

**SNir 1
2021-2022****Afficheur OLED**

Pré-requis : **PlatformIO** doit être installé dans votre profil.

1- Installation de la librairie

Avec l'outil PlatformIO, vous allez ajouter la librairie permettant la gestion de l'afficheur SSD1306 OLED 0,96" (128x64) avec un ESP32.

Recherche de la librairie :

```
pio lib search SSD1306Wire
```

ESP8266 and ESP32 OLED driver for SSD1306 displays

=====

#ID: 2978

I2C display driver for **SSD1306 OLED** displays connected to ESP8266, ESP32, Mbed-OS. The following geometries are currently supported: **128x64**, 128x32, 64x48. The init sequence was inspired by Adafruit's library for the same display.

Keywords: display

Compatible frameworks: Arduino

Compatible platforms: Espressif 8266, **Espressif 32**

Authors: ThingPulse, Fabrice Weinberg

nRF52_OLED

=====

#ID: 10787

An I2C/SPI display driver for SSD1306/SH1106 oled displays connected to an nRF52

Keywords: ssd1306, oled, display, i2c, spi

Compatible frameworks: Arduino

Compatible platforms: Nordic nRF52

Authors: Bernd Giesecke

Installation de manière globale de la première librairie, elle sera disponible pour d'autre projet par la suite.

```
pio lib -g install 2978
```

Elle s'installe dans votre dossier personnel : ``${HOMEDIR}`/.platformio/lib`

Le lien **github** pour l'obtenir par ailleurs : <https://github.com/ThingPulse/esp8266-oled-ssd1306>

On y trouve des exemples et de la documentation sur la librairie.

2- Connexion de l'afficheur sur votre cible ESP32



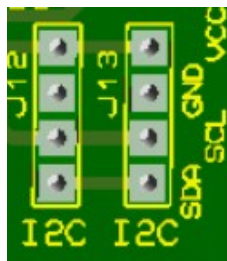
La carte Equilibreuse ESP32 dispose de deux connecteurs J12 et J13 pour le bus I2C, un peut être utilisé pour connecter l'afficheur.

Pour le fonctionnement du bus positionnez les cavaliers suivant :

Carte ESP32 cavaliers entre J10 et J11		
SCL	J10-2	J11-2
SDA	J10-5	J11-5
GND	J10-14	J11-14
3.3 V	J10-15	J11-15

Effectuez la connexion entre la carte ESP32 et la carte IOT en utilisant des fils de type male-femelle en respectant le brochage.

Les connecteurs J12 et J13 sont identiques sur la carte ESP32.



Carte IOT	Carte ESP32 connecteur I2C
SDA	Broche – J12 SDA
SCL	Broche – J12 SCL
GND	Broche – J12 GND
3V	Broche – J12 VCC

3- Préparation du projet

Réalisez un projet nommé **Afficheur** avec **PlatformIO** et votre environnement de développement.

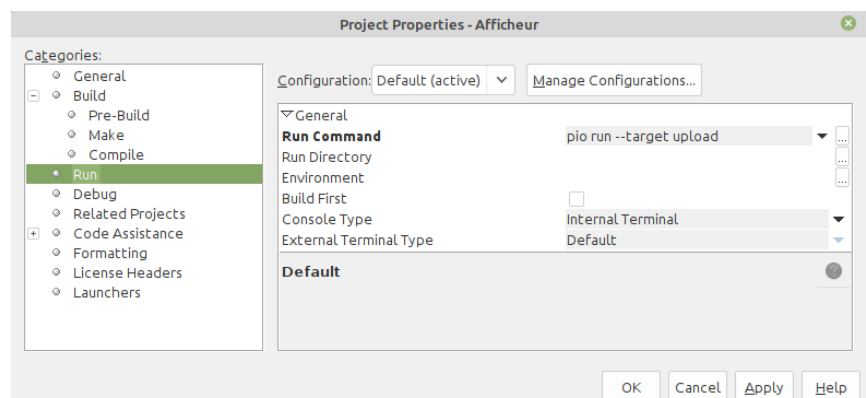
Pour cela, créez un répertoire portant le nom de votre projet **Afficheur**, ouvrez une console dans ce répertoire et lancez la commande suivante pour créez le projet :

```
pio project init --ide netbeans --board lolin32
```

