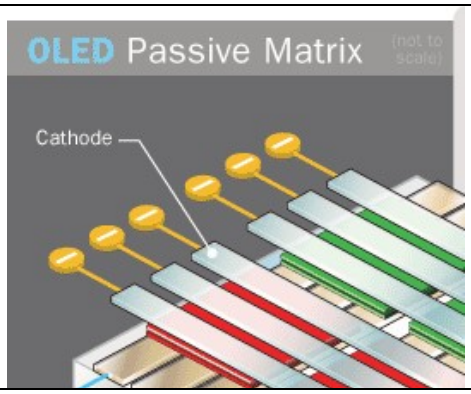
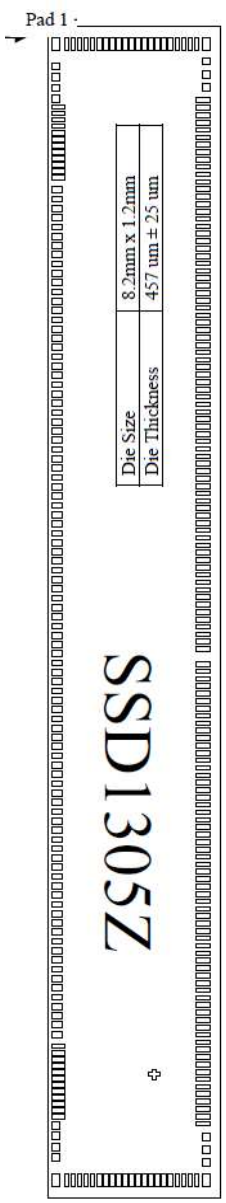
Découverte de l'Oled SSD1306

On parle d'écrans Oled de taille considérable, d'écrans souples pour tablette, la technologie promet beaucoup. Programmer l'écran d'un téléphone mobile nécessite une équipe de spécialistes, mais on peut se faire plaisir à comprendre et programmer un écran Oled miniature en C dans l'environnement de programmation Arduino.

Technologie

Pour afficher un pixel sur un écran, on utilse le même principe que pour accéder un bit dans une mémoire d'ordinateur. Une grille de conducteurs permet de sélectionner un point d'intersection et de l'exciter avec une tension qui allume un pixel.
On trouve plus d'explications sous https://www.ecranflexible.com/definition-technologie-oled et sous https://electronics.howstuffworks.com/oled3.htm

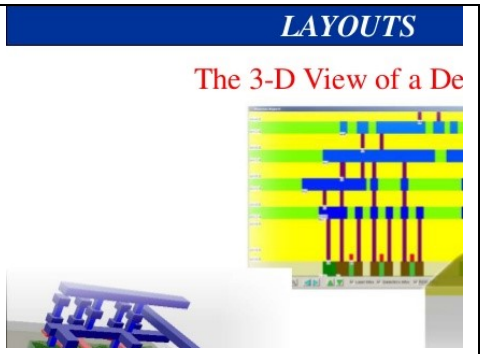


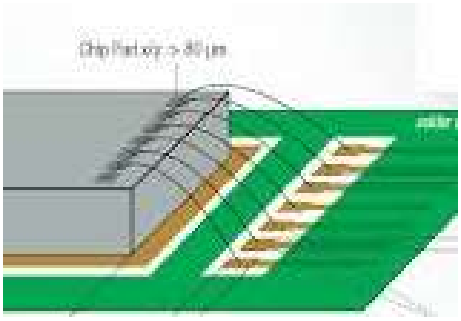


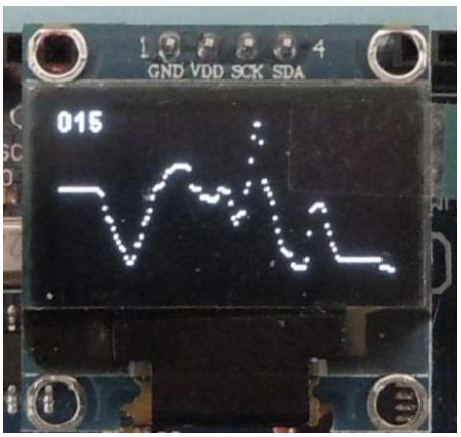
On devine que pour faire un petit écran de 64x128, comme celui que l'on va utiliser, il faut 64+128 fils qui partent de l'électronique. La puce silicium de commande, a une dimension de 8.2x1.2mm (la dimension de la lettre I dans le titre) avec tous les points de contact autour, comme le montre le dessin du fabricant.

