CH-1092 Belmont, Switzerland
Tel +41 21 728-6156, Fax 728-6157
Email info@didel.com, www.didel.com

Nouveau : Tous nos produits avec leur doc
Accessibles en 2-3 clicks : www.didel.com/Galerie

Expérience avec le Oled I2C SSD1306

Programmes sous www.didel.com/Oled1306.zip
Compatibilité 128x32 en fin de document

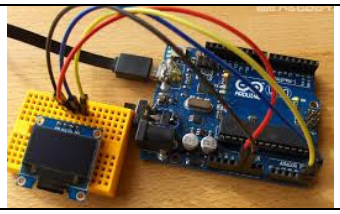
1 Introduction

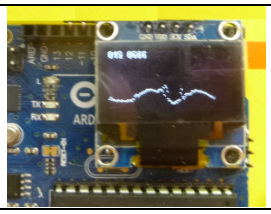
Ce document s'adresse à des intéressés à la programmation, qui connaissent Arduino et veulent approfondir leur connaissance. Même ceux qui ont suivi le MOOC EPFL (et peut-être surtout ceux-là) peuvent profiter de notre progression pas-à-pas et des explications associées. Arduino est excellent pour voir ce que l'on peut faire, mais c'est comme IKEA: vous câblez comme sur le dessin, vous chargez le programme, et vous êtes content.

L'affichage Oled SSD1306 est très utile pour visualiser des signaux, remplaçant le terminal Arduino, avec l'avantage de faire partie de l'application contrairement au terminal série.

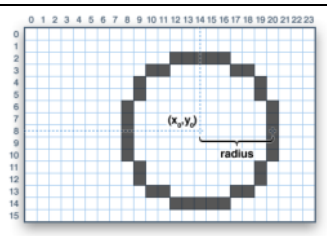
2 Oled 1306 I2C Arduino

Le composant qui va nous faire plaisir est l'affichage Oled 1306 I2C facile à obtenir pour une dizaine de francs. Il se branche traditionnellement sur les sorties I2C (A4 A5), mais notre soft offre plus de liberté, ce qui permet de le câbler ou cela nous arrange.





Pour utiliser la librairie AdaFruit; chercher "adafruit-gfx-graphics-library". Le graphisme est orienté texte et géométrie.
La documentation Didel 2015 utilise cette libraire et propose des exemples
www.didel.com/diduino/OledI2C.pdf
www.didel.com/diduino/OledI2C.zip



3 Librairie compacte LibOledPix

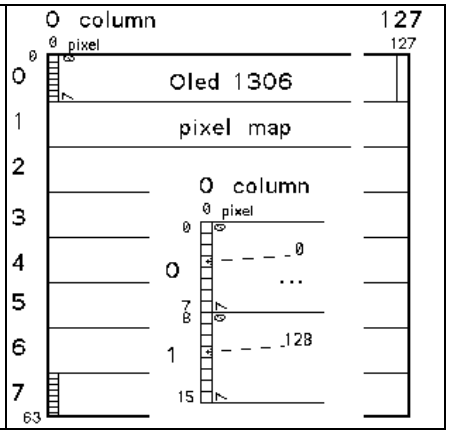
La librairie AdaFruit est volumineuse et nécessite 1 kbyte de mémoire RAM. **LibOledPix** écrit directement sur le Oled, les programmes sont plus courts, le chargement plus rapide, les fonctions plus simples à mémoriser. De plus, le câblage est possible sur 2 pins quelconques d'un même port.

Pour observer les valeurs d'un capteur sur un robot qui roule, suivre le robot avec un portable qui affiche les valeurs sur le terminal série n'est pas une solution. Transmettre par Wifi ou BT n'est maîtrisé que par des spécialistes, et il faut alors plus performant qu'un Arduino !
Un Oled est simple et sympa, comme vous allez voir.

