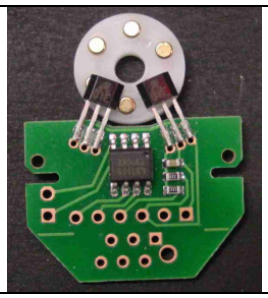
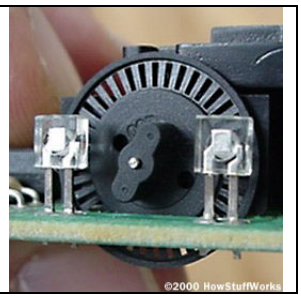
CH-1092 Belmont, Switzerland
Tel +41 21 728-6156, Fax 728-6157
Email info@didel.com, www.didel.com

www.didel.com/Encodeurs.pdf

Encodeurs

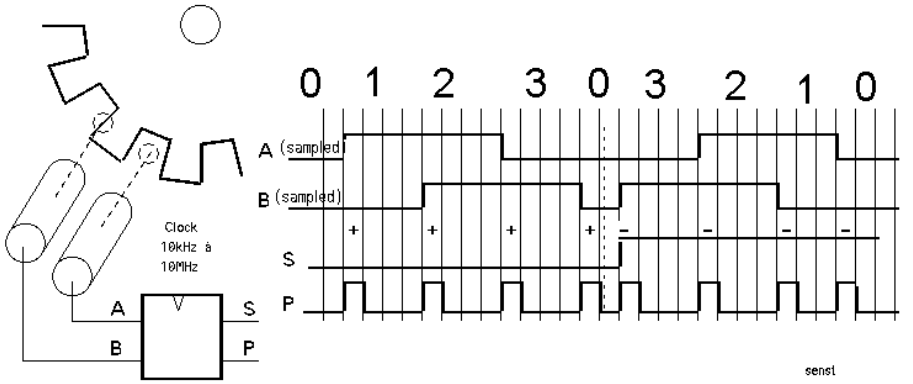
1 Introduction

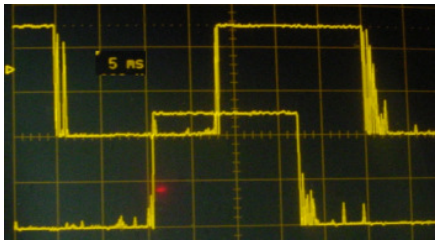
Les encodeurs ressemblent à des potentiomètres, mais ils ont au moins 4 fils, 2 pour l'alimentation et 2 pour les signaux en quadrature de phase. On les trouve aussi sur des axes de moteurs ou roues de robot pour connaître le déplacement. Ils étaient le cœur des anciennes souris.			

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.

