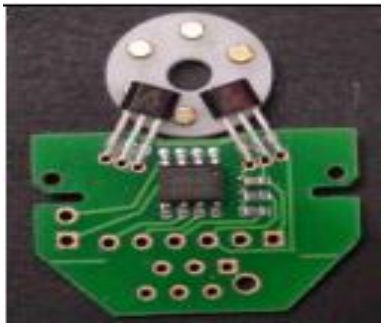
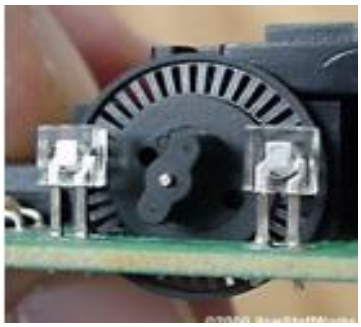
Encodeurs

1 Introduction

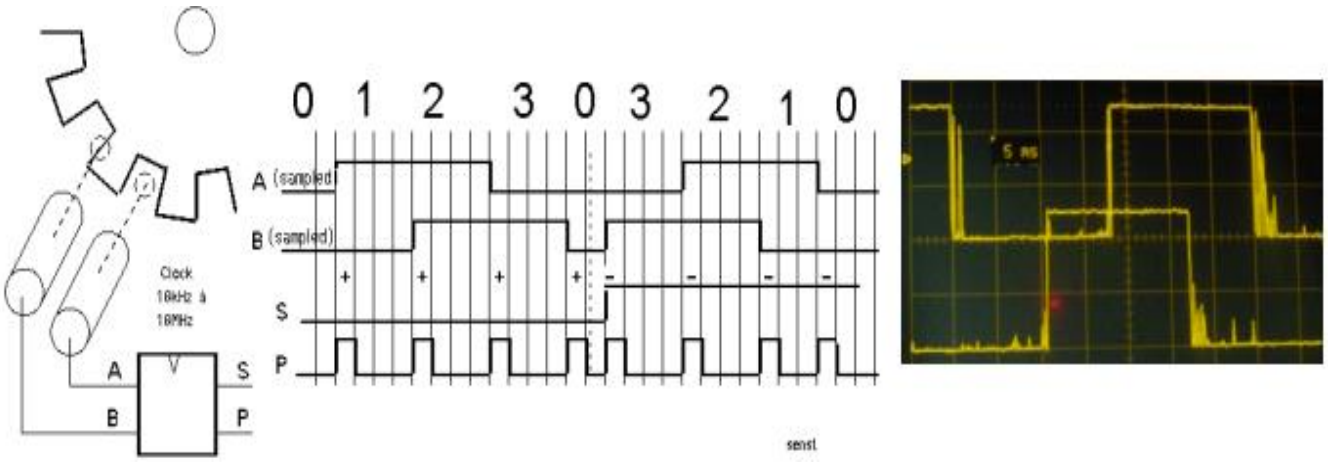
Les encodeurs ressemblent à des potentiomètres, mais ils ont au moins 4 fils, 2 pour l'alimentation et 2 pour les signaux en quadrature de phase. On les trouve aussi sur des axes de moteurs ou roues de robot pour connaître le déplacement. Ils étaient le cœur des anciennes souris.

Ce texte montre à des débutants en programmation C/Arduino comment mieux programmer. Il devrait aussi intéresser les "experts" en Arduino, qui manquent parfois de bonne compréhension du temps réel et d'un souci d'optimisation du code. Les exemples de programmes que l'on trouve cherchent à utiliser l'interruption directe, inadaptée s'il y a des rebonds. D'autres ne décodent pas toutes les transitions.

2 Principe du décodage des signaux

Les signaux sont générés par deux capteurs optique, magnétique ou mécanique. Les capteurs mécaniques ont des rebonds de contact, éliminés par échantillonnage. Filter avec un circuit électronique est absurde si on peut le faire par logiciel. Il faut bien comprendre qu'il y a une période d'échantillonnage, qui permet au programme de comparer et suivre l'évolution des signaux, et une durée maximale des rebonds de contact. La période d'échantillonnage doit être supérieure à la durée des rebonds.



Le contrôleur échantillonne les signaux A et B et prend la décision d'incrémenter ou décrémenter un compteur selon l'état précédent . L'algorithme le plus simple s'appuie sur le graphe des états.