Utilisons la librairie OledPix. Si vous connaissez bien la librairie AdaFruit, c'est facile de transposer les noms, les fonctions de base sont similaires.
Cette librairie a la forme d'un fichier inclus. Il n'y a rien à installer. Chaque exemple que vous trouvez dans le zip contient les fichiers nécessaires. Le croquis contient le .ino et les .h, Vous pouvez modifier et sauver avec Tool – ArchiveSketch. Très pratique puisque la date est mentionnée, un back-up sans effort!
Si on modifie un fichier inclus, la modification n'est pas reportée dans les autres croquis qui ajoutent ce fichier. C'est parfois embêtant pour reporter partout une amélioration ou correction. Mais c'est excellent pour apprendre à programmer en modifiant les programmes; ce que l'on modifie ne touche que le programme chargé.

4 Ecran du SSD1306

Voir http://www.didel.com/OledLib.pdf pour des explications générales pour le SSD1306 (en anglais). On utilise ici la librairie OledPix, et le but est d'utiliser l'affichage pour aider à la mise au point de programmes utilisant des capteurs et des machines d'état.
Il faut savoir que les nombres et textes ont une coordonnée de début en ligne/colonne et les pixels en x/y. Abscisse y et colonne sont identiques. Les lignes de texte vont de 0 à 7 et les pixels de 0 à 63, origine en haut.
Avant de positionner un caractère ou un nombre, il faut spécifier sa position avec la fonction LiCol (li,co);. La fonction Dot(x,y); travaille avec des coordonnées plus fines.



Exemple `LiCo(2,0); Car('a');` place un a en début de 2e ligne.
Par contre `LiCo(7,100); DecBig(123);` qui cherche à mettre en bas d'écran un nombre décimal de 3 digits en gros caractères ne pourra pas afficher tout le nombre, il a besoin de 30 colonnes.
Si les noms des fonctions de la libraire ne conviennent pas, c'est facile de redéclarer des synonymes,

5 Variantes de la librairie

Ne nous laissons pas troubler par des variantes qui doivent être mentionnées. Sur les processeurs, une interface I2C réserve 2 pins et de la logique interne facilite l'utilisation de l'I2C, en particulier lorsqu'il y a des interruptions. On peut aussi programmer I2C sur n'importe quelle pin (on parle de "bit-banging", abbrégé ici bb). Cela coûte quelques instructions, mais en fait moins que les librairies qui ajoutent un buffer en prévision des interruptions. Comme débutant, la solution bb nous convient très bien. Elle a l'avantage que l'on peut se brancher sur les pins qui nous arrangent. OledPix.h est de ce type.

Il y a encore deux variantes de notre librairie. Ecrire sur l'écran directement comme on le fait avec OledPix est simple, mais on ne peut pas relire. De plus, on écrit des bytes, et pas des pixels.
La librairie GFX et notre librairie OledMap (en annexe) travaillent avec un bitmap en mémoire de 1kilobytes dans lequel on peut écrire, lire, modifier. C'est rapide, mais pour voir les modifications, il faut copier tout le bitmap, c'est très lent !
Notre librairie OledPix écrit directement. Il y a parfois des pixels écrasés et cela ne convient pas pour dessiner des figures géométriques, mais pour interagir avec des capteurs et mettre à jour l'écran en temps réel, c'est le plus efficace.
Le but est l'apprentissage, puis une aide au dépannage et c'est bien de laisser les lignes I2C libres pour les applications professionnelles.
Il faudra inclure le fichier "OledPix.h" et mettre dans le setup la fonction SetupOledPix();
OledPix.h contient les définitions pour le câblage (7 lignes à modifier pour déplacer le Oled)) et des fonctions graphiques qui seront vues progressivement.

6 Câblage

La librairie OledPix permet de câbler le 1306 sur n'importe quelle pin. L'affichage ne consomme que quelques mA, on va donc alimenter par les pins. Toutefois, les timings sont critiques et on ne peut pas utiliser les fonctions pinMode et digitalWrite. Il faut des définitions en C, expliquées dans www.didel.com/coursera/LC1.pdf

Les définitions pour ce câblage I2C sont les suivantes. On remarque que tout est déclaré dans les 7 premières lignes: choix du port et des pins. Tout ce qui suit est indépendant du câblage.

