Ecran OLED I2C

D.Moreaux

28 mars 2021

1 Ecrans OLED U8G2

La bibliothèque U8G2 permet de gérer un ensemble de petits écrans OLED (32x128, 64x128, ...) connectés à l'arduino en SPI [1] ou en I2C [2].

Elle permet d'utiliser les ports SPI ou I2C hardware de l'arduino ou de gérer les protocoles au niveau logiciel sur des broches différentes. Le port I2C utilise A4 pour SDA et A5 pour SCL et le port SPI utilise les ports 13 (SCK), 12 (MISO), 11 (MOSI) et généralement 10 pour CS (aussi appelé SS dans certains cas). A noter que les ports MISO, MOSI et SCK sont aussi disponibles sur le connecteur 6 broches qui sert à programmer le microcontroleur directement (les trois autres broches sont RST (Reset), GND et +5V).

Pour utiliser la bibliothèque U8G2, il faut d'abord l'installer (U8g2, à ne pas confondre avec l'ancienne bibliothèque U8glib).

Cette bibliothèque propose pas loin de 700 fontes mais seules celles utilisées seront chargées dans l'arduino.

Le type d'écran (dimensions, contrôleur, mode de communication) ainsi que le mode de travail (buffer de 128, 256 ou 1024 octets) sont déterminés lors de la création de l'objet qui servira à accéder à la bibliothèque.

Dans le cas de l'écran utilisé au cours, on utilisera

```
U8G2_SSD1306_128x64_NONAME_1_HW_I2C u8g2(U8G2_R0, U8X8_PIN_NONE);
U8G2_SSD1306_128x64_NONAME_2_HW_I2C u8g2(U8G2_R0, U8X8_PIN_NONE);
U8G2_SSD1306_128x64_NONAME_F_HW_I2C u8g2(U8G2_R0, U8X8_PIN_NONE);
```

1. La connexion SPI est une connexion synchrone utilisant 4 fils : SCK (horloge), MISO (Master In Slave Out, de l'écran à l'arduino), MOSI (Master Out Slave In, de l'arduino à l'écran) et CS (Chip Select). Chaque composant sur la chaîne SPI a son propre signal CS
2. La connexion I2C est une connexion synchrone utilisant 2 fils : SCL (Horloge) et SDA (Données). Les périphériques disposent d'une adresse et le sens des données est déterminé par les commandes envoyées par l'Arduino

La première ligne utilisera un buffer de 128 octets (un groupe de 8 lignes de 128 pixels), la seconde de 256 octets (deux groupes de lignes) et la troisième un buffer complet (1024 octets, 8 groupes de 8 lignes de 128 pixels)

Dans le buffer, chaque octet représente une ligne verticale de 8 pixels. Si on utilise les modes 1 et 2, il sera donc nécessaire d'envoyer respectivement 8 et 4 fois des informations à l'écran, le mode F enverra la totalité en une fois.

La valeur U8G2_R0 indique la rotation de l'écran. HW I2C indique que l'on utilise le port I2C hardware (les deux broches dédiées ou les broches A4 et A5). Attention, certains contrôleurs sont très similaires mais peuvent faire apparaître un décallage au niveau image.

2 Initialisation et squelette du code d'affichage

Pour utiliser la bibliothèque U8g2, on commencera par les lignes suivantes :

```
#include <U8g2lib.h>

#ifdef U8X8_HAVE_HW_SPI
#include <SPI.h>
#endif
#ifdef U8X8_HAVE_HW_I2C
#include <Wire.h>
#endif

U8G2_SSD1306_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0, U8X8_PIN_NONE);
```

Au niveau du setup, on appellera `u8g2.begin();` qui initialisera l'écran, l'effacera et le préparera à recevoir son contenu.

Dans le mode full (F), on utilisera la structure suivante pour mettre à jour l'affichage

```
u8g2.clearBuffer();
// Insérer ici les commandes de tracé
u8g2.sendBuffer();
```

Dans les modes 1 et 2, il sera nécessaire d'envoyer l'image en plusieurs morceaux. Pour ce faire, il faudra la retracer pour chaque morceau. On utilisera dès lors le code suivant :

```
u8g2.firstPage();
do {
```

```
  // Insérer ici les commandes de tracé
} while(u8d2.nextPage);
```

3 Tracé

3.1 Texte

setFont Avant d'afficher du texte, il faut choisir une fonte à l'aide de `u8g8.setFont(fontname)`. Le nom de fonte définira la taille des caractères et leur aspect. Les fontes seront stockées dans la mémoire programme à la compilation. La liste des fontes disponibles se trouve sur le site de la bibliothèque : https ://github.com/olikraus/u8g2/wiki/fntlistall