La structure C "switch-case" permet de décider pour chaque état ce qu'il faut faire s'il y a changement d'état. Dans l'état 0 par exemple, il y a 3 possibilités car on ne peut pas avoir P1 et P2 qui passent ensemble à 1 :

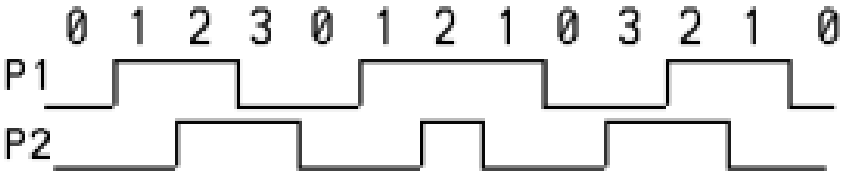
P1 = 1 on passe à l'état 1 et on compte

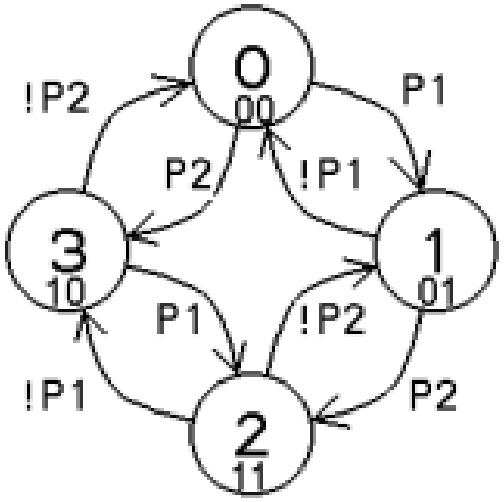
P2 = 1 on passe dans l'état 3 et on décompte

P1 = 0 et P2 = 0 on reste dans l'état 0

P1=1 et P2=1 erreur de timing

Si les signaux ont des rebonds de contact, comme pour un encodeur mécanique, l'échantillonnage doit se faire tous les 5 à 10ms, ce qui limite la vitesse de rotation, de toute façon faible avec un encodeur remplaçant un potentiométre.





Pour tester l'algorithme, il faut choisir comment afficher le résultat. Arduino offre le terminal série, qui appelle la librairie Serial. Le problème est que l'on doit échantillonner toutes les 10 ms avec un encodeur manuel, pour ne pas rater des pas. Si on affiche à chaque cycle; cela remplit l'écran et montre mal l'évolution. La solution est d'effectuer 100 cycles encodeurs, avec leur échantillonnage à 10ms, avant d'afficher le résultat sur le terminal série, donc une fois par secondes. Mais attention, quelle est la durée d'affichage? Si elle est trop longue, on va rater des pas.

A 9600bit/s, il faut 1ms pour afficher une lettre. Il faut à peu près le même temps pour convertir la valeur binaire de la variable 16 bits en décimal, donc l'affichage des 6 chiffres (au plus) prendra 7 millisecondes. Plus si on ajoute du texte. Toutes les secondes, l'échantillonnage se fera avec une période de 17ms, c'est ce qui va fixer la vitesse de rotation maximale.

3 Premier programme de test

```
// TestEncoArduino.ino
// (le programme se trouve sous www.didel.com/Encodeur.zip)
// L'encodeur est câblé sur les pins Arduino 14 et 15
#define P1 14 //const Int P1=14; si vous préférez
#define P2 15
int cnt =0 ; byte etat = 0;
void setup() {
 Serial.begin(9600);
 pinMode (P1, INPUT);
 pinMode (P2, INPUT);
}
void loop () {
 for (int i=0;i<100;i++) { // display every 100 scan
 delay (10);// sample delay (debounce)
 switch (etat) {
 case 0 : // P1=0 P2=0
 if (digitalRead (P1)) { etat = 1 ; cnt++ ; }
 if (digitalRead (P2)) { etat = 3 ; cnt-- ; }
 break;
 case 1 : // P1=1 P2=0
```

```
if (!digitalRead (P1)) { etat = 0 ; cnt-- ; }

if (digitalRead (P2)) { etat = 2 ; cnt++ ; }

break;

case 2 : // P1=1 P2=1

if (!digitalRead (P1)) { etat = 3 ; cnt++ ; }

if (!digitalRead (P2)) { etat = 1 ; cnt-- ; }

break;

case 3 : // P1=0 P2=1

if (digitalRead (P1)) { etat = 2 ; cnt-- ; }

if (!digitalRead (P2)) { etat = 0 ; cnt++ ; }

break; } // end switch

} // end for

Serial.println(cnt);

}
```

