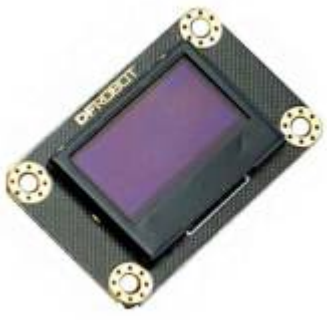
0,96" 128x64 OLED 2864 Display module - SSD1306 (I2C)

[Mise à jour le 20/8/2023]

- **Ressources**
  - **Wiki DFRobot** : Gravity: I2C OLED-2864 Display
  - **Distribué** par Mouser
- **Lectures connexes**
  - Les afficheurs graphiques : généralités et primitives
  - Bibliothèques - Arduino Library List
  - Bibliothèque - Adafruit GFX Graphics Library
  - Adafruit 1,3" 128x64 OLED FeatherWing - SH1107 + 3 buttons (I2C)
  - Adafruit 1.8" 128x160 Color TFT LCD display with MicroSD Card v2 - ST7735R (SPI)

---

1. Description

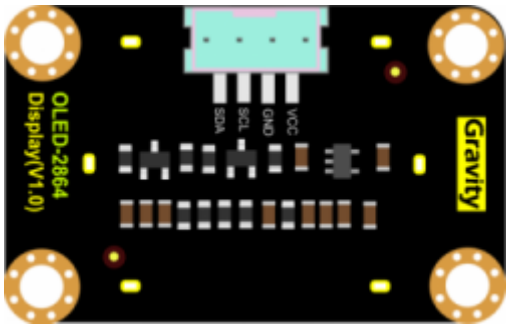


Un écran **OLED** fonctionne sans rétroéclairage. Ainsi, il peut afficher des niveaux de **noir profond** et peut être plus mince et plus léger qu'un écran à cristaux liquides (LCD). Dans des conditions de **faible luminosité ambiante**, telles qu'une pièce sombre, un écran OLED peut obtenir un taux de contraste plus élevé qu'un écran LCD.

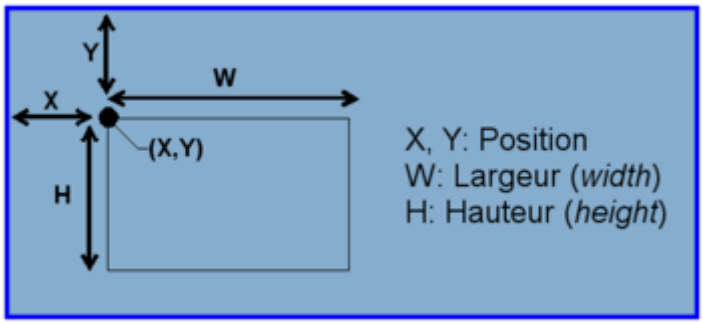
La technologie OLED est utilisée dans des applications commerciales telles que les écrans pour téléphones mobiles et lecteurs multimédias portables, les autoradios et les appareils photo numériques, entre autres.

L'écran **Gravity OLED 2864** est un module d'affichage autolumineux à **fond bleu**. La zone d'affichage est de **0,96“** et utilise une puce SSD1306. Il prend en charge les communications **I2C** et les fréquences de rafraîchissement allant jusqu'à 60 Hz. Le module utilise l’interface commune Gravity I2C pour une utilisation plug and play simplifiée. DFRobot

- **Caractéristiques**
  - **Diagonale** : 0,96”
  - **Luminosité** : 60 (typ.) Cd / m^2
  - **Contrôleur** : SSD1306
  - **Résolution** : 128 x 64
  - **Connectique** : 4 broches (alimentation et bus I2C)
  - **Bus I2C** : adresse 0x3C
  - **Tension d'alimentation** : 3,3V ~ 5V
  - **Consommation maximale** : 20mA @ 3v
  - **Dimensions** : 41.2×26.2x8mm
- **Brochage**



- **Organisation de l'écran**



  - **x** : position du point par rapport au côté gauche de l'écran.
  - **y** : position du point par rapport au dessus de l'écran.
  - **w** : largeur (du mot Width).
  - **h** : hauteur (du mot Height).
  - **c** : couleur (1=point allumé, 0=point éteint)

2. Programmation

Les programmes suivant sont codés :

- En langage **C** sur Arduino Uno ou compatible
- En langage **MicroPython** sur Raspberry Pi Pico, ESP32 etc.