On sait la quantité invraisemblable de transistors que l'ont peut faire apparaître sur un substrat de silicium, et interconnectés par des ponts de conducteurs faits par usinage chimique.

Entre la puce intégrée et le circuit souple, il faut souder par pression des fils de 20 microns de diamètre sur un circuit imprimé souple. Un ajoute les condensateurs et résistances qui ne peuvent pas être intégrés sur le circuit, et les conducteurs qui correspondent au type de transfert souhaité sont envoyés sur le connecteur du module .
Le résultat est un écran de 30x32mm avec 4 pins, deux pour alimenter en courant, 3-5V, ~5mA, et deux pour transférer l'information par I2C.







Expérimenter

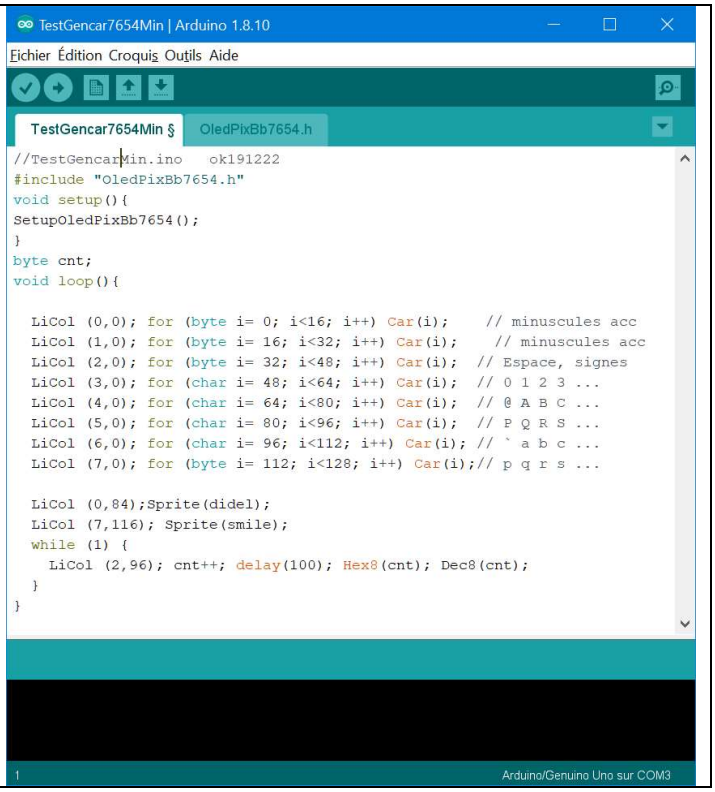
Pour jouer avec l'écran Oled, il faut une carte avec un microprocesseur que l'on va programmer. Programmer, c'est écrire des instructions dans un éditeur, dans un langage pour lequel on a un compilateur qui va traduire en "code machine" (du binaire). Un programme "loader" va transférer ces bits dans le processeur. L'environnement Arduino est facile à utiliser et vous pouvez vous perdre dans tous les ‘’Instructables‘’ pleins de fils et de vidéos naives. Si vous voulez nous suivre, il vous faut n'importe quelle carte compatible Arduino UNO (Diduino is the best of course) et un OLED SSD1306 avec ses 4 pins.

Vous avez repéré les pins 7654 sur la carte. Insérez le Oled; deux pins seront programmées avec du 0 et du 5V (la consommation est de quelques mA) et deux pins avec les signaux voulus. L’installation d’Arduino est expliquée par exemple sous https://www.didel.com/educ/EduC-Install.pdf avec un premier exemple de 3 lignes à copier et exécuter.