4 Avertissement

Si ce programme ne fonctionne pas, regardez d'abord les signaux avec un bon oscilloscope avant de dire qu'il n'est pas utilisable. Utilisez 2 boutons-poussoirs, appuyez sur les boutons suivant la séquence et observez le résultat sur le terminal. Avec un codeur qui tourne sans à-coups cela doit aussi fonctionner. Mais nous avons remarqué que cela ne fonctionnait pas avec un codeur Grove ayant des clicks et 4

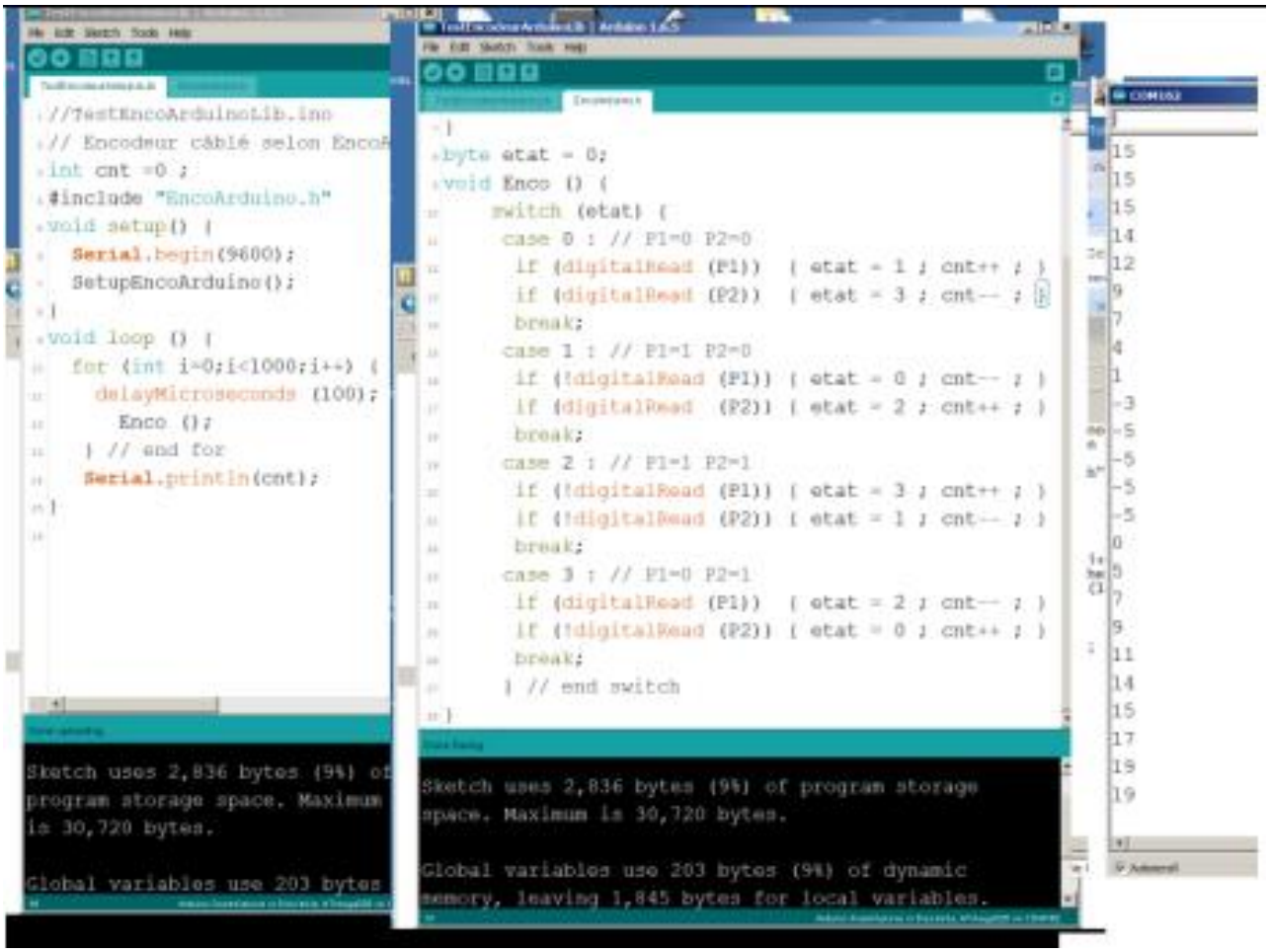
transitions à chaque clic. Les transitions peuvent être aussi proches que 1 ms, et sont donc filtrées.