Dans la plupart des cas, la variante "r" des fontes (caractères ASCII de 32 à 127) est suffisante. Si on désire disposer des caractères accentués, il sera nécessaire d'utiliser une variante "f". La taille est la taille de la lettre A majuscule

Nom	taille	commentaire
u8g2_font_tinytim_tr	5	
u8g2_font_courR08_tr	6	Normal
u8g2_font_courB08_tr	6	Gras
u8g2_font_amstrad_cpc_extended _8r	8	Style Amstrad CPC

Le nombre de fontes est énorme et inclus également des fontes spécialisées dans les icônes ou dans le support UTF-8.

drawGlyph/drawStr/drawUTF8 La fonction `u8g8.drawGlyph(x,y,utf)` permet d'afficher un caractère UTF-8 à une position donnée de l'écran. *utf* est la valeur du caractère, resteinte à la première page (page 0) de unicode (de 0 à 65535).

La fonction `u8g8.drawStr(x,y,str)` permet d'afficher une chaîne de caractères dont les codes vont de 0 à 255. Pour les codes supérieurs à 127, il sera nécessaire d'utiliser la notation `\x80` pour insérer le caractère dans la chaîne (l'environnement Arduino travaille en UTF-8)

La fonction `u8g8.drawUTF8(x,y,str)` permet elle d'afficher une chaîne de caractères encodés en UTF-8. A noter que la fonction `strlen` ne fonctionne pas sur du texte UTF-8

print La fonction `u8g8.print(...)` fonctionne comme les autres fonction `print` de la bibliothèque Arduino : elle accepte une chaîne mais aussi une

valeur numérique ou autre, elle écrit à la position du curseur et avance le curseur et elle supporte la macro F().

Le support pour l'UTF-8 peut être activé à l'aide de `u8g8.enableUTF8Print()` et désactivé à l'aide de `u8g8.disableUTF8Print()`.

Le curseur peut être positionné à l'aide de `u8g8.setCursor(x,y)` ou être amené à la position (0,0) à l'aide de `u8g8.home()`. A noter que la position des caractères par rapport au curseur dépend du mode d'affichage.

Les fonctions `u8g8.setFontPosBaseline()`, `u8g8.setFontPosBottom()`, `u8g8.setFontPosTop()` et `u8g8.setFontPosCenter()` permettent d'indiquer où se trouve le curseur sur le caractère. La valeur par défaut, Baseline, situe le curseur sur la ligne d'écriture (les lettres tels que p et q descendent plus bas). Une valeur Top permet de positionner par rapport au coin supérieur gauche des caractères (idéal si on désire utiliser `home()`).

options La fonction `u8g8.setFontMode(trans)` permet, si la fonte est transparente, d'activer la transparence. Si trans vaut 0, les points éteints de la fonte seront remplacés par la couleur de fond.

La fonction `u8g8.setDrawColor(color)` permet d'afficher en vidéo inverse(0), en vidéo normale(1) ou en XOR(2). Cette fonction affecte tous les tracés sauf `clear()`.

La fonction `u8g8.setFontDirection(dir)` permet de changer la direction de tracé pour les `drawStr`. 0 est le sens "normal", 1 représente une écriture vers le bas, 2 à l'envers et 3 vers le haut.

3.2 Interface Utilisateur

La bibliothèque dispose de fonctions pour faciliter une interface utilisateur. Pour se faire, lors du `u8g8.begin()` il faudra indiquer un certain nombre de broches qui serviront d'entrées pour des boutons de navigation.

La fonction `begin()` peut prendre de 3 à 6 paramètres (les trois premiers sont nécessaires pour les interfaces) qui indiquent les broches où sont connectés (dans l'ordre) les boutons SELECT, NEXT, PREV, UP, DOWN, HOME. Il s'agit de boutons reliés à la masse (entrées en INPUT_PULLUP).

La fonction `getMenuEvent()` retourne 0 si aucun des boutons n'est enfoncé ou une des valeurs définies par les macros suivantes :

```
U8X8_MSG_GPIO_MENU_SELECT
U8X8_MSG_GPIO_MENU_NEXT
U8X8_MSG_GPIO_MENU_PREV
U8X8_MSG_GPIO_MENU_HOME
U8X8_MSG_GPIO_MENU_UP
```

`U8X8_MSG_GPIO_MENU_DOWN`

`u8g8.userInterfaceInputValue(titre,pre,uint8\_t *val, lo,hi,dig,post)`

permet de demander à l'utilisateur d'entrer une valeur 8 bits.

Les chaînes titre, pre et post correspondent respectivement à la description (multilignes, séparer les lignes à l'aide de
n), le texte avant la valeur entrée et le texte après la valeur.

Les valeurs lo et hi donnent les valeurs minimales et maximales acceptables et val est un pointeur vers un entier qui permettra de récupérer la valeur entrée. dig indique le nombre de chiffres (de 1 à 3).