Pour progresser rapidement, on va charger des exemples et les modifier. Ces exemples sont dans un zip, qu'il faut "dézipper" et mettre dans un dossier de travail.
Chargez www.didel.com/DecouverteOled.zip dans un dossier (de nom *Oled* par exemple) que vous saurez trouver facilement quand vous serez dans l’éditeur Arduino, et dézipper. Si c'est nouveau, demandez quelques explications à un habitué des PC !
Vous devez voir 3 dossiers qui contiennent chacun, si le ‘’unzip’’ est bon, le programme principal avec l'extension .ino et un fichier inséré avec l'extension .h.

Connectez la carte et chargez Arduino. Ouvrez le dossier TestGencar et cliquez sur TestGencar.ino. Si Arduino est déjà chargé, le menu "fichier-ouvrir" permet de partir à la recherche du dossier et de son exécutable *.ino.

Voila ce qui vient sur l’écran (le nom a changé). TestGencar.ino apparaît dans la fenêtre et on voit dans un onglet le fichier inclus. C’est un fichier de "librairie" qui regroupe des fonctions qu'un débutant ne peut pas écrire.
On reconnait la structure Arduino, avec le setup() et le loop(). Le #include évite de surcharger le début du programme avec quantité d’information que l’on ne comprend pas. Pas de librairie Adafruit à aller chercher (et elle demande d'autres pins et des fils).
Notez aussi qu’il n’y a pas de ‘’digitalWrite’’ mais des ‘’bitSet’’ plus efficace, que vous ne verrez de toute façon pas dans le programme. Les définitions et les fonctions complexes sont cachées dans les fichiers inclus. Le programme principal décrit ce qu'il fait, avec des noms clairs, et non pas en listant quels numéros de pins doivent être secoués!



Le programme utilise la fonction Car(i) pour afficher les codes 0 à 127. Il faut plusieurs boucles ‘’for’’. On voit que les minuscules et chiffres sont en format 4x7 et les majuscules en 5x7. Les lignes ont 8 de haut, ce qui fixe l’interligne.
En général, on n’utilise pas la valeur numérique (code) des caractères ASCII, mais le signe typographique qui est converti en code par le compilateur. Par exemple Car(‘a’); affiche a et Text (‘’texte’); affiche le texte texte.
Attention, les codes 0 à 31 ne sont pas des caractères ASCII, donc ne sont pas connus du compilateur ; On doit écrire `Car(20)` ; pour afficher le caractère é.

Pour positionner l'endroit ou on veut écrire, LiCol(ligne,colonne); demande la ligne, de 0 à 7, et la colonne, de 0 - 127 du début de l'affichage. Il y a en plus des ordres pour afficher des nombres. Si les variables ont été déclarées byte, Bin8(v8), Hex8(v8), Dec8(v8); affichent en binaire, en hexadécimal et en décimal. Si elles ont été déclarées int, Hex16(v16), Dec9999(v16); affichent en hexadécimal et en décimal, avec la limitation du Dec9999 (qui affiche ???? si >9999).

Pourquoi cette anomalie du Dec9999()? Un nombre de 16 bits va de 0 à 0xFFFF en hexa, 0 à 65535 en décimal. Pour simplifier, pour ne pas prendre trop de place sur l'écran, on se limite à 4 digits, maximum 9999.

On voit que le programme, après avoir écrit tous les codes, positionne en 3e ligne à droite un compteur affiché en hexa 8 bits Hex8() et décimal Dec8().

Et naturellement, il faut de la pub! C'est en fait assez amusant de prendre du papier quadrillé, dessiner un Sprite et le coder. Les données (smile, sad, didel), sont prédéfinies dans les fichiers inclus *pix*.h. Chargez www.didel.com/LibrairiesOled.pdf pour apprendre à faire vos propres MySprites() et tout savoir sur les contraintes de l'écran et sur les fonctions à disposition.