Ce programme ne fonctionne pas non plus avec un codeur moteur à haute résolution comme le RomEnco (Didel-Solarbotics), car la supression des rebonds et l'utilisation du terminal Arduino fait perdre des pas. Facile de réduire l'échantillonnage à 100 microsecondes et le logiciel peut suivre jusqu'à 10 000 transitions par seconde (RomEnco en a 4000 a vitesse max). En augmentant les itérations dans la boucle "for" on affiche aussi le compte à chaque seconde, mais chaque fois que le terminal série affiche un nombre, des transitions sont perdues. Vous ne le remarquerez pas, s'est un petit pourcentage, mais comprenez ce que vous faites.

5 Fonction EncoArduino.h

Maintenant qu'on a un programme testé, il faut "encapsuler", c'est-à-dire créer une fonction Enco(); (la première version s'apellera EncoArduino(); Il suffit alors de se souvenir plus que de 2 choses: la fonction EncoArduino(); met à jour la variable int cnt;. Cette fonctions utilise les signaux P1 P2 qu'il faut déclarer et initialiser dans le SetupEnco();. Le fichier EncoArduino.h est inclus comme les librairies Arduino, mais fait partie du croquis. Il est juste dans une fenêtre à part. Cette pratique n'est pas courante, mais elle est très efficace (voir en annexe).

On voit que le programme principal montre uniquement ce que l'on veut faire: lire l'encodeur et afficher sur le terminal tous les 100 cycles. La variable cnt est globale, elle est utilisée par le fichier Enco.h et par le programme, donc déclarée tout au début. La variable etat est locale à Enco.h, donc définie dans Enco.h et on peut l'oublier. Arduino cache ces concepts, portant essentiels pour faire une application sérieuse.



6 Affichage sur Tell et fonction Enco.h

Utilisons le module Tell embarqué à la place de l'écran. Cela permettra d'être autonome.

Tell affiche 4 digits avec une fonction qui prend 3ms.

Le câblage est illustré sur la photo, et avec notre approche modulaire, il faut inclure la librairie Tell.h, appeler son setup, et savoir que Tell(xx); affiche la variable int xx;

//TestEncoTell.ino

int cnt =0 ;

#include "Enco.h"

```
#include "Tell.h"

void setup() {

 SetupEnco();

 SetupTell();

}

void loop () {

 delay (7); // suppr rebonds

 Enco ();

 Tell (cnt);

}
```

On voit que la boucle for a été supprimée, car l'affichage ne défile pas; il dure 3ms. Avec la suppression des rebonds, on est toujours limité à 100 transistions par secondes. Remarquez que la fonction Enco a été modifiée pour ne plus utiliser les fonctions Arduino, donc être portable et rapide. Si vous n'êtes pas à l'aise avec les opérations logiques, faites confiance, comme avec beaucoup d'autres fonctions que vous n'avez pas cherché à comprendre.

La fonction EncoArduino(); dure 10us avec les fonctions Arduino, Enco(); dure1.5 us avec les instructions C directes.

7 Affichage sur un Oled

Ajoutons un affichage Oled SSD1306, sur les pins 8 à 11, gardons les pins 4 à 7 pour un 2e moteur éventuel. Un 2e encodeur peut être ajouté sur les pins 16 et 17. Il faudra appeler une 2e fonction, Enco2.h avec les pins16, 17 déclarées et un compteur cnt2. Oled offre une liberté de représentation que l'on n'a pas avec Tell. Mais le temps d'écriture devient important, 25ms avec le programme ci-dessous, donc on est limité à 40 transitions par secondes, ce qui est compatible avec un potentiomètre-encodeur.

```
int cnt =0 ;

#include "Enco.h"

#include "Tell.h"

#include "OledI2Cbb.h"

#include "OledPix.h"

void setup() {

 SetupEnco();
```

```
 SetupTell();

 SetupI2Cbb();

 SetupOledPix();

}

byte x;

void loop () {

 Enco ();

 Tell (cnt); // 3ms

 LiCol(1,20);BigHex16(cnt);

 DDot (x,(cnt&0x63));

 if (x++==128) {x=0; Clear();}

}
```

8 Interruptions

L'approche usuelle pour suivre un encodeur par interruption est de demander au processeur de déclencher une interruption à chaque transition des 2 entrées. Les microcontrôleurs peuvent usuellement faire cela. Les rebonds créent aussi des interruptions et cela devient compliqué. Si on veut un 2e encodeur, ce n'et plus possible.

Notre approche ici est d'utiliser une interruption synchrone, qui n'a pas de limitation dans le nombre d'encodeurs, et qui permet d'ajouter facilement d'autres sources d'interruptions.

Le Timer2 est initialisé pour créer une interruption régulière. La routine d'interruption appelle la fonction Enco(); suffisamment souvent pour ne pas perdre des pas, mais pas trop souvent pour filtrer les rebonds.