- Arduino
- Micropython

Bibliothèque OakOLED

En programmation C, C++ l'utilisation des méthodes de la classe **Adafruit_GFX** sur cet afficheur peut se faire par l'intermédiaire de la bibliothèque **OakOLED**. Comme OakOLED dérive de Adafruit GFX Graphics il suffit de créer un objet OakOLED pour accéder aux méthodes de Adafruit_GFX.

Installer OakOLED avec le gestionnaire de bibliothèques de l'IDE. Les méthodes de la classe sont décrites **ici**.

Exemple : “Hello World”

helloWorld.cpp

```
// Exemple d'utilisation de la bibliothèque OakOLED
// Description : Affiche "hello, world"

#include "Wire.h"          // Bibliothèques nécessaires
#include "Adafruit_GFX.h"
#include "OakOLED.h"

OakOLED oled;             // Construction d'un objet OakOLED

void setup() {
  Serial.begin(115200);
  oled.begin();

  oled.setTextSize(1);
  oled.setTextColor(1);
  oled.setCursor(0, 0);

  oled.println("Hello, World!");
  oled.display();
}

void loop() {
  delay(10);
}
```

2.1 Présentation

Bibliothèque SSD1306

Les exemples de code de cette page ont été testés sur un afficheur Gravity OLED-2864 (128x64) de DFROBOT et un Module afficheur OLED 0,96'' TF052 connectés à un **ESP32** ou à un **Raspberry Pi Pico**. Ces afficheurs sont pilotés par un circuit SOLOMON SYSTECH SSD1306. Sa mise en oeuvre nécessite une bibliothèque.

Télécharger la bibliothèque SSD1306 pour MicroPython et la copier dans un fichier nommé ssd1306.py à installer dans la carte à µC. Celle bibliothèque hérite de framebuf. On dispose donc des méthodes ci-dessous pour dessiner des formes sur le buffer d'impression.

- **Méthodes de la bibliothèque SSD1306**

Prototype	**Description**
fill(c)	Remplit l'écran en **noir (c=1)** ou en **blanc (c=0)**
rect (x , y , w , h , c[, f])	Dessine un rectangle de largeur w et de hauteur h dans la couleur c au point (x,y). Le paramètre facultatif f peut être défini sur True pour remplir le rectangle. Sinon, seul un contour d'un pixel est dessiné.
hline(x,y,w,c)	Dessine une ligne horizontale de longueur w dans la couleur c au point (x,y)
vline(x,y,h,c)	Dessine une ligne verticale de longueur h dans la couleur c au point (x,y)
line(x1,y1,x2,y2,c)	Dessine une ligne dans la couleur c entre les points (x1,y1) et (x2,y2)
pixels (x , y[, c])	Affiche un pixel dans la couleur c au point (x,y). Si c n'est pas donné, obtient la valeur de couleur du pixel spécifié. Si c est donné, définit le pixel spécifié sur la couleur donnée.
scroll(depl.horiz, depl. vert.)	Déplace le contenu de l'écran de n points
show()	Transfert le contenu du buffer d'affichage sur l'écran
text("text",x,y,c)	Affiche le texte "text" dans la couleur c au point (x,y). En noir (c=1) ou en blanc (c=0). **La hauteur d'un caractère occupe 8px**.
poly (x , y , coordonnées , c[, f])	Étant donné une liste de coordonnées, dessine un polygone fermé arbitraire (convexe ou concave) à l'emplacement x, y donné en utilisant la couleur donnée. Les coordonnées doivent être spécifiées sous forme de tableau d'entiers, par exemple .array('h', [x0, y0, x1, y1, … xn, yn]) Le paramètre facultatif f peut être défini sur True pour remplir le polygone. Sinon, seul un contour d'un pixel est dessiné.
ellipse(x , y , xr , an , c[, f , m])	Dessine une ellipse à l'emplacement donné. Les rayons xr et yr définissent la géométrie ; des valeurs égales entraînent le dessin d'un cercle. Le paramètre c définit la couleur. Le paramètre optionnel f peut être défini sur True pour remplir l'ellipse sinon, seul un contour d'un pixel est dessiné.

A l'exception de scroll(), les méthodes ci-dessus "écrivent" dans le **tampon d'affichage**. Il faut vider le tampon avec **show()** pour que les caractères apparaissent à l'écran.

2.2 Mise en oeuvre

2.2.1 Configurations

Dans les exemples de cette page, les microcontrôleurs accèdent à l'afficheur via le bus I2C. Il faut au préalable le **configurer** et créer une instance de la classe SSD1306 comme ci-dessous.