Programme graphique

On veut dessiner un segment oblique sur l'écran. La fonction Dot(x,y); dessine le point ; il faut savoir que

- L'origine des coordonnées est en haut à gauche, ce qui n'est pas dans les habitudes scolaires. Une soustraction par rapport au bas de l'écran rétablit nos habitudes
- x vaut 0 à 127, y vaut 0 à 63

On doit déclarer x et y comme des variables 8 bits.
On doit initialiser la position du début du segment.
On doit décider quelle évolution donner aux coordonnées.
Décidons de faire une oblique de 20 pixels qui monte à 45 degrés depuis le centre.
La boucle qui génère l'oblique est

```
void loop () {
  x=10; y=32;
  for (byte i=0 ; i<20 ; i++) {Dot (x,63-y); x++; y++; }
}
```

Le programme complet est

```
//TestOblique.ino Segment oblique
#include "OledPixBb7654.h"
void setup () {
  SetupOledPixBb7654();
}
byte x,y;   // il faut déclarer
 les variables avant la boucle
void loop() {
 x=64; y=32;
 for (byte i=0 ; i<20 ; i++) {
   Dot (x,63-y); x++; y++;
 }
}
```

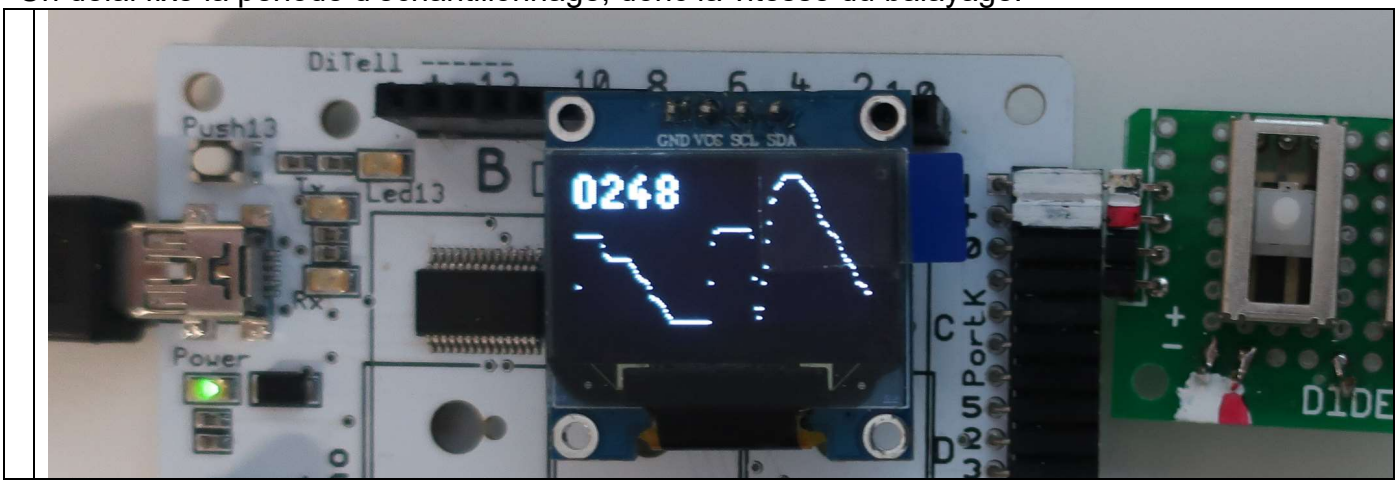
Pour bien de familiariser, il faut modifier l'origine, l'angle, l'amplitude et vérifier que le résultat est celui qui a été prévu. Que se passe-t-il si on dépasse les bords?
Si on dessine un rectangle, on voit qu'il y a un probléme avec les verticales. L'organisation des bytes sur le Oled est de 128 bytes verticaux par ligne, très efficace pour afficher les caractères. Mais si on dessine un seul pixel, on écrase les 7 autres pixels du même byte. Si on pouvait lire le mot de 8 bits, on lirait et superposerait avant de ré-écrire. Mais voilà, le SSD1306 est ''write-only''. Adafruit et d'autres écrivent tout dans une mémoire ''bit-map'' de 1 kilo-byte, et ont toute la liberté de programmation nécessaire ; mais ensuite il faut transférer tout le bit-map et cela prend 80ms. Tracer un graphique comme dans le programme suivant serait trop lent ; pour un graphique, on écrit dans la prochaine colonne et c'est bon.

