MicroPython
pour les microcontrôleurs
Afficheur riquiqui

Günter Spanner (Allemagne)

Voici un extrait du livre de Günter Spanner, « MicroPython for Microcontrollers » (Elektor 2021). Ce passage décrit l'association d'une carte à microcontrôleur ESP32 et d'un afficheur OLED SSD1306, le tout animé avec du code en MicroPython. Vous découvrirez ainsi la puissance et la facilité d'utilisation du langage MicroPython. Pour la pratique, réalisez l'afficheur d'ECG et l'horloge numérique.

Pour des tests et des applications simples, afficher des textes et des valeurs numériques envoyés sur le port série est largement suffisant. Toutefois, cette méthode exige la présence d'un PC ou au moins d'un ordinateur portable. Si cela dure plusieurs jours ou semaines, l'usage de ces appareils entraîne une consommation électrique inutile. En outre, réserver un ordinateur à chaque projet avec un microcontrôleur n'est évidemment ni envisageable ni faisable en pratique.

Pour juste afficher l'heure ou les valeurs de mesure de capteurs d'environnement, il est alors préférable de le faire sur un petit afficheur géré par le contrôleur. Très répandu, le modèle OLED SSD1306 a une taille de 2,4 cm (0,96 pouce) et une résolution de 128×64 pixels.

Pour cet afficheur, MicroPython contient par défaut un pilote lorsqu'il est chargé dans l'ESP32. Ce pilote permet d'afficher du texte et des valeurs numériques ainsi que des graphiques simples. Le SSD1306 est équipé d'une mémoire vive interne et d'un oscillateur embarqué et n'a besoin d'aucun composant externe supplémentaire. De plus, il dispose d'une commande de luminosité à 256 niveaux. Ses principales caractéristiques sont les suivantes :

- Résolution : matrice de 128×64 pixels
- Tension d'alimentation : 1,65 V à 3,3 V
- Plage de température de fonctionnement : –40 °C à +85 °C
- Tampon d'affichage SRAM de 128×64 bits intégré
- Fonction de défilement continu sur les axes horizontal et vertical
- Oscillateur embarqué

Les écrans OLED se passent de rétroéclairage, car chaque pixel de l'écran est capable d'émettre de la lumière. Cela les rend faciles à lire même en cas de conditions d'éclairage ambiant défavorables. En outre, leur contraste est nettement supérieur à celui des écrans à cristaux liquides (LCD). Comme un pixel ne consomme de l'énergie que lorsqu'il s'allume, ils sont très efficaces sur le plan énergétique. Les versions les plus simples du SSD1306 n'ont que quatre broches, suffisantes pour gérer l'affichage via le bus I²C. Certaines versions disposent de broches de réinitialisation ou d'une interface SPI supplémentaire, mais la version simple suffit pour la plupart des applications. Le tableau suivant spécifie toutes les connexions nécessaires.

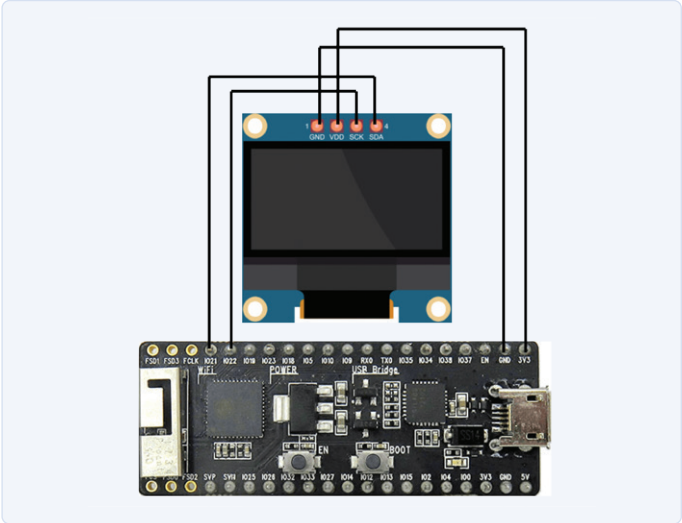

Figure 1. Afficheur SSD1306 connecté à la carte à microcontrôleur ESP32.