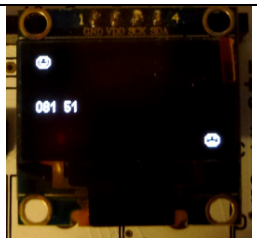
```
#define Ddr  DDRD
#define Port PORTD
#define Pin  PIND
#define bCk 5
#define bDa 4
#define bGnd 7
#define bVcc 6  // la suite est générale
#define Ck1  bitClear (Ddr,bCk)
#define Ck0  bitSet (Ddr,bCk)
#define Da1  bitClear (Ddr,bDa)
#define Da0  bitSet (Ddr,bDa)
```

```
void SetupI2C() {
 Ddr  |=  (1<<bGnd|1<<bVcc|1<<bCk|1<<bDa) ;
 Port |=  (1<<bVcc) ;
 Port &= ~(1<<bGnd|1<<bCk|1<<bDa) ;
}
```

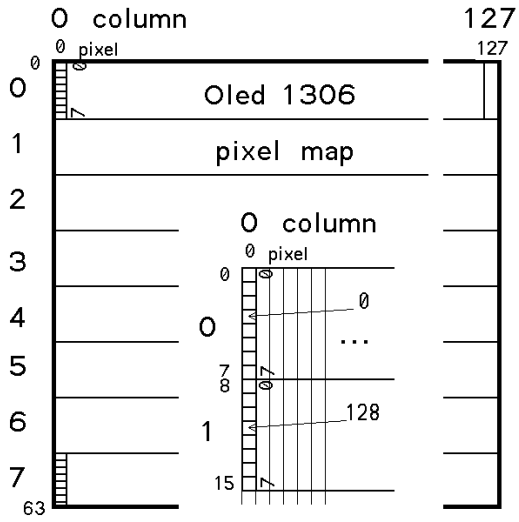
Si les opérations logiques ne vous sont pas familères, http://www.didel.com/C/OperationsLogiques.pdf donne les bases nécessaires. Arduino est très lourd et peu efficace quand on doit agir sur plusieurs bits d'un même mot. Et trop lent dans notre cas.

7 Afficher des nombres

Tous les programmes auront la même initialisation, complétée par l'initialisation des capteurs et actionneurs dans d'autres documents.
Ce premier programme positionne deux sprites selon la grille écran (section 4) affiche en binaire, hexadécimal et décimal une variable 8 bits qui augmente. La fonction Licol() positionne le pointeur pour la prochaine écriture. Les 2 librairies doivent être appelées et initialisées. Le nom byte n'est pas toujours connu du compilateur, il faut le redéclarer.



```
//TestLibOledPix  170601
typedef uint8_t byte;
#include "OledI2Cbb.h"
#include "OledPix.h"
void setup(){
  SetupI2Cbb();
  SetupOledPix();
}
byte var = 33; // -- variable globale
void loop(){
  LiCol(0,0);  Sprite(smile);  // top
  LiCol(4,0);  Dec8(var);
  LiCol(4,30); Hex8(var);
  (var++);  // on compte  var++;     // on
compte
  LiCol(7,118); Sprite(sad); // bas d'écran
  delay(500);
}
```



Les fonctions à disposition sont

Licol(li,co);	. Place le pointeur ligne li (0 à 7) et en colonne co (0 à 127, mais il faut la place pour finir)
Bin8(v);	. Affiche la variable 8 bits (byte, char, int8_t, uint8_t) en binaire
Hex8(v);	. Affiche la variable 8 bits (byte, char, int8_t, uint8_t) en binaire
Hex16(v);	. Affiche la variable 16 bits (int, int16_t uint6_t) en binaire
Dec8(v);	. Affiche la variable 8 bits (byte, char, int8_t, uint8_t) en binaire
Dec9999(v);	. Affiche la variable 16 bits (int, int16_t uint6_t) en décimal, limite 0x270F=9999 note: Dec16 aurait utilisé 5 chiffres, et c'est rare que l'on doive afficher des nombres plus grands que 9999 en interagissant avec des capteurs.

En double taille (utilise la ligne en-dessus, donc lignes 2, 4, 6, 8):

Big(); (ou BigCar) BigBin8(); BigHex8(); BigHex16(); BigDec8(); BigDec9999();

Modifiez le programme que vous venez de charger pour essayer les fonction. Sauvez sous un autre nom quand vous êtres satisfait du test.