Ouvrez ensuite le projet avec NetBeans. dans les propriétés du projet (Clic droit sur le projet), dans la catégorie **Run** modifier la commande de run avec la ligne suivante :

```
pio run --target upload
```

Pour avoir l'auto-complétion dans le code source, ajoutez dans les options de NetBeans (**Tools Options**) la ligne indiquant le chemin de la librairie SSD1306Wire dans la rubrique **C++ Compiler** par



Exemple :

```
/home/USERS/PROFS/pcruchet/.platformio/lib/ESP8266 and ESP32 OLED driver for SSD1306 displays/src
```

Les librairies se trouvent dans le répertoire caché **.platformio/lib** de votre dossier personnel lorsqu'elles sont installées de manière globale. Vous devez descendre jusqu'aux répertoires contenant les fichiers .h et .cpp de la librairie ici **.platformio/lib/ESP8266 and ESP32 OLED driver for SSD1306 displays/src**.

4- Exemple d'utilisation de la librairie

Voici un premier exemple qui permet d'initialiser l'afficheur, de retourner verticalement l'affichage, de sélectionner une police de caractères et d'afficher un premier message.

```
#include <Arduino.h>
#include <SSD1306Wire.h>

SSD1306Wire display(0x3c, SDA, SCL, GEOMETRY_128_64);

void setup()
{
    display.init();
    display.flipScreenVertically();
    display.setFont(ArialMT_Plain_24);
    display.drawString(0, 0, "Hello world");
    display.display();
}

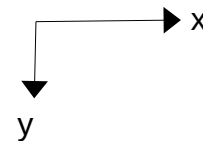
void loop()
{
}
```

La classe **SSD1306Wire** est instanciée avec le nom **display**. L'adresse de l'afficheur **0x3c** est transmise au constructeur ainsi que le numéro des broches **SDA** et **SCL**, deux constantes initialisées avec les valeurs par défaut des broches pour le bus I2C. Le dernier paramètre du constructeur indique grâce à la constante **GEOMETRY_128_64** la dimension de l'afficheur en pixels.

L'affichage se fait en x,y à partir du pixel 0,0 jusqu'au pixel 127,63

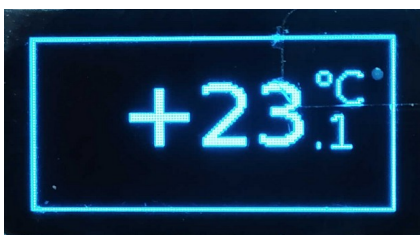
Attention, la largeur de l'écran est bien 128 et sa hauteur 64.

L'affichage est tamponné, il est nécessaire d'utiliser la méthode **display()** pour faire apparaître l'image ou le texte.



5- Application

L'application à réaliser consiste, dans un premier temps, à mettre au point une fonction chargée d'afficher la température en degré Celsius sur l'afficheur OLED comme le montre la figure ci-dessous :



La partie entière de la température utilise la police **DejaVu_Sans_32** avec affichage systématique du signe à la position.

L'unité « °C » utilise la police **DejaVu_Sans_16** à la position

Les dixièmes de degrés utilise la même police. L'écriture se fait à la même position que l'unité, seule la coordonnée en y est décalé de la hauteur de la police soit 16 pixels.