*Exemple pour un **ESP32 Feather Huzzah***

main.py

```
from machine import Pin, SoftI2C
import ssd1306
import time
import urandom

# Configuration du bus i2c sur l'ESP32 Feather Huzzah
i2c = SoftI2C(scl=Pin(22), sda=Pin(23),freq=100000)

# Dimension de l'afficheur oled (ssd1306)
oled_width = 128 # px
oled_height = 64 # px

# Construction de l'objet oled
oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)
```

*Exemple pour un **Raspberry Pi Pico***

main.py

```
# Modifications à apporter au code ci-dessus
from machine import Pin, I2C
...

# Configuration du bus i2c sur le Raspberry Pi Pico
# Accès au connecteur I2C0 du shield Grove
i2c = I2C(0,sda=Pin(8), scl=Pin(9),freq=400_000)
# Accès au connecteur I2C1 du shield Grove
#i2c = I2C(1,sda=Pin(6), scl=Pin(7),freq=400_000)
...
```

Une fois créée, l'instance de la classe SSD1306 est utilisable pour afficher du texte et des graphiques sur l'écran.

2.2.2 Démos

- **Démo 1** : affichage d'un texte

main.py

```
# A ajouter au code du §2.1
# Division de l'afficheur en 8 lignes et 9 colonnes
lin_hight = 9
col_width = 8
def text_write(text, lin, col):
    oled.text(text,col*col_width,lin*lin_hight)

oled.fill(0) # Extinction de l'afficheur
text_write("MicroPython",1,2);
text_write("sur",3,6)
text_write("ESP32",5,5)
oled.show()

time.sleep(1)
```

- **Démo 2** : affichage de lignes

main.py

```
# A ajouter au code du §2.1
# ligne horizontale : hline(pos_x1,pos_y1,pos_x2,pos_y2,c)
# ligne verticale : vline(pos_x1,pos_y1,pos_x2,pos_y2,c)
# c=0=>noir, c=1=>blanc

oled.hline(0,0,oled_width-1,1)
oled.hline(0,oled_height-1,oled_width-1,1)
oled.vline(0,0,oled_height,1)
oled.vline(oled_width-1,0,oled_height,1)
oled.show()

time.sleep(3)
```

- **Démo 3** : affichage aléatoire de pixels

main.py

```
# A ajouter au code du §2.1
oled.fill(0) # Ecran noir

for n in range(50):
    pos_x = urandom.randint(1,oled_width)
    pos_y = urandom.randint(1,oled_height)
```

```
    oled.pixel(pos_x,pos_y,1)
oled.show()

time.sleep(3)
```

- **Démo 4** : affichage d'une icône

main.py

```
# A ajouter au code du §2.1
ICON = [
    [0,0,0,0,1,1,1,0,0,0,0],
    [0,0,0,1,0,0,0,1,0,0,0],
    [0,0,1,0,0,0,0,0,1,0,0],
    [0,1,0,0,0,0,0,0,0,1,0],
    [1,0,0,0,0,0,0,0,0,0,1],
    [1,1,1,1,1,1,1,1,1,1,1],
    [0,0,0,0,1,1,1,0,0,0,0],
    [0,0,0,1,0,1,0,1,0,0,0],
    [0,0,1,0,0,1,0,0,1,0,0],
    [0,1,0,0,0,1,0,0,0,1,0],
    [0,0,0,0,1,1,1,0,0,0,0],
]

# Démo 4a : affichage d'une icône au centre
oled.fill(0) # Extinction de l'afficheur
for y in range (11):
    for x in range (11):
        oled.pixel(x+64,y+32,ICON[y][x])
oled.show()

time.sleep(2)

# Démo 4b : affichage d'icônes au hazard
for n in range(12):
    pos_x = urandom.randint(1,oled_width-12)
    pos_y = urandom.randint(1,oled_height-12)
    for y, ligne in enumerate(ICON):
        for x, c in enumerate(ligne):
            oled.pixel(x+pos_x,y+pos_y,c)
oled.show()

time.sleep(3)
```

TELECHARGER

Télécharger le projet MICROPYTHON_ESP32_SSD1306_DEMO pour Thonny et la **vidéo** des démos.

- **Démo 5** : affichage d'une icône avec canal alpha
  Modifier l'exemple du site MCHobby en prenant en compte le code du §2.2.1 pour un ESP32 ou un Raspberry Pi Pico.

From:
http://webge.fr/dokuwiki/ - **WEBGE Wikis**

Permanent link:
http://webge.fr/dokuwiki/doku.php?id=materiels:afficheurs:ard0_96&rev=1692601227

Last update: **2023/08/21 09:00**