Les noms ne vous plaisent pas? Vous pouvez leur donner un synonyme avec le #define, par exemple
#define BD9() BigDec9999() // attention, pas de ; final

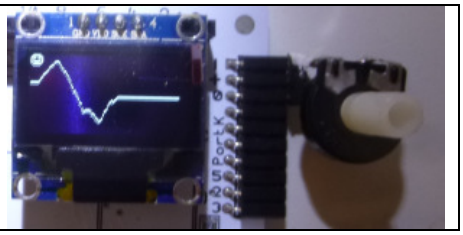
8 Afficher des points et des graphiques

L'origine des coordonnées est en haut à gauche (section 4).
Utiliser l'origine "scolaire" nécessite une simple soustraction: x= (63-x).
Ecrire un point efface les autres pixels du byte écran qu'il contient. Cela n'est pas gênant pour la seule application qui nous intéresse: visualiser des suites de mesures.
Les primitives sont

Dot(x,y); DDot(x,y); Vline(x); Hline(y);

DDot (x,y) affiche 2 points superposés. C'est pratique si on veut visualiser 2 courbes.

Vous avez un potentiomètre à brancher sur A0?
analogRead(A0); donne une valeur de 0 à 1023. On peut diviser pour rester dans les 64 pixels verticaux. On peut saturer pour rester dans une zone.
x progresse toutes les 50ms par exemple. Quand x arrive en fin d'écran, on peut effacer ou non.



Voila le programme à essayer si vous n'êtes pas encore en confiance pour l'inventer.

```
//TestPotOledPix  170520
typedef uint8_t byte;
#include "OledI2Cbb.h"
#include "OledPix.h"
void setup(){
  SetupI2Cbb();
  SetupOledPix();
}
byte x,y;
void loop(){
  LiCol(0,0); Sprite(smile);
  y = analogRead(A0)/16;  // 0-1023 --> 0-63
  Dot (x,y);
  x++; if(x==128) { x=0; Clear(); }
  delay(200);
}
```

Exercices: diagonales, segments horizontaux et verticaux, etc
Ecrire des fonctions qui dessinent un bout de segment.

9 Afficher des lutins (sprite) et des textes

L'écran est fait de bytes alignés verticalement. Licol (li,co); positionne le pointeur sur un byte. Facile de copier une table à partir de cette position.
Pour le lutin smile, la table est

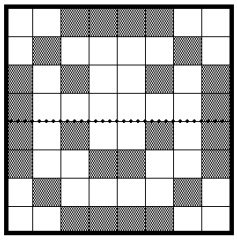
```
  byte  mysmile[]= {0x3c,0x42,0x95,0xa5,0xa1,0xa1,
0xa5,0x95,0x42,0x3c};
```

L'instruction pour copier à partir du ponteur est

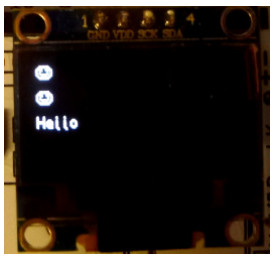
```
 MySprite[mysmile];
```

L'utilisation de la fonction Sprite, délicate pour un débutant, sera expliquée dans un autre document.
Pour un texte, on procède de façon similaire. Il faut déclarer un textes comme une table. Les caractères ASCII de 0x20 à 0x7F sont connus, mais pas les \n et autres.
Le programme TestTextOledPix.ino permet de tester et compléter.
Après le setup, on a

```
 byte  mysmile[]= {0x3c,0x42,0x95,0xa5,0xa1, \
                   0xa1,0xa5,0x95,0x42,0x3c};
void loop(){
  LiCol(0,0); Sprite(smile);
  LiCol(2,0); MySprite(mysmile);
  LiCol(4,0); Text("Hello");
  delay(200);
}
```



3c 42 95 a5 a1 a1 95 42 3c



10 SSD1306 en format 128x32

En appelant la fonction `DoubleH();` (qui envoie les commandes 0xda et 0x02) on limite le nombre de lignes à 4. Avec le 128x64, les lignes sont dédoublées, les caractères sont plus lisibles avec le même temps d'écriture. Big est encore plus grand.
Avec un 128x32, la présentation est normale.

Signalez les difficultés de compréhension et manque d'information. Merci.

jdn 170601/171115 /200208