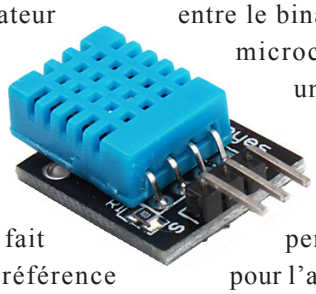
APPRENDRE LE LANGAGE C EN S'AMUSANT AVEC ARDUINO.

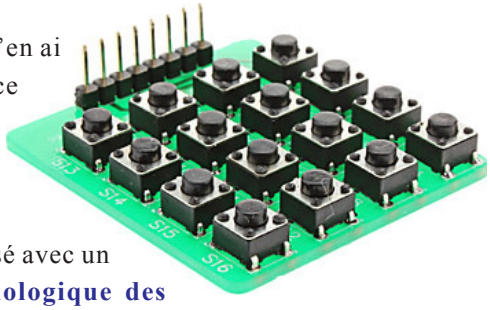
Pratiquant les microprocesseur ainsi que les microcontrôleurs depuis des années, c'est à travers le binaire pur et dur que j'ai eu le privilège de me passionner pour ces composants enthousiasmant. Cette activité remonte à des lustres, quand en France de tels circuits intégrés n'étaient pas encore disponibles et qu'il fallait les faire venir d'outre Atlantique. Les ouvrages de vulgarisation n'existaient pas, et c'est "octet par octet" que l'on finissait par comprendre les arcanes de ces étranges gros insectes. Le temps a passé, et les applications personnelles se sont étoffées jusqu'à gérer des feux de compétition sportives nationales ou un simulateur de pilotage maison pour piloter des avions ou des vaisseaux spatiaux.

L'évolution incite à changer régulièrement de processeur, passant de l'architecture Von Neumann à l'architecture Harvard. C'est une remise en cause permanente, avec des circuits intégrés de plus en plus puissants, mais aussi de plus en plus complexes. Avec le temps, le binaire finit par être indigeste, et ce d'autant plus qu'avec l'âge la mémoire devient volatile et que ne pas pratiquer régulièrement rend les reprises pénibles.



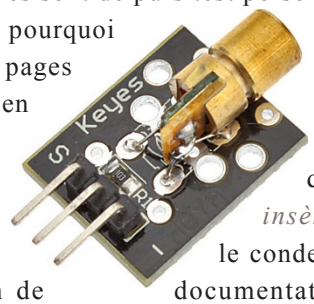
Alors, intercaler un compilateur entre le binaire et nos expérimentations décrites en langage évolué devient très séduisant. Si de plus un microcontrôleur particulier devient disponible "clef en main", la programmation redevient un vrai loisir, celui de programmer juste pour le plaisir.

Bien qu'ayant déjà pratiqué ce type d'approche, notamment avec comme langage BASIC, PASCAL, L.S.E, j'avais vraiment envie d'expérimenter le langage C très populaire et réputé "orienté système". Alors l'existence d'Arduino et de cette communauté internationale à carrément fait pencher la balance du bon coté. Je me suis commandé un Arduino sur Internet, un livre de référence pour l'accompagner et commencé mes premières expériences. L'existence d'une foule de petits modules, de Shields, de bibliothèques pour les mettre en œuvre ne pouvait que séduire … Mais … Le livre dédié à Arduino que j'avais acheté est très creux, vraiment "vide" et ne m'a pratiquement rien appris sur le langage C. Heureusement qu'il y a les forums !



S'il est enfantin de commander sur internet un quelconque module, et j'en ai approvisionné une tripotée pour faire joujou avec le langage C, force est de constater qu'ils ne sont pratiquement jamais accompagnés de leur schéma et des documentations associées. Il faut fureter sur la Toile, ce qui n'est pas mon fort. Parfois, trouver le bon lien relève de la chance.

Ce document n'est en rien un tutoriel et n'a absolument pas été réalisé avec un quelconque souci de pédagogie. **Ce n'est qu'un recueil chronologique des différentes expériences** qui ici ont aboutit et dont j'ai ultra résumé l'aspect documentaire à des fins purement personnelles. Certaines sont de purs test personnels, d'autres utilisent des modules du commerce. Du coup, quoi que bourré de faiblesses, pourquoi ne pas faire profiter la communauté de ces informations, même si elles sont étalées dans les pages sans critère de "consistance" visant un apprentissage progressif. La finalité de cette mise en ligne n'est pas de vous aider à apprendre le langage C, ni de faire de la publicité pour des produits commerciaux, raison pour laquelle je ne précise pas les références exactes de ces modules. *(Toutefois vous les reconnaitrez très facilement car chaque fois j'en insère une petite photographie dans ce recueil)* C'est juste de mettre à votre disposition le condensé de mes expériences. Vous aurez ainsi la certitude d'avoir dès le départ un minimum de documentation sur ces produits, et surtout, au moins un programme qui les accompagne pour pouvoir les tester.

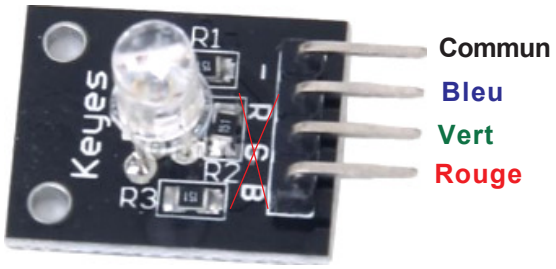


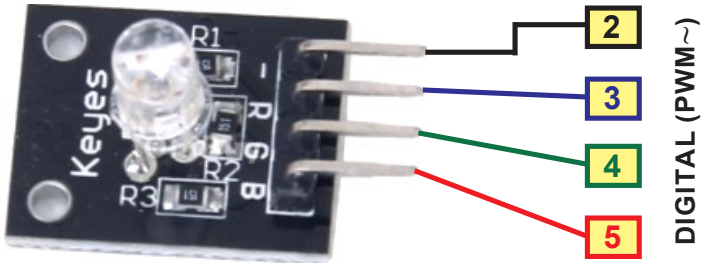
Ce petit livre n'a pas d'autre but.

Amicalement :
Nulentout.

Caractéristiques techniques du module "Keyes" :

• Attention la sérigraphie est fausse pour le brochage. (***R*** *correspond au BLEU et* ***B*** *au ROUGE)*
• La consommation en courant est fonction de la couleur de la LED. *(Voir tableau)*





	Rouge	Vert	Bleue
5 Vcc	14 mA	10 mA	11 mA
3,3 Vcc	7 mA	3 mA	3 mA

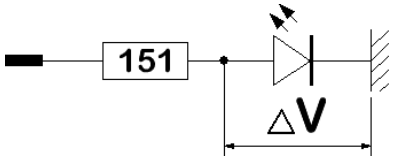
Exemple de programme :

```
/* Test du petit module avec la LED tricolore.
   Pour pouvoir brancher directement sur la platine ARDUINO,
   la sortie PWM n°2 sert de masse. */

int const Moins = 2;  // Sortie 2 pour le moins.
int const Bleu = 3;   // Sortie 3 pour piloter le BLEU.
int const Vert = 4;   // Sortie 4 pour piloter le VERT.
int const Rouge = 5;  // Sortie 5 pour piloter le ROUGE.
const int LED = 13;   // Broche de la LED U.C.

void setup()
 { analogWrite(Moins, 0);     // Tension nulle pour la masse.
  pinMode(LED, OUTPUT);    // LED est une sortie.
  digitalWrite(LED, LOW); }  // Éteint la LED de l'U.C.

void loop()
  { LUMIERE(0, 0, 1); // Bleu.
   LUMIERE(0, 1, 0); // Vert.
   LUMIERE(1, 0, 0); // Rouge.
   LUMIERE(1, 1, 1); // Blanc.
   LUMIERE(1, 1, 0); // Orange.
   LUMIERE(1, 0, 1); // Violet.
   LUMIERE(0, 1, 1); // Bleu/vert.
   LUMIERE(0, 0, 0);} // Éteint.

void LUMIERE(int R, int V, int B)
    {   for (int i = 0; i <= 255; i++)
           { analogWrite(Rouge, R * i);
             analogWrite(Vert, (V * i));
             analogWrite(Bleu, B * i);
             delay(5); }
       for (int i = 255; i > 0 ; i--)
          { analogWrite(Rouge, R * i);
            analogWrite(Vert, (V * i));
            analogWrite(Bleu, B * i);
            delay(5); }   }
```

NOTE 1 : Le petit programme donné en exemple fait augmenter puis baisser linéairement la tension sur les broches du module LED. Pour simplifier les branchements et permettre d'enficher directement le module sur la platine ARDUINO, le commun est branché sur la sortie analogique **2**.

NOTE 2 : Le courant dans les LED est fonction de la tension aux bornes ΔV et de leur seuil de conductivité. Pour la LED rouge et pour la LED bleue l'éclairement est progressif. Par contre, la LED verte présente un fonctionnement "binaire".

	Rouge	Vert	Bleue
ΔV	2,2 V	3,1 V	3 V



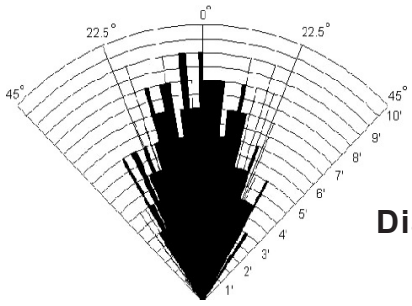
Caractéristiques techniques du module HC-SR04 :

- Alimentation : 5Vcc.
- Consommation en utilisation : 15 mA.
- Gamme de distance : 2 cm à 4 m.
- Résolution : 3 mm.
- Angle de mesure : < 15°.

Mise en œuvre :

Il faut envoyer une impulsion niveau haut *(+ 5v)* d'au moins 10 µs sur la broche Trig Input pour déclencher une mesure. *(Voir Fig.1)* En retour la sortie Output ou *(Echo)*, va restituer une impulsion + 5v dont la durée est proportionnelle à la distance si le module détecte un objet. Afin de pouvoir calculer la distance en cm, on utilisera la formule suivante : Distance = (durée de l'impulsion en µs) / 58.

Le câblage du module à l'Arduino sera le suivant :
- Broche 12 de l'Arduino vers Trig Input. *(Trig)*
- Broche 11 de l'Arduino vers Output. *(Echo)*



Exemple de programme :

```
/* Utilisation du capteur Ultrason HC-SR04 */
// Définition des broches utilisées.
int Trig = 12;
int Echo = 11;
long Lecture_Echo;
long Distance;
void setup()
{
pinMode(Trig, OUTPUT);
digitalWrite(Trig, LOW);
pinMode(Echo, INPUT);
Serial.begin(9600);
}
void loop()
{
digitalWrite(Trig, HIGH);
delayMicroseconds(10);
digitalWrite(Trig, LOW);
Lecture_Echo = pulseIn(Echo, HIGH);
Distance = Lecture_Echo / 58;
Serial.print("Distance en cm : ");
Serial.printcr(Distance);
// Temporisation entre deux TX sur la ligne série.
delay(500);
}
```

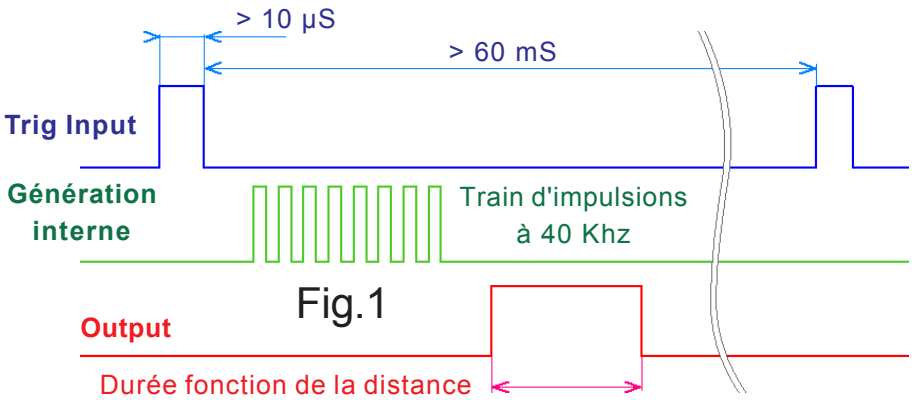


Fig.1

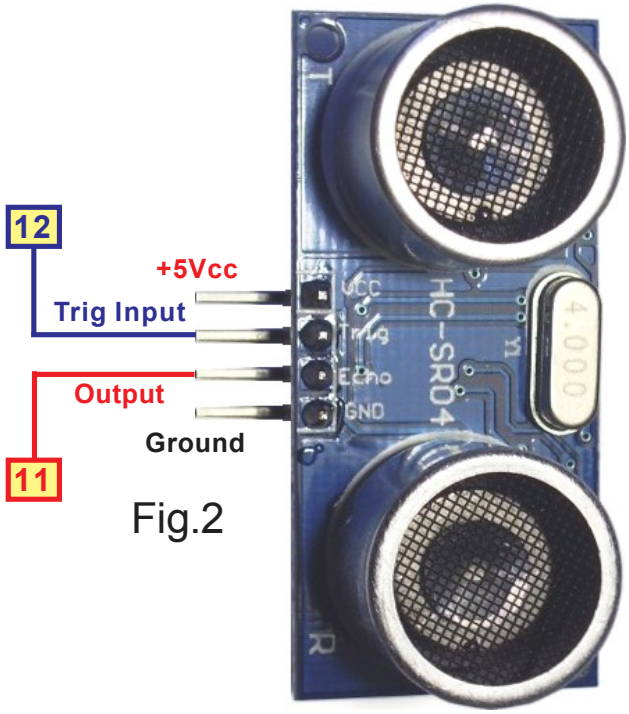


Fig.2

Utilisation :

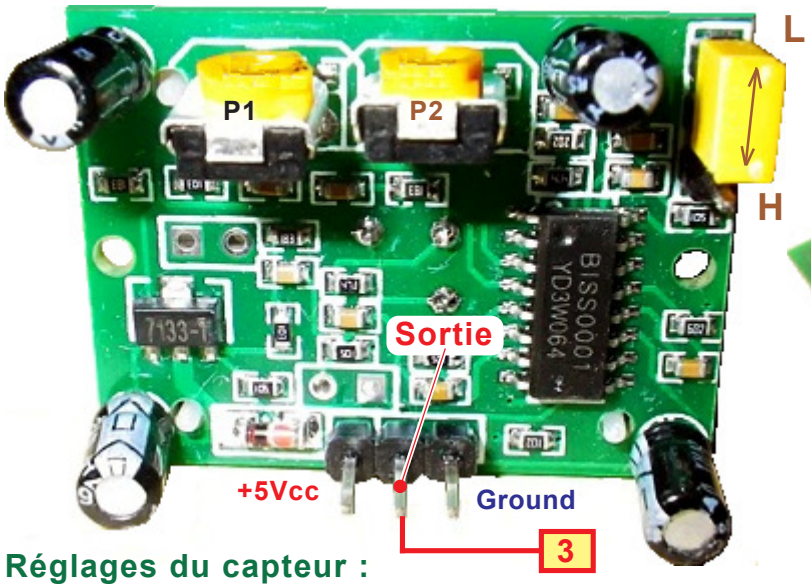
• Brancher le transducteur.
• Activer le programme.
• Outil > Moniteur série >
Le moniteur série affiche en boucle la distance mesurée.

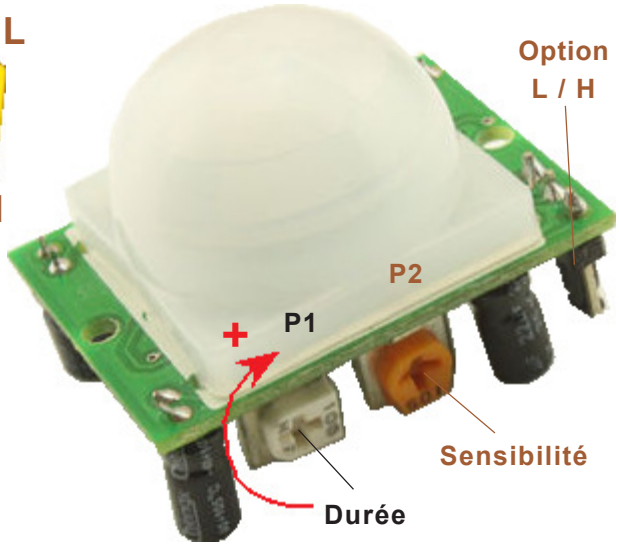
Autre exemple > radar directionnel :
http://www.voilelec.com/pages/arduino_radar.php

Caractéristiques techniques du module infrarouge HC SR 501 :

Le capteur PIR *(Passive Infra Red)* détecte une source d'infrarouge placée dans son champ de capture, détecte ses mouvements et déclenche une réponse. Le composant comporte deux capteurs "symétriques" car on cherches à détecter un mouvement et non des niveaux IR moyens. Les deux senseurs sont câblés de sorte que leurs effets s'annulent. Si l'un reçoit plus ou moins de rayonnement infrarouge que l'autre, la sortie prend alors un état Haut ou un état Bas.

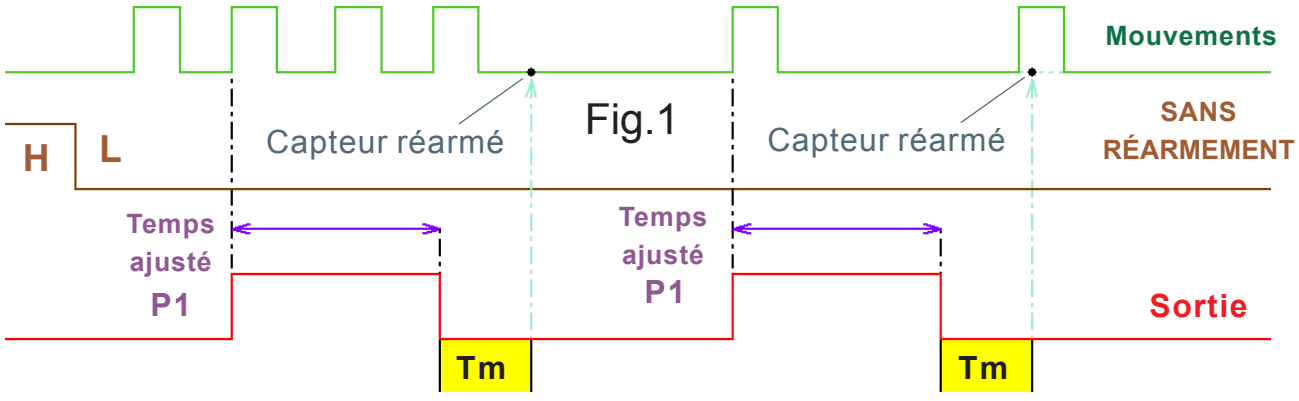
- Tension de fonctionnement : 4,5Vcc à 20Vcc.
- Consommation statique de 50 µA à 65 µA.
- Niveaux de sortie : Haut : 3.3 Vcc, Bas : 0 V.
- Temps de délai ajustable de 0.3 secondes à 18 secondes. *(5 secondes par défaut)*
- Temps mort 0.2 secondes.
- Déclenchement : **L** sans répétition, **H** répétition possible. *(Valeur H par défaut)*
- Portée de détection : Angle de moins de 120°, 3 m par défaut, 7 m MAXI.





Réglages du capteur :

- **P1** : Ajustement du délai pendant lequel la sortie reste verrouillée sur HIGH après une détection de mouvement. *(Mini ≈ 2526 mS à ≈ 192100 mS MAXI > Voir Exemple de programme.)* Le temps de latence avant le redéclenchement suivant est d'environ 2.5 secondes par défaut. *(Temps mort **Tm**)*
- **P2** : Ajustement de la sensibilité du capteur de 3 m à 7 m. *(Sens identique à celui de **P1** pour augmenter)*
- Option **L** *(Fig.1)* : C'est le mode SANS RÉARMEMENT, la sortie passe à l'état HIGH lors du premier contact, y reste durant toute la durée réglée par **P1**. Pendant ce temps le capteur ne réagit plus. La durée de l'état Haut en sortie est alors constante. *(Exemple : Déclencher une alarme)*
- Option **H** *(Fig.2)* : AVEC RÉARMEMENT, la sortie passe à l'état HIGH lors du premier contact, y reste durant toute la durée réglée par **P1**, mais chaque nouvelle détection remet le chronométrage de durée à zéro. C'est une détection en continu, le capteur reste stable à l'état haut tant que la source d'infra-rouges est mobile. *(Exemple : maintenir une ampoule éclairée le temps nécessaire)*



Notice et chéma sur : http://www.elecfreaks.com/wiki/index.php?title=PIR_Motion_Sensor_Module:DYP-ME003

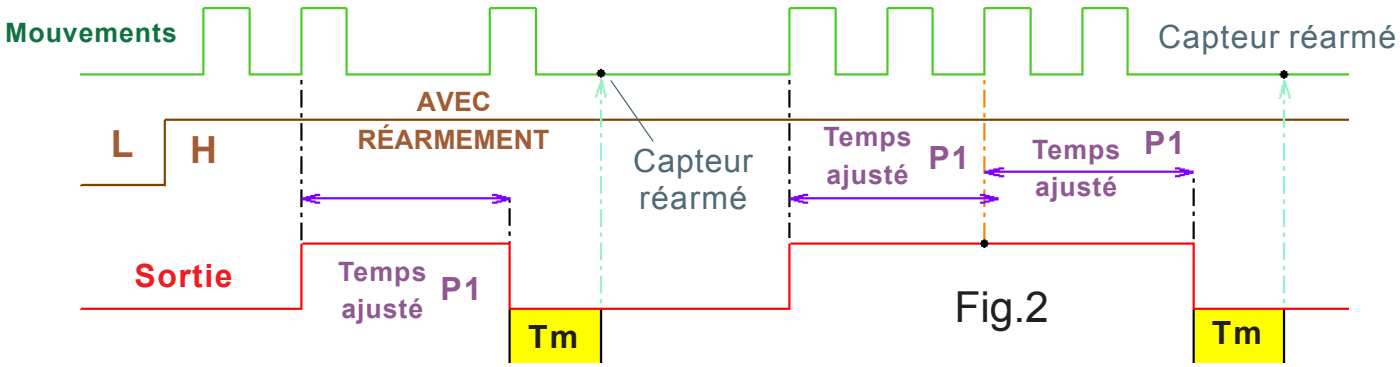


Fig.2

Exemple de programme avec mise en mémoire :

```
// Utilisation du détecteur de présence SKU:SEN0018.
// Détecteur type HC SR 501. La sortie du module est branchée sur le port 3.
/* Ce programme permet de mesurer la temporisation ajustée avec P1.
   Il permet en outre de tester l'option L "AVEC RÉARMEMENT". */

const int LED = 13;           // Broche pour la LED.
const int Capteur = 3;        // Broche pour lire la sortie du PIR.
int EtatCapteur = LOW;        // Au départ aucun mouvement détecté.
int Memoire = 0;              // Variable de mémorisation d'état du capteur.
long DebutMillis = 0;         //Heure de début du mouvement..

void setup() {
  pinMode(LED, OUTPUT);    // LED est une sortie.
  pinMode(Capteur, INPUT); // Capteur est une entrée.
  Serial.begin(9600); }

void loop()
{
Memoire = digitalRead(Capteur); // Lecture du capteur.
unsigned long CourantMillis = millis();  // Heure de début.
if  (Memoire == HIGH)
   { digitalWrite(LED, HIGH);       // Allume la LED
     if (EtatCapteur == LOW)        // Front montant détecté.
          { Serial.println("============================");
         Serial.print("Debut de mouvement >  ");
         Serial.println(CourantMillis);
         // On ne veut écrire que si l'état a changé, pas s'il est stable.
         EtatCapteur = HIGH;
         DebutMillis = CourantMillis;  // Mémoriser début du signal
        }
    }
else { digitalWrite(LED, LOW);       // Éteint la LED
       if (EtatCapteur == HIGH)      // Front descendant détecté.
          { Serial.println("Fin de mouvement > ");
            Serial.println(CourantMillis);
           unsigned long duree = CourantMillis - DebutMillis;  // Calcul de la durée.
            Serial.print("Duree >>> "); Serial.println(duree);
           // On ne veut écrire que si l'état a changé, pas s'il est stable.
            EtatCapteur = LOW;     // N'écrire que si changement d'état.
          }
      }
}
```

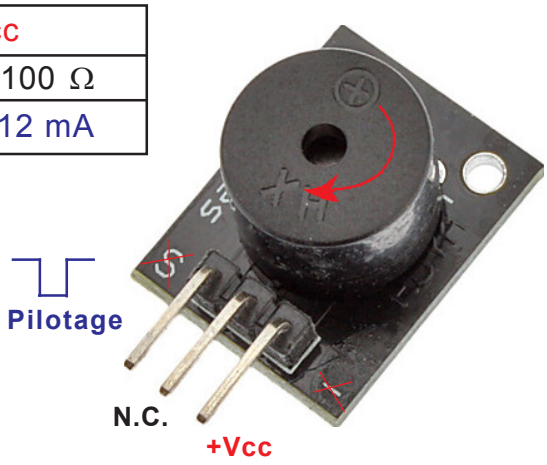
Ce programme allume et éteint la DEL disponible sur la carte ARDUINO en broche 13, mais il permet de mesurer la durée du délai ajustée avec **P1** et indiquée en page précédente. L'affichage se fait sur la ligne série activable avec **[CTRL] [MAJ] M**.

Caractéristiques techniques du module pour Arduino :

• Attention la sérigraphie est fausse pour le brochage. *(- correspond au + et S au pilotage)*
• La consommation en courant est fonction des conditions d'utilisation. *(Voir tableau)*

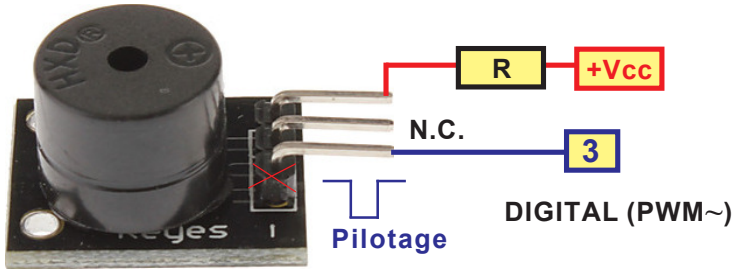
+Vcc	3,3 Vcc		5 Vcc	
R	0	47 Ω	0	100 Ω
I	13,5 mA	13 mA	20 mA	12 mA

Avec une tension de 5 Vcc la résistance de 100 Ω permet de diminuer le courant de pratiquement moitié, et avec lui le bruit fait par le BUZZER.
Sur 3,3V la résistance ne s'impose pas, car son effet est négligeable. Une valeur plus élevée aboutit rapidement à un mauvais fonctionnement.



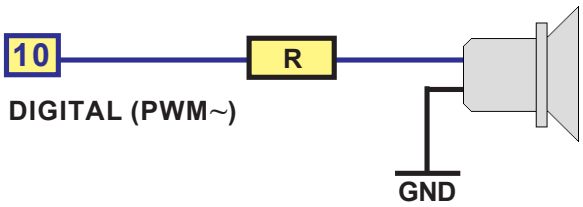
Exemple de programme :

```
/* Test du petit module BUZZER actif */
/* Génération d'un SOS en code morse sur le BUZZER*/
/* Répétiteur de contrôle sur la LED du module ARDUINO*/

int const Pilotage =  3;   //Pilotage du BUZZER en broche 3.
const int LED = 13;  // Broche de la LED U.C.

void setup()
{ pinMode(Pilotage, OUTPUT);
  pinMode(LED, OUTPUT);      // LED est une sortie.
  digitalWrite(Pilotage, HIGH); // Silence.
  digitalWrite(LED, LOW); }   // Éteint la LED de l'U.C.

void loop()
  {S(); ESPACE(); O() ; ESPACE(); S(); delay(3000);}

void POINT()
  { digitalWrite(Pilotage, LOW);
    digitalWrite(LED, HIGH);   // Allume la LED de l'U.C.
    delay(100); digitalWrite(Pilotage, HIGH);
    digitalWrite(LED, LOW); }   // Éteint la LED de l'U.C. }

void TRAIT()
  { digitalWrite(Pilotage, LOW);
    digitalWrite(LED, HIGH);   // Allume la LED de l'U.C.
    delay(300); digitalWrite(Pilotage, HIGH);
    digitalWrite(LED, LOW); }  // Éteint la LED de l'U.C.

void S()
  {  for (int i = 1; i < 4 ; i++)
       {POINT();delay(100);}}

void O()
  {  for (int i = 1; i < 4 ; i++)
       {TRAIT();delay(100);}}

void  ESPACE()
  {delay(400);}
```



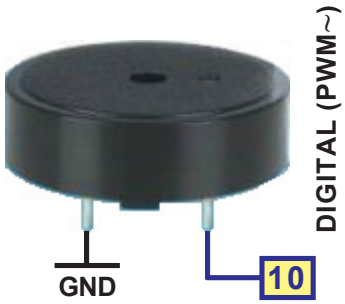
Caractéristiques techniques du composant HPR-227 :

Tous les composants de ce type présentent des caractéristiques voisines.
- Les valeurs citées ici sont celles mesurées sur le type HPR-227.
- Test effectué avec une onde de signaux carrés en niveaux TTL.
- Fréquence de résonance : ≈ 3380Hz. *(Mesurée au courant maximal)*
- Courant maximal à la résonance : ≈ 20 µA.