Le programme échantillonne à 100 Hz . L'interruption dure 2.5 microsecondes toutes les 10 ms et met à jour la variable cnt. Si dans le programme principal on appelle la fonction d'affichage Tell() et les fonctions Oled. Est-ce qu'elles peuvent être interrompues pour 2.5 microsecondes? La réponse est oui, mais il faut l'avoir lue dans les spécifications. La boucle d'attente dans le programme principal ne dépends plus de Enco(), mais seulement de notre perception visuelle de l'affichage.

```
//TestEncoInter.ino

void setup () {

 SetupEnco ();

 SetupTimer2();

 TCCR2A = 0; //setup timer2

 TCCR2B = 0b00000111; // clk/1024

 TIMSK2 = 0b00000001; // TOIE2
```

```
 sei(); // autorise les interruptions
}
volatile int cnt;
ISR (TIMER2_OVF_vect) {
 TCNT2 = -160; // pour 100 hz si clk/1024
 Enco();
}
void loop () {
 Serial.println(cnt);
 delay (500);
}
```

Facile de remplacer le terminal par Tell ou Oled est facile. Il n'y a plus à se préoccuper du timing (TestEncoInterTellOled.ino).

Pour le codeur d'un moteur rapide, modifiez l'initialisation du timer. Avec une interruption toutes les 60µs, le moteur Rome BO10 avec RomEnco (12 transitions par tour) pourrait tourner à 1300 tours / seconde (pas RPM) en utilisant 2,5 µs de temps de processeur tous les 60 µs.

9 Autres actions par interruption

L'utilisation d'un seul timer pour gérer plusieurs tâches est appelée programmation synchrone. Cela évite le problème des collision multiples d interruptions. Ajouter un deuxième, troisième, ... encodeur? Facile! Ajouter du PFM sur les moteurs, du PWM sur Leds? La programmation en C est très puissante. Les bibliothèques Arduino sont comme les meubles d'Ikea. c'est bien, mais pour des applications très précises. Voir LibX (github nicoud Libx) qui s'appelera DidLib lorsqu'il sera pleinement documenté.

II Fichiers inclus sous Arduino

Une bonne pratique de programmation, quand les programmes deviennent longs et que des parties de programmes sont utilisables dans différents programmes, est de mettre les définitions, fonctions, modules dans des fichiers séparés et les appeler avec l'ordre #include. Le mécanisme est bien connu pour les librairies Arduino, qui sont dans une zone mémoire réservée et connue du compilateur.

Prenons l'exemple de l'ensemble des définitions pour les exercices du cours Microcontrôleur. On crée un fichier LcDef.h qui les contient, et dans le programme, on note #include "LcDef.h " pour que le préprocesseur ajoute ce fichier avant d'envoyer le programme au compilateur. Pour le set-up, on préfère définir une fonction qui est appelée dans le setup. On ajoute dans le setup les initialisations éventuellement nécessaires, comme Serial.begin(9600); De cette façon, on a dans le fichier LcDef.h tout ce qui concerne l'aspect matériel de l'application.

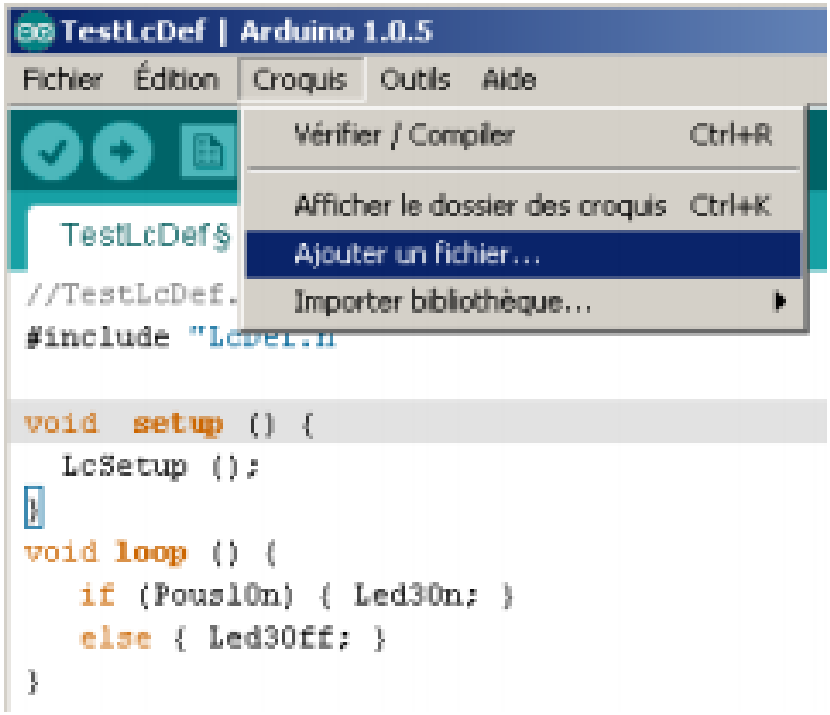
On peut parler de librairies propres (elles sont entre "" alors que les librairies communes sont entre <>). La contrainte pour les librairies propres est que les fichiers insérés doivent se trouver dans le même dossier (sketch, croquis) que le programme. Le sketch du programme contient le fichier .ino et les fichiers .h associés. Dans l'index "Sketch" on peut aller chercher des fichiers à ajouter dans d'autres dossiers, mais on ne peut pas les enlever, il faut intervenir avec le gestionnaire de fichiers.