La fonction retournera 1 si une valeur est entrée (et validée à l'aide de SELECT) et 0 si l'entrée a été abandonnée à l'aide de HOME.

`u8g8.userInterfaceSelectionList(title,start,opt)` permet de demander à l'utilisateur de choisir une option dans une liste. Le titre (sur une ligne) sera affiché sur le haut de l'écran, la chaîne opt contient les options séparées par un
n et start indique l'option sélectionnée par défaut (de 1 à n).

Cette fonction retourne une valeur de 1 à n si une option a été choisie et 0 si on a annulé à l'aide de HOME

`u8g8.userInterfaceMessage(title1,title2,title3, but)` permet d'afficher un message accompagné de un ou plusieurs boutons.

title1 et title3 sont multilignes (séparées par
n), title2 est sur une simple ligne. but est une chaîne qui contient le ou les boutons (séparés par
n).

Cette fonction retournera une valeur de 1 à n pour indiquer le bouton sélectionné ou 0 si ona a annulé à l'aide de HOME.

3.3 Dessin

Les fonctions de tracés peuvent être restreintes à la zone définie dans un rectangle défini par `u8g8.setClipWindow(x0,y0,x1,y1)`. Les tracés qui sorte du rectangle de (x0,y0) inclus à (x1,y1) exclu seront ignorés. Utiliser `u8g8.setMaxClipWindow()` pour restaurer l'affichage normal.

drawFrame(x,y,w,h)	trace un rectangle
drawBox(x,y,w,h)	trace un rectangle rempli
drawCircle(x,y,r)	trace un cercle de rayon r et de centre (x,y). Un paramètre optionnel permet de préciser les quadrants à afficher
drawDisc(x,y,r)	trace un cercle plein. Un paramètre optionnel permet de préciser les quadrants
drawEllipse(x,y,rx,ry)	trace une ellipse. Un paramètre optionnel permet de préciser les quadrants
drawFilledEllipse(x,y,rx,ry)	trace une ellipse remplie. Un paramètre optionnel permet de préciser les quadrants
drawHLine(x,y,w)	trace une ligne horizontale
drawVLine(x,y,h)	trace une ligne verticale
drawLine(x0,y0,x1,y1)	trace une ligne quelconque
drawPixel(x,y)	trace un pixel
drawRFrame(x,y,w,h,r)	trace un rectangle à coins arrondis. r est le rayon des arrondis
drawRBox(x,y,w,h,r)	trace un rectangle plein à coins arrondis
drawTriangle(x0,y0,x1,y1,x2,y2)	trace un triangle plein

Les tracés de cercle et d'ellipse permettent un paramètre supplémentaire qui indique un ou plusieurs quadrants à afficher. Les valeurs définies ci-dessous correspondent chacune à un quadrant, si on veut en afficher plusieurs, il faut les combiner avec `X`.

```
U8G2_DRAW_UPPER_RIGHT
U8G2_DRAW_UPPER_LEFT
U8G2_DRAW_LOWER_LEFT
U8G2_DRAW_LOWER_RIGHT
U8G2_DRAW_ALL
```

Si le paramètre est absent, il considérera une valeur ALL par défaut.

4 Images

La bibliothèque U8g8 permet également d'afficher des images. Ces images doivent être au format XBM. Ce format est un format standard unix qui dé-

fini une image comme un morceau de code C pouvant être inclus directement qui contient deux #define pour les dimensions et un tableau pour les données.

Le tableau sera défini comme (XXXX sera le nom de l'image)

```
static␣unsigned␣char␣XXXX_bits[]␣=␣{␣...
static␣const␣unsigned␣char␣XXXX_bits[]␣U8X8_PROGMEM␣=␣{␣...
```

La deuxième forme permet de stocker le tableau dans la mémoire programme (pour ne pas encombrer la RAM) et sera préférée sur les AVR ATMega328 (Arduino, Arduino Nano, Arduino Micro, RGBDuino, ...), ATMega32U8 (Arduino Leonardo, Arduino Micro Pro) et ATMega2560 (Arduino Mega).

L'image sera affichée par une des commandes

```
void u8g2.drawXBM(x, y, w, h, bitmap)
void u8g2.drawXBMP(x, y, w, h, XXXX_bits)
```

Les #define seront XXXX_width et XXXX_height et peuvent être utilisées directement dans la fonction `drawXBM`. La version `drawXBMP` sera utilisée si le tableau est situé en mémoire programme.

Le post suivant https://sandhansblog.wordpress.com/2017/04/16/interfacing-displaying-a-custom-graphic-on-an-0-96-i2c-oled/ décrit comment convertir une image dans le format XBM à l'aide de TheGimp.