Il est tout à fait concevable d'utiliser un petit haut-parleur moyennant d'intercaler une résistance de limitation de courant. Le son est un peu moins aigü. L'essai a été effectué avec une valeur de **R** de 380 Ω. Il importe dans ce cas de vérifier que durant les périodes de silence la sortie 10 soit bien à 0Vcc.

Contrairement aux composants actifs intégrant une électronique pour générer un signal B.F, le sens de branchement pour un BUZZER piezzo électrique est indifférent.



Exemple de programme :

```
// Génère les tonalités "d'occupation de fréquence".
// ************************************************
// * Constantes publiques pour définir les notes. *
// ************************************************
 #define NOTE_A3  220 // L'ordre des déclarations n'est pas
 #define NOTE_B3  247 // important mais il faut impérativement
 #define NOTE_C4  262 // une ligne par constante et pas de
 #define NOTE_D4  294 // point virgule comme séparateur.
 #define NOTE_E4  330
int Melodie[] = {
  NOTE_D4, 0, NOTE_B3,NOTE_D4, NOTE_C4, NOTE_A3,NOTE_D4,
  NOTE_E4, 0, NOTE_D4, NOTE_D4};
 // Durée de la note : 4 = noire, 8 = croche, etc :
int Durees[] = {
  2, 8, 4, 4, 4, 4, 4, 2,16, 4, 2 }; // Durée inversement proportionnelle à la valeur.
void setup() {} ;
void loop() {
  // Générer les notes de la mélodie.
  for (int Note = 0; Note < 12; Note++) {
   // Pour calculer la durée de la note prendre une seconde,
   // Diviser par le type de note.
   // Exemple : Noire = 1000 / 4, Croche = 1000/8 etc.
    int Duree = 1000/Durees[Note];
    tone(10, Melodie[Note],Duree);
    // Utilisation de la broche de sortie n°10.
    // Pour distinguer les notes il faut un délai entre deux :
    // (Durée de la note) + 30 % semble convenable.
    int Pause_entre_notes = Duree * 1.30;
   delay(Pause_entre_notes);
    // Stopper la génération du son.
    noTone(10);}  // Utilisation de la broche de sortie n°10.
    delay(2000); }
```

NOTE : Les définitions des constantes pour générer les notes sont extraites d'une table plus générale donnée en page suivante. Elles sont issues d'une bibliothèque disponible en ligne, mais il vaut mieux n'en extraire que celles qui sont utilisées et ainsi alléger la taille du programme.

Beaucoup de programmes musicaux utilisent la procédures standard tone qui utilise trois paramètres. tone(n° de broche ~, Note, Duree). Note est une valeur qui définit la période du signal généré. Duree est une valeur qui définit le temps de génération. Généralement une boucle passe comme paramètre Noteet Duree dont les valeurs sont puisées dans une table. *(Exemple voir page précédente.)* L'instruction noTone(n° de broche ~); repasse la sortie à l'état repos. Le contenu de la bibliothèque "pitches.h" est listé ci dessous, mais si la mélodie ne comporte que peu de notes, il vaut mieux n'en copier que les lignes concernées au lieu d'un #include "pitches.h" qui insère la totalité dans le programme.

```
// *************************************************
// * Constantes publiques pour définir les notes. *
// *************************************************

#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
```

Autre exemple * trouvé en ligne :

```
#define c1 3830    // 261 Hz
#define d1 3400    // 294 Hz
#define e1 3038    // 329 Hz
#define f1 2864    // 349 Hz
#define g1 2550    // 392 Hz
#define a1 2272    // 440 Hz
#define b1 2028    // 493 Hz
#define C2 1912    // 523 Hz
#define R 0 // définir un silence.
```

```
#define c 3830    // 261 Hz
#define d 3400    // 294 Hz
#define e 3038    // 329 Hz
#define f 2864    // 349 Hz
#define g 2550    // 392 Hz
#define C2 2272   // 440 Hz
#define D2 2028   // 493 Hz
#define E2 1912   // 523 Hz
```

ATTENTION : Ci-contre l'exemple * trouvé sur Internet. Mais ne passe pas à la compilation car certaines constantes définies avec **un seul caractère** sont refusées. Par exemple, d et e sont acceptés mais pas c.

Caractéristiques techniques de la diode LASER :

- C'est une simple diode LASER avec une résistance de 103 Ω en série.
- Longueur d'onde 650 nm. *(Couleur rubis)*
- Puissance lumineuse 2 à 5 mW.
- Le +5Vcc peut être appliqué directement sur la broche d'alimentation, mais pour augmenter la durée de vie de la diode LASER il est possible d'insérer une résistance. Le tableau ci-dessous précise le courant d'alimentation en fonction de R, la tension d'alimentation étant de +Vcc.

R	0	22 Ω	47 Ω	82 Ω	100 Ω
I	22 mA	19 mA	17 mA	14 mA	14 mA

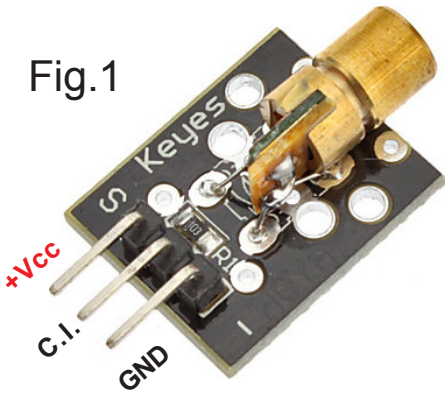


Fig.1

Exemple de programme n°1 :

Un premier exemple est donné dans le programme TEST_LASER_morse.ino qui fonctionne comme l'exemple du BUZZER actif mais c'est le module LASER qui est piloté au lieu du BUZZER. La Fig.2 indique le branchement à effectuer.

Ce petit programme se contente de générer un SOS en code morse sur la sortie binaire 12. La LED de la carte ARDUINO sert de répétiteur visuel local. le LASER s'illumine à la cadence du SOS généré sur 12.

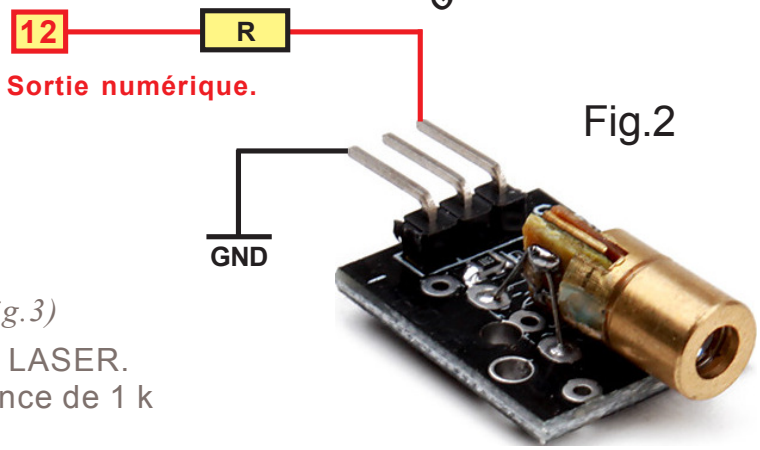


Fig.2

Exemple de programme n°2 :

(Voir schéma de la barrière optique sur la Fig.3)

```
// Exemple simple d'utilisation de la diode LASER.
// Le LASER éclaire une LDR, une résistance de 1 k
//    définit le niveau "haut".

const int LED = 13;  // Broche pour la LED.
byte VAL = 0;  // Valeur mesurée sur l'entrée analogique.

void setup() {
 pinMode(LED, OUTPUT);  // LED est une sortie.
 digitalWrite(LED, LOW);   // Éteint la LED
 Serial.begin(9600); }

void loop() {
 VAL = analogRead(0); // Lecture de l'entrée analogique 0.
 if (VAL < 50)
   {Serial.print("LAZER vu : ");Serial.println(VAL);
    digitalWrite(LED,HIGH); }
   else {Serial.print("LASER non vu : ");Serial.println(VAL);
    digitalWrite(LED,LOW); } ;
 delay(100);
}
```

Ce petit programme utilise une photorésistance LDR illuminée par le faisceau du LASER. Quand le faisceau est intercepté, la tension chute aux environs de 0,3Vcc, alors qu'elle est supérieure à 1,5Vcc quand elle est éclairée. Une entré analogique mesure la tension **E**, un niveau de référence permet de discriminer les deux états.

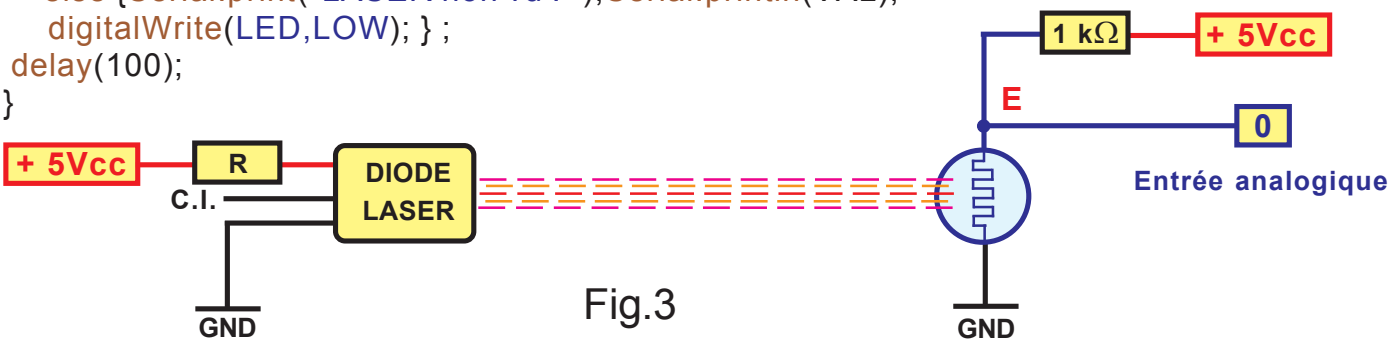


Fig.3

Incontestablement le petit module équipé d'une thermistance CTN est le plus rudimentaire des trois dispositifs décrits dans ces pages. Il est constitué d'un résistor de 10 kΩ mis en série avec une thermistance à coefficient de température négatif. La Fig.1 donne le schéma du petit circuit imprimé, la Fig.2 précise les branchements à effectuer pour interfacer avec la carte ARDUINO sur l'entrée analogique **0**.

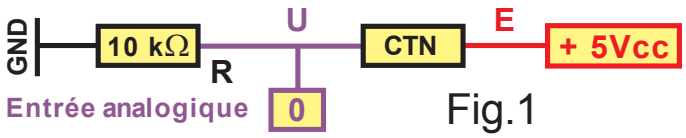


Fig.1

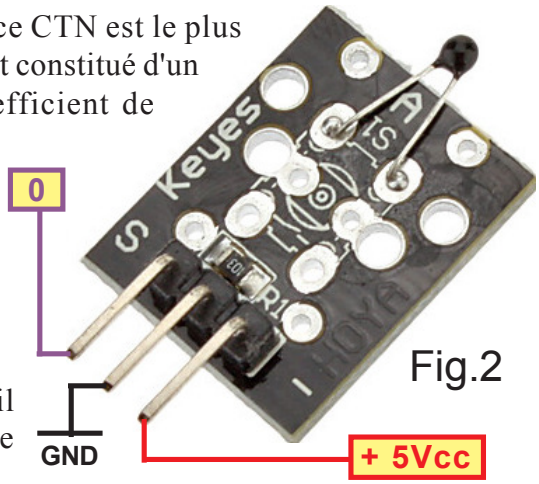


Fig.2

Mesurer la valeur de la thermistance n'est pas difficile puisqu'il s'agit d'un simple dispositif potentiométrique. La tension mesurée étant **U**, on peut facilement en déduire **CTN** en fonction de **R** :

$$\frac{E}{U} = \frac{R + CTN}{R} \Rightarrow \frac{E * R}{U} = R + CTN \Rightarrow CTN = \frac{E * R}{U} - R$$

Avec R = 10000 et E = 5 Vcc.
D'ou le calcul @.

Exemple de programme :

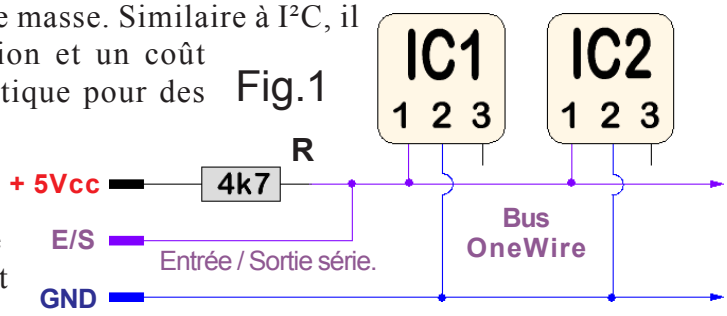
```
// Exemple simple de mesure de la valeur d'une CTN.

int Entree_de_mesure = 0; // Entrée analogique 0 utilisée.
float Tension_mesuree;    // Variable pour calculer U.
// float pour avoir U avec deux chiffres décimaux.
unsigned int RTH;             // Calcul de la valeur de la CTN.
// unsigned int pour avoir RTH affiché sans décimales.
unsigned int R = 10000;       // Valeur de la résistance R entre CTN et masse.

void setup() {Serial.begin(9600);
  Serial.println("Mesures sur une thermistance CTN."); }

void loop() {
  Tension_mesuree = analogRead(Entree_de_mesure);
  Tension_mesuree = Tension_mesuree * 5 / 1023;
  // Pour 5 Vcc sur l'entrée la conversion numérique donne 1023.
  Serial.print("Tension mesuree : "); Serial.print(Tension_mesuree);
  Serial.print(" Vcc.  >>>  Resistance CTN = ");
  RTH = ((R * 5) / Tension_mesuree) - R;  @
  Serial.print(RTH); Serial.println(" ohms.");
  delay(500); }
```

Ce petit programme mesure la tension **U** et la convertit en Volts. Il affiche cette valeur sur la ligne série. Puis il en déduit la valeur de la thermistance CTN par utilisation de la formule @. Cette valeur est alors affichée sur la voie série. La boucle infinie effectue deux mesures par seconde.

Autant mesurer la valeur de la CTN est aisé avec ARDUINO, autant la convertir en température relève d'un calcul plus délicat. La thermistance n'a pas un comportement linéaire et il importe dans la conversion d'utiliser une fonction de type logarithmique. Il existe bien des bibliothèques sur le sujet, mais les utiliser alourdira forcément le programme. On trouve des explications en ligne, par exemple sur : http://web.cecs.pdx.edu/~eas199/B/howto/thermistorArduino/thermistorArduino.pdf
Mais vu le faible coût des composants avec traitement numérique intégré tels que le DS-18B20, il devient inutile de s'engager dans des programmes compliqués et de taille inutile. Par contre, un montage basé sur un seuil, ou une comparaison reste simple et justifie pleinement l'utilisation d'un composant aussi rudimentaire. Par exemple un diviseur potentiométrique permet sur l'une des entrées analogiques de donner une consigne, une autre entrée mesure la température avec une CTN, et le programme établit une comparaison. Il peut alors commander une interface de puissance pour réguler le chauffage d'un appareil spécifique. Par exemple thermostater l'enceinte d'un générateur étalon de fréquence à quartz.

Fabriqué par Dallas Semiconductor, ce composant complexe utilise pour dialoguer avec un microcontrôleur, un **bus nommé OneWire**. OneWire est un bus conçu par Dallas Semiconductor qui permet de connecter en série, en parallèle ou en étoile des modules avec seulement deux fils : Un fil de données de type "collecteurs ouverts" et un fil de masse. Similaire à I²C, il présente cependant des vitesses de transmission et un coût inférieurs. Il est généralement utilisé en domotique pour des thermomètres ou d'autres instruments de mesures météorologiques. Pour dialoguer avec les modules utilisant le bus OneWire, Dallas Semiconductor met à la disposition d'ARDUINO une bibliothèque spécifique nommée <OneWire.h> qu'il faut importer dans l'environnement de développement.

Fig.1



Caractéristiques techniques du capteur de température de DALLAS :

La documentation complète avec les protocoles de dialogue est disponible dans le fichier DS18B20.pdf avec en Fig.2 le brochage du composant. La broche DQ est celle qui se branche sur la ligne série avec forçage de niveau haut par **R**.

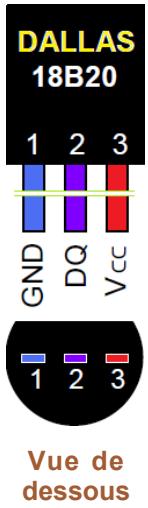


Fig.2

- Interface OneWire® avec une seule broche pour le port de communication.
- Chaque composant dispose d'un code d'identification de 64 bits stocké dans une ROM.
- Pas besoin de composants externes.
- Source de courant comprise entre 3Vcc et 5.5Vcc.
- Mesures des températures comprises entre -55 °C et +125 °C. *(-67 °F à +257 °F)*
- Précision ± 0,5°C entre -10 °C à +85 °C.
- Résolution définie par l'utilisateur entre 9 et 12 bits.
- Temps de conversion 750 mS MAX pour 12 bits.
- Paramètres d'alarme définissables par l'utilisateur dans une mémoire non volatile.
- Commande pour identifier un capteur dont la mesure déborde les limites de l'alarme programmée.
- Disponible en boitier linéaire 8 broches ou en TO-92.

Exemple de programme:

Le programme Temperature_DS_18B20.ino très détaillé exploite la bibliothèque OneWire.h et liste sur la ligne série *(Voir Fig.3)* le résultat des mesures par utilisation de la broche d'E/S binaire n°2. La Fig.4 montre les trois branchements à effectuer pour relier le petit module DS-18B20 à la carte ARDUINO.

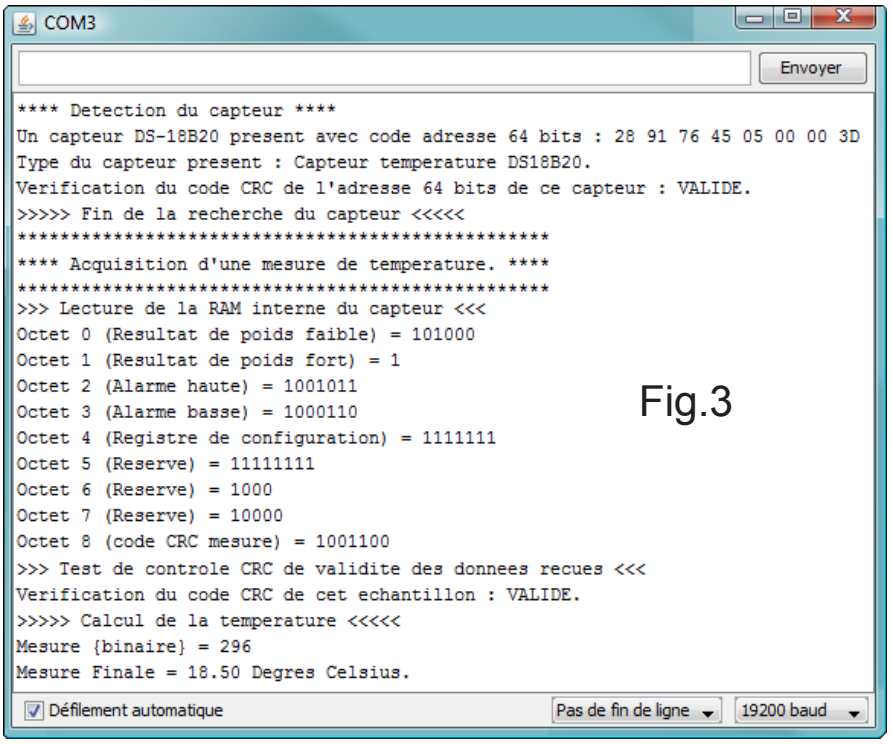
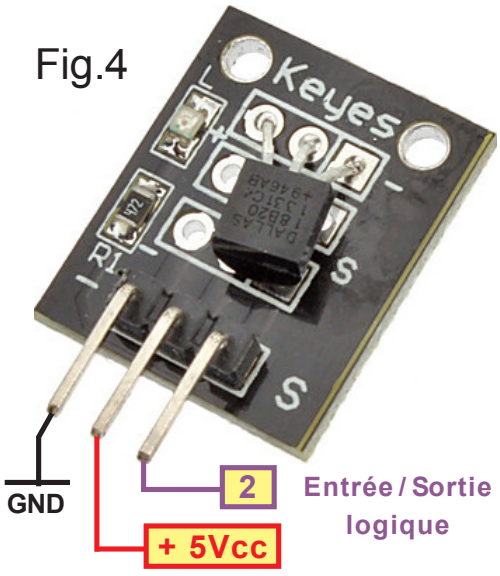


Fig.3

Si utilisé en "mono-coup", le résultat d'une mesure est déclenché au moment de l'activation du moniteur série ou à convenance sur RESET.

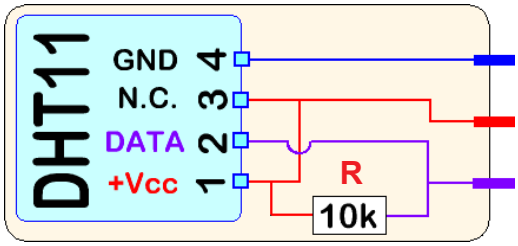
Fig.4



Montré sur la Fig.2, ce module de mesure est constitué d'un senseur de température à base d'une thermistance à coefficient négatif de température et d'un capteur d'humidité résistif. Un microcontrôleur intégré s'occupe de faire les mesures, de les convertir et de les transmettre. Chaque module est calibré en usine et ses paramètres de calibration sont stockés dans la mémoire ROM du microcontrôleur local. Le DHT11 fait partie de ces circuits numériques qui dialoguent au moyen d'une seule ligne série bidirectionnelle "à collecteur ouvert" de type OneWire. *(Un fil)*

Caractéristiques techniques du capteur d'humidité DHT11 :

Fig.1



La Fig.1 donne le brochage du composant ainsi que celui du petit module en circuit imprimé. La Fig.2 indique comment établir la liaison avec une carte ARDUINO. La mise en œuvre sur ARDUINO est d'autant plus facile que la bibliothèque DHT11.h est disponible en ligne. Elle se charge des protocoles de dialogue de type série. La ligne filaire peut faire jusqu'à 20 m de longueur entre le capteur et l'U.C. de traitement utilisée, dans ce cas il faut diminuer la valeur de **R** à 4,7 kΩ. Le format des données série est de 40 bits.

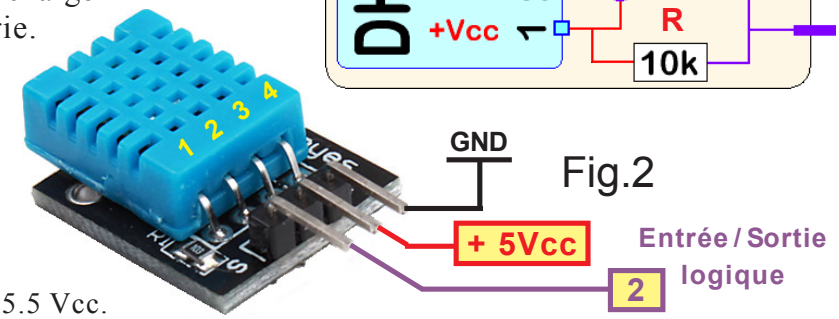


Fig.2

- Mesure de l'humidité relative. *(RH)*
- Alimentation comprise entre 3 Vcc et 5.5 Vcc.
- Courant d'alimentation : 100 µA maximum en veille, 2.5 mA maximum en dialogue.
- Mesures des températures comprises entre 0 °C et 50 °C.
- Mesures d'humidité entre 30 % et 90% RH à 0°C.
  entre 20 % et 90% RH à 25°C.
  entre 20 % et 80% RH à 50°C.
- Résolution 1 % RH. • Précision à 25 °C : ± 5% RH.

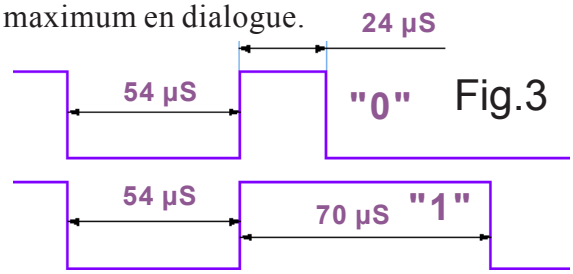


Fig.3

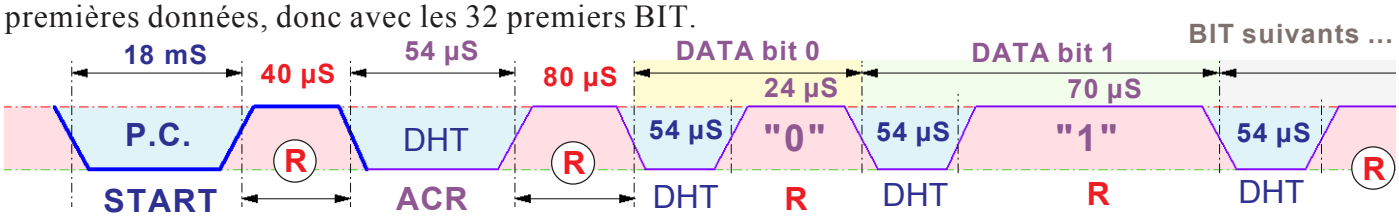
DIALOGUE sur la ligne DATA.

Lorsque le P.C. envoie un signal de démarrage, le DHT11 passe du mode faible consommation électrique au fonctionnement en mode dialogue et envoie l'accusé de réception suivi des données. *(Si le signal de départ est valide)* Il repasse ensuite en mode veille jusqu'à la prochaine sollicitation.

1) Demander une mesure : Forcer l'état "0" durant 18 mS puis état "1" durant 40 µS.
2) Accusé de réception du DHT11 : Forcer l'état "0" durant 54 µS puis état "1" durant 80 µS.
3) Lecture des données : Les données suivent l'accusé de réception et sont codées dans une série de 40 bits sous forme de cinq valeurs de 8 bits chacune dont voici le format :

8 BIT	8 BIT	8 BIT	8 BIT	8 BIT
RH *(Partie entière)*	RH *(Partie décimale)*	T°C *(Partie entière)*	T°C *(Partie décimale)*	Check Sum

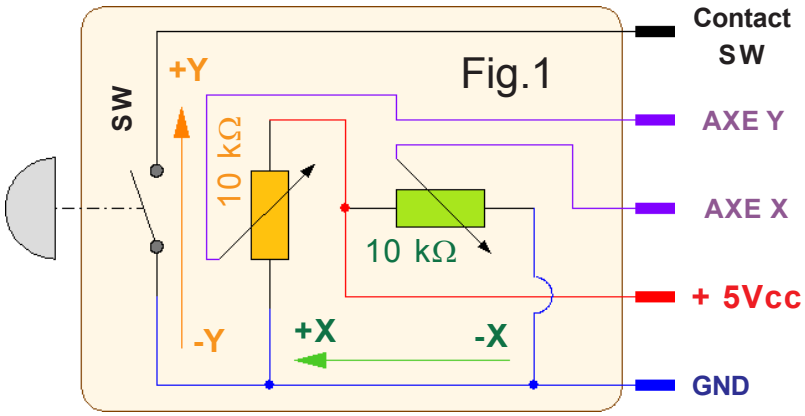
La Fig.3 donne le format des BIT de données, la Check Sum est calculée par la somme des quatre premières données, donc avec les 32 premiers BIT.



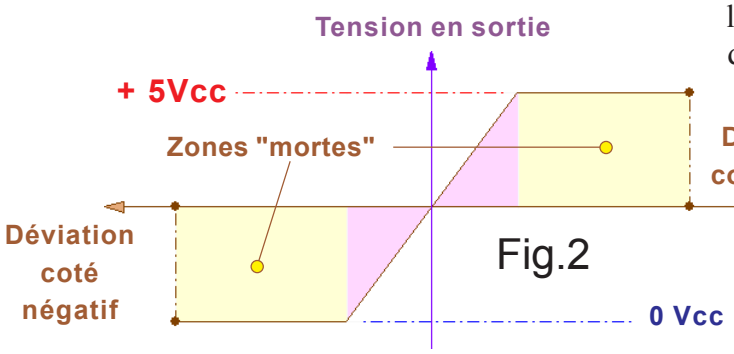
* En bleu clair l'état zéro forcé par le P.C. ou en réponse par le DHT11.
* En rose pâle l'état "1" maintenu par la résistance de forçage **R**. *(Pull Up)*
* En bleu épais la requête du P.C. et en violet fin la réponse du capteur.

Exemple de programme:

Le programme Humidistance_DHT11.ino très détaillé exploite la bibliothèque DHT11.h et liste sur la ligne série *(Boucle infinie ≈ 2 secondes)* le résultat des mesures par utilisation de la broche d'E/S binaire n°2. La Fig.2 montre les trois branchements à effectuer pour relier le petit module à ARDUINO.