Exemple; vous voulez clignoter la Led3.

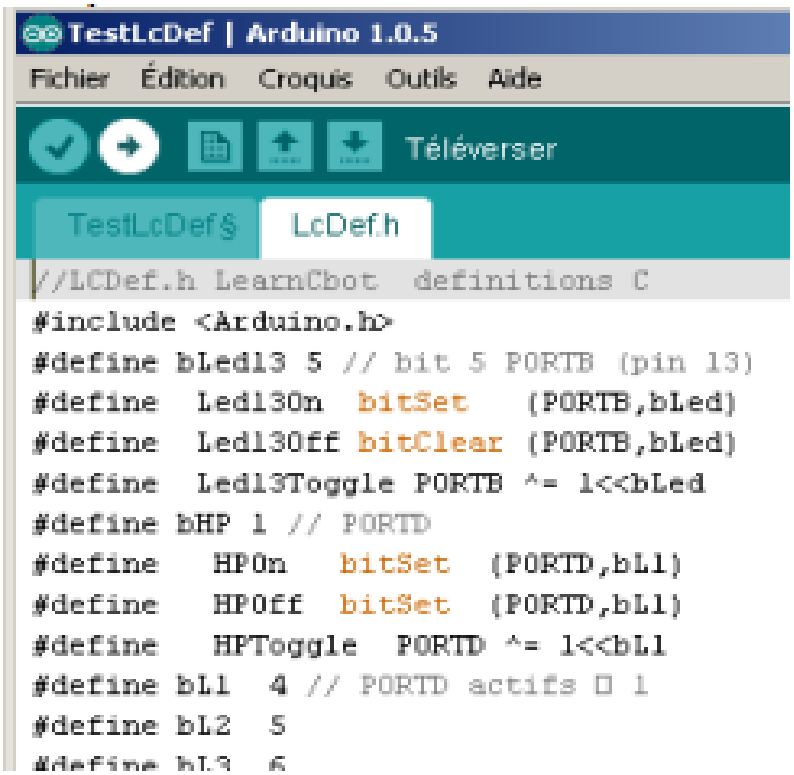
Il faut se référer avant la boucle de programme aux deux éléments prédéfinis:

#include "LcDef.h" et LcSetup ();

Ensuite, il faut ajouter le fichier LcDef.h, préparé quelque part sur le disque, dans le dossier du programme.



On peut ensuite observer les fichiers importés, et il n'est pas nécessaire de sélectionner la première fenêtre pour recompiler.



On peut naturellement faire des fichiers importés avec des fonctions. La contrainte est que les variables utilisées doivent être définies dans un module précédemment appelé.

La coupure en morceaux d'un programme existant n'est pas évidente. Il faut parfois passer par un éditeur comme NotePad++. Sous l'onglet "sketch" le menu "add file" permet d'ajouter un fichier, mais apparemment on ne peut pas ajouter un fichier vide que l'on nomme à ce moment. Dans Energia, l'option "New Tab" crée un fichier vide. Pour des modifications successives de programmes, c'est facile, les fichiers inclus sont automatiquement transportés. Ne pas oublier de sauver à chaque étape. A noter que si le fichier inclus comporte des noms de registres ou des fonctions Arduino, il faut ajouter la ligne #include .