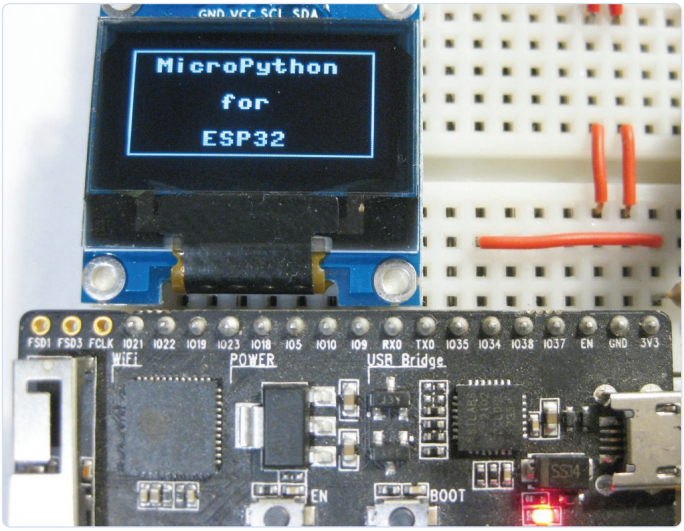

Figure 2. Affichage de texte et graphique sur le SSD1306.

broche OLED	broche ESP32
VDD	3V3
GND	GND
SCK	GPIO 22
SDA	GPIO 21

La **figure 1** montre le schéma de câblage correspondant. Le programme suivant affiche un texte et un élément graphique simple (un cadre) :

```
# SSD1306_DEMO.py

from machine import Pin, I2C
from ssd1306 import SSD1306_I2C

i2c=I2C(-1,scl=Pin(22),sda=Pin(21))
# I2C Pin assignment
oled_width=128
oled_height=64
oled = SSD1306_I2C(oled_width, oled_height, i2c)
lin_hight = 9
col_width = 8
def text_write(text,lin, col):
oled.text(text,col*col_width,lin*lin_hight)

oled.fill(0)
text_write("MicroPython", 1, 2)
text_write("for", 3, 6)
text_write("ESP32", 5, 5)

oled.rect(5, 5, 116, 52, 1)
oled.show()
```

La **figure 2** montre le résultat sur l'afficheur. Ensuite, pour piloter l'affichage, il faut importer les modules requis. Comme déjà mentionné, les bibliothèques *machine* et *SD1306* sont en principe déjà disponibles en tant que bibliothèques standard. En cas de nécessité, un fichier *ssd1306.py* peut être téléchargé séparément sur la carte. La déclaration des broches pour le bus I²C se fait ainsi :

```
i2c = I2C (-1, scl = Pin (22), sda = Pin (21))
```

Le nombre de pixels disponibles sur le module connecté est défini par les variables :

```
oled_width = 128
oled_height = 64
```

Le paramètre '`-1`' indique que le module utilisé n'a pas de broches de réinitialisation ou d'interruption. Avec ces informations passées en arguments, on crée un objet `SSD1306_I2C` appelé `oled :`

```
oled = ssd1306.SSD1306_I2C (oled_width, oled_height,
   i2c)
```

L'écran est maintenant prêt à fonctionner. La fonction `text()` est utilisée pour afficher des informations sur l'écran. L'affichage est rafraîchi avec la méthode `show()`. La fonction `text()` accepte les arguments suivants :

- Message (texte)
- Coordonnées X et Y du début du texte en pixels
- Couleur du texte (option) : 0 = noir (sombre) et 1 = blanc (clair)

L'instruction suivante affiche un message en blanc ou bleu sur fond sombre. Le texte commence à la position X = 0 et Y = 0 :

```
oled.text ('MicroPython!', 0, 0)
```

La méthode `show()` rend les modifications visibles sur l'écran. La bibliothèque contient également d'autres méthodes utiles. Avec `oled.fill(1)`, on allume tous les pixels (écran complètement clair) alors qu'avec `oled.fill(0)`, on les éteint tous (écran complètement sombre).