Ajoutez deux fichiers à votre projet **Afficheur.h** et **Afficheur.cpp**. Le premier contiendra l'entête des fonctions et l'inclusion de la librairie SSD1306Wire :

```
#ifndef AFFICHEUR_H
#define AFFICHEUR_H

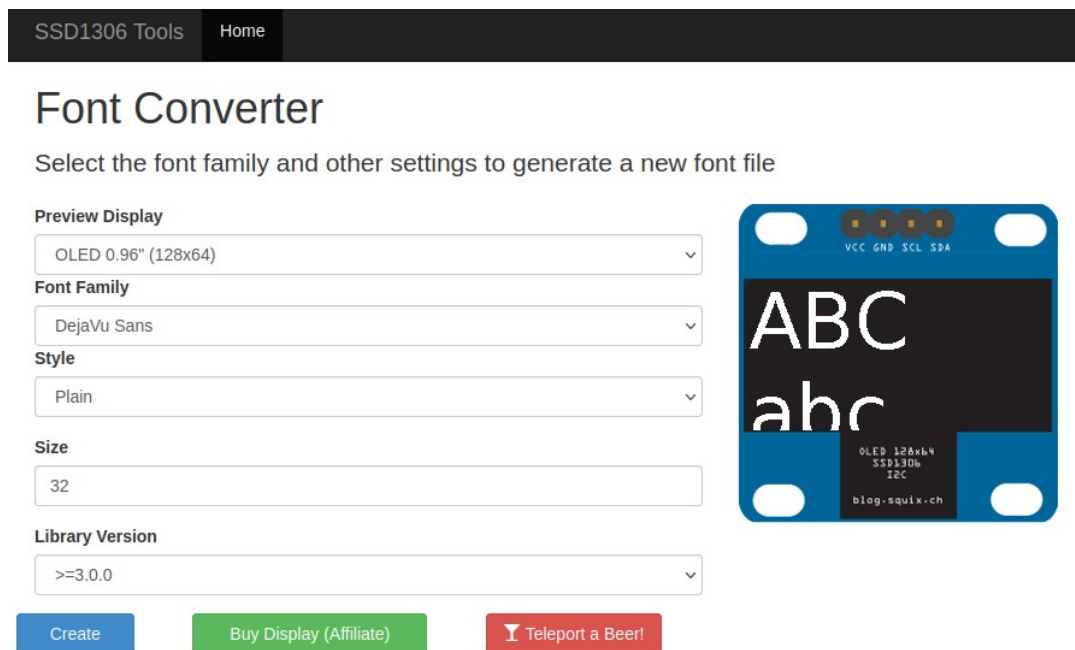
#include <SSD1306Wire.h>
void Initialiser();
void AfficherTemperature(const int _temperature, const int _dixieme, const uint16_t _dx=25, const uint16_t _dy=10);

#endif // AFFICHEUR_H
```

Le second fichier contiendra la définition des deux fonctions.

5.1- Création des polices de caractères

A partir du lien <https://oledisplay.squix.ch/>, réalisez dans un premier temps la police **DejaVu_Sans_32**



Cliquez sur **Create** et copiez dans un fichier nouveau fichier nommé **Font.h** l'ensemble du code généré. Vous apporterez les modification suivante à votre fichier :

```
#ifndef FONT_H
#define FONT_H
#include <Arduino.h>
const uint8_t DejaVu_Sans_32[] PROGMEM = {
    0x20, // Width: 32
    0x26, // Height: 38
    0x20, // First Char: 32
    0xE0, // Numbers of Chars: 224
    ...
}
```

Toujours dans le même fichier, procédez de manière similaire pour la deuxième police **DejaVu_Sans_16**.

Ajoutez l'inclusion du fichier **Font.h** dans le fichier **Afficheur.h**

5.2- Réalisation des affichages

Préparez sur une feuille de papier la position de chaque élément que vous allez devoir afficher en fonction des polices utilisées et de la taille de l'afficheur.

Dans le fichier **Afficheur.cpp**, déclarez de manière globale une instance de la classe SSD130006Wire et ajoutez la définition des deux fonctions :

```
SSD1306Wire display(0x3c,SDA,SCL,GEOMETRY_128_64);

void Initialiser()
{
}

void AfficherTemperature(const int _temperature, const int _dixieme, const uint16_t _dx, const uint16_t _dy)
{
}

}
```

La fonction **Initialiser()** doit initialiser l'afficheur et éventuellement retourner l'affichage verticalement.