Programmation sous C

Le changement est mineur si vous utilisez

un compilateur C à la place d'Arduino.

```
#include "LcDef.h"

int main () {

 LcSetup ();

 while (1) { // remplace void loop () {

 . . .

 }

}
```

III Expérience avec le Oled I2C SSD1306

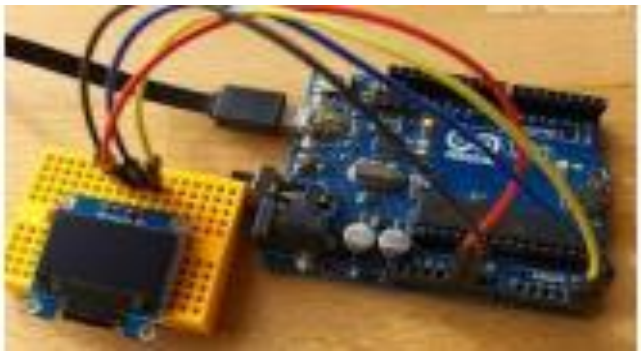
1 Introduction

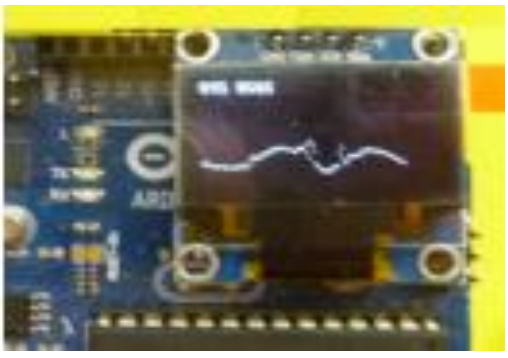
Ce document s'adresse à des intéressés à la programmation, qui connaissent Arduino et veulent approfondir leur connaissance. Même ceux qui ont suivi le MOOC EPFL (et peut-être surtout ceux-là) peuvent profiter de notre progression pas-à-pas et des explications associées. Arduino est excellent pour voir 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.



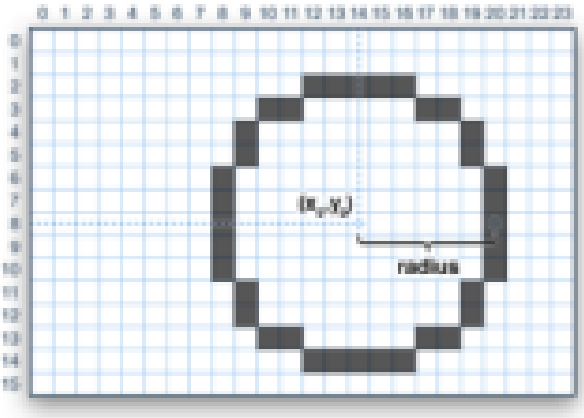


Pour utiliser la librairie AdaFruit; chercher "adafruit-gfx-graphics-library". Le graphisme est orienté texte et géométrie.

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.

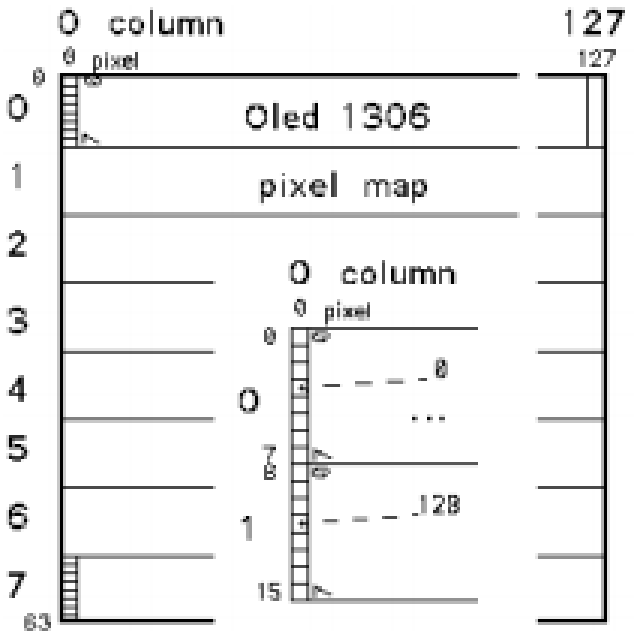
Pour observer les valeurs d'un capteur sur un robot qui roule, suivre le robot avec un portable qui affiche les valeurs sur le terminal série n'est pas une solution. Transmettre par Wifi ou BT n'est maîtrisé que par des spécialistes, et il faut alors plus performant qu'un Arduino ! Un Oled est simple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

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.