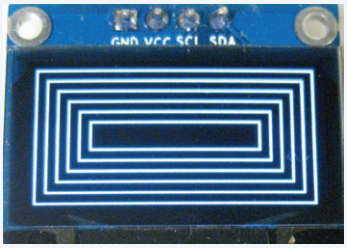

Figure 3. Cadres concentriques dessinés sur un écran OLED.

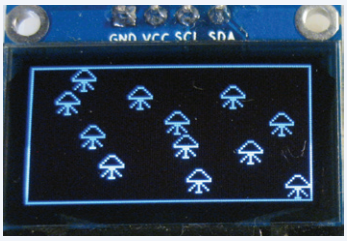

Figure 4. Éléments graphiques sur l'écran OLED.

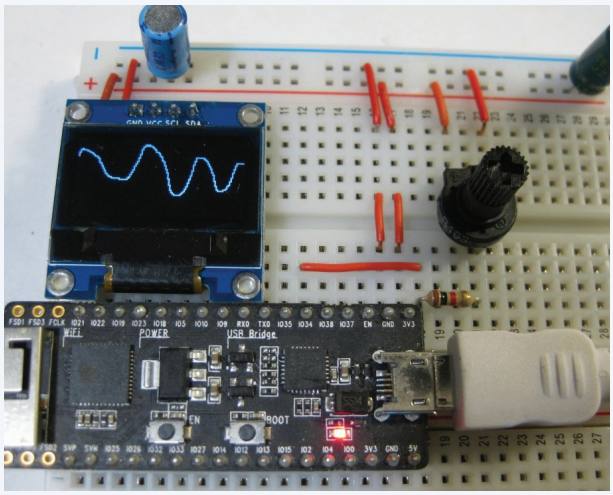

Figure 5. Affichage de courbe sur l'écran OLED.

Listage 1. Démonstration de l'affichage de bitmap sur le SSD1306

```
# SSD1306_bitmap_DEMO.py

from machine import Pin, I2C
import ssd1306
import urandom

i2c = I2C(-1, scl=Pin(22), sda=Pin(21))
oled_width = 128
oled_height = 64
oled = ssd1306.SSD1306_I2C(oled_width, oled_height,
   i2c)

# frame
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, 1)
oled.vline(oled_width-1, 0, oled_height, 1)
oled.show()
```

```
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],
]
for n in range(12):
  xofs = urandom.randint(1, oled_width-12)
  yofs = urandom.randint(1, oled_height-12)
  for y, row in enumerate(ICON):
    for x, c in enumerate(row):
      oled.pixel(x+xofs, y+yofs, c)
oled.show()
```

Listage 2. Affichage d'ECG en continu.

```
# Rolling_ECG_display.py
from machine import Pin, ADC, I2C from time import
   sleep
import ssd1306

i2c = I2C(-1, scl=Pin(22), sda=Pin(21))
  oled_width = 128
  oled_height = 64
  oled = ssd1306.SSD1306_I2C(oled_width, oled_
   height, i2c)

  pot = ADC(Pin(34))
  pot.atten(ADC.ATTN_11DB) #Full range: 3.3v vMax =
   3.4
  dotPos_old = int(oled_height/2)
while True:
  pot_value = pot.read()
  voltage = 0.000816*pot_value + 0.037822 #
   print(voltage)
dotPos_new = int(voltage/vMax*oled_height)

oled.line(0, dotPos_new, 0, dotPos_old, 1)
oled.scroll(1, 0)
oled.line(0, dotPos_new, 0, dotPos_old, 0)

dotPos_old = dotPos_new oled.pixel(0, int(oled_
   height/2), 1)
oled.show()
```

La méthode `pixel()` permet des représentations graphiques. Elle accepte les arguments suivants :

- Coordonnée X : position horizontale du pixel
- Coordonnée Y : position verticale du pixel
- Couleur du pixel : 0 = noir, 1 = blanc

Pour allumer un seul pixel dans le coin supérieur gauche :

```
oled.pixel (0, 0, 1)
```

Avec

```
oled.invert (True)
```

on inverse les couleurs sur l'écran : le blanc devient noir et vice versa. On revient aux couleurs d'origine avec `oled.invert(False)`.