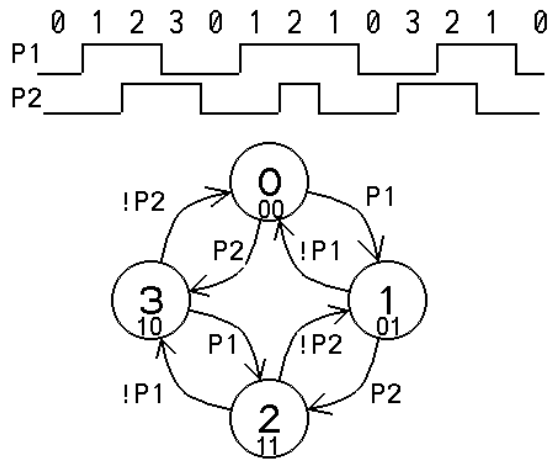
2 Principe du décodage des signaux

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);
}
```

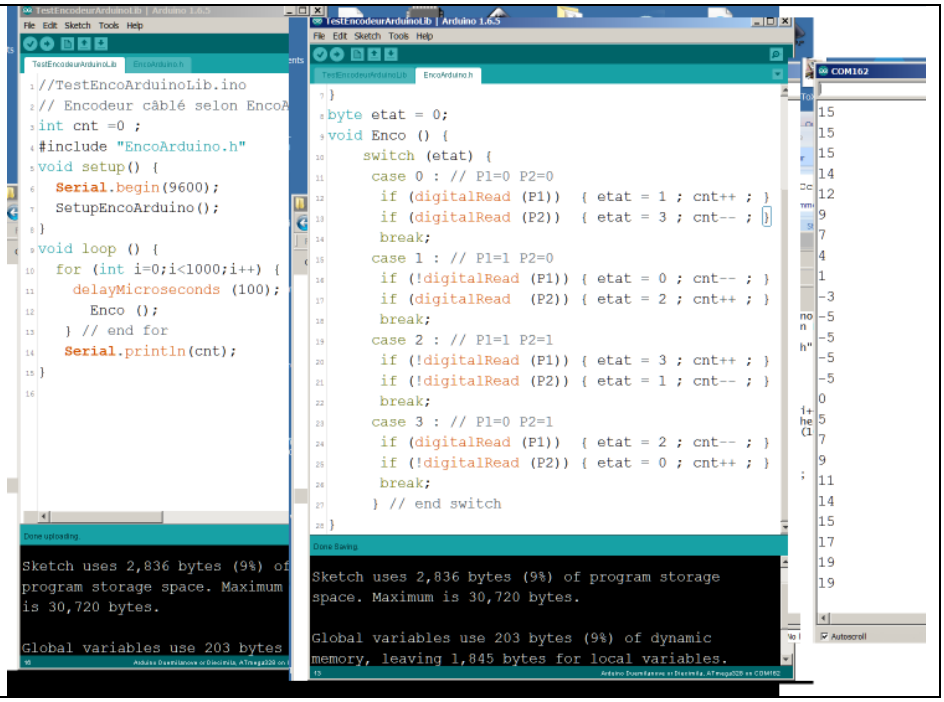
4 Avertissement

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).

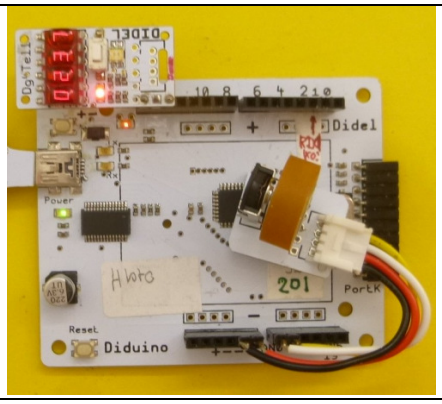
On voit que le programme principal montre uniquement ce que l'on veut faire:
lire l'encodeur et afficher sur le terminal tous les 100 cycles.
La variable `cnt` est globale, elle est utilisée par le fichier Enco.h et par le programme, donc déclarée tout au début.
La variable `etat` est locale à Enco.h, 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);
}
```



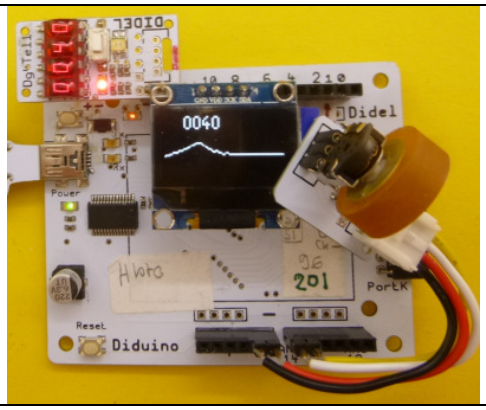
On voit que la boucle for a été supprimée, car l'affichage ne défile pas; il dure 3ms. Avec la suppression des rebonds, on est toujours limité à 100 transistions par secondes.
Remarquez que la fonction Enco a été modifiée pour ne plus utiliser les fonctions Arduino, donc être portable et rapide. Si vous 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 (voir www.didel.com/Oled1306.pdf).
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.

```
//TestEncoTellOled.ino
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 document www.didel.com/coursera/LC5.pdf explique ce timer 2 de l'AVR328.
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.

jdn 170621

Fichiers inclus: voir www.didel.com/coursera/FichiersInclus.pdf