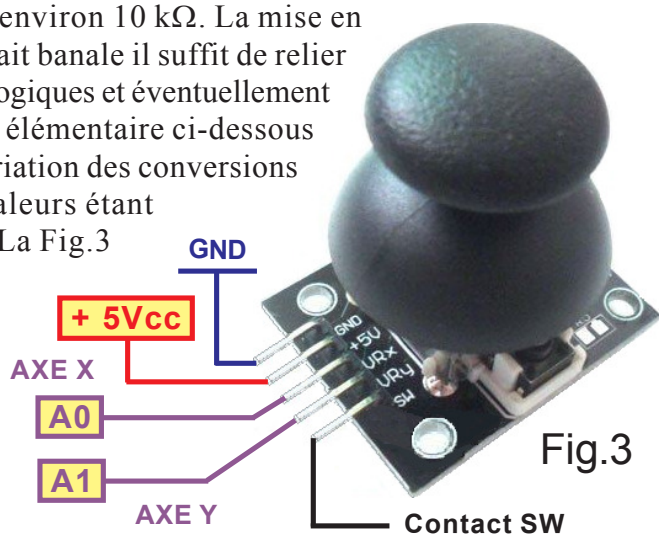


Difficile de concevoir plus simple. Comme le montre la Fig.1 le mini joystick est constitué de deux potentiomètres d'environ 10 kΩ branchés entre le **+ 5Vcc** et la masse **GND** et d'un contacteur élémentaire **SW**. La Fig.1 présente l'orientation géométrique des deux axes ainsi que leurs sens de variations. Sur ce dessin le circuit imprimé est vu de dessus coté composants. On remarque que l'axe des X'X est alors en orientation inverse de celle généralement adoptée pour l'usage des repères conventionnels de sens direct. Mais rien n'empêche d'inverser les rôles joués par les axes X'X et Y'Y pour retrouver des sens habituels pour les variations. La Fig.2 présente l'allure de l'évolution de tension en sortie en fonction de l'angle d'inclinaison du mini-manche. Cette courbe représentative est globalement linéaire, mais à peine un tiers de la course totale participe à la variation, un tiers de chaque extrémité de la course se comporte en zone "morte", l'évolution restant figée à la valeur maximale ou minimale. Notons au passage que si le contact **SW** est utilisé pour servir de consigne d'initialisation par exemple, il faudra forcer un niveau "1" en reliant la broche **SW** au **+ 5Vcc** par l'intermédiaire d'une résistance d'environ 10 kΩ. La mise en œuvre ne présente pas de difficulté, et de façon tout à fait banale il suffit de relier les deux sorties potentiométriques vers deux entrées analogiques et éventuellement l'inverseur **SW** vers une entrée binaire. Le programme élémentaire ci-dessous permet de se faire facilement une idée des profils de variation des conversions numériques effectuées sur les axes X'X et Y'Y, les valeurs étant affichées en boucle continue rapide sur la voie série. La Fig.3 ci-contre précise les branchements à effectuer.



Exemple de programme :

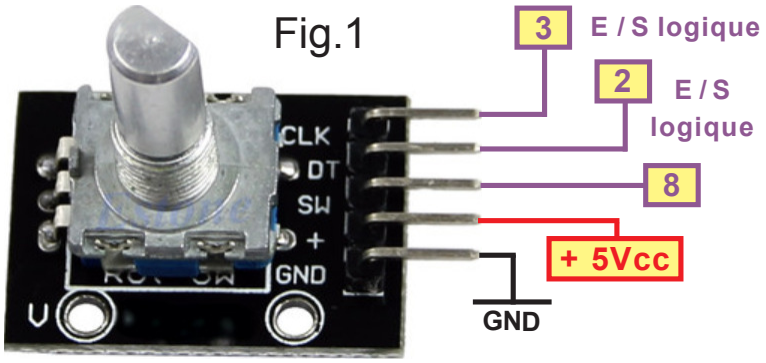
```
// Exemple d'utilisation du mini-joystick.
void setup() {
  Serial.begin(19200);}

void loop() {
  int VarX = analogRead(A0);
  int VarY = analogRead(A1);
  Serial.print("X'X = ");
  Serial.print(VarX, DEC);
  Serial.print(" / ");
  Serial.print("Y'Y = ");
  Serial.println(VarY, DEC); }
```



Ce petit programme liste en boucle infinie rapide les valeurs numériques retournées par les CNA sur les entrées A0 et A1 branchées respectivement sur les sorties X'X et Y'Y

Autres exemples de programmes en ligne :

Exemple simple avec un affichage sur écran LCD :
http://www.instructables.com/id/Arduino-Joystick-Breadboard-with-LCD-Output/step5/
Exemple avec pilotage d'une petite caméra :
http://arduino4projects.com/joystick-controlled-camera-using-arduino/
Exemples variés :
http://tronixstuff.com/tag/joystick/



Le KY-40 est un codeur **sans butée** à 20 points par tour pourvu d'un "bouton poussoir" par appui sur la tige de commande qui permet d'envoyer une consigne de RAZ par exemple. La sortie se fait par deux lignes pilotées par des capteurs de type codage Gray. La Fig.1 donne le schéma des branchements à réaliser sur ARDUINO pour utiliser le petit programme donné en exemple.

Caractéristiques techniques du module KY-040 :

- Consommation maximale : 10 mA sous 5 Vcc.
- Température de fonctionnement : - 30 à + 70 °C.
- Température de stockage : - 40 à + 85 °C.
- Durée de vie du capteur de rotation : Minimum 30 000 cycles.
- Durée de vie du contact de RAZ : Minimum 20 000 cycles.
- Résistance de passage du contact de RAZ : 3 Ω maximum.

Fonctionnement :

Concrètement les trois sorties sont alimentées au + 5 Vcc par des résistances de 10 kΩ. La sortie **SW** est celle de l'inverseur piloté par appui sur la tige du rotor. Les deux sorties **CLK** et **DT** sont en réalité deux sorties classiques avec déphasage de 90° souvent nommées **A** et **B** pour ce type de capteur. Le déphasage de 90° électriques des signaux **CLK** et **DT** permet de déterminer le sens de rotation. *(Voir Fig.2)*

Exemple de programme :

Ce petit programme procède au comptage ou au décomptage en fonction du sens de rotation du bouton. Appuyer sur la tige de rotation provoque la remise à zéro du compteur. Les valeurs affichées sont limitées entre zéro et la valeur déclarée par la constante **Max**.

```
/* Exemple d'utilisation du codeur rotatif KY-40 */
/*      Codeur sans butée 20 pas par tour       */
/* Branchements à effectuer sur ARDUINO :
   GND  -> GND
   +    -> +5V
   SW   -> D8
   DT   -> D2
   CLK  -> D3
   */
enum assignation_des_broches
  { Codeur_A = 2,   // DT
    Codeur_B = 3,   // CLK
    Bouton_de_RAZ = 8 }; // SW
volatile unsigned int Position_Codeur = 0;  // Compteur dèstiné à la "molette".
unsigned int Derniere_position = 1;         // Gestion des rotations.
boolean Changement = false;         // Gestion anti-rebonds.

boolean A_set = false;
boolean B_set = false;
long Max = 10; // Valeur comptée maximale.

void setup() {
  pinMode(Codeur_A, INPUT);
  pinMode(Codeur_B, INPUT);
  pinMode(Bouton_de_RAZ, INPUT);
  digitalWrite(Bouton_de_RAZ, HIGH);
  // Broche de l'interruption 0 (E/S 2)
  attachInterrupt(0, doEncoderA, CHANGE);
```

```
  // Broche de l'interruption 1 (E/S 3)
  attachInterrupt(1, doEncoderB, CHANGE);
  Serial.begin(9600); }
void loop() {
  Changement = true;  // RESET l'anti-rebonds.
  if (Derniere_position != Position_Codeur) { // Si différents, afficher.
    Serial.print("Position : ");
    Serial.print(Position_Codeur, DEC);
    if (Position_Codeur == 0) Serial.print( " >>> Comptage en butee nulle.") ;
    if (Position_Codeur == Max) Serial.print( " >>> Comptage en butee maximale.") ;
    Serial.println() ;
     Derniere_position = Position_Codeur; }
  if (digitalRead(Bouton_de_RAZ) == LOW )
     {Position_Codeur = 0; } }  // Remise à zéro si on appuie sur la tige du capteur.
// Interruption quand A change d'état avant B.
void doEncoderA(){
  if ( Changement ) delay (5);  // Attendre un peu la fin du rebond.
  // Test pour savoir si l'état a changé.
  if( digitalRead(Codeur_A) != A_set ) {  // Anti-rebonds encore une fois.
     A_set = !A_set;
    // Incrémenter le compteur si A avant B.
     if ( A_set && !B_set )
        Position_Codeur += 1;
     if (Position_Codeur > Max )
        Position_Codeur=Max;
        Changement = false; } }  // Changement pris en compte.
// Interruption quand B change d'état avant A.
void doEncoderB(){
  if ( Changement ) delay (5);  // Attendre un peu la fin du rebond.
  // Test pour savoir si l'état a changé.
  if( digitalRead(Codeur_B) != B_set ) {  // Anti-rebonds encore une fois.
     B_set = !B_set;
    // Décrémenter le compteur si B avant A.
     if (Position_Codeur == 0)
        return;
     if( B_set && !A_set ) Position_Codeur -= 1;
      Changement = false; } }
```

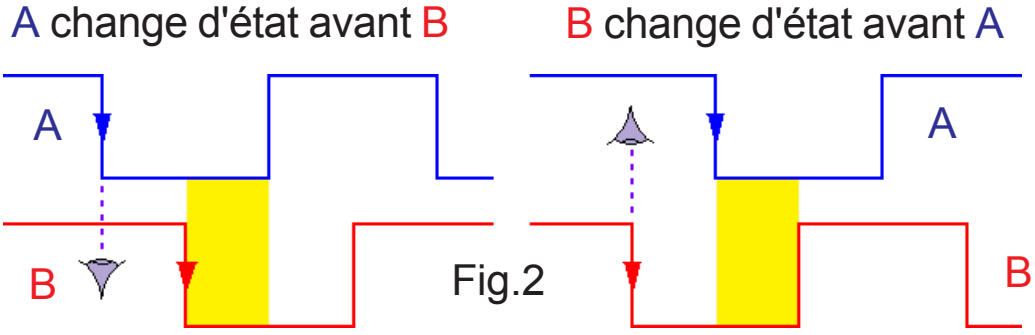
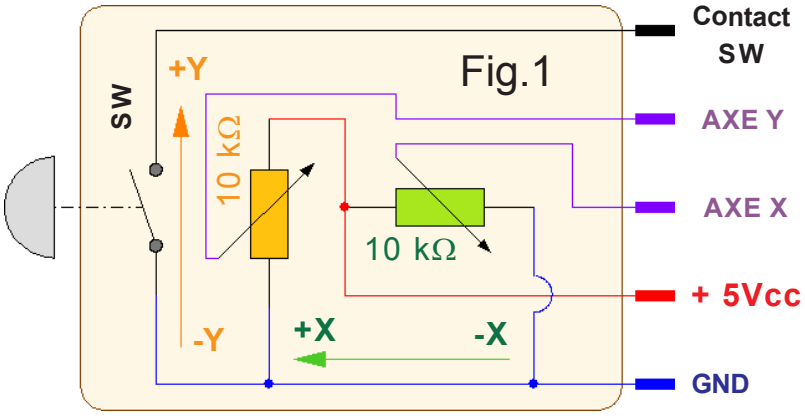


Fig.2

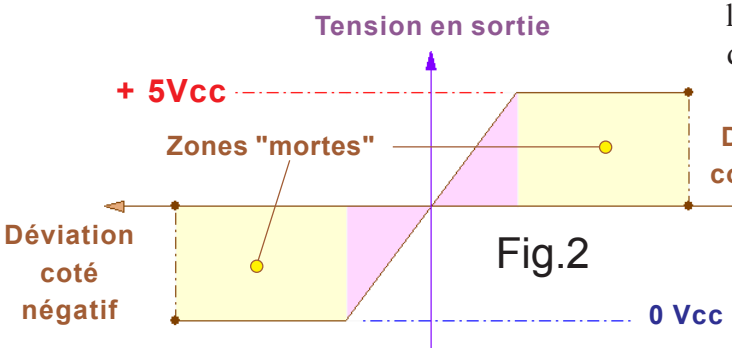
Comme montré Fig.2 la détermination du sens de rotation peut se faire par l'ordre chronologique des transitions du front descendant par exemple.

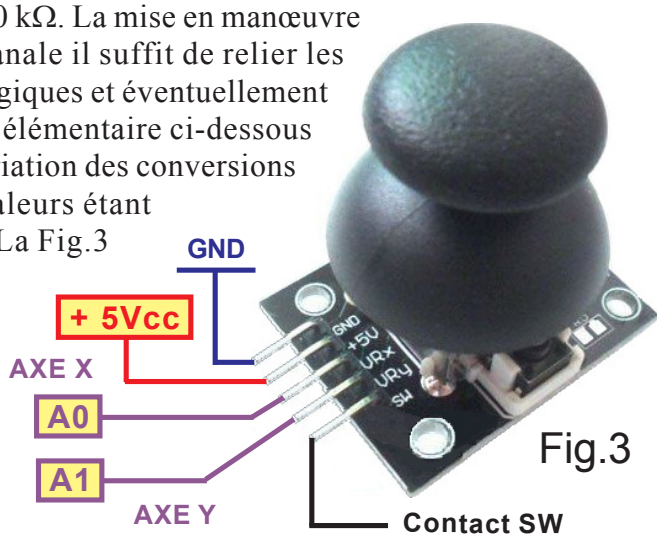
Autre possibilité de détermination du sens de rotation :

- Dans un sens de rotation pendant le front montant du signal **A**, le signal **B** est à 0.
- Dans l'autre sens de rotation pendant le front montant du signal **A**, le signal **B** est à 1.



Difficile de concevoir plus simple. Comme le montre la Fig.1 le mini joystick est constitué de deux potentiomètres d'environ 10 kΩ branchés entre le **+ 5Vcc** et la masse **GND** et d'un contacteur élémentaire **SW**. La Fig.1 présente l'orientation géométrique des deux axes ainsi que leurs sens de variation. Sur ce dessin le circuit imprimé est vu de dessus coté composants. On remarque que l'axe des X'X est alors en sens inverse de celui adopté généralement pour les repères conventionnels de sens direct. Mais rien n'empêche d'inverser les rôles joués par les axes X'X et Y'Y pour retrouver des sens habituels pour les variations. La Fig.2 présente l'allure de la variation de tension en sortie en fonction de l'angle d'inclinaison du mini-manche. Cette courbe représentative est globalement linéaire, mais à peine un tiers de la course totale participe à variation, un tiers de chaque extrémité de la course se comporte en zone "morte" l'évolution restant figée à la valeur maximale ou minimale. Notons au passage que si le contact **SW** est utilisé pour servir de consigne d'initialisation par exemple, il faudra forcer un niveau "1" en reliant la broche **SW** au **+ 5Vcc** par l'intermédiaire d'une résistance d'environ 10 kΩ. La mise en manœuvre ne présente pas de difficulté, et de façon tout à fait banale il suffit de relier les deux sorties potentiométriques vers deux entrées analogiques et éventuellement l'inverseur SW vers une entrée binaire. Le programme élémentaire ci-dessous permet de se faire facilement une idée des profils de variation des conversions numériques effectuées sur les axes X'X et Y'Y, les valeurs étant affichées en boucle continue rapide sur la voie série. La Fig.3 ci-contre précise les branchements à effectuer.





Exemple de programme:

```
// Exemple d'utilisation du mini-joystick.
void setup() {
  Serial.begin(19200);}

void loop() {
  int VarX = analogRead(A0);
  int VarY = analogRead(A1);
  Serial.print("X = ");
  Serial.print(VarX, DEC);
  Serial.print(" / ");
  Serial.print("Y = ");
  Serial.println(VarY, DEC); }
```

Ce petit programme liste en boucle infinie rapide les valeurs numériques retournées par les CNA sur les entrées A0 et A1 branchées respectivement sur les sorties X'X et Y'Y

Autres exemples de programmes en ligne :

Exemple simple avec un affichage sur écran LCD :
http://www.instructables.com/id/Arduino-Joystick-Breadboard-with-LCD-Output/step5/
Exemple avec pilotage d'une petite caméra :
http://arduino4projects.com/joystick-controlled-camera-using-arduino/
Exemples variés :
http://tronixstuff.com/tag/joystick/

Déjà utilisé en page 7, le petit Buzzer piézoélectrique HPR-227 fonctionne également en source de tensions s'il est branché sur une ligne à haute impédance. Quand un matériau de cette famille est soumis à des tensions variables, il se dilate ou se contracte imperceptiblement à la fréquence des signaux. C'est suffisant pour faire vibrer l'air environnant et générer un son audible. L'amplitude de la déformation est directement fonction de la tension appliquée à ses bornes. Réciproquement, si le matériau est soumis à des variations de pression, des charges électriques apparaissent à ses bornes. Hors toute vibration ou choc aura cet effet sur un composant de ce type, car, par définition, un son est constitué de variations dynamiques de pression dans l'air ambiant. Ces variations de pression influences le Buzzer qui peut alors générer un signal électrique. On va dans cette petite application utiliser cette propriété physique. Si le HPR-227 est relié à une entrée à haute impédance telle que **A0** par exemple, les bruits ambiant vont générer des tensions à ses bornes qui seront lues par l'entrée analogique. Il faut toutefois prendre deux précautions élémentaires, l'une pour se prémunir de l'électricité statique, l'autre pour protéger l'entrée analogique utilisée sur Arduino. Considérons la Fig.1 sur laquelle on retrouve le Buzzer piézoélectrique noté **Bz** et représenté comme un quartz. *(Le quartz a été le tout premier matériau découvert présentant ces particularités.)* On retrouve également la résistance de 390Ω qui limite le courant de pointe pouvant être généré par une sortie. Mais maintenant c'est sur une entrée analogique que se trouve branché notre "effecteur" qui devient un capteur. Les entrées analogiques de l'ATmega328 présentent une impédance d'entrée élevée. **Bz** est également une entité électrique avec une impédance très élevée. Si on ne branchait pas les composants représentés en rouge, les fils de liaison se comporteraient comme des antennes. Ces dernier récupéreraient toute les tension électrostatique des environs. L'entrée **A0** serait alors soumise à des fluctuations aléatoires de faux signal. Pour juguler ce phénomène, il suffit de diminuer l'impédance d'entrée à une valeur faible, mais sans pour autant "abrutir" **Bz** qui ne peut délivrer que des courants infimes. La valeur de **1MΩ** pour **R** semble un bon compromis. L'entrée **A0** ne sera plus victime de tensions parasites, mais elle se trouve potentiellement en danger. En effet, si un composant piézoélectrique ne peut pas fournir un courant notable, par contre les tensions qui apparaissent à ses bornes peuvent s'avérer très élevées : Des centaines de vol sur un choc énergique. *(C'est le principe des allume-gaz.)* Pour être certain d'éliminer définitivement ce risque, même s'il n'est statistiquement pas très probable, la diode Zener **DZ**, *(Du nom de son inventeur)* va empêcher toute tension négative de plus de 0,6Vcc. C'est le seuil de conduction classique. Mais "attaquée" en inverse, sa "tension de claquage" est très faible sur ce type de diode. J'ai utilisé un composant où la tension en inverses limitra le signal à +5,1Vcc mais tout modèle entre 2Vcc et 5,5Vcc conviendra.

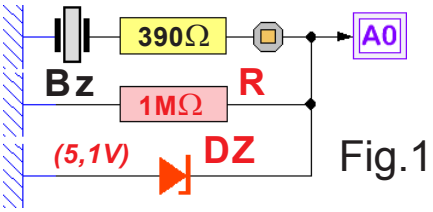
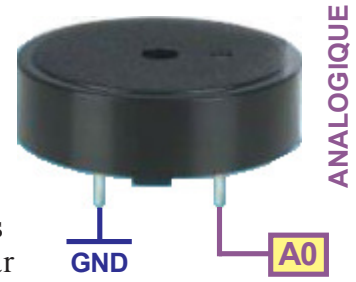


Fig.1



Sans particularités extraordinaire, le programme PIEZZO_en_capteur_de_sons.ino utilise toutefois une ou deux petites subtilités pas toujours exploitées. Par exemple les trois déclarations des constantes qui définissent les rôles joués par les broches sont données par des directives #define. En particulier, il faut translater la plage de valeur de la variable Capteur comprise entre [0 et 1023] en une valeur de tension lue sur **A0** qui se situe dans la plage [0 à 5000]. La valeur de 5000 est choisie pour avoir une grandeur exprimée en mV qui puisse ensuite être divisée par 1000 sans perdre les décimales. Pour effectuer cette transposition dans les fourchettes de valeurs indiquées, on a utilisé la fonction map bien commode. Chaque fois que le capteur détecte un bruit, l'état d'éclairement de la LED branchée sur la sortie binaire **D2** s'inversera. Simultanément, la valeur de la tension mesurée sur **A0** sera envoyée sur la ligne série USB. En fonction de la valeur de la constant Seuil_detection on pourra si on le désire rendre le capteur extrêmement sensible. Par exemple dans le programme PIEZZO_en_capteur_de_sons.ino ce seuil de détection est imposé à 2mV. Le plus petit souffle, la plus "tendre caresse" avec le capteur ou la platine d'expérimentation va déclencher le basculement du processus. Dans ces conditions, si l'on désire pouvoir lire la valeur de la tension il faut l'exprimer en mV. Mais l'on peut fort bien vouloir utiliser notre détecteur non plus comme touche sensitive, mais comme détecteur de choc en ambiance très bruyante. On doit alors donner à la constante Seuil_detection une valeur bien plus élevée, par exemple 1 Volts. Attention, c'est en mV que cette constante est exprimée. Il faut donc coder 1000 pour ce choix. Avec le HPR-227 il faut alors "toquer" directement sur **Bz** avec un stylet rigide, *(Manche d'un petit tournevis par exemple)* pour déclencher une détection de choc. Naturellement, cette valeur de seuil sera ajustée expérimentalement en fonction de l'effet désiré et du modèle de capteur utilisé. On peut remplacer **Bz** par un petit hautparleur, dans ce cas les éléments de protection représentés en rouge sur la Fig.1 ne sont plus du tout utiles.

Comme montré sur le dessin de la Fig.2 ci-dessous c'est un clavier de type "multiplexé ligne / colonne". Attention, il n'y a par de résistance de forçage de niveau, il faut les prévoir en extérieur pour l'utiliser de façon classique. Toutefois, une exploitation banale de type ligne / colonne imposerait de mobiliser 4 sorties et 4 entrées sur la carte ARDUINO. Moyennant de faire appel à une entrée analogique et à une série de résistances placées en parallèle et constituant un diviseur de tension il devient possible de n'utiliser qu'une seule entrée. L'idée de base pour effectuer la différenciation des lignes se fait alors par l'application de valeurs de tensions différentes sur ces dernières. La Fig.3 donne le principe utilisé. Les résistances de 1 kΩ alimentent les colonnes avec le **+ 5Vcc** lorsque l'une des sorties binaires passe au niveau haut. Les tensions notées sur la Fig.3 sont celles obtenues quand on appui sur l'un des boutons poussoir situés sur la ligne testée. En réalité les valeurs de résistances utilisées privilégient des éléments très courants, les tensions mesurées sont fonction de la chute de potentiel d'environ 0,5 Vcc au passage du courant dans les diodes **D**. Dans la pratique, les tensions présentes seront légèrement plus faibles car les sorties binaires ne fournissent pas exactement la valeur de + 5 Vcc quand elles sont forcées au niveau "1". Pour l'entrée analogique on peut utiliser à convenance n'importe laquelle puisque toutes présentent le même comportement. Pour les sorties binaires, celles qui sont proposées sur le schéma de la Fig.3 ne sont pas choisies au hasard. Ce sont celles qui ne peuvent que fonctionner en binaire "statique". Ainsi on laisse disponibles les sorties de type PWM et TX/RX pour un maximum de souplesse dans l'application envisagée. Si aucun des boutons poussoir de la colonne n'est appuyé, la tension présente et le courant consommé seront nuls, il est ainsi facile d'agencer une routine d'exploration de ce petit clavier. Le petit tableau ci-dessous résume les résultats obtenus avec les valeurs données dans le schéma.

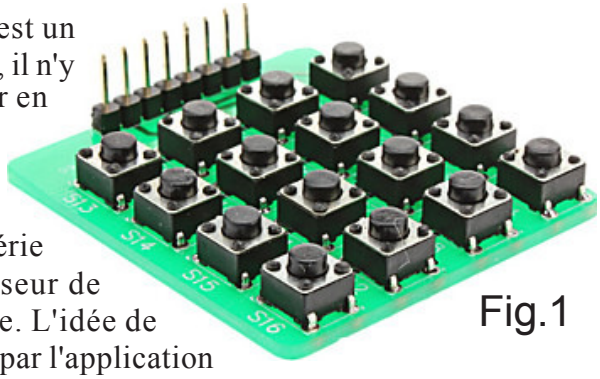
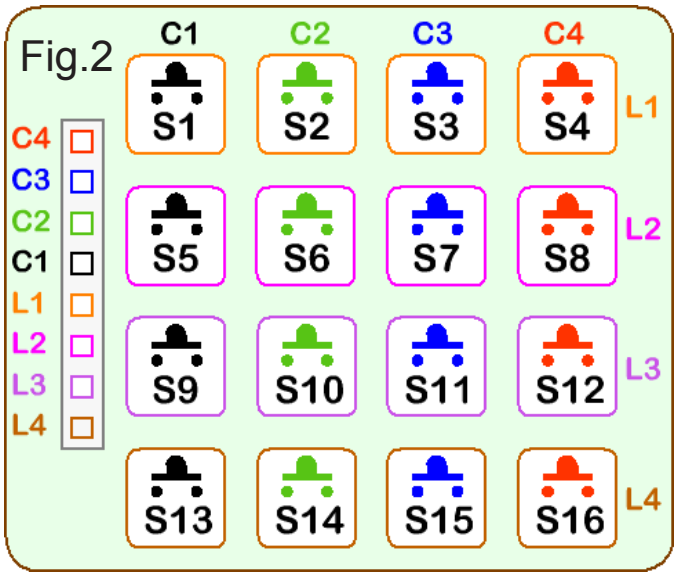


Fig.1

Fig.2



Tensions et numérisation.

SW	Tension	CAN
Aucun	0 Vcc	0
S1	3.95 Vcc	806
S2	3,1 Vcc	631
S3	1,95 Vcc	392
S4	0,95 Vcc	180

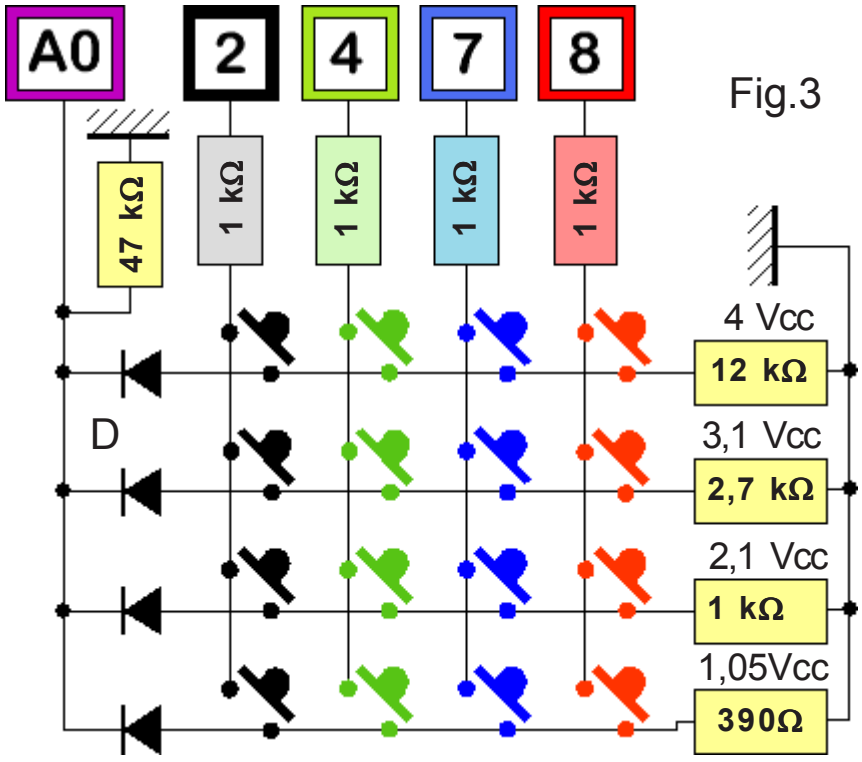
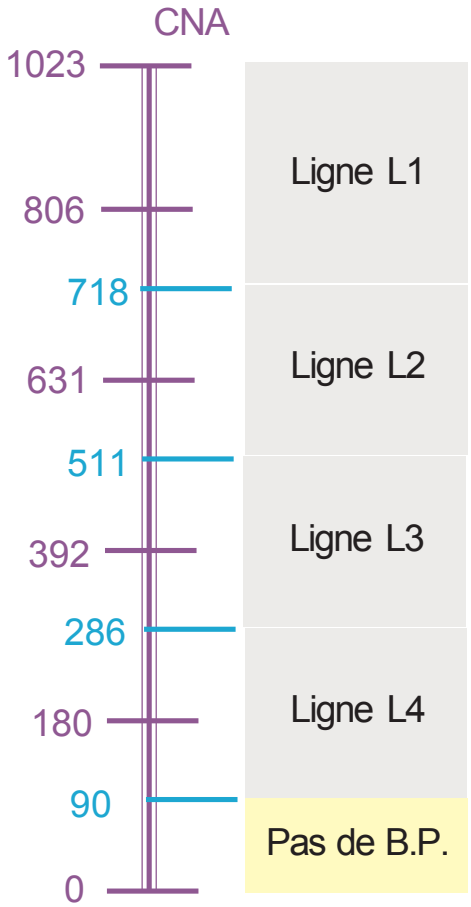


Fig.3

Exemple de programme :

```
// Test du petit clavier à 16 boutons poussoirs.
const byte Entree_mesuree = 0; // Entrée analogique 0 utilisée.
const byte C1 = 2;  // Colonne C1 sur la sortie 2.
const byte C2 = 4;  // Colonne C2 sur la sortie 4.
const byte C3 = 7;  // Colonne C3 sur la sortie 7.
const byte C4 = 8;  // Colonne C4 sur la sortie 8.
byte SW = 0 ; // Bouton poussoir utilisé
int CNA; // Mesure analogique retournée par le CNA.

void setup() {
   Serial.begin(19200);
   // 19200 bauds semble le maximum par ma ligne USB.
   Serial.println(">>> Test du petit clavier 4 x 4 <<<");
   Serial.println();
   pinMode(C1, OUTPUT);
   pinMode(C2, OUTPUT);
   pinMode(C3, OUTPUT);
   pinMode(C4, OUTPUT);
   digitalWrite(C1, LOW);
   digitalWrite(C2, LOW);
   digitalWrite(C3, LOW);
   digitalWrite(C4, LOW); }

void loop() {
  TesteColonne(C1);
  TesteColonne(C2);
  TesteColonne(C3);
  TesteColonne(C4);}

void TesteColonne(byte Colonne) {
  digitalWrite(Colonne, HIGH); // Passe la colonne au +5Vcc.
  CNA = analogRead(Entree_mesuree);
  if (CNA > 90) {CodeSW(Colonne);} ;
// Code et affiche le n° du B.P. inscrit sur le petit circuit imprimé.
  digitalWrite(Colonne, LOW);} // Repasse la colonne à zéro volts.

void CodeSW(byte Colonne) {
  SW = 13;
  if (Colonne == C2) {SW = 14;} ;
  if (Colonne == C3) {SW = 15;} ;
  if (Colonne == C4) {SW = 16;} ;
  if (CNA > 286) {SW = SW - 4;} ;
  if (CNA > 511) {SW = SW - 4;} ;
  if (CNA > 718) {SW = SW - 4;} ;
  Serial.print("Bouton S"); Serial.println(SW);
  while (analogRead(Entree_mesuree) > 90)
        {delay (5); }  } // Attendre le relâcher.
```

En bleu clair les seuils utilisés pour établir les plages de valeurs définissant les B.P. utilisés. Les valeurs données dans le petit tableau de la page 1/2 sont des valeurs moyennes issues de plusieurs mesures.