Représentations graphiques

En plus des instructions de gestion des pixels, d'autres commandes graphiques sont disponibles. Pour tracer des lignes horizontales et verticales, utilisez `.hline()` ou `.vline()` qui acceptent comme arguments la position de départ XY, la longueur et la couleur de la ligne.

Le petit programme suivant dessine des cadres rectangulaires concentriques sur l'écran. La **figure 3** montre le résultat.

```
# SSD1306_frames.py
from machine import Pin, I2C
import ssd1306

i2c = I2C(-1,scl=Pin(22),sda=Pin(21))
oled_width = 128
oled_height = 64
oled = ssd1306.SSD1306_I2C(oled_width, oled_height,
   i2c)

for n in [0,5,10,15,20,25]:
oled.hline(n, n, oled_width-1-2*n, 1-2*n)
oled.hline(n, oled_height-1-n, oled_width-1-2*n,
   1-2*n)
oled.vline(n, n, oled_height-1-2*n, 1-2*n)
oled.vline(oled_width-1-n, n, oled_height-2*n, 1-2*n)
oled.show()
```

Les diagonales sont tracées en utilisant une ligne (x1, y1, x2, y2, c) entre deux points définis (x1, y1) et (x2, y2). Le paramètre `c` indique la couleur de la ligne dessinée. Les graphiques simples peuvent être écrits pixel par pixel dans la mémoire tampon de l'écran. Le programme du **listage 1** en fournit un exemple dont l'affichage est reproduit sur la **figure 4**.

Afficheur OLED comme traceur de courbes

Outre les bitmaps, les données de mesure peuvent être affichées graphiquement sur l'écran OLED, c'est-à-dire qu'il est possible d'effectuer des affichages graphiques indépendants sans passer par la fonction de traceur de Thonny (EDI utilisé pour MicroPython), Le programme du **listage 2** fournit un affichage à défilement comparable aux ECG professionnels dans les hôpitaux.

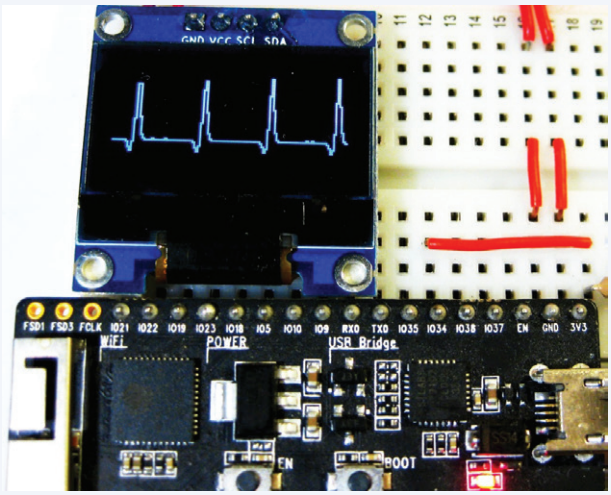

Figure 6. Un électrocardiogramme sur l'écran OLED.

En connectant un potentiomètre à l'entrée 34 du convertisseur A-N, on peut enregistrer la valeur mesurée. Après le démarrage du programme, les valeurs de tension sont affichées sur l'écran en continu grâce à la fonction de défilement intégrée. L'instruction

```
oled.scroll(1, 0)
```

déplace l'ensemble du contenu de l'écran d'un pixel. Pour avoir une courbe continue au lieu de pixels isolés, on les relie par des segments en utilisant les deux variables :

```
dotPos_old and dotPos_new
```

Elles servent à tracer une ligne entre la valeur courante et la valeur précédente. Puis, l'affichage est décalé d'un pixel, la valeur courante est transférée dans la mémoire tampon :

```
dotPos_old = dotPos_new
```

et le cycle recommence. La **figure 5** montre un exemple de valeurs de potentiomètre qui changent continuellement. Si vous disposez d'un amplificateur d'ECG, vous pouvez enregistrer les signaux électriques du cœur humain et obtenir ainsi l'électrocardiogramme typique bien connu dans les unités de soins intensifs des hôpitaux (**fig. 6**).