Réalisez la fonction **AfficherTemperature()** en plusieurs étapes que vous validerez étape par étape par un appel de la fonction dans la fonction **setup()** de votre programme après avoir appelé la fonction **Initialiser()**.

Étape 1

- Dessiner dans un premier temps le rectangle, il doit occuper l'ensemble de l'écran de l'afficheur.

Étape 2

A partir de maintenant, les affichages tiennent compte des paramètres `_dx` et `_dy`

- Afficher la valeur entière de la température dans la police de 32 pixels en utilisant la méthode **drawString()**. Remarque pour transformer un entier en String : `String(_temperature)`

Étape 3

- Après avoir obtenu la largeur en pixel de la partie entière de la température avec la méthode **getStringWidth()**, affichez l'unité « °C » dans la police de 16.

Étape 4

- Enfin, affichez les dixièmes de degré dans la même police de 16.

Étape 5

- On souhaite afficher le signe « + » lorsque la température est positive (0 compris).

5.3- Test de la fonction

A l'aide des boutons poussoirs présents sur votre carte ESP32, vous allez réaliser le test de votre fonction **AfficherTemperature()**.

- BP1** → augmente la partie entière d'une unité,
- BP2** → décrémente la partie entière d'une unité,
- BP3** → augmente les dixièmes de degré [0 → 9]
- BP4** → décrémente les dixièmes de degré [9 → 0]

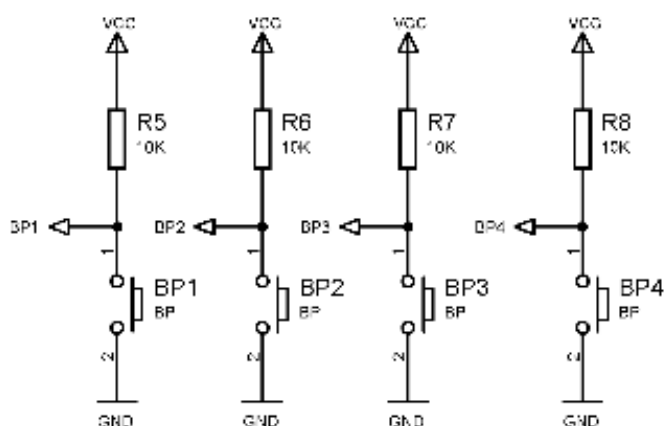
Reprenez la classe **Clavier** étudié en cours ajoutez là à votre projet puis déclarez l'instance avant la fonction **setup()**.

Ajoutez également deux variables entière `temperature` et `dixieme globale` après l'instance de **Clavier**.

Pour coder le corps de la fonction **loop()**, vous utiliserez l'instruction **switch() case...** pour traiter le cas de chaque boutons-poussoirs.

Effectuez les corrections si nécessaire dans votre fonction pour que l'affichage soit satisfaisant.

Pour rappel :



Adaptez l'énumération `TOUCHE_CLAVIER` en utilisant les noms BP1, BP2...

BP1	GPIO36
BP2	GPIO39
BP3	GPIO34
BP4	GPIO35

6- Description de la librairie

6.1- Méthodes pour le contrôle de l'afficheur

void init();	Initialise l'afficheur
void end();	Libère la mémoire utilisée par l'afficheur
void resetDisplay(void);	Ré-initialise l'afficheur
void reconnect(void);	Connecte à nouveau l'afficheur au bus I2C
void displayOn(void);	Allume l'écran
void displayOff(void);	Éteint l'écran
void clear(void);	Efface le tampon de l'afficheur
void display(void);	Écrit le contenu du tampon dans la mémoire de l'afficheur
void invertDisplay(void);	Inverse le mode d'affichage
void normalDisplay(void);	Affiche dans le mode normal
void flipScreenVertically();	Retourne le sens d'affichage verticalement
void mirrorScreen();	Dessine l'écran en miroir
void setBrightness(uint8_t);	Fixe la luminosité de l'afficheur
void setContrast(uint8_t contrast, uint8_t precharge = 241, uint8_t comdetect = 64);	Fixe le contraste de l'afficheur