Dès que l'on appuie sur l'un des boutons poussoirs son n° d'ordre est affiché sur la ligne série. La procédure attend ensuite son relâcher avant de reprendre le balayage des colonnes.

Note : *La variable Colonne est passée systématiquement en paramètre, mais elle pourrait parfaitement rester une variable globale déclarée en tête de programme.*

Autres exemples :

http://forum.arduino.cc/index.php?topic=121361.0

http://morgan-blog.over-blog.com/article-arduino-clavier-matriciel-anti-rebond-96811880.html

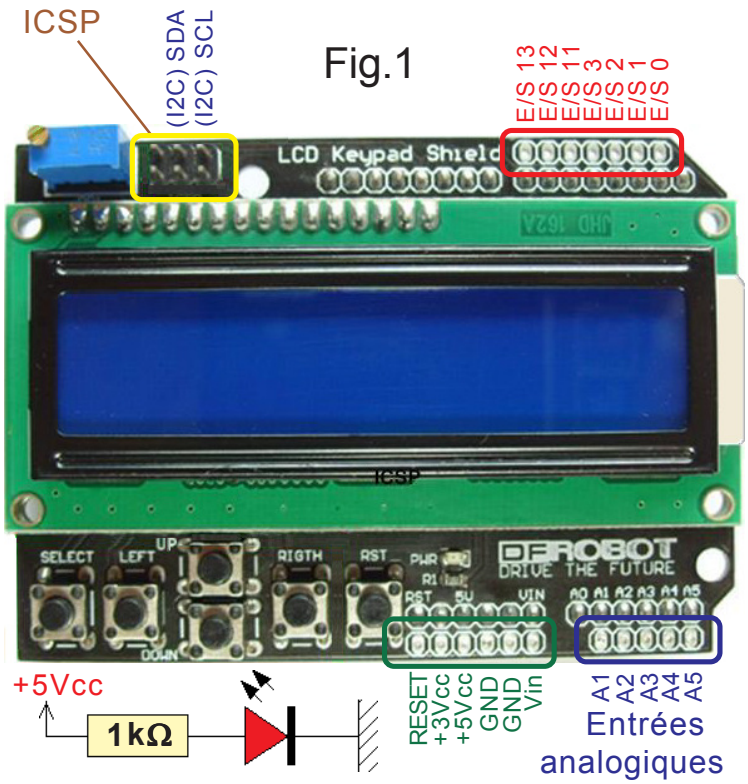


Fig.1

L'architecture du module LCD qui se branche directement sur les connecteurs d'Arduino est donnée sur la Fig.2 qui en précise les divers branchements. La bibliothèque LiquidCrystal, permet de définir les E/S utilisées par l'interface qui relie le module LCD du projet à la carte Arduino. Le bouton RESET du SHIELD est en parallèle avec celui du microcontrôleur. On peut noter sur la Fig.1 que certaines broches des connecteurs d'Arduino peuvent être reprises moyennant de souder des lignes de picots idoines aux emplacements prévus à cet effet. Une petite LED rouge confirme la présence du +5Vcc sur la carte de l'afficheur LCD. On remarque sur la Fig.2 que pour économiser des E/S sur le microcontrôleur l'afficheur est utilisé avec le minimum de liaisons d'interface. Il est figé en mode écriture sur sa broche **5**, les commandes sont transmises sur les entrées **DB4** à **DB7** en pilotage séquentiel sur quatre bits. Les entrées **DB0** à **DB3** sont inutilisées. Le rétro éclairage peut être piloté par le transistor **T** qui sans utilisation de la sortie **10** est saturé par la résistance **R** de 4,7KΩ. Mais le module commercialisé présente un danger incontestable autant pour le transistor **T** que pour la sortie d'Arduino. En effet, il n'y a pas de résistance de limitation. Si la sortie **10** du microcontrôleur passe à l'état "1", le courant I base passe à environ 85 mA qui correspond à celui d'un court-circuit sur la broche du ATmega328. Deux solutions sont envisageables : Soit on n'utilise pas la possibilité de piloter le rétro éclairage, soit pour l'éteindre on force la broche **10** en sortie et immédiatement on y impose un état "0", ou pour l'allumer on la configure en entrée. Soit on modifie le module électronique. Pour ma part, comme montré sur la Fig.3, la broche qui va en **10** est coupée au ras du connecteur pour l'isoler. Piloter le rétro éclairage se fait par une diode **D** soudée sur le dessus du circuit imprimé entre l'ancienne broche **10** et l'E/S **13** la plus proche sur le coté gauche du connecteur. **Il faut impérativement utiliser une diode au germanium** dont le seuil de conduction est faible, de l'ordre de 0,3Vcc ou **T** ne sera pas bloqué. Quand la sortie **13** passe à l'état "1" la diode est bloquée. Quand **13** passe à l'état "0" elle draine les 3 mA qui traversent **R**.

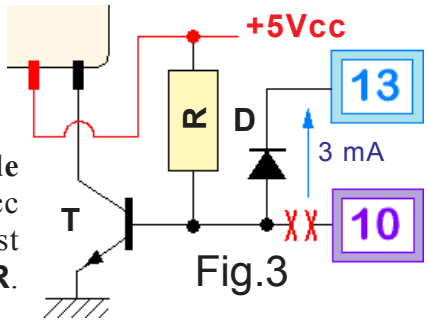


Fig.3

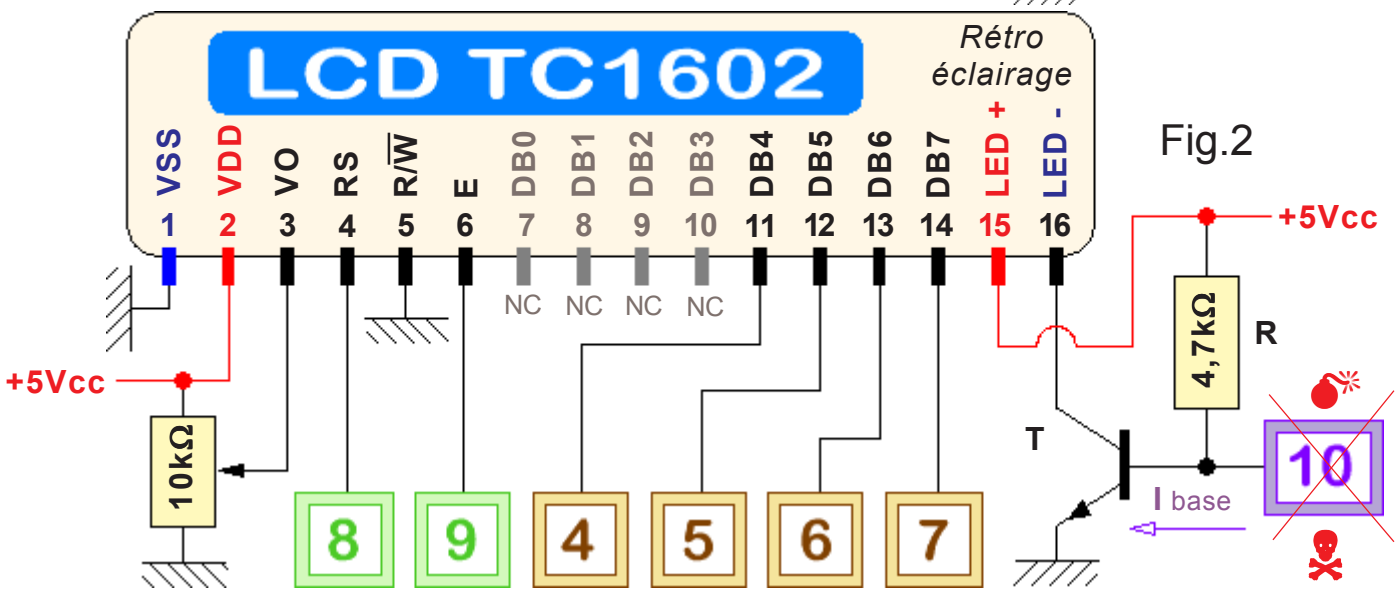


Fig.2

Exemple de programme :

```
// Test de la librairie "LiquidCrystal".
// Le rétro éclairage clignote à 1Hz.
#include <LiquidCrystal.h>
//*****************************
int Heures=0;
int Minutes=0;
int Secondes=0;
const int Pilotage_RETRO = 13;
boolean lumineux = true;

void setup()  {
   pinMode(Pilotage_RETRO, OUTPUT);
   digitalWrite(Pilotage_RETRO, HIGH);
   LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
   lcd.begin(16,2);
   lcd.setCursor(0,0);
   lcd.print("TEST DE COMPTAGE"); }

void loop() {
   delay(1000);
   LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
   lcd.begin(16,2);
   lcd.setCursor(0,0);
   if (lumineux) {digitalWrite(Pilotage_RETRO, LOW);}
     else {digitalWrite(Pilotage_RETRO, HIGH);} ;
   lcd.print(">>> HORLOGE. <<<");
    Secondes ++;
   if (Secondes > 59) {Minutes ++; Secondes = 0;};
   if (Minutes > 59) {Heures ++; Minutes = 0;};
   if (Heures > 23) {Heures = 0;};
   lcd.setCursor(0,1);
   if (Heures < 10) {lcd.print("0");};
   lcd.print(Heures); lcd.print(" H");
   lcd.setCursor(5,1);
   if (Minutes < 10) {lcd.print("0");};
   lcd.print(Minutes); lcd.print(" min");
   lcd.setCursor(12,1);
   if (Secondes < 10) {lcd.print("0");};
   lcd.print(Secondes); lcd.print(" S"); }
```

Ce petit programme se contente d'incrémenter une fois par seconde un compteur horaire affiché sur l'écran LCD du module électronique. Pour tester la modification donnée en Fig.3 et apportée au module le rétro éclairage est inversé à chaque seconde.

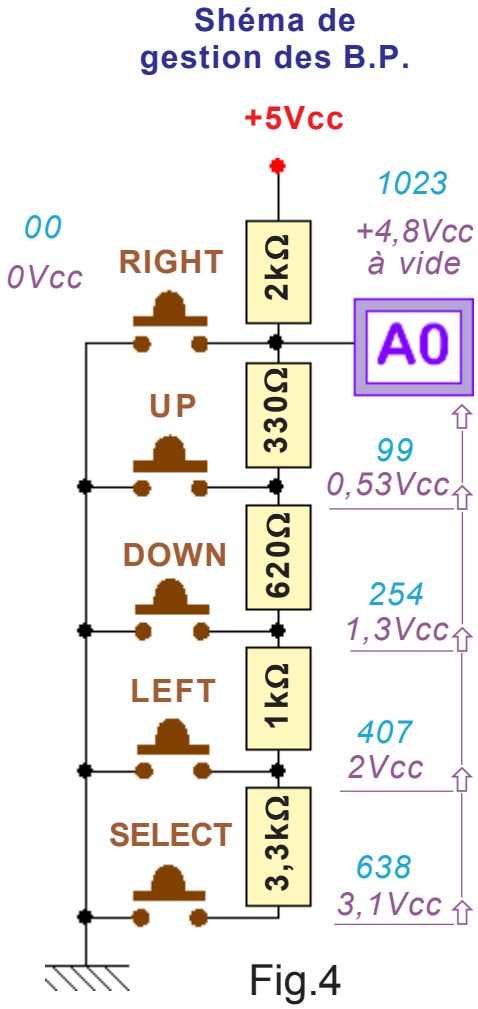
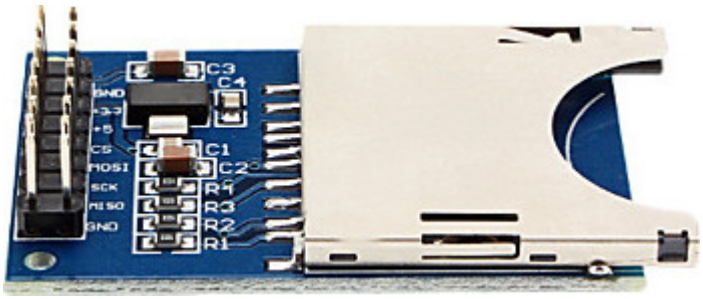


Fig.4

Représenté sur la Fig.4 ci-dessus, on trouve le schéma électrique adopté pour prendre en compte les cinq boutons poussoir disponibles sur le module électronique, qui reprend l'idée d'un diviseur de tension. Comme déjà adopté pour le mini-clavier 4 x 4, l'idée consiste à n'utiliser qu'une seule entrée analogique et à différencier les B.P. appuyés par une tension propre définie par une chaine de résistances. Sur la Fig.4 sont précisées en violet les tensions mesurées sur la broche **A0** en fonction de l'état d'activation des divers boutons. En bleu clair figurent les conversions analogiques / numériques qui en résultent, celle à prendre en compte pour déterminer les plages de valeurs de détermination. Le petit programme Test_LCD_sur_voie_serie.ino reprend l'exemple de programme listé ci-avant mais ne fait plus clignoter le rétro éclairage et retourne sur la voie série la valeur de la numérisation issue de l'entrée **A0**. C'est ce programme qui a fourni les valeurs portées sur la Fig.4 en bleu clair.

Autre exemple de programme associant un écran LCD et un capteur de température DS-18B20 :
http://web.ncf.ca/ch865/Arduino/Thermometre_ds18b20_LCD.ino



Comme pour tous les systèmes électroniques qui font l'objet d'une bibliothèque spécifique, la programmation pour un lecteur de carte SDRAM s'avère très simple, les routines de base étant déjà écrites. La Fig.1 présente le brochage de ce type de composants. Les cartes SD et les périphériques hôtes communiquent avec une interface série SPI par chaines de bits synchrones cadencés par un signal d'horloge fourni par le dispositif d'accueil. *(Trames de 48 bits)* La colonne Liaisons précise les branchements vers ARDUINO qui sont tributaires des routines incluses dans la bibliothèque **SD.h** qui

SD	mini	µSD	Nom	Fonction de la broche. *(Piste cuivrée)*	Liaisons
1	1	2	$\overline{CS}$	Sélection de la carte. *(Logique négative)*	10 > tampon
2	2	3	DI	Entrée série SPI. *(MOSI)*	11 > tampon
3	3		VSS	Masse. *(GND)*	GND
4	4	4	VDD	+ Alimentation. *(+3.3)*	*Non branché*
5	5	5	CLK	Horloge de la SPI. *(SCK)*	13 > tampon
6	6	6	VSS	Masse. *(GND)*	GND
7	7	7	DO	Données séries de la SPI. *(MISO)*	12
8	8	8	nIRQ	*(Non utilisé pour les cartes mémoire)*	
9	9	1	NC	Non utilisé.	

10 et 11 : N.C. *(Reserved)*

+5vcc d'Arduino va sur +5 du lecteur de carte

http://en.wikipedia.org/wiki/Secure_Digital_card

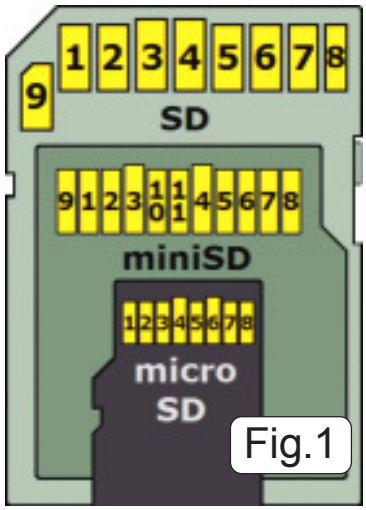


Fig.1

fait partie des librairies de base dans l'environnement IDE. Les E/S utilisées libèrent les deux lignes TX et RX ainsi qu'un maximum de sorties de type PWM. Le module lecteur de cartes SD est muni de deux rangées de broches mâles identiques, autorisant ainsi le branchement simultané de plusieurs périphériques de type SPI. Comme les cartes mémoire SD fonctionnent avec une tension nominale de 3,3Vcc le petit module intègre une régulateur local qui alimenté avec la broche +5Vcc d'Arduino pour fournir du 3,3Vcc stabilisé. Pour éviter toute perturbation de la mémoire "SD Card" à la mise sous tension, il est fortement recommandé de placer des résistances de forçage *(Pull-up)* vers le +3.3Vcc pour toutes les lignes. La Fig.2 présente le schéma électronique de la petite platine. On constate que seule la ligne **CS** n'a pas de résistance de forçage de 10kΩ. Les quatre condensateurs servent aux découplages sur l'entrée du régulateur par **C3** et sur la ligne d'alimentation 3,3Vcc avec **C1**, **C2** et **C4**. Le boitier métallique du "socket" de la carte SD est relié à la masse. La broche **WP** est un simple contact électrique qui s'établit avec la masse lorsqu'une petite carte mémoire est insérée dans le support. Il n'est pas recommandé de drainer trop de courant sur les broches notées **3.3** car le régulateur intégré AMS1117 n'est pas muni de radiateur.

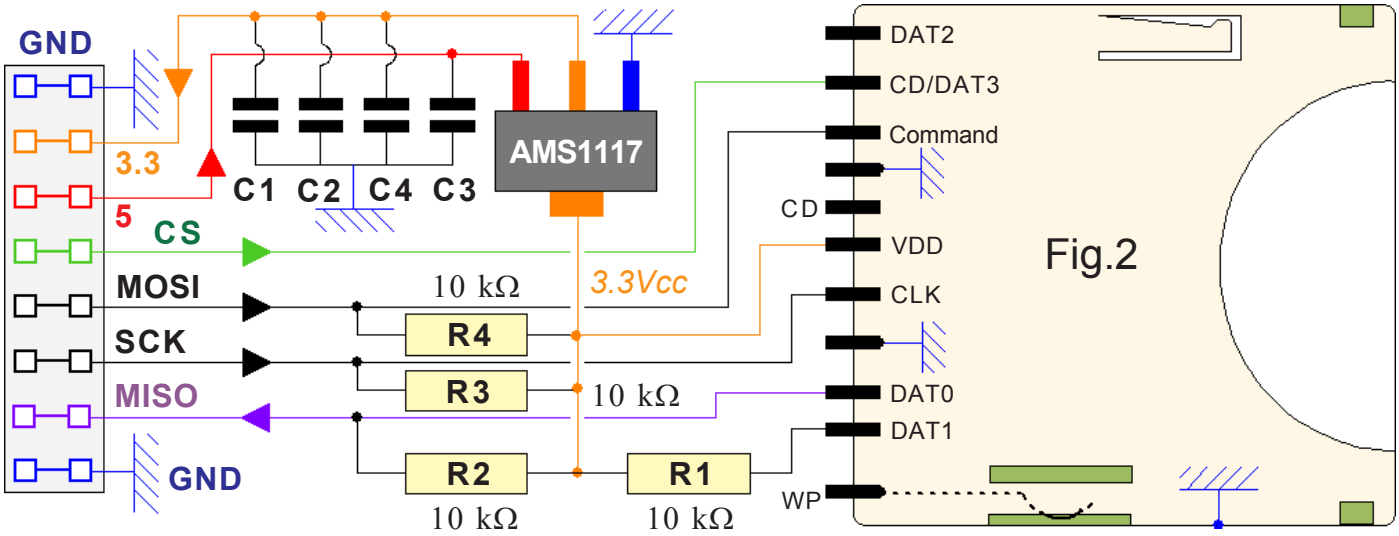


Fig.2

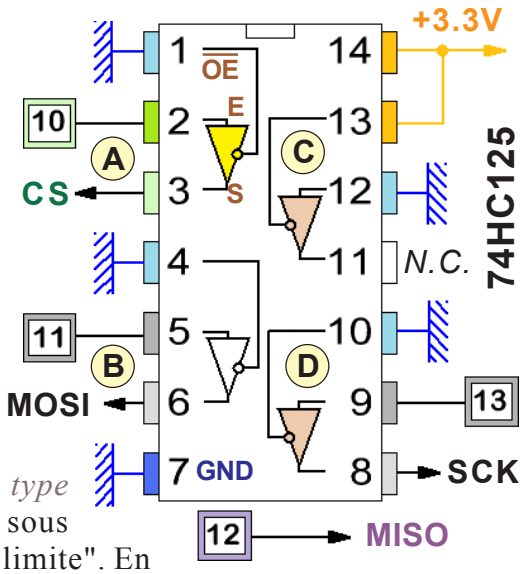
Problème d'interfaçage avec la carte ARDUINO.

Les sorties **SCK**, **MOSI** et **CS** issus de la carte Arduino fournissent des signaux dont la tension à l'état "1" fait pratiquement 5Vcc alors que les entrées des cartes SD ne sont prévues que pour 3,3Vcc. On peut brancher directement les sorties **10**, **11** et **13** sur le module électronique, le fonctionnement est correct. Mais ce n'est vraiment pas conseillé, car c'est forcément au détriment de la longévité des mémoires utilisées. Les résistances **R1** à **R4** assurent un niveau de 3,3Vcc quand les broches ne sont pas branchées ou au démarrage du processeur ATmega328. Ensuite, à l'état "1" des sorties **10**, **11** et **13** force la tension au maximum, soit 5,5Vcc. Il faut impérativement abaisser cette tension à celle préconisée par les fournisseurs de SD RAM. Trois solutions ont été expérimentées ici avec succès.

Solution n°1 :

Elle consiste à utiliser un opérateur logique 74HC125.C'est est un quadruple tampon qui recopie en sortie l'état en entrée avec pour tension à l'état "1"celle de l'alimentation si Output Enable est forcée à l'état logique "0". La sortie passe en mode isolé haute impédance si Output Enable est portée à l'état logique "1". La tension d'alimentation être comprise entre +2Vcc et +6Vcc pour le type HC. Des trois solutions c'est la plus rationnelle, les signaux fournis n'étant pas dégradés dans les transitoires de commutation. La Fig.3 présente le schéma adopté, le **+3.3V** étant branché sur la sortie régulée de la platine Arduino. Noter que l'opérateur **C** est en mode sortie isolée. Son entrée est mise à la masse pour éviter toute tension statique sur cette dernière. Ne disposant pas de 74HC125, j'ai utilisé un circuit intégré de type DM74125. *(Analogue au type 74HCT125)* Comme ce composant est prévu pour fonctionner sous +5Vcc et non à une tension plus faible, son comportement était "limite". En particulier les deux opérateurs **C** et **D** n'étaient pas capables de fournir en sortie pour la broche **CS** une tension suffisamment basse. **CS** a donc été branché sur l'opérateur **A**, le fonctionnement étant alors correct. Mais dans l'hypothèse d'une utilisation fiable avec toutes sortes de cartes SD, il faudrait utiliser un 74HC125.

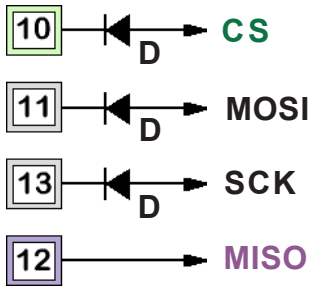
Fig.3



Solution n°2 :

C'est la plus simple à mettre en œuvre puisqu'elle ne comporte que trois diodes de type "faible courant". Les résistances de forçage présentes sur le petit circuit imprimé maintiennent l'état "1" sur les trois entrées de la carte mémoire. Elles empêchent le +5V des sorties **10**, **11** et **13** de se propager sur la carte SD. Par contre, quand des signaux "0" sont imposés, les diodes maintiennent une tension suffisamment faible pour être interprétée correctement par la mémoire. Seul inconvénient de ce montage : Sa fiabilité n'a pas été vérifiée avec un nombre suffisant de cartes mémoires.

Fig.4



Solution n°3 :

Comme montré sur la Fig.5 elle consiste à abaisser à 3,3Vccs le signal de 5Vcc émis par Arduino avec un simple diviseur de tensions constitué de deux résistances. Cette solution semble convenable pour les deux broches **MOSI** et **SCK**, mais pas pour la liaison avec **CS**. En effet, l'état "0" ne fournit pas une tension suffisamment faible pour qu'elle soit interprétée correctement. Pour les couples **R1** / **R2** on peut utiliser à loisir des valeurs de 1,8kΩ / 3,3kΩ ou la combinaison 2,7kΩ / 4,7kΩ qui donne également satisfaction. Pour la broche **CS** l'idée consiste, comme montré sur la Fig.6 à utiliser un transistor de commutation **T2**. Une résistance de forçage **R3** est ajoutée par mesure de sécurité mais elle n'est pas indispensable du tout. Sa valeur n'est vraiment pas critique et peut être comprise entre 4,7kΩ et 47kΩ. Le transistor pour petit signaux **T2** inversant le signal logique, pour rétablir la "parité il faut le faire précéder d'un autre inverseur. Dans ce but on va intercaler un autre transistor de commutation pour petits signaux **T1**. Tout transistor NPN de

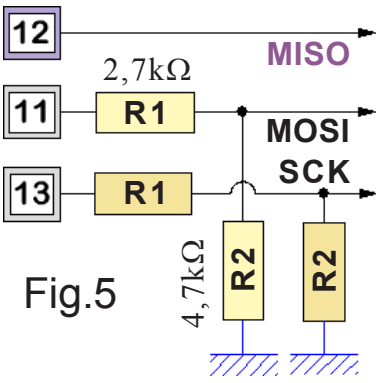
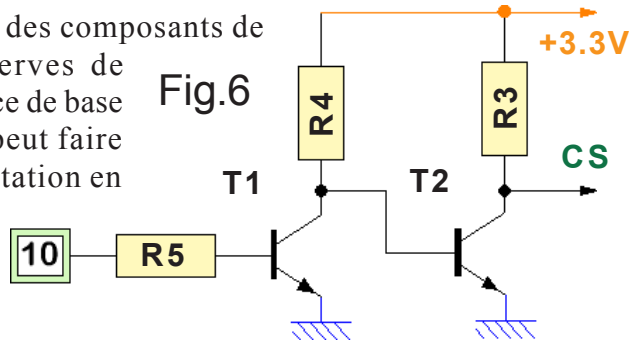


Fig.5

faible puissance convient. Pour ma part j'ai testé avec des composants de récupération CII267Q qui "encombrent" mes réserves de composants dédiés aux loisirs électroniques. La résistance de base **R4** qui du 3,3Vcc permet de saturer le transistor **T2** peut faire entre 10kΩ et 47kΩ. Quand à **R5** la résistance de limitation en courant pour **T1**, on peut adopter entre 22kΩ et 47kΩ. Comme pour le circuit de la solution n°2, ce montage a donné satisfaction, mais il n'a pas été testé avec un nombre suffisant de mémoire pour en vérifier la fiabilité.