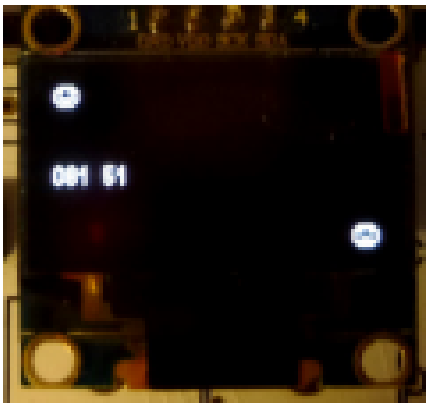
```
#define Ddr DDRD
#define Port PORTD
#define Pin PIND
#define bCk 5
#define bDa 4
#define bGnd 7
```

```
#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) ;

}
```

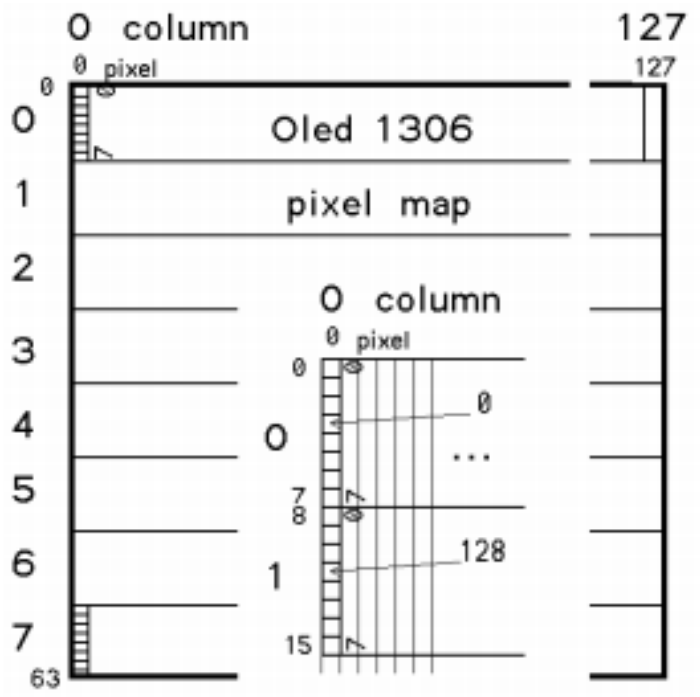
Si les opérations logiques ne vous sont pas familères, http://www.didel.com/C/OperationsLogiques.pdf donne les bases nécessaires. Arduino est très lourd et peu efficace quand on doit agir sur plusieurs bits d'un même mot. Et trop lent dans notre cas.

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=9999note: 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).

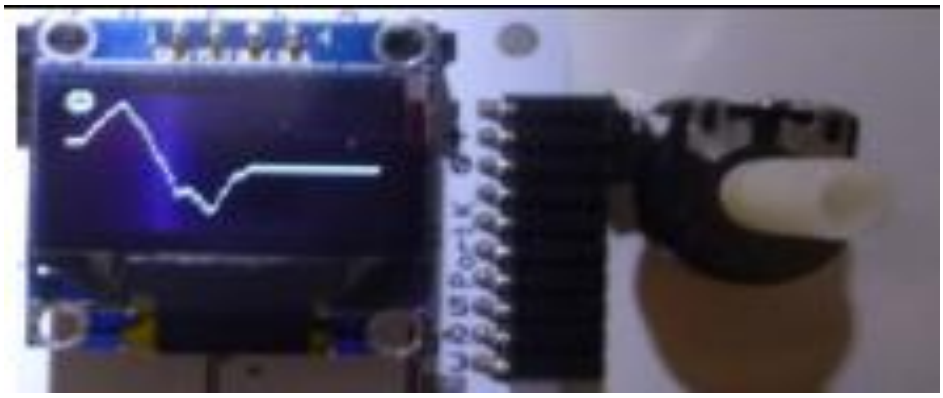
Ecrire un point efface les autres pixels du byte écran qu'il contient. Cela n'est pas gênant pour la seule application qui nous intéresse: visualiser des suites de mesures.

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];

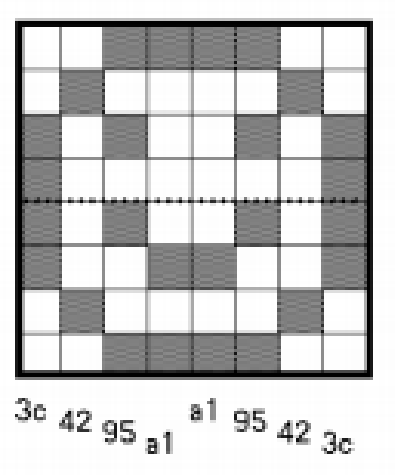
L'utilisation de la fonction Sprite, délicate pour un débutant, sera expliquée dans un autre document.

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);
}
```



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.

Références :

https://www.didel.com