Pour un affichage normal la valeur de la luminosité et du contraste est 100

Pour un affichage vraiment faible la valeur du contraste et de la luminosité est 10

6.2- Méthodes pour l'affichage de pixels

void setColor(OLEDDISPLAY_COLOR color);	Fixe la couleur pour toutes les opérations sur les pixels valeurs possibles : BLACK, WHITE, INVERSE
void setPixel(int16_t x, int16_t y);	Dessine un pixel à la position donnée
void drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1);	Dessine une ligne entre les points 0 et 1
void drawRect(int16_t x, int16_t y, int16_t width, int16_t height);	Dessine le bord d'un rectangle à la position indiquée
void fillRect(int16_t x, int16_t y, int16_t width, int16_t height);	Rempli le rectangle
void drawCircle(int16_t x, int16_t y, int16_t radius);	Dessine un cercle
void fillCircle(int16_t x, int16_t y, int16_t radius);	Rempli le cercle
void drawTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2, int16_t y2);	Dessine le bord d'un triangle
void fillTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2, int16_t y2);	Rempli le triangle
void drawHorizontalLine(int16_t x, int16_t y, int16_t length);	Dessine une ligne horizontale
void drawVerticalLine(int16_t x, int16_t y, int16_t length);	Dessine une ligne verticale
void drawProgressBar(uint16_t x, uint16_t y, uint16_t width, uint16_t height, uint8_t progress);	Dessine une barre de progression arrondie avec les dimensions extérieures données. progress est une valeur non signée comprise entre 0 et 100

6.3- Méthodes pour l'affichage d'images

<code>void drawFastImage(int16_t x, int16_t y, int16_t width, int16_t height, const uint8_t *image);</code>	Dessine une image bitmap
<code>void drawXbm(int16_t x, int16_t y, int16_t width, int16_t height, const uint8_t *xbm);</code>	Dessin une image au format XBM

Pour créer une image Bitmap : <https://miliohm.com/how-to-draw-or-print-bitmap-to-oled-display-arduino/>

Format d'image XBM : https://fr.wikipedia.org/wiki/X_BitMap

6.4- Méthodes pour le texte

<code>void drawString(int16_t x, int16_t y, const String &text);</code>	Affiche le texte à la position indiquée
<code>void drawStringf(int16_t x, int16_t y, char* buffer, String format, ...);</code>	Affiche un texte formaté comme printf à la position indiquée.
<code>void drawStringMaxWidth(int16_t x, int16_t y, int16_t maxLineWidth, const String &text);</code>	Affiche une chaîne avec une largeur maximale à l'emplacement donné. Si la chaîne donnée est plus large que la largeur spécifiée le texte sera renvoyé à la ligne suivante avec un espace ou un tiret
<code>uint16_t getStringWidth(const char* text, uint16_t length);</code> <code>uint16_t getStringWidth(const String &text);</code>	Renvoie la largeur de la chaîne en pixel suivant la police actuelle
<code>void setTextAlignment(OLEDDISPLAY_TEXT_ALIGNMENT textAlignment);</code>	Spécifie l'alignement du texte par rapport au point d'ancrage. Les valeurs possibles sont : par défaut TEXT_ALIGN_LEFT (à partir du point en haut à gauche), TEXT_ALIGN_CENTER , TEXT_ALIGN_RIGHT , TEXT_ALIGN_CENTER_BOTH
<code>void setFont(const uint8_t* fontData);</code>	Détermine la police de caractères. Les polices disponibles par défaut sont : ArialMT_Plain_10, ArialMT_Plain_16, ArialMT_Plain_24

Pour créer de nouvelle police utiliser l'outil de création de police : <http://oleddisplay.squix.ch>

Seules les principales méthodes de la classe **SSD1306Wire**, d'autres fonctions existent, la description se trouve soit dans le github de l'auteur ou dans les fichiers .h de la librairie.