Fig.6



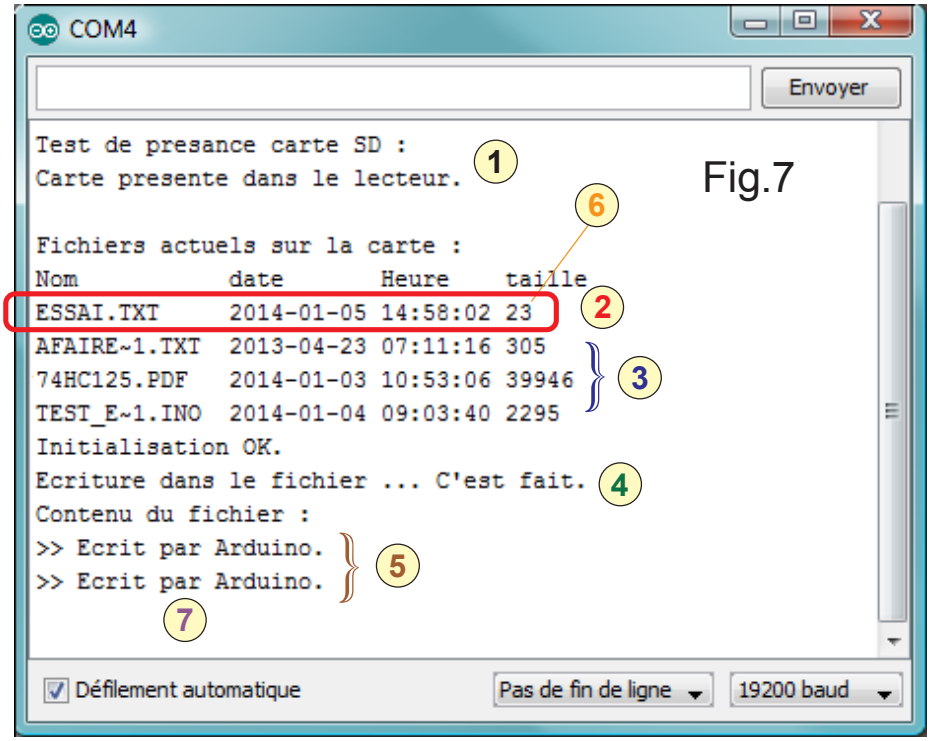
Le plus simple de tous est celui avec les trois diodes, mais s'il ne convient pas, il serait préférable d'utiliser la solution n°1. Ce n'est que par manque de disponibilité du 74HC125 qu'il serait envisageable de mettre alors en œuvre la solution avec deux transistors.

Un exemple d'utilisation du SHEILD pour lecteur de mémoire "SD Card".

Deux bibliothèques sont disponibles pour se servir des cartes mémoire SD avec un Arduino. Les deux font appel aux mêmes broches d'E/S binaires. La bibliothèque SD.h est fournie en standard dans l'environnement IDE. La librairie SD.h permet de lire et d'écrire les cartes mémoires SD. Elle est basée sur la bibliothèque sdfatlib.h de ***William Greiman*** qui donne accès à davantage de fonctions, notamment des fonctions d'accès aux informations de la carte SD. Ceci étant précisé, SD.h est déjà largement suffisante pour couvrir des applications très étoffées. *(Aller voir sur : http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.LibrairieSD pour de plus amples informations)* Le petit programme donné en page 25 permet de tester le petit circuit imprimé lecteur de carte SD en utilisant la "ligne série USB" *(Voir Fig.7)* pour afficher les résultats des tests.

Sur un RESET ce petit logiciel élémentaire commence par tester la présence d'une carte mémoire sur le "socket". Si le test confirme la présence d'une mémoire SD, après compte rendu en **1** il y a lecture de son contenu puis affichage sur le canal série de la liste des fichiers en **3**, y compris ceux qui sont dans des sous-répertoires. Pour chaque fichier son nom, sa date, son heure et sa taille sont précisés. Puis il y a un test d'écriture sur la carte, d'un petit texte dans le fichier de nom ESSAI.TXT qui sera créé lors d'une première lecture. ATTENTION : La création du fichier et l'écriture dans ce dernier se font même si la carte est protégée en écriture par son petit curseur latéral. La création ou l'ouverture du fichier ESSAI.TXT est confirmée en **4**. Puis il y a lecture du contenu de ce fichier texte et affichage en **5**.
La taille de 23 caractères indiquée en **6** peut sembler contradictoire avec le contenu montré en **5** qui fait en réalité le double en nombre de caractères. C'est normal puisque l'écriture de la phrase **7** qui fait 23 caractères se fait après lecture et exploitation de la directory inscrite dans la carte mémoire SD.
Chaque RESET provoque la lecture de la carte mémoire, affiche son contenu puis ajoute une ligne de plus dans le fichier ESSAI.TXT dont on peut corroborer la taille par un calcul simple :
Chaque phrase fait 21 caractères plus 2 pour le CR et pour le LF. Si par exemple **6** indique la valeur 115, le fichier comporte alors 115 divisé par 23 soit 5 fois cette phrase avant le test d'écriture. Il y aura donc six fois le texte de test lors de la lecture du contenu.

Fig.7



Exemple de programme :

```
/* Expérimentations avec le lecteur de carte SD.
 CS -> Broche 10.      MOSI -> Broche 11.
 MISO -> Broche 12.    SCK -> Broche 13.     */

#include <SD.h> // Bibliothèque présente par défaut dans l'IDE.

const int chipSelect = 10 ;  //Broche 10 reliée au CS du module SD.
Sd2Card card ;
SdVolume volume ;
SdFile root ;
File myFile ;

void setup() {
  Serial.begin(19200);  while (!Serial) { ; }
  Serial.println("\nTest de presance carte SD : ");
  pinMode(10, OUTPUT);
  // Vérification de la présence carte.
  if (!card.init(SPI_HALF_SPEED, chipSelect))
    { Serial.println("Pas de carte dans le lecteur."); return; }
  else { Serial.println("Carte presente dans le lecteur."); }
  // Vérification des données.
  if (!volume.init(card)) { Serial.println("La carte n'est pas formatee."); return; }
  // Afficher la liste des fichiers présents.
  Serial.println("\nFichiers actuels sur la carte :");
  Serial.println("Nom        date      Heure    taille    ");
  root.openRoot(volume);
  root.ls(LS_R | LS_DATE | LS_SIZE);
  // On se prépare à intervenir dans les fichiers.
  if (!SD.begin(chipSelect)) { Serial.println("Echec de l'initialisation."); return; }
  Serial.println("Initialisation OK.");
  // Ouverture du fichier pour écriture. (Il sera créé s'il n'existait pas)
  myFile = SD.open("ESSAI.TXT", FILE_WRITE);
  // Si l'ouverture du fichier a fonctionné, on va y ajouter du texte.
  if (myFile)
    { Serial.println("Ecriture dans le fichier ... ");
      myFile.println(">> Ecrit par Arduino.");  // Ajouter une phrase dans le fichier.
      myFile.close();  // Fermeture du fichier.
      Serial.println("C'est fait."); }
  else { // si l'ouverture du fichier a échoué on le précise.
        Serial.println("Impossible d'ouvrir le fichier pour l'ecriture"); }
  // On ouvre le fichier, et on vérifie son contenu.
  myFile = SD.open("ESSAI.TXT");
  if (myFile) { Serial.println("Contenu du fichier :");
          // Lire le fichier jusqu'à ce qu'il n'y a rien d'autre dans ce dernier.
          while (myFile.available()) {Serial.write(myFile.read()); };
          myFile.close(); }  // Fermeture du fichier.
  else { Serial.println ("Impossible d'ouvrir le fichier pour sa lecture"); }
          // Si l'ouverture du fichier a échoué, le préciser.
}
void loop(void) {  }
```

NOTE : "\n" avant le texte dans un Serial.println provoque un retour à la ligne **avant** d'afficher le texte qui suit.

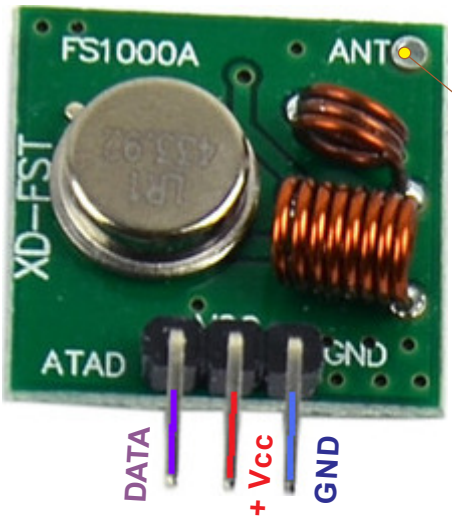
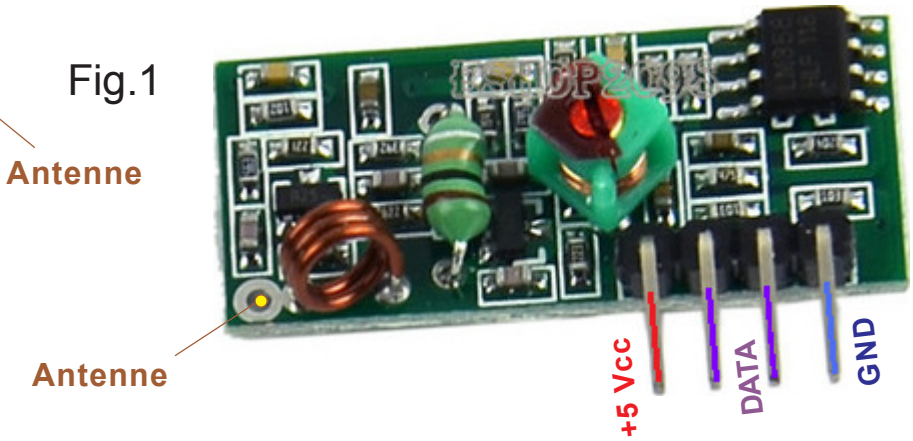


Fig.1



Ces deux petits modules de très petites dimensions ne sont strictement pas spécifiques au système de développement Arduino, mais ils peuvent servir pour tout ensemble électronique devant faire dialoguer deux entités à distance, quelles que soient leurs technologies. Leur fréquence de fonctionnement est de 433.92MHz, mais il existe aussi des paires accordées sur 315 MHz. La dispersion de fréquence est inférieure à 150KHz. L'émetteur peut être alimenté entre 3,5Vcc et 12Vcc, sa portée étant d'autant plus grande qu'il est alimenté sous une tension **Vcc** plus élevée.

Caractéristiques techniques du transmetteur :

- Alimentation : 3,5Vcc à 12 Vcc.
- Consommation : 20mA à 28mA.
- Portée : Environ 40m en intérieur, et 100m à l'air libre. *(Moyenne avec un transmetteur sous +5Vcc)*
- Température de fonctionnement : Entre -10°C et +70°C.
- Mode de résonance : SAW. *(Sound Wave Resonance)*
- Mode de modulation : ASK / OOK.
- Puissance VHF : 10mW à 25mW. *(25mW sous 12Vcc)*
- Débit maximal de données : 10Kb/s.
- Débit de transfert 4Kb/s.
- Antenne extérieure de 25cm.

Caractéristiques techniques du récepteur :

- Alimentation : 5Vcc avec un courant statique de 4mA.
- Température de fonctionnement : Entre -10°C et +70°C.
- Bande passante : 2MHz.
- Récepteur simple à superréaction.
- Antenne de réception extérieure de 32cm droite ou enroulée en spirale.

Modulation ASK/OOK.

Comme montré en Fig.2, initialement l'ASK *(Amplitude Shift Keying)* est une modulation en amplitude de la fréquence porteuse. L'alternative OOK *(On Off Keying)* est une forme binaire de la modulation d'amplitude. Elle consiste à couper la porteuse lors des états "0" et à transmettre en VHF lors des états "1". L'ASK impose une consommation d'énergie permanente, alors que le procédé OOK rend l'émetteur moins gourmand en énergie. Le débit de données en mode OOK est limité par le temps de démarrage de l'oscillateur. En OOK les perturbations en réception sont plus importantes qu'en mode ASK.

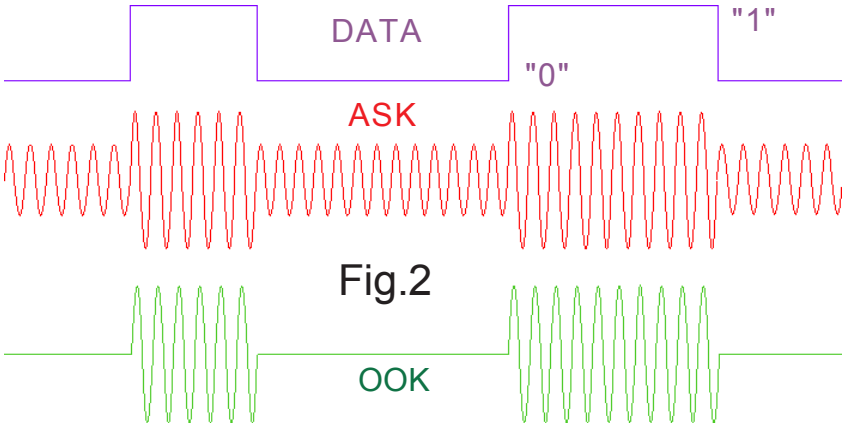


Fig.2

TESTS DE BASE.

Programme pour étudier l'émetteur :

TEST_TX_433MHz.ino

```
/* Test du petit module 433MHz */

const int LED = 13;  // Broche de la DEL U.C.
const byte Impulsion = 6;  // Sortie binaire 6 utilisée.

void setup()
{ pinMode(LED, OUTPUT);      // LED est une sortie.
  digitalWrite(LED, LOW); }  // Éteint la LED de l'U.C.

void loop() {
   digitalWrite(LED, LOW);   // Éteint la LED de l'U.C.
   for (int i=0; i<1001; i++)// Générer une pulse courte durant 1S.
     {digitalWrite(Impulsion, HIGH);
      delayMicroseconds(250); // Etat "1" durant 250µS.
      digitalWrite(Impulsion, LOW);
      delayMicroseconds(750);} // Etat "0" pour compléter à 1 mS.
   digitalWrite(LED, HIGH);   // Allume la LED de l'U.C.
   for (int i=0; i<1001; i++)// Générer une pulse longue durant 1S.
     {digitalWrite(Impulsion, HIGH);
      delayMicroseconds(450); // Etat "1" durant 450µS.
      digitalWrite(Impulsion, LOW);
      delayMicroseconds(550); }; } // Etat "0" pour compléter à 1 mS.
```

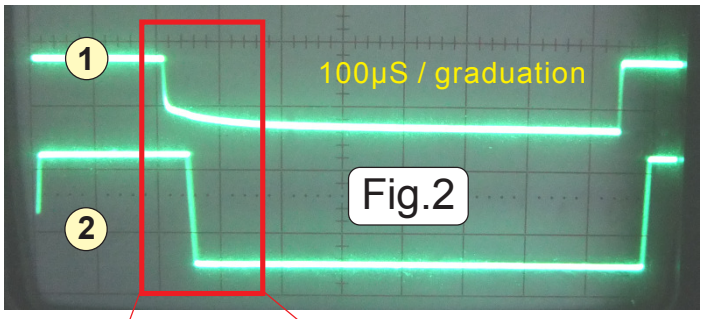
Ce petit programme génère comme montré sur la Fig.2 des impulsions de 250 µs suivies d'un état logique zéro pour aboutir à un cycle d'une milliseconde. Il y a génération de ces créneaux courts durant environ une seconde. Puis, toujours sur une seconde il y a création des impulsions longues de la Fig.3 d'une durée de 450 µS.

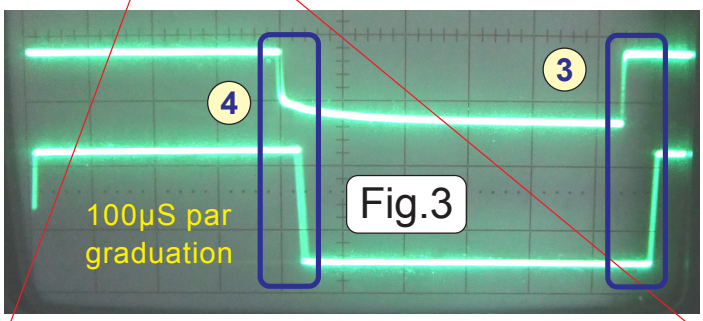
Programme sur l'émetteur pour utiliser l'oscilloscope :

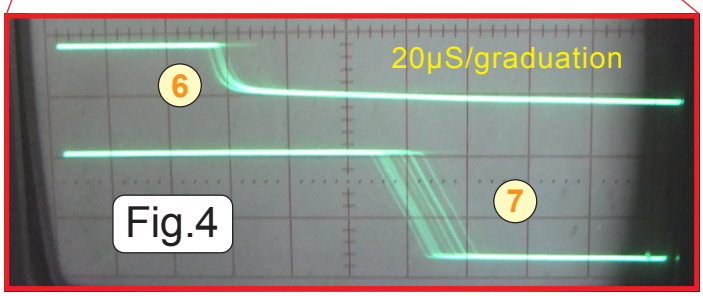
TEST_TX_433MHz_oscilloscope.ino

```
/* Test du petit module 433MHz */

const byte Impulsion = 6;  // Broche PWM 6 utilisée.

void setup() {}

void loop()
  { digitalWrite(Impulsion, HIGH);
    delayMicroseconds(250);
    digitalWrite(Impulsion, LOW);
    delayMicroseconds(750);}
```

La Fig.4 présente le signal observé avec l'oscilloscope en balayage sur mode retardé pour "étaler" le signal descendant. Pour rendre la visualisation plus stable, seule l'impulsion courte de 250 µS est générée sans fin par le programme simplifié.







Premiers constats :

Sur les trois figures la courbe du haut **1** correspond au signal logique envoyé par Arduino au petit émetteur 433MHz. La courbe du bas **2** est relative au signal restitué par le petit récepteur sur sa sortie. On observe qu'entre le signal de pilotage **1** de transition montante **3** et celui issu de récepteur **2** il y a un retard d'environ 60µS. Un retard analogue se produit également lors de la transition descendante comme visualisé en **4**.

On constate que les signaux de pilotage **6** sont affectés d'un certain jitter. *(JITTER : Faible variation de phase d'un signal électrique)*

On retrouve cette instabilité sur le signal **7** restitué par le récepteur. On peut noter également qu'un signal binaire de fréquence relativement élevée et assez "aléatoire" sort du récepteur quand ce dernier ne reçoit pas un signal cohérent issu de l'émetteur.

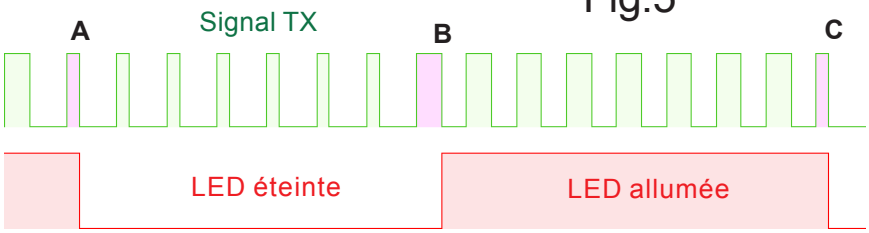
Programme pour tester le récepteur :

Ce petit programme est assez élémentaire. Il consiste à maintenir allumée la LED d'essai de la platine Arduino lorsque le signal envoyé par l'émetteur génère des impulsions longues de 450µS, et à éteindre cette dernière lorsque le pilotage est effectué avec les impulsions courtes de 250µS. Ces durées à l'état logique "1" sont suffisamment différentes pour facilement les différencier.

TEST_RX_433MHz.ino

```
/* Test du petit module de réception 433 MHz */
const int LED = 13;  // Broche de la DEL U.C.
const byte EntreePulse = 2;  // Broche 2 pour mesurer T.
int Periode;

void setup(){
  pinMode(LED, OUTPUT);       // LED est une sortie.
  pinMode(EntreePulse, INPUT); // EtreePulse est une entrée.
  digitalWrite(LED, LOW);     // Éteint la LED de l'U.C.
  Serial.begin(19200); }

void loop() {
  Serial.println(pulseIn(EntreePulse,HIGH));
  if (pulseIn(EntreePulse,HIGH) < 350)
    {digitalWrite(LED, LOW);}
  else {digitalWrite(LED, HIGH);} }
```

Fig.5



COMMENTAIRES :

L'instruction pulseIn(EntréeBINAIRE,ETAT) mesure la durée d'une impulsion dont on précise l'entrée binaire utilisée, et s'il faut mesurer durant l'état "1" ou durant l'état "0". Cette instruction retourne un entier dont la valeur exprime la durée mesurée en µS. Pour expérimenter cette fonction, l'instruction **Serial**.println(pulseIn(EntreePulse,HIGH)); est ajoutée au programme pour avoir les valeurs retournées sur la voie série, le programme TEST_TX_433MHz.ino étant activé sur l'émetteur. Les valeurs affichées sur la voie série oscillent entre 226 à 271 pour l'impulsion courte, et entre 440 à 447 pour l'impulsion longue. On retrouve les fluctuations résultant du phénomène de JITTER. Pour différencier les impulsions courtes des longues, on teste à la valeur moyenne entre 250 et 450 soit 350. On pare ainsi au maximum les fluctuations résultant du "jitter". La première impulsion inférieure à 350µS en **A** de la Fig.5 éteint la LED. Puis la première qui sera supérieure à 350 en **B** allume la LED. Enfin le cycle recommence en **C**, ainsi la LED d'essai du système Arduino clignote à environ un demi-Hertz.

Problèmes de réception :

Le petit récepteur 433MHz est particulièrement sensible aux parasites hertziens de toutes natures. De plus, cette fréquence est affectée à une foule de dispositifs en tous genres. Par exemple, ici la petite station météo fonctionne sur cette même longueur d'onde. Du coup, les divers capteurs de cette station météo ainsi que la base qui les interroge provoquent régulièrement des parasites sur le récepteur d'Arduino. La réciproque est vraie. Le transmetteur d'Arduino perturbe la station météo qui perd les informations issues de certains de ces capteurs. Du reste le sujet de discution sur le forum suivant montre bien que ce problème de parasitage est relativement général :

http://fr.openclassrooms.com/forum/sujet/reception-en-433mhz-composant-grille-arduino

• Enfin les liens suivants conduisent à des exemples de transmission de données :

Didacticiel très pédagogique et en V.F. avec utilisation d'une sonde de température DS18B20 :
http://forum.pcinpact.com/topic/165594-raspberry-pi-fabriquons-des-trucs/page-5#entry2754679

Autre exemple d'utilisation avec échange de données :
http://www.pjrc.com/teensy/td_libs_VirtualWire.html

Représenté en Fig.1 ce module intègre une horloge temps réel pilotée par quartz, qui compte les heures, les minutes, les secondes. Le circuit intégré gère les jours, les semaines, la date du mois, et les années. La correction pour les années bissextiles est valable jusqu'en 2099. ***Une mémoire RAM locale à usage général de 31 octets est disponible pour l'utilisateur.***

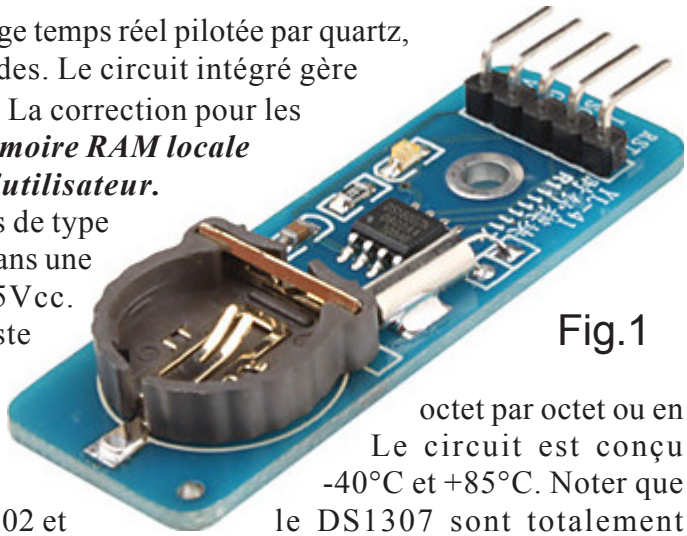


Fig.1

Les échanges de données se font par un bus à trois fils de type SPI. Compatible TTL le circuit DS-1302 fonctionne dans une plage d'alimentation comprise entre 2Vcc et 5,5Vcc. Alimenté sous 2Vcc il consomme un courant qui reste très inférieur à 0,3µA. Les données de l'horloge, du calendrier ou de la RAM peuvent être lues ou écrites octet par octet ou en mode rafales comportant un certain nombre d'octets. Le circuit est conçu pour une ambiance industrielle et fonctionne entre -40°C et +85°C. Noter que l'interface et les protocoles de dialogue entre le DS1302 et le DS1307 sont totalement différents. Une broche spécifique est prévue pour le raccordement à une pile de secours en alimentation. Comme montré sur la Fig.2 le petit module est complété par une LED de témoin d'alimentation +Vcc. Les Entrées / Sorties sont choisies pour pouvoir utiliser simultanément le SHIELD de l'afficheur LCD.

Protocoles de dialogue avec le DS1302 :

Contrairement à tous les exemples trouvés sur Internet, se contenter de réunir les trois broches de pilotage aux E/S d'Arduino n'est pas suffisant. Le fonctionnement ne devient correct que si les trois lignes de dialogue de la SPI sont drainées à l'état "1" par trois résistances de 10KΩ au +Vcc.

RESET : Cette broche nommée RST agit comme un "chip select". Pour écrire ou lire dans la "puce" du DS1302 cette ligne doit être maintenue à l'état "1" jusqu'à la fin de la lecture / écriture. Lorsque cette ligne est à l'état"0" toutes les lectures et écritures sur le DS1302 sont désactivées.

I/O : C'est la ligne de données bidirectionnelle du protocole SPI. Lorsque l'entrée RESET est haute, elle devient active. Les données série sont échangées par mots de 8 bits. Une écriture ou cycle de lecture consiste à envoyer un octet de commande, suivi d'une lecture ou d'écriture d'octets. L'octet de commande indique l'adresse à lire ou à écrire.

SCLK : Il s'agit de l'horloge de synchronisation du système. Les spécifications sur la fiche de données signalent une fréquence d'horloge de valeur maximale de 2 MHz sous une alimentation de 5Vcc. Mais certains internautes précisent qu'ils n'ont pas pu atteindre cette haute performance. La fréquence maximale d'échantillonnage semble se situer aux environs de 500 kHz.

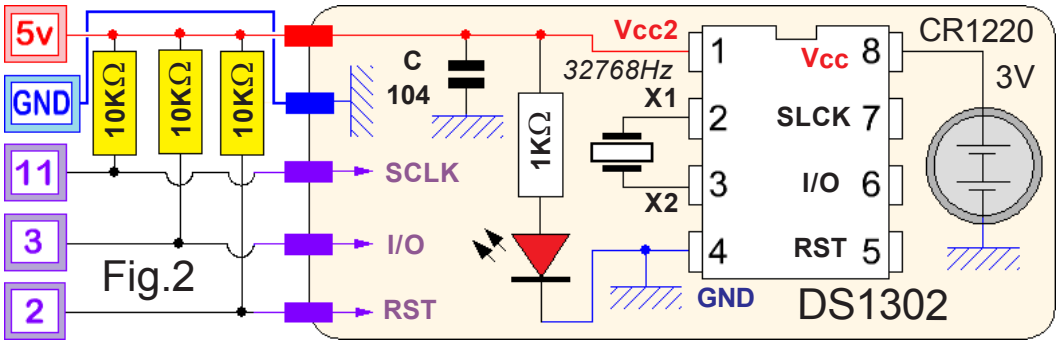


Fig.2

(Connecteur vu de dessous)

BIBLIOTHÈQUES POUR LE DS1302 :

Deux bibliothèques sont disponibles. La première est celle de base et permet d'initialiser le circuit intégré et fournit les routines de base pour effectuer les dialogues. La deuxième plus complète intègre diverses routines facilitant les échanges de données. Initialement les deux ont été installées, mais vraisemblablement la plus complète est peut être suffisante, ce qui reste à vérifier car elle fait appel à la première. Les deux étant nommées DS1302.h j'ai renommée la plus complète DS1302bis.h pour les différencier. Quand on installe DS1302bis.h, elle semble être renommée DS1302.h dans <user>.

Librairie de base avec exemple d'initialisation : http://blog.trendsmix.com/?attachment_id=148

Librairie avec quatre exemples d'utilisation du DS1302 dont un pour l'exploitation de la RAM utilisateur : http://www.henningkarlsen.com/electronics/library.php?id=5

Exemple de programme :

```
// Exemple d'utilisation de l'horloge/Calendrier DS1302.
// Le module est sauvegardé par batterie.
// Les lignes notées >>> sont à valider si l'on veut réinitialiser l'horloge.
// En l'état ce programme se contente de lister les valeurs sur la ligne série.

#include <stdio.h>
#include <string.h>
#include <DS1302.h>
```

Ce petit programme donné en exemple affiche en boucle sur la voie série la date et l'heure. L'affichage est rafraichi une fois par seconde. Si on désire initialiser la date et l'heure il suffit de mettre à jour la ligne @ et de valider les quatre lignes bleu clair >>>.

```
// Dédinir les E/S sur Arduino.
uint8_t LigneRST  = 2;
uint8_t LigneIO   = 3;
uint8_t LigneSCLK = 11;
```

} Compatible avec l'utilisation simultanée du module LCD.

```
// Créer des "registres tampon".
char TamponDeDonnees[50]; char JOUR[10];

// Créer "l'objet" DS1302.
DS1302 rtc(LigneRST, LigneIO, LigneSCLK);

void setup() { Serial.begin(19200); }
// ========== Bloc à inclure si mise à l'heure désirée. ==========
/* Initialiser une nouvelle fois le circuit intégré en désactivant
   la protection en écriture et le "drapeau d'arrêt de l'horloge.
   Ces directives ne doivent pas toujours être appelées. Voir en
   détail la fiche d'exploitation du DS1302. */
// >>> rtc.write_protect(false);
// >>> rtc.halt(false);
  // Définir la date et l'heure à transmettre au DS1302.
  // Ici le 10 Janvier 2014 à 10h 22 min 40 secondes.
// >>> Time t(2014, 1, 10, 10, 22, 40, 5);  @
  // Transmettre ces informations au circuit intégré.
// >>> rtc.time(t);
//================================================================

void loop() { AfficheLeTemps(); delay(1000); }

void AfficheLeTemps() { // Télécharger la date et l'heure préservée dans le DS1302.
  Time t = rtc.time();
  // Nommer les jours de la semaine.
  memset(JOUR, 0, sizeof(JOUR));  // Effacer le tampon pour le jour.
  switch (t.day) {
    case 1: strcpy(JOUR, "Lundi"); break;
    case 2: strcpy(JOUR, "Mardi"); break;
    case 3: strcpy(JOUR, "Mercredi"); break;
    case 4: strcpy(JOUR, "Jeudi"); break;
    case 5: strcpy(JOUR, "Vendredi"); break;
    case 6: strcpy(JOUR, "Samedi"); break;
    case 7: strcpy(JOUR, "Dimanche"); break; }
 // Formater la date et l'heure et l'introduire dans le tampon des données.
  snprintf(TamponDeDonnees, sizeof(TamponDeDonnees),
  "%s %02d-%02d-%04d %02d:%02d:%02d", JOUR, t.date, t.mon, t.yr, t.hr, t.min, t.sec);
  // Affiche la chaine des données sur la ligne série.
  Serial.println(TamponDeDonnees); }
```

@ **IMPORTANT :** Quand on initialise une date, le DS1302 ne peut pas savoir quel est le JOUR, il faut le lui préciser avec le dernier paramètre. 1 pour Lundi, 2 pour mardi, 3 pour Mercredi etc.