Horloge numérique avec affichage OLED

Une autre application d'affichage utile est celle de l'heure. Elle transforme l'ESP32 et le SSD1306 en horloge numérique. La fonction `time()` du module de l'heure renvoie le nombre de secondes depuis la mise sous tension ou la réinitialisation de la carte. Au moyen de la variable

```
time_offset=20*3600+00*60+0 # hh+mm+ss
```

on spécifie une heure de début en secondes. Pour la valeur ci-dessus, l'horloge commence à 20:00:00. La fonction `time_text()` :

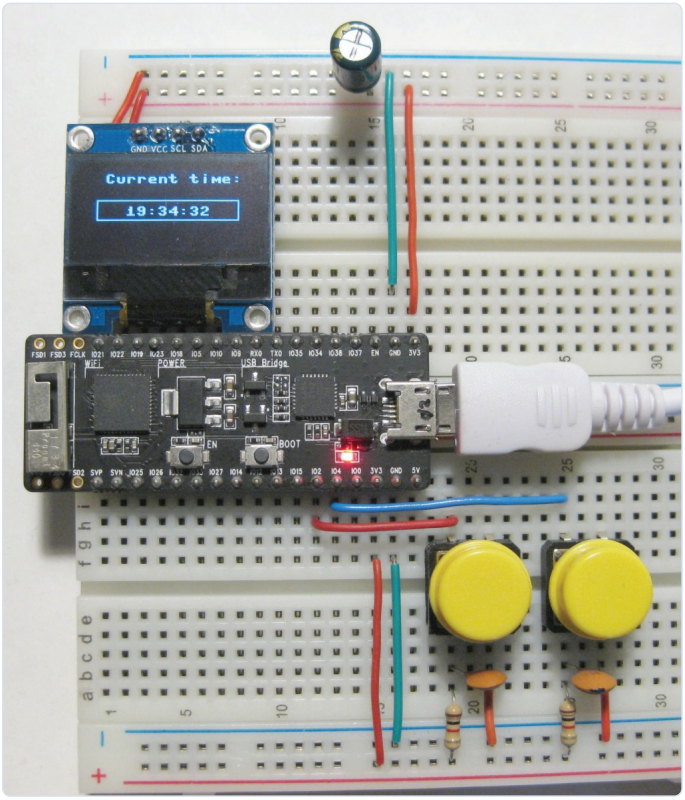

Figure 7. Horloge numérique avec affichage OLED.