Afficher un diagramme des temps

Notre 3e programme lit la tension analogique de la pin 14/A0. Chargez le programme TestOledA0. Il affiche la valeur 10 bits (donc < 1024, < 9999) et fait en plus un graphique.

Dot (x,y); allume un pixel en colonne x (0 à 127) et ligne y (0 à 63) mais en partant du haut comme pour les lignes de texte, numérotées de 0 à 7. Pour retrouver les habitudes scolaires

avec l'origine en bas à droite, on écrit Dot (x,63-y). En bout de ligne, on efface et recommence. Un délai fixe la période d'échantillonnage, donc la vitesse du balayage.

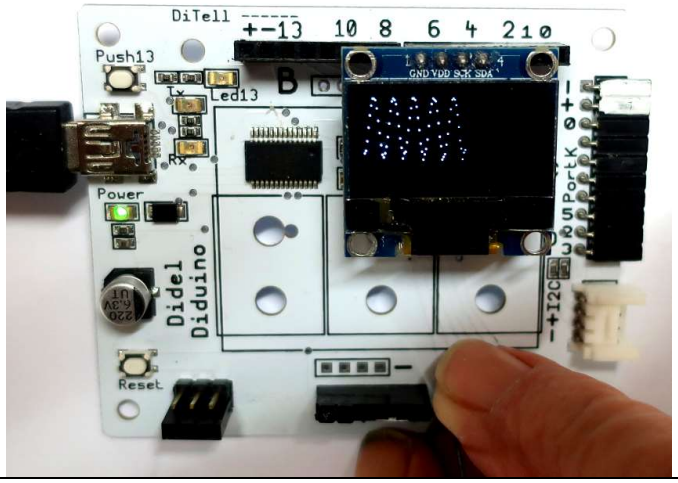


Si vous avez un potentiomètre, il faut connecter le point milieu sur A0 et les extrémités au + et -. Avec un LDR ou une photodiode, il faut une résistance pour faire diviseur de tension.

Vous avez vraiment rien ? Alors on va faire plus intéressant avec une épingle, un fil métallique quelconque inséré dans la broche A0. Cette broche est à haute impédance. En la touchant, votre corps capte le 220 Volts votre tension est une sinusoide à 50 Hz. Pour la visualiser, il faut supprimer tous les retards.

```
void loop(){
  pot = analogRead(A0);
  //  LiCol(1,0); BigDec9999(pot);
  Dot (x,63-pot/16);
  if (x++==128) {x=0; Clear();}
  //  delay(50);
}
```

Analysez l'image. Quelle est la durée de la boucle ? Où intervenir pour la réduire ?
Vous voyez maintenant la différence entre programmer en Arduino et programmer C temps réel ? C'est comme être passager dans un autocar avec un mauvais chauffeur ou apprendre à faire du VTT.



Pour accélérer la boucle et voir une plus jolie courbe, on peut chercher à intervenir sur le logiciel ; il est déjà bien optimisé. Mais I2C peut être accéléré. Tout est sous notre contrôle en ''bit-bang''. Dans le ''driver I2C'', fichier inclus OledPixBb7654.h, on reconnaît la boucle d'attente qui fixe la fréquence I2C :

```
  for (byte j=0; j<10; j++)  //  12us period
```

soit 80 kHz. Le Oled SSD1306 est garanti à 200 Khz et on peut peut-être aller plus loin. Modifions la limite j<10. On obtient une plus jolie courbe, parfois assez stable (pourquoi ?).

Essayons encore quelque chose. Remettons la valeur j<10 initiale (il faut toujours revenir en arrière après un essai, après avoir éventuellement sauvé la configuration essayée sous un autre nom). Pas d'affichage de la valeur, mais un delay(19) ; Je vois une sinusoide pour les valeurs entre 17 et 20 ms. Mon explication : la lecture de la tension est instantanée ; en échantillonnant la sinusoide du 50Hz avec une durée proche d'un multiple de la période, on a un bête effet stroboscopique.

Si cela vous plaît, il existe quantités de capteurs et un Oled vous permettra de mieux faire connaissance. Une liste de nos documents se trouve sous www.didel.com/prof/Oled.html.

Jdn 200103