Un petit projet complet avec le DS1302.

Sans prétention, ce petit programme permet de réaliser une horloge / Calendrier avec le module d'affichage LCD. Pour simplifier la programmation, la date et l'heure sont initialisées une première fois. Puis le programme est modifié pour passer en remarque les lignes d'initialisation, et le microcontrôleur est rechargé. C'est le DS1302 qui est sauvegardé avec une pile lithium, qui se charge de conserver les données. Ainsi sur coupure courant et redémarrage, l'heure et la date ne sont pas perdues. Sur un RESET même comportement. Comme une dérive de l'heure n'est pas exclue, les boutons LEFT, RIGHT permettent d'incrémenter ou de décrémenter les secondes. UP et DOWN sont utilisées pour ajuster de la même façon les minutes. Enfin le bouton SELECT permet d'allumer ou d'éteindre le rétro éclairage. Comme la ligne série n'est pas utilisée, l'E/S binaire n°0 change d'état à chaque affichage dans la boucle de base. Elle peut servir a allumer une LED ou à brancher un périodemètre pour affiner les affichages à pratiquement une seconde entre chaque rafraichissement. Le programme est listé dans les pages 32 et 33.
Le matériel reprend strictement sans changement celui de la Fig.2 donnée en page 20. On se contente d'ajouter le module horloge comme décrit en Fig.2 de la page 29. Eventuellement une LED avec sa résistance de limitation de courant est insérée entre le +5Vcc et la sortie binaire n°0. Naturellement, pour pouvoir facilement se brancher sur Arduino "à travers" le Shield LCD, il faut sur le dessus de ce dernier souder des petits connecteurs sur les pastilles prévues à cet effet.

STRUCTURE DU PROGRAMME.

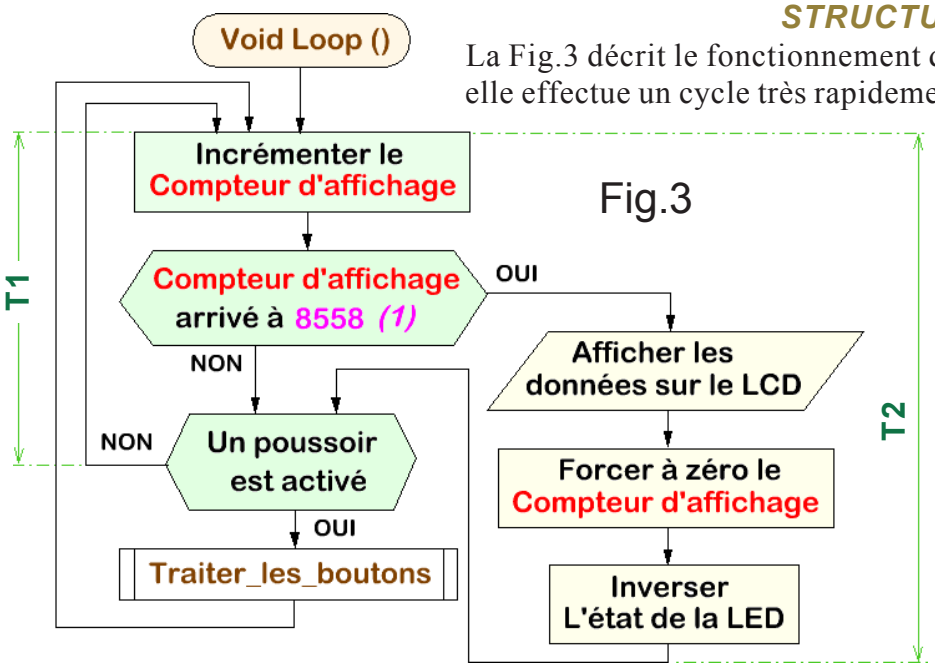


Fig.3

La Fig.3 décrit le fonctionnement de la boucle de base. Globalement elle effectue un cycle très rapidement, la prise en compte des boutons poussoir est immédiate. La valeur **(1)** a été affinée pour que le passage dans la branche jaune se fasse une fois par seconde le plus précisément possible. Avec un test pour une valeur > 8557 on arrive à une boucle de période **T1** très stable de 1.000027 secondes en moyenne. La boucle verte est donc parcourue en 1168µS. Quand il y a affichage, on passe par la séquence jaune, la durée devenant **T2**, avec un changement d'environ 40 µS. On constate que l'ajustement précis du Compteur d'affichage permet un rafraichissement extrêmement précis sur le LCD, à exactement une fois par seconde. Pour affiner cette valeur, une première expérience avec 10000 a été expérimentée. Une mesure au périodemètre suivie d'un calcul de proportionnalité ont permis de se rapprocher de la valeur optimale. Trois essais supplémentaires ont conduit à l'approximation la plus précise de 8557 comme valeur de test dans le PGM. Une mention particulière s'impose pour justifier la gestion des boutons poussoirs. La Fig.4 donne en bleu clair les valeurs de numérisation sur l'entrée analogique A0. Pour parer totalement les fluctuations inévitables de la valeur restituée par le convertisseur analogique / numérique, la technique classique consiste à établir les zones de détermination en plaçant les seuils de discrimination à moitié "distance" entre des valeurs mesurées voisines. Ces valeurs moyennes sont repérées en violet sur la Fig.4 et se retrouvent dans les tests effectués par le programme.

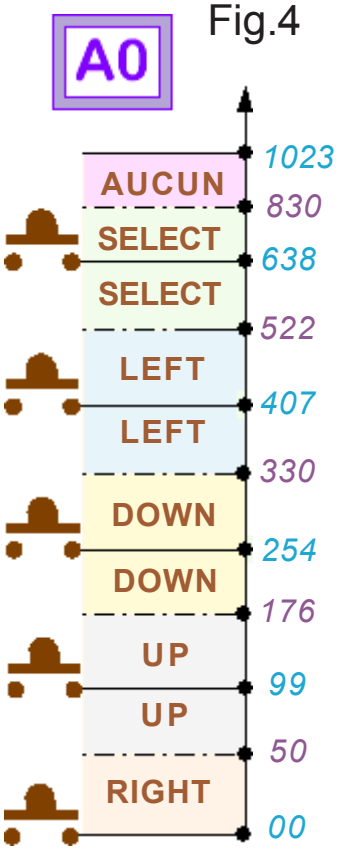


Fig.4

Programme pour le projet d'horloge / Calendrier :

```
// Exemple d'utilisation de l'horloge/Calendrier DS1302.
// Le module est sauvegardé par batterie.
// Les lignes notées >>> sont à valider si l'on veut réinitialiser l'horloge.
// Ce programme affiche la date et l'heure sur le LCD.
// Les boutons permettent d'allumer ou d'éteindre le rétroé clairage et
// d'incrémenter ou de décrémenter les minutes et les secondes.

#include <stdio.h>
#include <string.h>
#include <DS1302.h>
#include <LiquidCrystal.h>

uint8_t LigneRST  = 2;  // Dédinir les E/S sur Arduino.
uint8_t LigneIO   = 3;
uint8_t LigneSCLK = 11;

const byte Pilotage_RETRO = 13;
const byte LED = 1;   // Une LED peut être branchée sur la sortie 1.
const byte Entree_mesuree = 0; // Entrée analogique 0 utilisée.
int CNA; // Mesure analogique retournée par le CNA.
boolean Lumineux = true;
boolean DEL_lumineuse = false;
int Compteur_Affichage = 0;

char JOUR[10]; // Créer des "registres tampon".

// Creer "l'objet" DS1302.
DS1302 rtc(LigneRST, LigneIO, LigneSCLK);

void setup() {
// ========== Bloc à inclure si mise à l'heure désirée. ==========
/* Initialiser une nouvelle fois le circuit intégré en désactivant
   la protection en écriture et le "drapeau d'arrêt de l'horloge.
   Ces directives ne doivent pas toujours être appelées. Voir en
   détail la fiche d'exploitation du DS1302. */
// >>> rtc.write_protect(false);
// >>> rtc.halt(false);
  // Définir la date et l'heure à transmettre au DS1302.
  // Ici le Dimanche 12 Janvier 2014 à 16h 47 min 40 secondes.
// >>> Time t(2014, 1, 12, 16, 47, 40, 7);
  // Dernier paramètre 7 : Pour Dimanche.
  // transmettre ces informations au cisduit intégré.
// >>> rtc.time(t);
//================================================================
   pinMode(Pilotage_RETRO, OUTPUT);
   digitalWrite(Pilotage_RETRO, HIGH);
   pinMode(LED, OUTPUT);
   digitalWrite(LED, LOW); }

void loop() {
  Compteur_Affichage = Compteur_Affichage + 1;
  if (Compteur_Affichage > 8557)
  { AfficheLeTemps(); Compteur_Affichage = 0;
    if (DEL_lumineuse) {digitalWrite(LED,LOW); DEL_lumineuse = false;}
              else {digitalWrite(LED,HIGH); DEL_lumineuse = true;}; }
```

Initialement c'est la broche A0 qui avait été utilisée. Le programme fonctionnait. mais quand on désirait le "téléverser", RX était perturbé et le transfert ne se faisait pas

IMPORTANT : Il faut impérativement débrancher A0, qund on Télécharge un programmme si elle est utilisée en sortie ou la ligne série est perturbée et le code "téléversé" n'est pas transmis.

Pour initialiser la date et l'heure du DS1302 il suffit de changer les paramètres roses et de valider les quatre lignes bleu clair.

NOTE : Le dessin du petit circuit imprimé qui concrétise les branchements de la Fig.2 est donné dans le petit livret sur les circuits imprimés .

```
  // Traiter les boutons poussoir.
  CNA = analogRead(Entree_mesuree);
  if (CNA < 830) { Traiter_les_boutons(); }}
void AfficheLeTemps() {// Télécharger la date et l'heure préservée dans le DS1302.
  LiquidCrystal lcd(8, 9, 4, 5, 6, 7); lcd.begin(16,2); lcd.setCursor(0,0);
  Time t = rtc.time();
 // Nommer les jours de la semaine.
  memset(JOUR, 0, sizeof(JOUR));  // Effacer le tampon pour le jour.
  switch (t.day) {
    case 1: strcpy(JOUR, " Lundi"); break;
    case 2: strcpy(JOUR, " Mardi"); break;
    case 3: strcpy(JOUR, "Mercredi"); break;
    case 4: strcpy(JOUR, " Jeudi"); break;
    case 5: strcpy(JOUR, "Vendredi"); break;
    case 6: strcpy(JOUR, " Samedi"); break;
    case 7: strcpy(JOUR, "Dimanche"); break; }
// Affiche la chaine des données sur le LCD.
 lcd.print(JOUR);  lcd.setCursor(9,0); lcd.print(t.date);  lcd.setCursor(12,0);
 if (t.mon == 1) {lcd.print("JANV");} ; if (t.mon == 2) {lcd.print("FEVR");} ;
 if (t.mon == 3) {lcd.print("MARS");} ; if (t.mon == 4) {lcd.print("AVRL");} ;
 if (t.mon == 5) {lcd.print("MAI ");} ; if (t.mon == 6) {lcd.print("JUIN");} ;
 if (t.mon == 7) {lcd.print("JUIL");} ; if(t.mon == 8) {lcd.print("AOUT");} ;
 if (t.mon == 9) {lcd.print("SEPT");} ; if (t.mon == 10) {lcd.print("OCTB");} ;
 if (t.mon == 11) {lcd.print("NOVB");} ; if (t.mon == 12) {lcd.print("DECB");} ;
 lcd.setCursor(0,1);   lcd.print(">> "); if (t.hr < 10) {lcd.print(" ");};
 lcd.print(t.hr); lcd.print("H ");   if(t.min < 10) {lcd.print(" ");};
 lcd.print(t.min); lcd.print("min ");   if(t.sec < 10L) {lcd.print(" ");}
 lcd.print(t.sec); lcd.print("s"); }
void Traiter_les_boutons () {
  //==================== Attendre le relacher. ========================
  while (analogRead(Entree_mesuree) < 830) {}; // Attendre le relacher.
  //====================== Bouton SELEXT ==============================
  if (CNA > 522) { if (Lumineux)
                 {DitalWrite(Pilotage_RETRO,LOW); Lumineux = false;}
               else {digitalWrite(Pilotage_RETRO, HIGH); Lumineux = true;} } ;
  //======================== Bouton LEFT ==============================
  if ((CNA > 330) && (CNA < 522 )) // Diminuer les secondes de 1.
    {Time t = rtc.time(); t.sec = t.sec - 1;
     rtc.time(t);} ;
  //======================== Bouton DOWN ==============================
  if ((CNA > 176) && (CNA < 330 )) // Diminuer les minutes de 1.
    {Time t = rtc.time(); t.min = t.min - 1;
     rtc.time(t);} ;
  //========================== Bouton UP ==============================
  if ((CNA > 50) && (CNA < 176 )) // Augmenter les minutes de 1.
    {Time t = rtc.time(); t.min = t.min + 1;
     rtc.time(t);} ;
  //======================== Bouton RIGHT =============================
  if (CNA < 50 ) // Augmenter les secondes de 1.
    {Time t = rtc.time(); t.sec = t.sec + 1; rtc.time(t);} ; }
```

Comme montré en vue de dessous sur la Fig.1, ce SHIELD est équipé d'origine d'un lecteur de mini-carte SD pour pouvoir exploiter des images "Bitmap" qui y seraient stockées. Il s'agit d'un vrai écran couleur qui permet le respect total de la teinte des images. Une bibliothèque complète permet facilement d'utiliser ce module, avec un exemple d'utilisation de l'écran tactile, et de plusieurs autres exemples pour la génération de tracés divers. *(Cercles de couleur, Droites de couleur, Combinaison des deux, Tracés colorés de textes de différentes tailles etc)* La Fig.2 représente le résultat de l'un de ces programmes qui transforme ce dispositif électronique en tablette à dessiner. Dans cet exemple, "cliquer" en **1** sur la bande latérale à droite permet de sélectionner la couleur du tracé. Le dessin est obtenu en déplaçant un stylet sur le reste de l'écran en **2**. L'image de la Fig.3 montre le résultat d'un autre programme qui est dérivé de ceux proposés, et qui fait défiler en boucle un certain nombre de photographies situées sur la mini-carte SD. On peut vérifier que les couleurs restituées sont vraiment très belles. Pour pouvoir être utilisées par le logiciel ces images doivent avoir exactement 320 x 240 pixels.

Fig.1

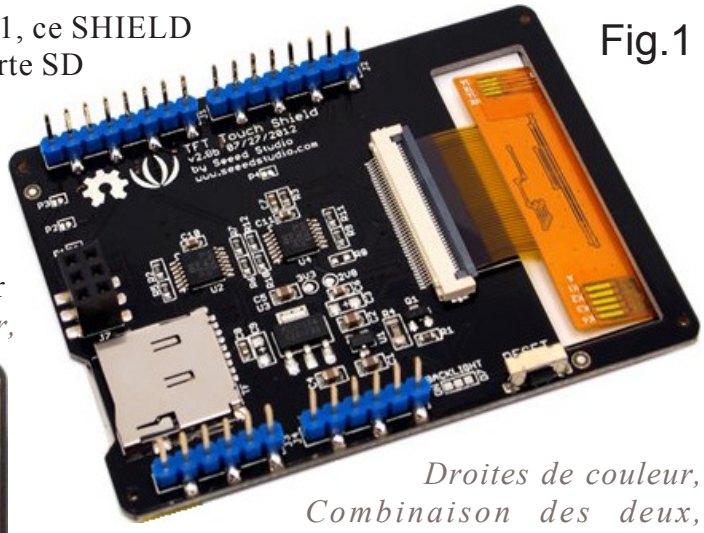
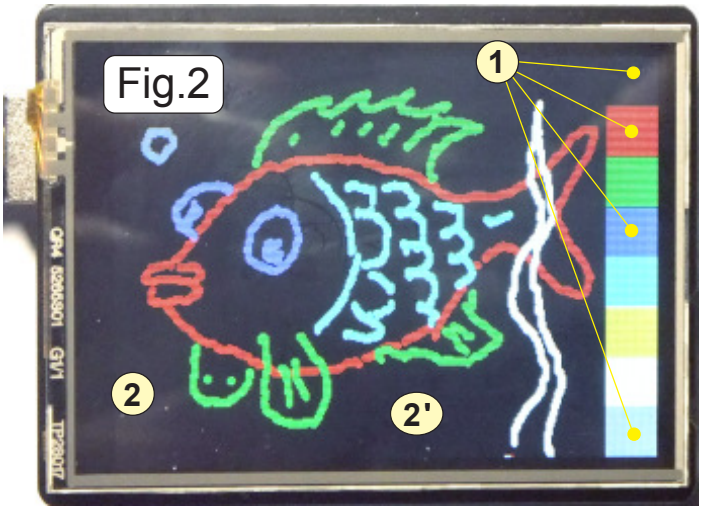


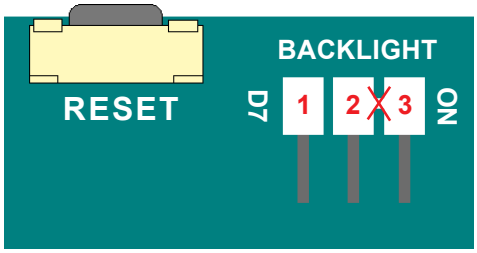
Fig.2



Caractéristiques techniques du module LCD :

- Alimentation : 4,5Vcc à 5,5Vcc.
- Courant maximal consommé : 250 mA maximum.
- Diagonale de l'écran : 2,8 pouces. (72 mm)
- Angle d'observation : 60° à 120°.
- Résolution de l'écran : 320 x 240 pixels.
- Nombre possible de couleurs : 65535.
- Éclairage arrière par LED dont la luminosité peut être contrôlable par programme.
- Contrôleur LCD utilisé : ILI9341.
- Capteur de contact digital sur toute la surface de l'écran.
- Interface de type SPI.

Fig.4



(@) : Attention, la voie série s'approprie D0 et D1 si elle est utilisée.

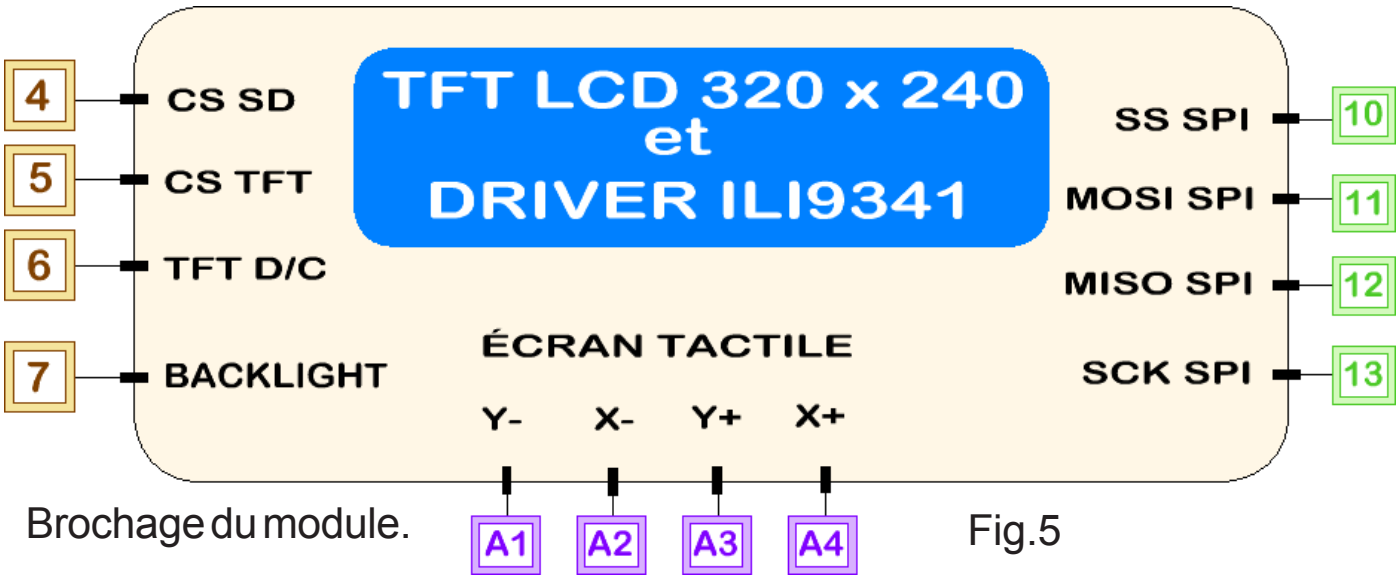
Pilotage du rétro éclairage.

La Fig.4 montre le circuit imprimé du module d'affichage vue dans la zone du petit bouton poussoir qui déporte le RESET. On y observe les trois pastilles **1**, **2** et **3** qui permettent de choisir entre un éclairage permanent et un éclairage piloté. D'origine les pastilles **2** et **3** sont pontées, **2** est donc alimenté en permanence par le +5Vcc. Couper la piste entre **2** et **3**, puis ponter **1** et **2**. Le rétro éclairage sera alors piloté comme une DEL quelconque par la broche D7 qui devra être déclarée en sortie de type binaire. Les broches D0, D1, D2, D3, D8, D9 ainsi que A4 et A5 restent disponibles. *(@)*

Fig.3

Branchements du SHIELD SLD10261P.

L'agencement du petit module électronique tient compte de l'utilisation en standard de la SPI. Comme on peut le vérifier sur la Fig.5 les Entrées/Sorties utilisées sont celles habituellement affectées à cette ligne de dialogue. Comme précisé sur ce dessin le rétro éclairage peut être piloté par la sortie binaire D7. Mais d'origine cette broche est laissée à la convenance du programmeur. La Fig.4 de la page 34 indique la modification à apporter au circuit imprimé si l'on désire une gestion par Arduino du rétro éclairage.



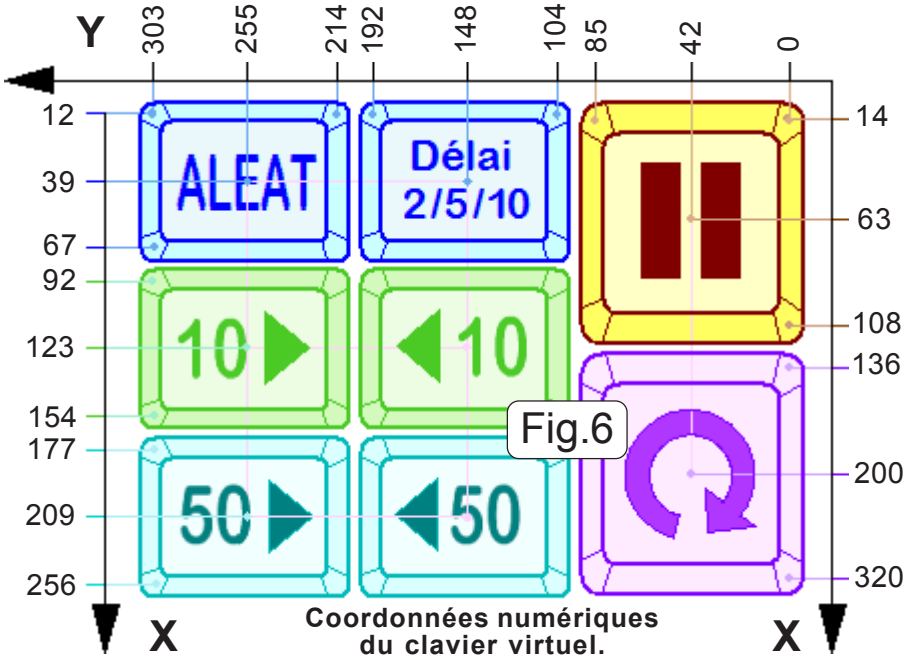
Brochage du module. Fig.5

Gestion de l'afficheur graphique :
D4 : Broche de sélection du lecteur de carte SD.
D5 : Broche de sélection du module graphique.
D6 : Broche de mode DONNÉES/COMMANDE.
D7 : Pilotage éventuel du rétro éclairage.

Gestion de l'écran tactile :
A1 : Tension de numérisation Y-.
A2 : Tension de numérisation X-.
A3 : Tension de numérisation Y+.
A4 : Tension de numérisation X+.

Écran tactile et coordonnées de numérisation.

Gérer l'écran tactile à partir de la bibliothèque s'avère relativement simple. La lecture des quatre tensions A1 à A4 une fois traitée fournit les coordonnées ainsi que la pression subie au point de contact de la surface sensible. Les valeurs proposées sur la Fig.6 sont celles obtenues en "cliquant" sur les coins et au centre des touches fictives. Le clavier virtuel utilisé ici est celui prévu pour le petit projet de cadre photographique interactif. Les valeurs portées sur la Fig.6 sont issues d'une moyenne effectuées sur plusieurs mesures, car le point sur lequel on échantillonne la lecture n'est jamais strictement le même. Le programme personnel Test_Ecran_Tactile.ino affiche le dessin du clavier virtuel "IMAGE00.bmp" puis liste sur la ligne série les coordonnées à chaque appui en un point quelconque de l'écran. Ce programme pourra être utilisé pour déterminer les coordonnées correspondant à des claviers différents dédiés à des applications particulières.

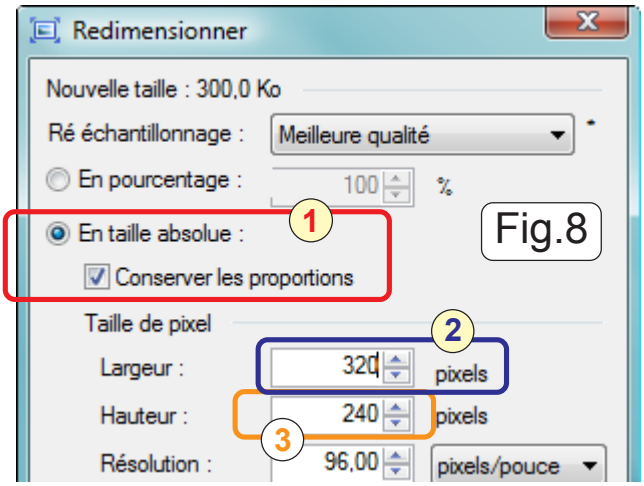


Traiter une image pour la visionner sur le petit écran LCD.

Fondamentalement la bibliothèque permet d'utiliser des images de type BMP qui doivent respecter le format de l'écran si l'on veut éviter des affichage aberrents. Il suffit de transformer la photographie pour aboutir à une image de 320 pixels de large et 240 pixels de haut, mais avec l'attitude représentée sur la Fig.7 pour qu'elle soit convenablement orientée. La technique pour adapter les photographies issues de l'appareil stéréoscopique est d'autant plus simple que le rapport Largeur/Hauteur des photographies est pratiquement idéal. Quand il faut "rogner" les images, c'est toujours faiblement ce qui ne les dénature pas.

1) Ouvrir PAINT.EXE et charger l'image à convertir.
2) Copier l'image dans PAINT.EXE et la coller dans le logiciel de traitement d'images PAINT NET.EXE.
3) Image > Redimensionner ...
4) La Fig.8 présente en **1** dans le cadre rouge les options les mieux adaptées pour changer le format de l'image.
   Il suffit d'imposer la largeur en **2** pour qu'en **3** automatiquement la Hauteur passe à environ 240 pixels.

Si la taille en Hauteur est inférieure à 240 pixels, imposer 240 dans sa fenêtre, ainsi c'est la largeur qui deviendra un peu supérieure aux 320 pixels désirés. L'important consiste à avoir au minimum les dimensions requises, et à couper dans PAINT le coté qui "déborde", car le module SLD10261P n'accepte pas des images de taille inférieure à celle de la définition de l'écran de visualisation.



5) Image > Faire pivoter de 90° à droite >
6) Copier l'image.
7) Revenir dans PAINT.EXE et la coller.
8) Si l'image déborde la taille sur l'une des deux dimensions, la recadrer avec Image > Attributs... puis la sauvegarder.