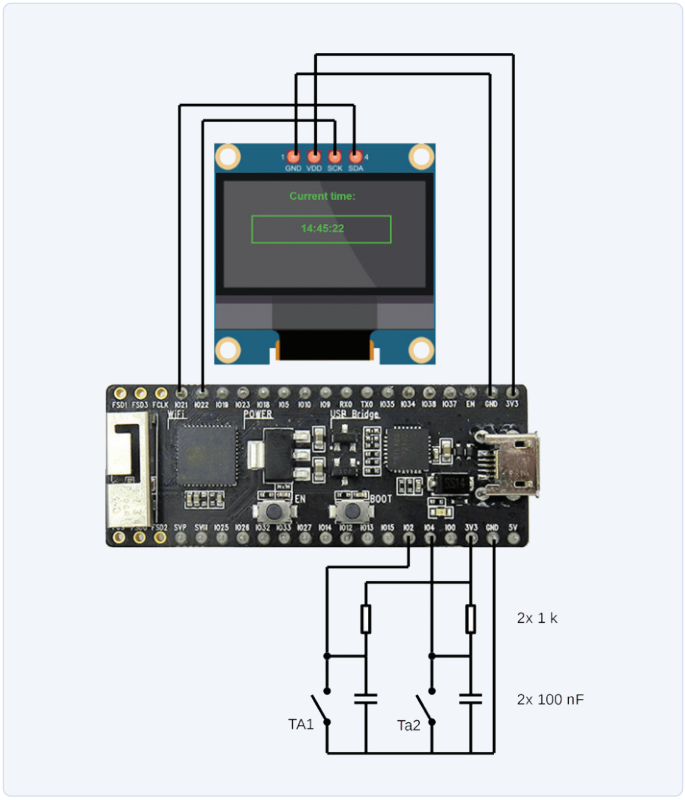

Figure 8. Schéma de l'horloge numérique programmée en MicroPython.

Listage 3. Horloge ESP32 / SSD1306 en MicroPython.

```
# setable_clock.py

from machine import Pin,I2C
import time
from ssd1306 import SSD1306_I2C

i2c = I2C(-1,scl=Pin(22),sda=Pin(21))
oled_width=128
oled_height=64
oled = SSD1306_I2C(oled_width,oled_height,i2c)

time_offset=20*3600+00*60+0 # hh+mm+ss
lin_hight=5
col_width=8

def handle_interrupt_min(pin):
  global time_offset
  time_offset+=60
  time.sleep(.2)

def handle_interrupt_hr(pin):
  global time_offset
  time_offset+=3600
  time.sleep(.2)

button_min = Pin(4, Pin.IN)
button_min.irq(trigger=Pin.IRQ_RISING,
   handler=handle_interrupt_min)

button_hr = Pin(2, Pin.IN)
button_hr.irq(trigger=Pin.IRQ_RISING,
   handler=handle_interrupt_hr)

def text_write(text, lin, col=0):
  oled.text(text, col*col_width, lin*lin_hight)

def time_text(time):
  secs=time%60
  mins=(time//60)%60
  hours=(time//3600)%24
  return "{:02d}:{:02d}:{:02d}".
   format(hours,mins,secs)

def show():
  oled.fill(0)
  text_write("Current time:",1,2)
  current_text = time_text(current_time)
  text_write(current_text,6,4)
  oled.rect(10,25,108,16,1)
  oled.show()

while True:
  current_time=time_offset+time.time()
  show()
```

```
def time_text(time):
secs=time%60
mins=(time//60)%60
hours=(time//3600)%24
return "{:02d}:{:02d}:{:02d}".format(hours,mins,secs)
```

décompose cette valeur en heures, minutes et secondes et retourne la représentation habituelle *hh:mm:ss*, (**fig. 7**). Deux boutons-poussoirs (Ta1 et Ta2) reliés aux ports 02 et 04 comme indiqué sur la **figure 8** permettent de régler l'horloge. Les boutons ont en parallèle des condensateurs de 100 nF pour l'anti-rebond. Les résistances de 1 kΩ servent de rappel. À chaque pression, les routines d'interruption associées :

```
def handle_interrupt_min(pin):
global time_offset
time_offset+=60
time.sleep(.2)
```

et

```
def handle_interrupt_hr(pin):
global time_offset
time_offset+=3600
time.sleep(.2)
```

ajoutent respectivement 60 s (1 min) ou 3600 s (1 h) à l'heure de début. Le **listage 3** donne le programme complet de l'horloge numérique.

(200581-04)

Note de l'éditeur : le livre « MicroPython for microcontrollers » est disponible à l'adresse suivante www.elektor.fr/micropython-for-microcontrollers.
Le code extrait du livre est disponible gratuitement sur la page de l'article [1].

Des questions, des commentaires ?
Contactez Elektor (redaction@elektor.fr).

Contributeurs
Texte et illustrations : **Günter Spanner**
Rédaction : **Jan Buiting**
Mise en page : **Giel Dols**
Traduction : **Helmut Müller**

LIEN

[1] **Code extrait du livre « MicroPython for microcontrollers » :** www.elektormagazine.fr/200581-04