Contraintes d'exploitation :

L'utilisation de la bibliothèque TFTv2.h impose un certain nombre de contraintes relatives à l'image à visualiser et au nom du fichier qui lui est attribué :

- Les noms d'images ne doivent pas dépasser 8 caractères. *(Auxquels on ajoute ".bmp")*
- Si l'image est grise et moirées, c'est que la définition ne fait pas exactement 320 x 240 pixels. La détérioration de l'image intervient que la définition soit plus petite ou plus grande.
- **Si un nom d'image n'est pas trouvé le programme affiche un écran gris et "se bloque"**.
- Par contre l'appel de ces images peut se faire dans un ordre quelconque.

Pour pouvoir ouvrir un fichier image il faut utiliser une instruction de type :

```
bmpFile = SD.open("IMAGE0.bmp");
```

S'il faut inclure le nom de chaque image dans le programme, la taille imputée dans l'espace programme devient rapidement rédhibitoire et annulera le bénéficier du volume disponible dans la carte SD. De plus on imagine facilement la lourdeur à écrire tous les noms des fichiers dans le programme d'exploitation. L'idée pour optimiser cet aspect du logiciel consiste à donner le même radical à toutes les images avec un n° de différentiation. Par exemple IMAGE25.bmp où 25 est sa signature. Le n° sera celui de la variable de la boucle qui visualise en continu. Le nom de l'image est alors construit avec le préfixe "IMAGE", l'indice de la boucle et le suffixe ".bmp" ces trois entités étant concaténées dans une variable de type chaîne.

Un petit projet complet avec le SLD10231P.

L'idée consiste à réaliser un petit cadre photographique numérique interactif. Ce programme n'est pas très ambitieux, mais il utilise les routines de lecture de la petite carte mémoire SD, l'affichage d'images au format BMP et l'utilisation de l'écran tactile pour changer le comportement du système. La Fig.9 présente l'organigramme de la boucle principale du programme utilisé. Comme expliqué en bas de la page 36 le nom du fichier image à visionner est construit en **1**. Si la touche virtuelle **ALEAT** *(**A** sur la Fig.10)* a été validée, le n° de l'image sera généré avec la fonction aléatoire dans la plage [1 - "NumImageMAX"]. En **2** on trouve une boucle dont le nombre de passages peut être modifié avec la touche **Délai** *(**B** sur la Fig.10)* qui en permutation circulaire génère la durée pendant laquelle il y a attente avant de passer à l'image suivante. Les temporisations sont respectivement de 2, 5 et 10 secondes. Chaque passage dans la boucle vérifie si l'écran tactile a été sollicité. Si c'est le cas, on passe en **3** qui fait afficher le clavier virtuel. Puis il y a attente d'un "clic" sur l'écran tactile. En fonction de la zone sur laquelle on a appuyé, il y aura traitement des touches de **A** à **H**. La touche **C** augmente de 10 le n° de la prochaine image qui sera affichée. Naturellement il y a une vérification de non débordement avec recyclage à l'image n°1. La touche **E** présente exactement le même comportement mais avec un incrément de 50 au lieu de 10. **D** et **F** sont des fonctions similaires mais qui font décaler de 10 ou 50 images en arrière avec test de débordement sur 1. La touche **G** ne fait que valider le booléen PAUSE, alors que la touche **H** le replace à FAUX. Noter que si le mode **ALEAT** est actif, les touches **C**, **D**, **E** et **F** sont sans effet puisque l'indice de la boucle principale n'est plus pris en compte pour générer le nom du fichier à visualiser. Quand la PAUSE est validée avec **G**, on tourne dans la boucle d'attente **5** jusqu'à ce que l'on clique sur l'écran. Si l'on clique sur la moitié droite de l'écran, on passe à l'image suivante pour retomber dans le circuit **5**. Si l'on clique sur la moitié gauche, c'est que l'on désire retrouver le fonctionnement en boucle continue. Le programme se contente de forcer à FAUX le booléen PAUSE. On retourne alors dans la boucle principale avec passage dans la temporisation **2**. Si le mode PAUSE est actif, la séquence de programme **2** est sautée, car pour pouvoir reprendre la boucle avec test du clavier virtuel il faut "cliquer" dans la moitié gauche de l'écran. Quand en **2** on clique sur l'écran il y a passage en **3** pour afficher le clavier graphique. Le traitement **4** commence par entrer dans une boucle pour attendre que l'on "clique" sur l'écran. Puis, en fonction de la zone influencée par cette sollicitation il y a traitement de la touche virtuelle conformément aux explications données ci dessus.

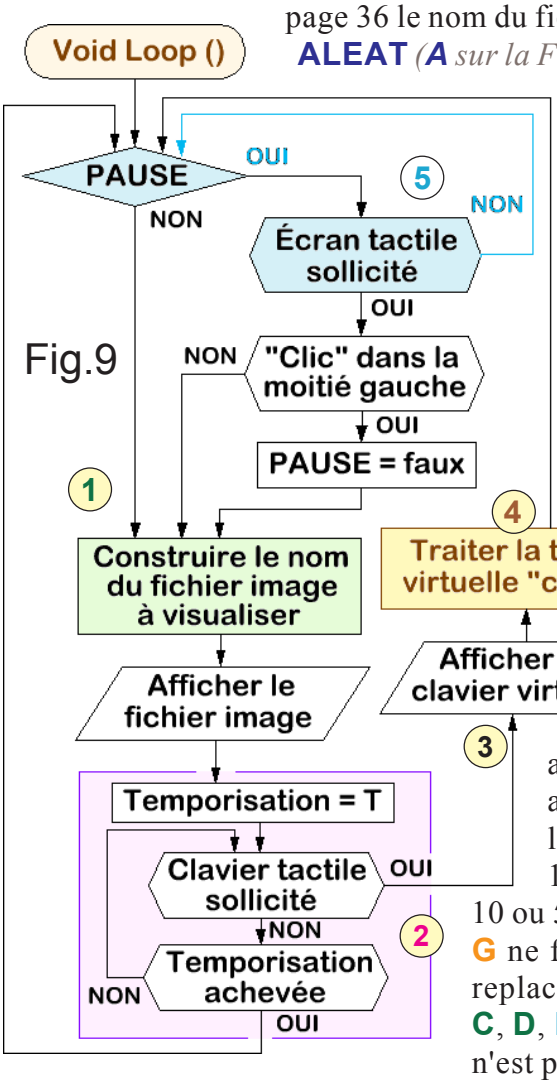


Fig.9

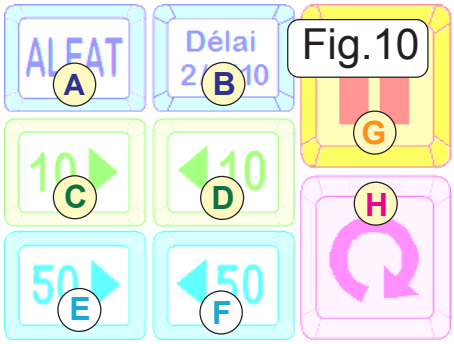


Fig.10

PROBLÈME : Je n'ai pas trouvé la séquence de programme qui vérifie de façon absolue que la surface sensible a été activée. Dans les deux boucles d'attente, celle en **4** ou celle **5**, au bout d'un certain temps il y a déclenchement intempestif comme si l'on avait "cliqué" sur l'écran tactile. La durée de cette attente varie, mais à un moment où à un autre le passage à l'image suivante se déclenche spontanément.

PISTE À EXPLORER : Actuellement en boucle continue on ne peut faire appel au clavier virtuel qu'en fin d'image dans la boucle d'attente **2**. Il faudrait analyser pour voir si un traitement de cette séquence **2** ne pourrait pas se traiter par une interruption, ainsi il deviendrait possible d'intervenir durant le balayage de l'écran qui construit l'image en cour d'affichage.

Actuellement tous les servomoteurs utilisés en maquettisme fonctionnent sur le même principe. Bien que les fournisseurs ne respectent pas forcément ce qui semble devenir une norme de fait, globalement tous les moteurs s'approchent de ce qui devient un standard. Ces moteurs intégrant une électronique d'interfaçage et un capteur de position se raccordent avec seulement trois fils. Le fil noir ou marron va à la masse, le fil central rouge va au +Vcc et le fil blanc ou orange reçoit le signal de pilotage.

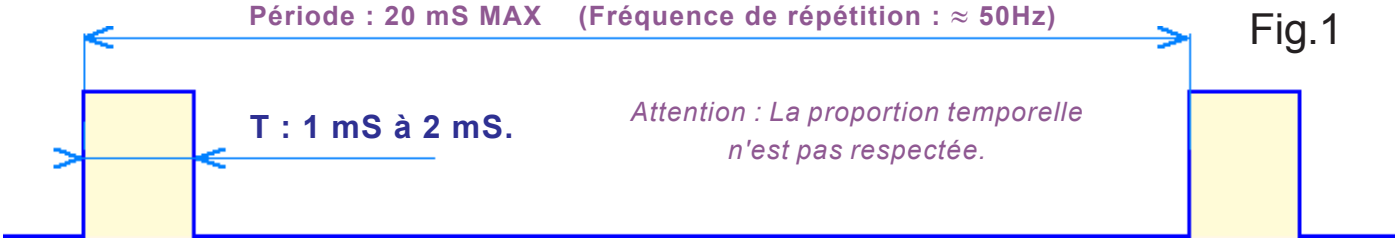


Fig.1

Comme montré sur la Fig.1 la fréquence de répétition standard choisie pour le pilotage est de 50 Hz, soit une période de 20 mS. C'est la durée **T** de l'impulsion à l'état "1" qui positionne le moteur en absolu par rapport à son amplitude possible de rotation. Mis à part les moteurs sans limitation de rotation angulaire, les plages les plus courantes avoisinent les 180°.


Fig.2

En standard :

- T = 1 mS : Le moteur se place en position minimale.
- T = 1,5 mS : Le moteur se place en position neutre. *(Position moyenne)*
- T = 2 mS : Le moteur se place en position maximale.

Caractéristiques techniques des deux moteurs utilisés :

Quand la consigne angulaire passe de 0° à 180° ou que la consigne en impulsion passe de 540µS à 2400µS, le moteur MC-1811 de la Fig.2 tourne dans le sens des aiguilles d'une montre. Par contre, le modèle RS2 MG/BB montré en Fig.3 tourne en sens inverse pour les mêmes variations de commande de pilotage.


Fig.3

ATTENTION : Un seul de ces moteurs branché sur le 5Vcc d'Arduino provoque de brusques baisses de tension lors des appels de courant. Ces perturbations sont parfaitement mesurables quand on branche un voltmètre quelconque sur le +5Vcc d'Arduino. Si la tension chute, c'est que le régulateur intégré à Arduino passe en limitation de courant. Outre que le microcontrôleur et les circuits intégrés périphériques ne fonctionnent pas dans des conditions fiables, le régulateur intégré à Arduino va chauffer, et ce d'autant plus qu'il n'est pas refroidi par radiateur. **Il est donc fortement recommandé d'alimenter les moteurs séparément** avec un bloc externe sans oublier naturellement de réunir les masses pour la référence des signaux de commande.

Moteur	Alimentation	Couple	Réaction	Paliers	Engrenages
MC-1811	4,8Vcc à 6Vcc	15 à 18cmN	0,19S (60°)	---	PTFE
RS2 MG/BB	4,8Vcc à 6Vcc	31 à 35cmN	0,1S (60°)	Roulements	Métalliques

PROGRAMMATION des servomoteurs.

Compte tenu de la simplicité des signaux à générer, la première approche logique consiste à utiliser les instructions classiques d'Entrée/Sortie d'Arduino. Comme la période doit avoisiner les 20mS, l'utilisation de la PWM d'origine n'est pas possible. En effet, la fréquence de répétition de la PWM d'Arduino fait environ 1kHz, donc une période vingt fois trop faible. Mais rien n'interdit de générer directement de la PWM sur une sortie binaire en usant des fonctions de temporisation de base. Le petit programme Commande_Servo_par_PWM.ino génère des impulsions calibrées correspondant au standard. Bien que le fonctionnement soit correct, le moteur est affecté d'un bruit de "grenaillage" prouvant qu'il corrige sans arrêt sa position. Ce phénomène n'est pas compréhensible en ce qui me concerne, car les signaux observés à l'oscilloscope sont correct et les informations envoyées sur la ligne série sont conformes. Les

transmissions série sont passées en remarque dès que les valeurs ont été vérifiées, car elles allongent la période de répétition du signal de commande. La librairie Servo.h fournie en standard dans l'environnement IDE corrige ce phénomène, donc inutile pour le moment de rechercher l'origine de ce problème.

Exemple de programme avec génération "binaire" du signal de PWM :

PROGRAMME : Commande_Servo_par_PWM.ino

```
// Test de pilotage d'un Servomoteur par gestion binaire de la PWM.
// Un potentiomètre sur une entrés analogique génère le signal de commande.

const byte Entree_mesuree = 5; // Entrée analogique 5 utilisée pour commander.
const byte Sortie_PWM = 3;  // Broche PWM 3 utilisée pour générer la commande.
float CNA; // Mesure analogique retournée par le CNA.
int Duree_Pulse; // Durée de l'impulsion de commande.

void setup() { pinMode(Sortie_PWM, OUTPUT);  }

void loop() {
  CNA = analogRead(Entree_mesuree); // Valeur directe si RS2 MG/BB.
  CNA = 1024-CNA; // Inverser le sens si le moteur utilisé est le MC-1811.
  Duree_Pulse = ((1800 * CNA)/1023)+500;
  digitalWrite(Sortie_PWM, HIGH); delayMicroseconds(Duree_Pulse);
  digitalWrite(Sortie_PWM, LOW);
  delay(15); // Délai constant pour commencer le 20mS.
  delayMicroseconds(5000 - Duree_Pulse); } // Ajustement pour une fréquence constante.
```

Le programme Commande_Servo_PWM_librairie.ino donné ci-dessous génère des commandes qui débordent volontairement du standard. Il permet de trouver les limites extrêmes du moteur utilisé.

Exemple de génération du signal de PWM avec la librairie Servo.h :

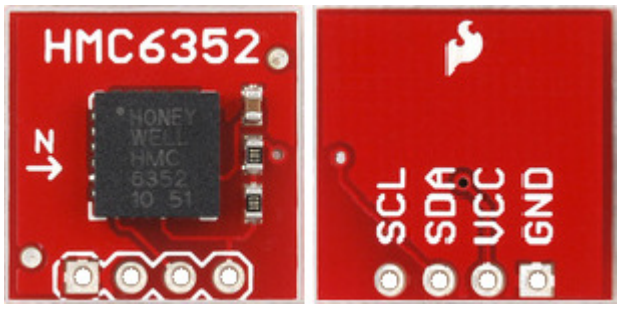
```
// Test de pilotage d'un Servomoteur par gestion librairie de la PWM.
// Un potentiomètre sur une entrés analogique génère le signal de commande.

#include <Servo.h> // Inclusion de la librairie Servo.
Servo Servomoteur; // déclare une variable de type Servo appelée Servomoteur.
const byte Entree_mesuree = 5; // Entrée analogique 5 utilisée pour commander.
float CNA; // Mesure analogique retournée par le CNA.
int Duree_Pulse; // Durée de l'impulsion de commande.

void setup()  { Servomoteur.attach(3,100,3000); // Associe le servomoteur à la broche 3.
   Serial.begin(19200);  }

void loop() {
  CNA = analogRead(Entree_mesuree); // Valeur directe si RS2 MG/BB.
  CNA = 1023-CNA; // Inverser le sens si le moteur utilisé est le MC-1811.
  Duree_Pulse = ((2900 * CNA)/1023)+100;
     (2900 = impus_max - impls_min)
  Servomoteur.writeMicroseconds(Duree_Pulse);
  // Positionne le moteur.
  Serial.print("Duree_Pulse = ");
Serial.println(Duree_Pulse); }
```

impls_min, impls_max

Limites mesurées sur les modèles utilisés		
Servomoteur	T mini	T MAXI
MC-1811	660 à 700	2000
RS2 MG/BB	570	2000

Exemple de pilotage ANGULAIRE avec la librairie Servo.h :

C'est probablement la façon la plus "naturelle" d'envoyer des commandes à un servomoteur. La commande NomMoteur.write(Angle) positionne directement le moteur à la valeur Angle indiquée en degrés et comprise entre 0 et 180. La fonction se charge de générer l'impulsion qui placera le moteur à une position proportionnelle en fonction de l'amplitude de rotation possible sur le modèle utilisé. Le petit programme Commande_Servo_par_Angles.ino donne un exemple d'utilisation, c'est toujours un potentiomètre branché sur une entrée analogique qui permet d'envoyer les consignes.



Comme tous les modules qui utilisent une SPI pour réaliser le dialogue entre Arduino, ce petit circuit est extrêmement facile à mettre en œuvre. Le schéma du petit module est donné en Fig.1 avec le circuit imprimé représenté par dessus. La flèche verte sur le dessin représente la direction du "Nord magnétique" de cette boussole statique. La petite flèche notée **N** sur le circuit imprimé définit ce pôle du circuit intégré. Quand cette flèche est orientée vers le Nord magnétique local, le HMC6352 retourne la valeur 0° pour le Cap.

Caractéristiques techniques du module HMC6352 :

- Tension de fonctionnement : 2,7Vcc à 5,2Vcc.
- Consommation 2mA. Maximum 10mA sous 5Vcc.
- Compas avec donnée de sortie en valeur décimale pour le CAP.
- Intègre deux axes magnétiques complets.
- Logiciel de traitement d'acquisition, de calibrage et de calcul de CAP inclus.
- Interface série Wire I²C.
- Fonctionne en esclave. Fréquence d'horloge possible 100kHz.
- Compensé en température.
- Peut être utilisé dans un environnement de champs magnétiques importants.

  ATTENTION : Bien que réputé protégé contre les champs magnétiques importants, sa mise en présence d'un aimant de gestion de la tête d'un disque dur, (Aimant très puissants) a engendré un mauvais fonctionnement du circuit. Pour retrouver sa sensibilité il a fallu soumettre la "puce" à un champs inverse, puis attendre plusieurs heures que la "rémanence" magnétique se dissipe.
- Plage de champ magnétique ±2Oe.

L'oersted (Symbole Oe) est l'unité ÉLECTROMAGNÉTIQUE CGS à trois dimensions d'excitation magnétique ou de champ magnétique. L'oersted, nommé ainsi en l'honneur de Hans Christian Ørsted.

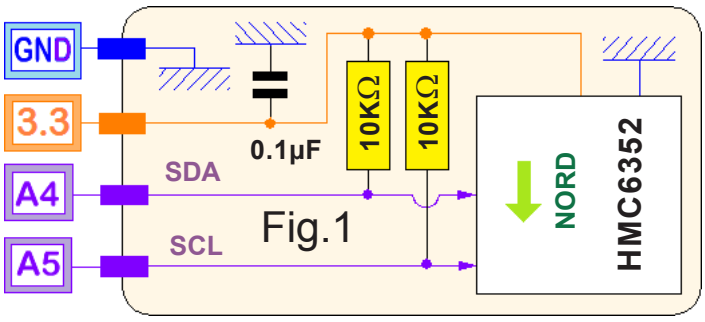


Fig.1

Outre le condensateur de découplage sur l'alimentation en 3,3Vcc on peut noter sur la Fig.1 que les deux lignes de la SPI sont forcées à l'état logique "1" par des résistances de 10 kΩ incluses. C'est un impératif puisqu'il n'y a pas d'autre module I²C en ligne, et que ce type de périphérique est à "collecteur ouvert" pour permettre le multiplexage de plusieurs esclaves.

ATTENTION : Le module boussole HMC6352 ne fournit une information de Cap correcte que s'il se trouve dans une attitude parfaitement horizontale. Il importe donc de veiller à cette contrainte lors des expérimentations. Conformément à tout dispositif influencé par un champs magnétique, il donnera une information relative aux "lignes de champ" locales. Toute masse magnétique ou aimant dans son environnement faussera l'information relative au champs magnétique terrestre.

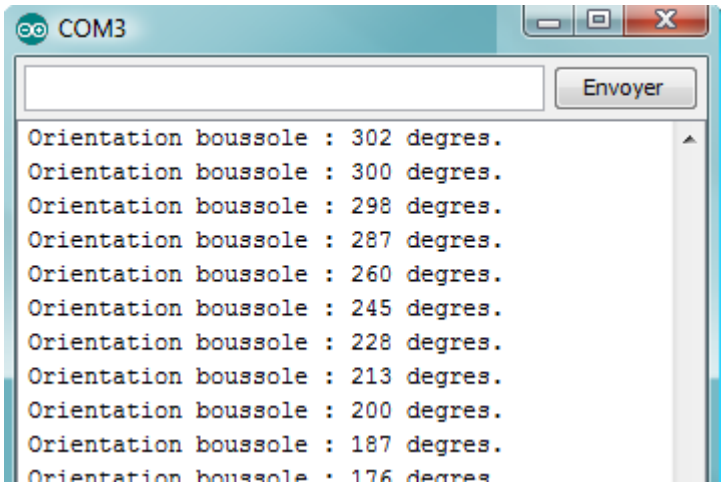
Exemple de programme :

Le petit programme Boussole_statique.ino permet de tester le dispositif. Il effectue environ 10 interrogations par secondes *(Suivis d'affichages sur la ligne série)* sur la ligne I²C et retourne l'information de Cap sur la ligne série USB. Dans ce programme la variable ValeurCAP *(De type int dans le programme)* est utilisée pour pouvoir s'affranchir des dixièmes de degrés lors de l'affichage sur la ligne série. Initialement, pour pouvoir tester ce petit module la bibliothèque spécialisée hmc6352.h avait été intégrée dans l'environnement de développement. Mais le petit programme d'essai n'en a pas besoin, il faut juste inclure les routines de gestion de la ligne série et de l'I²C.

https://www.sparkfun.com/products/7915
http://www.incertitudes.fr/robot/Matos-Arduino-Web.html

Liens pour obtenir de la documentation sur le HMC6352.

Fig.2




```
// Petit exemple d'utilisation de la boussole statique HMC6352.
// La boucle principale lit la valeur de l'orientation du module
// et liste "le CAP" sur la ligne série.
#include <Wire.h>
#include <OneWire.h> // Librairie pour le bus "OneWire".
/* La valeur par défaut (Usine) de l'adresse du HMC6352 esclave 7 bits
   est 42 HEX pour les opérations d'écriture, ou 43 HEX pour la lecture. */
int Adresse_du_HMC6352 = 0x42; // Écrire à cette adresse.
int Lecture_Adresse = 0x41; //"A" en hexadécimal correspond à une commande.
int ValeurCAP;

void setup() {
/* Wire librairie utilise une adresse sur 7 BITs. Si le module utilisé
   utilise une adresse sur 8 BITs il suffit d'effectuer un décalage à
   droite pour obtenir une adresse qui reste dans la plage 0 à 127. */

Adresse_du_HMC6352 = Adresse_du_HMC6352 >> 1;
// L'adresse Ox42 est < 127 mais cette instruction est nécessaire.
Serial.begin(19200);
Wire.begin(); }

void loop() {
// Saisie de la donnée.
Wire.beginTransmission(Adresse_du_HMC6352);
Wire.write(Lecture_Adresse); // Commande de lecture.
Wire.endTransmission();
// Le HMC6352 a besoin d'un délai de 6ms quand il reçoit une commande.
delay(6);
Wire.requestFrom(Adresse_du_HMC6352, 2);
// Saisie des deux octets de donnée "MSB" et "LSB".
byte MSB = Wire.read(); byte LSB = Wire.read();
// ========== Traite la donnée et calcule l'orientation. ==========
// La donnée issue de deux octets est concaténée, "MSB" étant décalé huit fois à gauche.
float CAP_construit = (MSB << 8) + LSB;
// La donnée est fournie en degrés et dixièmes de degrés d'où la division par 10.
float CAP_fractionnaire = CAP_construit / 10;
// ========== Affiche l'orientation sur la ligne série. ==========
Serial.print("Orientation boussole : "); ValeurCAP = CAP_fractionnaire;
Serial.print(ValeurCAP); Serial.println(" degres.");
delay(100); } // Environ dix affichages par seconde.
```

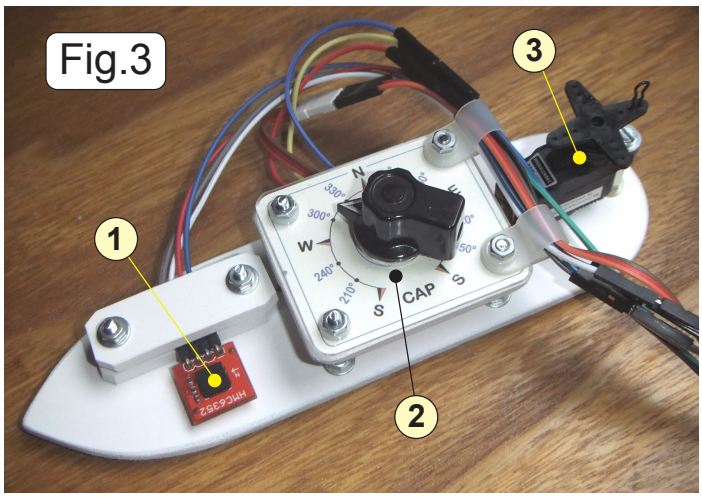
Un petit projet complet avec le HMC6352.

Dans sa version la plus élémentaire, le projet consiste à simuler la barre automatique d'une petite maquette de bateau en n'incluant que les éléments strictement indispensables, des petits perfectionnements seront ensuite ajoutés pour étoffer la programmation. La structure globale de base est montrée sur la Fig.3 est comporte trois éléments fondamentaux :

1) Le détecteur de la ROUTE magnétique actuelle.

2) Un potentiomètre pour indiquer le CAP que le bateau devra suivre.

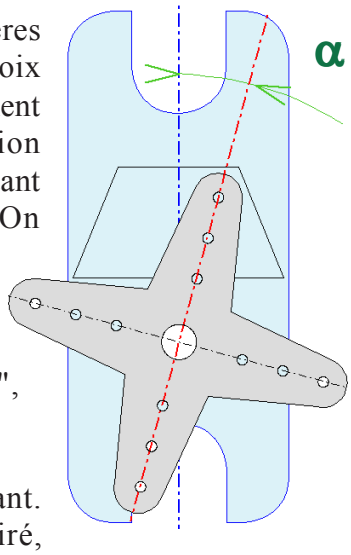
3) Un servomoteur qui peut entrainer directement le safran fixé sur la sortie de son arbre ou par un palonnier relié à la croix. Sur la maquette de la Fig.3 la partie arrière du servomoteur comporte un "trombone" vertical qui symbolise la position du safran qui plongerait dans l'eau à la poupe.



Non visible sur la Fig.3 il faut impérativement prévoir un autre potentiomètre qui sera chargé de définir l'amplitude de la plage neutre. C'est l'angle de déviation de part et d'autre de la ROUTE qui est toléré et qui n'engendre pas de correction de cap. Cette tolérance est impérative si l'on ne veut pas que le servomoteur soit en corrections permanentes. En fonction du clapot et des conditions de navigation, cette zone neutre de chaque coté de la route imposée sera comprise entre 2° minimum et 15° maximum.

Contraintes imposées pour la gestion de la barre.

Fig.4



Comme sur tout navire la barre doit être gérée en respectant plusieurs critères parfois "contradictoire". Certains sont évidents, d'autres résultent de choix personnels. Comme déjà précisé ci-avant, pour éviter un fonctionnement permanent du servomoteur qui engagerait le bateau dans des alternances de correction permanentes, une zone neutre sera définie en fonction de l'altération de cap résultant des conditions de navigation. *(Allure du bateau, état du plan d'eau etc.)* On suppose ici que la maquette est un voilier, ce qui induit des problèmes bien plus nombreux que s'il s'agit d'un bateau à moteur. *(Dérive, instabilité de cap au portant, virement de bord face au vent prohibé etc.)*

- Si la commande de CAP n'engendre pas un problème de virer "Face au vent", tourner du coté où l'écart angulaire est le plus faible.
- Angle de barre maximal : 30°. *(Rappel le plus énergique)*
- Angle de barre *(Rappel)* d'autant plus grand que l'écart angulaire est important.

Il faut "tempérer" la correction de route quand le bateau approche le cap désiré, car le délai de réaction du servomoteur et surtout l'inertie de bateau vont forcer un dépassement de la consigne. L'automatisme passerait alors dans une alternance permanente de surcompensation.

- Correction possible de dérive en fonction de l'allure adoptée par le voilier et d'un courant traversier.
- Correction angulaire "du zéro" pour tenir compte de la non symétrie du palonnier. *(Voir Fig.4)*

Correction de symétrie : La croix servant de palonnier ne peut adopter sur l'arbre de sortie du servomoteur qu'un certain nombre de positions imposées par des cannelures. Comme montré sur la Fig.4, quand le servomoteur reçoit une consigne de "zéro", la croix peut présenter un décalage **α** par rapport à l'axe longitudinal du moteur, donc un angle parasite par rapport à la symétrie de la barre quand la commande est au neutre. Pour compenser ce détail mécanique il suffit de retrancher systématiquement l'angle **α** dans la consigne de positionnement du moteur.

Conversion Analogique / Cap Désiré en degrés.

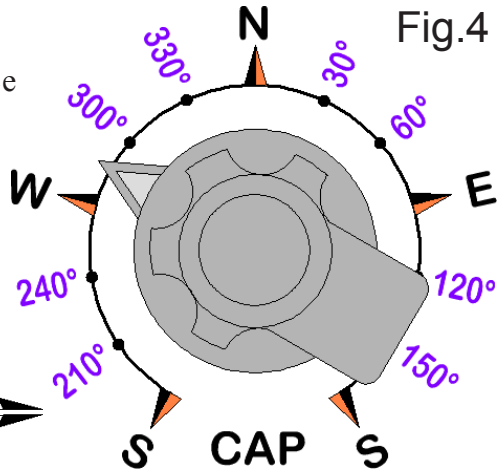


Fig.4

Réaliser cette conversion n'est vraiment pas compliqué. Le potentiomètre utilisé sur l'une des entrées analogiques fournit une valeur comprise entre 0 et 1024 une fois la numérisation effectuée. Comme représenté sur la Fig.4, la rotation du bouton de commande se fait sur une échelle qui va de 181° à 180° en passant par 360°/0° à moitié course. On désire donc un traitement qui aboutit à la transposition de la Fig.5 donnée ci-dessous.

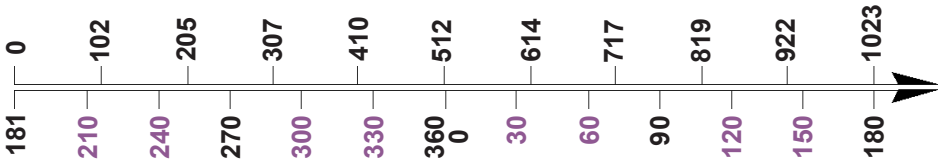


Fig.5

Le codage de cette transposition reste facile :

```
void DeterminerCommandeDeCAP() {
  CAP = analogRead(Commande_du_CAP); // Mesure de la "consigne CAP".
  if (CAP >= 512) {CAP = ((CAP - 511) * 180) / 512;}
        else {CAP = ((CAP * 180) / 512) + 181;} }
```

La "dualité" 0/360 n'est pas traitée car elle ne présente pas d'inconvénient dans le programme.

Linéarisation du potentiomètre.

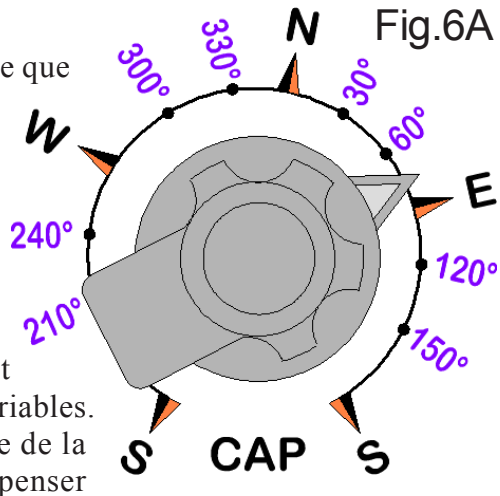


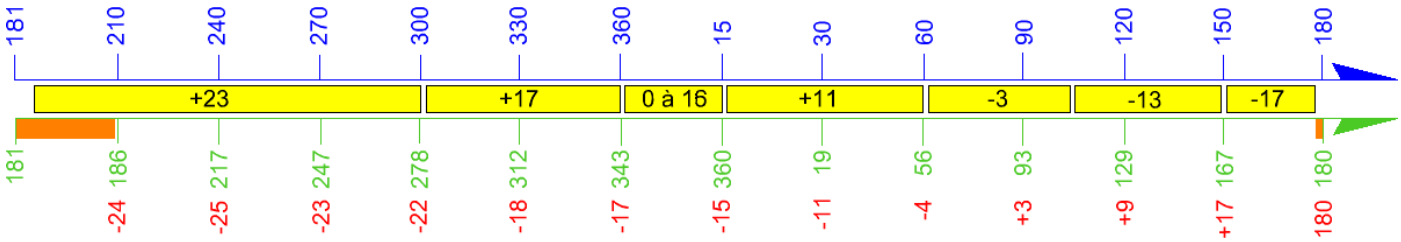
Fig.6A

Imprimer les graduations comme représenté sur la Fig.4 suppose que le potentiomètre utilisé présente une variation réellement linéaire en fonction de la position angulaire de son "axe de commande". Mais comme montré sur la Fig.6A la réalité est bien différente, et c'est assez ennuyeux pour vérifier le comportement du programme car la consigne de CAP ne correspond pas à la valeur pointée par le bouton flèche. Deux solutions sont envisageables. On peut corriger le dessin des graduations comme montré sur la Fig.6A et réimprimer, mais ce n'est pas une solution esthétique. Le nord n'est plus "vers le haut" et surtout les graduations ont des espacements variables. On peut corriger les valeurs renvoyées par la procédure de saisie de la valeur du CAP. L'idée consiste comme montré sur la Fig.6B à compenser en partie par valeur moyenne de l'écart constaté, en procédant par secteurs avec la procédure ci-dessous.

```
void CorrigerCommandeNonLineaire() {
  int CapLineaire = CAP;
  if ((CAP >= 185) && (CAP <= 278)) {CapLineaire = CAP + 23;};
  if ((CAP > 278) && (CAP <= 343)) {CapLineaire = CAP + 17;};
  if ((CAP > 343) && (CAP <= 360)) {CapLineaire = 16 - (360-CAP);};
  if ((CAP >= 0) && (CAP <= 56)) {CapLineaire = CAP + 11;};
  if ((CAP > 56) && (CAP <= 110)) {CapLineaire = CAP - 3;};
  if ((CAP > 110) && (CAP <= 167)) {CapLineaire = CAP -13;};
  if ((CAP > 167) && (CAP < 178)) {CapLineaire = CAP -17;};
  CAP = CapLineaire; }
```

Valeurs pointées par le bouton flèche.
Valeurs retournées par la numérisation.
Écarts à corriger.
Dans les rectangles jaunes la compensation partielle effectuée.

Fig.6B



En orange les plages non corrigées car correspondent aux zones "mortes" sur le potentiomètre utilisé.

Déterminer de quel coté orienter la barre.

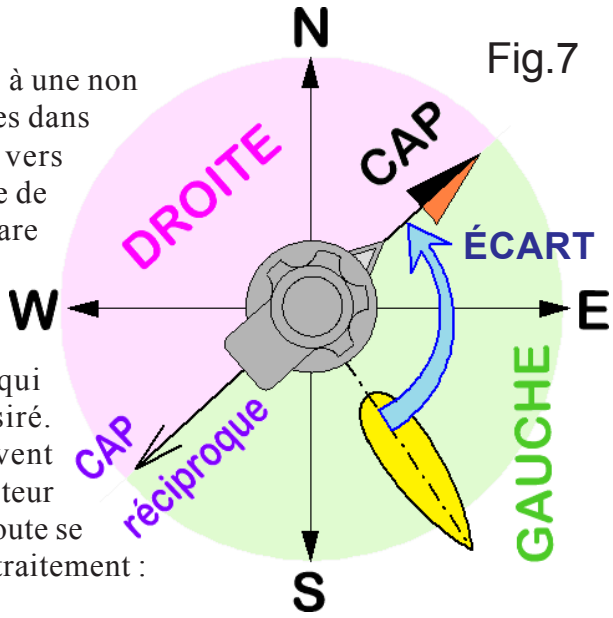


Fig.7

Après bien des essais infructueux probablement dus à une non homogénéité initiale des types de variables incluses dans les expressions de tests, la solution pour trouver le coté vers lequel il faut faire évoluer le bateau pour emprunter l'angle de correction le plus court est relativement simple. On compare le COMPAS à la consigne de CAP et l'on établit le changement d'état du booléen par rapport au CAP réciproque à +180°. Sur la Fig.7 l'arc représenté par la flèche curviligne bleue représente l'angle le plus faible qui permet au bateau jaune (COMPAS) de venir au CAP désiré. Tout le secteur vert clair correspond aux amures qui doivent engendrer une correction de route vers la GAUCHE. Le secteur complémentaire rose clair correspond aux amures dont la route se corrige par une déviation à DROITE. Voici la fonction de traitement :

```
boolean DeterminerBabordTribord(long CAP, long COMPAS) {
  if (COMPAS <= 180) {DROITE = (CAP > COMPAS) && (CAP < COMPAS + 180); }
    else {DROITE = (CAP > COMPAS) || (CAP < COMPAS - 180); }
  return(DROITE); }
```

Amplitude de rotation de la barre.

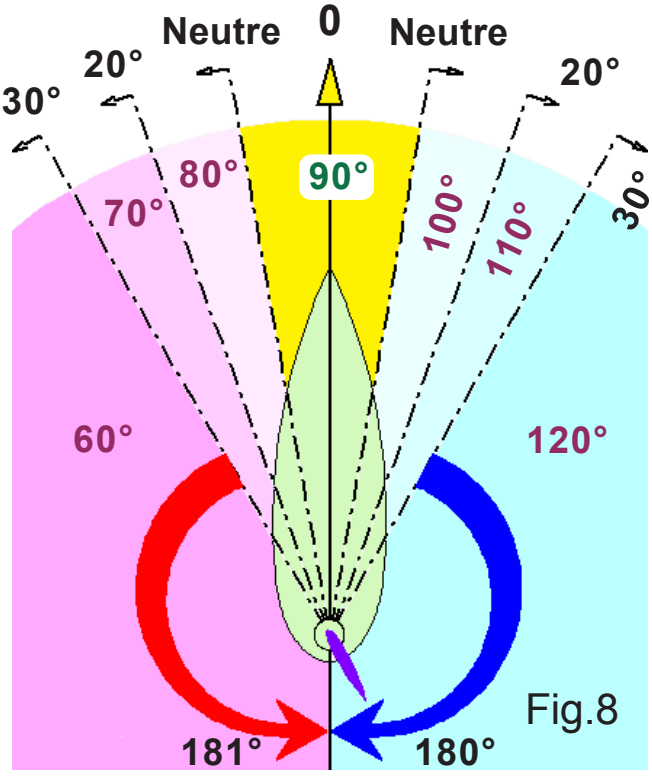


Fig.8

Les consignes d'angle représentées en violet sur la Fig.8 sont à rapprocher des valeurs indiquées sur la Fig.9 qui traduisent le comportement du servomoteur utilisé. Sur la Fig.9 en vert et en violet sont données les valeurs de consigne fournies à la librairie standard Servo.h sachant que le servomoteur utilisé couvre 180° de rotation à pleine échelle. La position 90° correspond au centrage de l'axe moteur. En bleu nous avons les déviations dans un sens et en rouge celles symétriques. Le sens de rotation dépend directement du modèle de servomoteur utilisé. Sur la Fig.9 la plage de -2° à +2° coloriée en jaune correspond à l'amplitude la plus faible de la zone neutre traitée par logiciel. Sur la Fig.8 le bateau est représenté en vert et par rapport à son axe longitudinal sont données en violet les consignes de pilotage pour le safran. La déviation de 30° de ce dernier représentée également en violet est probablement exagérée et doit par les tourbillons provoqués casser l'erre du bateau. Dans la pratique il ne faudrait pas dépasser un ordre de grandeur qui se situe aux environs de 20°, mais en caricaturant dans un premier temps les consignes envoyées à l'automatisme on en observe plus facilement son comportement sur la maquette de validation du programme. La plage jaune correspond à la zone neutre de déviation du bateau qui maintien la barre dans l'axe. Cette plage angulaire est variable dans une fourchette de valeurs comprises entre 2° et 15°. En noir sont précisées les valeurs de l'ÉCART entre la route magnétique actuelle du bateau et la commande de CAP. Les déviations sont traitées par secteur *(Colorés sur la Fig.8)* la flèche sur les rayons précisant que la borne est incluse.

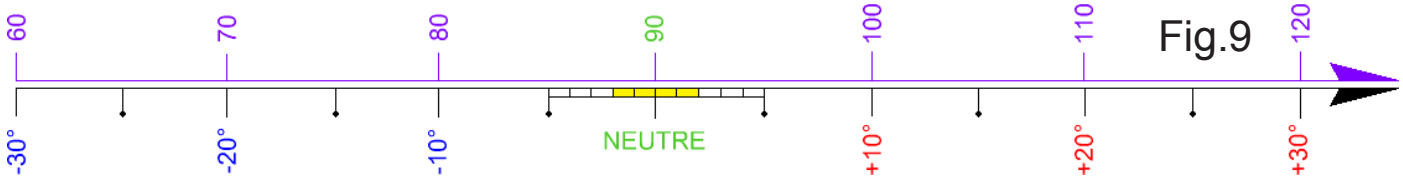
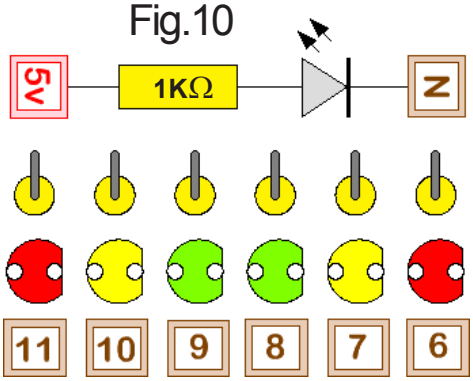


Fig.9

Visualisation de la consigne envoyée au servomoteur.

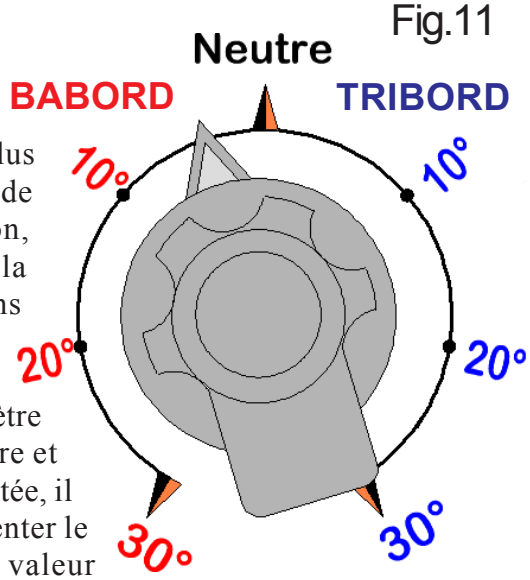
Fig.10



Comme Arduino Uno présente bien plus d'Entrées/Sorties que n'en exige le programme de base, y compris avec prise en compte des fonctions complémentaires envisagées, un groupe de LED dont le schéma est donné en Fig.10 permet de visualiser en couleur l'amplitude de la consigne envoyée au Servomoteur et de quel coté est orientée la barre. Une fois déclarées les Entrées/Sorties, ce petit complément est peu gourmant en code. Il suffit d'une procédure qui éteint toutes les LED et qui sera invoquée trois fois dans le programme. L'allumage des LED n'est constitué que de six instructions de forçage de niveau sur une sortie binaire, instruction incorporée directement dans les séquences de détermination de la consigne sermoteur. Plus la consigne dévie la barre, plus la couleur de la LED devient "chaude". Le schéma électronique montre le circuit utilisé pour chaque LED et explique pourquoi dans le programme il faut un état "1" pour éteindre et un état "0" pour allumer. Pour les LED vertes il faudrait adopter une résistance de valeur plus faible, car leur tension de conduction étant plus élevée que pour la jaune ou la rouge, la clarté est moindre. Dès que la barre est ramenée au centre toutes les LED sont éteintes. Idem pour le mode MANUEL où l'on pilote "linéairement" la barre.

Traitement informatique pour barrer manuellement.

Fig.11



Pour des raisons évidente de sécurité, l'homme doit pouvoir à tout moment reprendre le contrôle sur les automatismes. Dans le cadre de ce petit projet cette faculté permet en outre de résoudre les changements de route quand la plus courte déviation fait franchir le lit du vent au voilier. Mais le mode MANUEL permet aussi "le plaisir" de barrer. Pour cette option, l'amplitude de déviation de la barre sera toujours de 30°, mais la consigne est "linéaire" et permet d'adopter toutes les positions intermédiaires. Il est évident que dans ce cas le potentiomètre devrait être basculé sur un mini manche, mais dans le cadre de ce petit programme expérimental on va se contenter du potentiomètre de consigne avec en Fig.11 la relation entre sa position angulaire et celle imposée au servomoteur. Dans la solution informatique adoptée, il faut tourner le bouton flèche vers le coté où l'on désire voir s'orienter le bateau. La Fig.12 présente la relation qu'il doit y avoir entre la valeur numérisée pour la commande de barre et la valeur de la consigne à envoyer au servomoteur. Comme déjà mentionné à plusieurs reprises en fonction du moteur utilisé il peut s'avérer nécessaire d'inverser "le sens" de variation, ce qui est le cas pour le MC-1811. C'est la ligne repérée (1) dans le listage de la procédure qui traite cet aspect du programme. Dans tous les cas il faut compenser la dissymétrie de calage du palonnier, c'est le terme correctif colorié en bleu clair dans le listage qui s'en charge. Ce décalage représenté en jaune sur la Fig.12 peut s'avérer positif ou négatif en fonction du modèle de moteur utilisé.

```
void DeterminerConsigneDeBARRE(){
  EteindreLesLEDdeDeviationBarre();
  CAP = analogRead(Commande_du_CAP); // Mesure de la "consigne CAP".
  CAP = 1023 - CAP; // Inverser le sens car Servomoteur MC-1811 utilisé.  (1)
  Consigne_Servomoteur = ((CAP *60) / 1023) + 60 + CorrectionSymetrie ; }
```

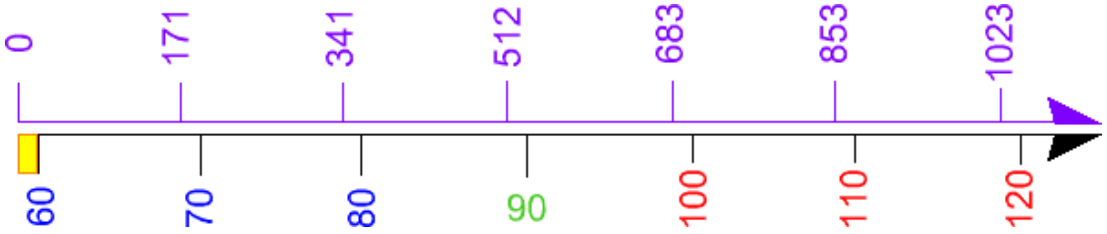
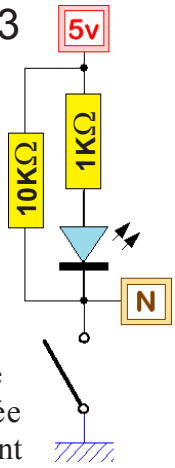


Fig.12

Traitement informatique pour compenser la dérive.

Fig.13



Exactement comme pour l'option de pilotage manuel, un inverseur utilisant le schéma donné en Fig.13 permet de valider ou non la compensation de la dérive. La LED témoin du mode compensé est allumée directement par l'inverseur ce qui économise l'utilisation d'une sortie. Pour parfaire le forçage d'un état "1" une résistance de 10KΩ est placée en parallèle avec le témoin lumineux. La Fig.14 ressemble en tout points à la Fig.11 mis à part les valeurs angulaires plus faibles pour les déviations maximale du bouton flèche. Le comportement attendu est tout à fait analogue. Pour le choix des compensations de dérive maximale, la valeur de 15° est estimée largement suffisante pour compenser un glissement latéral résultant du VENT ajouté à un éventuel courant traversier sur la zone de navigation. Le bouton flèche indique de quel coté doit être dévié le CAP pour compenser la dérive. Si cette dernière résulte du VENT, le bouton est tourné vers sa direction. Dans tout les cas on oriente la proue à l'opposé de la dérive constatée. Par exemple sur la Fig.14 la dérive est estimée aux environs de 12°. Le CAP sera donc augmenté de cette valeur vers la droite puisque le bateau glisse ver bâbord.

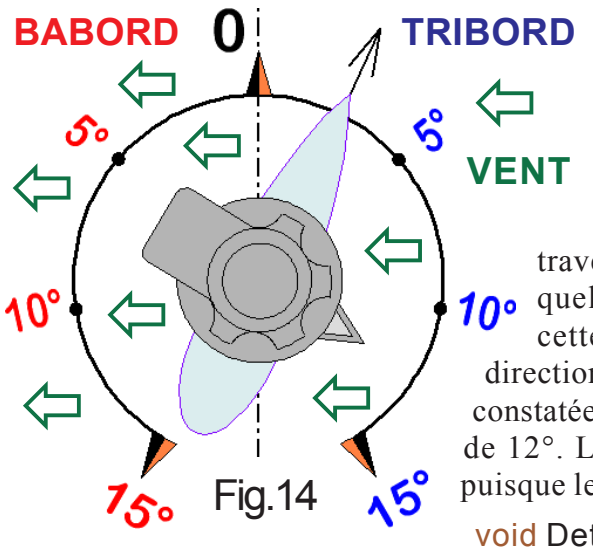


Fig.14

Le traitement informatique pour compenser la dérive est pratiquement élémentaire.

```
void DeterminerConsigneANTIDERIVE(){
  GrandeurDERIVE = analogRead(Mesurer_DERIVE) ;
  GrandeurDERIVE = ((GrandeurDERIVE * 30) / 1023) - 15 ; }
void CalculeCAPavecCompensationDERIVE() {
  CAP = CAP + GrandeurDERIVE;
  if (CAP < 0) {CAP = CAP + 360;}; }
```

Résumé des ENTRÉES / SORTIES adoptées pour ce petit projet.

ENTRÉES ANALOGIQUES :

ATTENTION : Le SERVOMOTEUR est alimenté "hors Arduino".

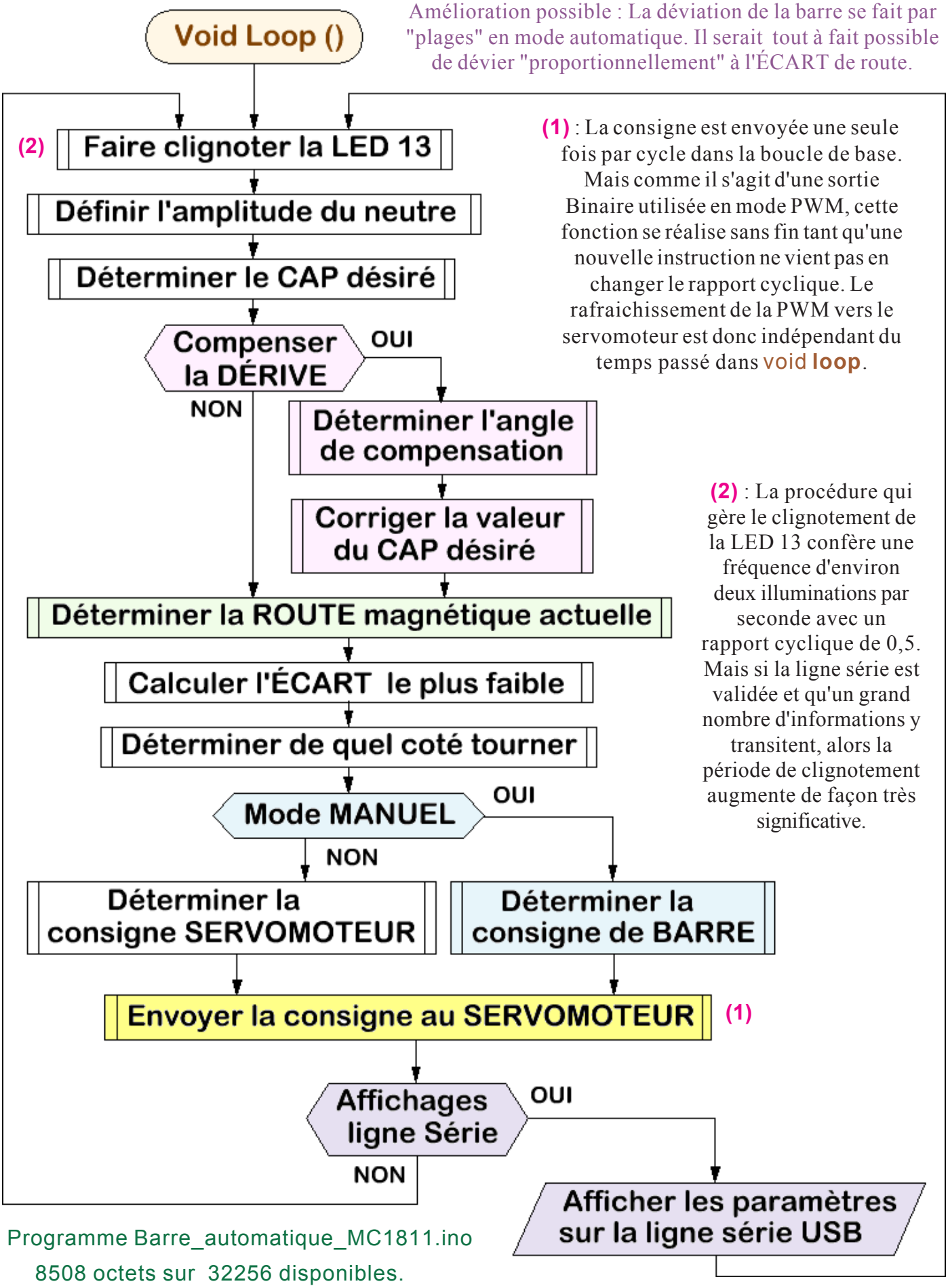
A0 : Mesure de l'amplitude angulaire pour le **Neutre**.
A1 : Mesure de la commande de **CAP désiré**.
A2 : Mesure de la **Compensation de dérive**.
A3 : Non utilisée. *(Disponible)*
A4 : SDA.
A5 : SCL.
} Dialoge SPI et boussole magnétique. *(Imposé par la librairie hmc6352.h)*

ENTRÉES / SORTIES BINAIRES :

D0 :
D1 :
} Réservées au travail sur la ligne série USB.
D2 : Broche en ENTRÉE binaire pour imposer le MODE MANUEL.
D3 : Sortie de type PWM pour piloter le SERVOMOTEUR.
D4 : Broche en ENTRÉE binaire pour compenser la DÉRIVE.
D5 : Pilotage en PWM d'une LED blanche proportionnellement à l'amplitude du **Neutre**.
D6 : LED rouge signalant consigne de barre à DROITE 30°.
D7 : LED rouge signalant consigne de barre à DROITE 20°.
D8 : LED rouge signalant consigne de barre à DROITE 10°.
D9 : LED rouge signalant consigne de barre à GAUCHE 10°.
D10 : LED rouge signalant consigne de barre à GAUCHE 20°.
D11 : LED rouge signalant consigne de barre à GAUCHE 30°.
D12 : Autorise ou bloque l'affichage en ligne série.
D13 : LED jaune qui clignote pour montrer le déroulement normal de void **LOOP**.

TRAITEMENT EFFECTUÉ DANS LA BOUCLE DE BASE

Amélioration possible : La déviation de la barre se fait par "plages" en mode automatique. Il serait tout à fait possible de dévier "proportionnellement" à l'ÉCART de route.



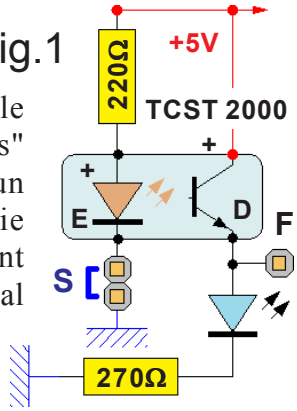
(1) : La consigne est envoyée une seule fois par cycle dans la boucle de base. Mais comme il s'agit d'une sortie Binaire utilisée en mode PWM, cette fonction se réalise sans fin tant qu'une nouvelle instruction ne vient pas en changer le rapport cyclique. Le rafraichissement de la PWM vers le servomoteur est donc indépendant du temps passé dans void **loop**.

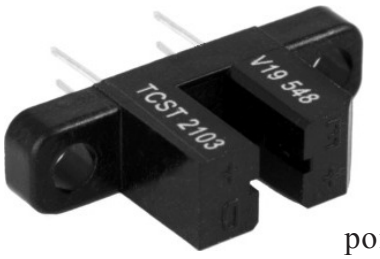
(2) : La procédure qui gère le clignotement de la LED 13 confère une fréquence d'environ deux illuminations par seconde avec un rapport cyclique de 0,5. Mais si la ligne série est validée et qu'un grand nombre d'informations y transitent, alors la période de clignotement augmente de façon très significative.

Programme Barre_automatique_MC1811.ino
8508 octets sur 32256 disponibles.

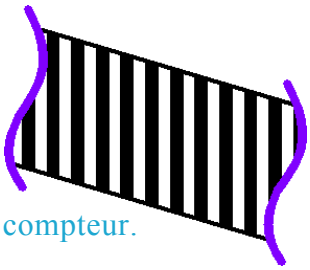
D'une utilisation simple, les fourches optiques n'imposent vraiment pas de faire appel à une quelconque librairie. Le schéma électrique adopté est donné en Fig.1 et commenté dans le livret sur les circuits imprimés. Le modèle TCST 2000 est expérimenté car il était disponible "dans les stocks de composants" mais tout autre modèle de fourche optique avec phototransistor présentera un comportement analogue. Compte tenu de la résistance de 270Ω insérée en série avec la LED bleue et sa tension de conduction, le phototransistor draine un courant d'environ 4mA. Par contre, pour fonctionner au nominal la LED interne à la fourche optique doit être alimentée aux environs de 20mA. Comme cette électronique est intégrée au petit SHIELD de développement, il n'est pas raisonnable de consommer cette énergie en permanence même si le dispositif TCST 2000 n'est pas utilisé par le programme en cours de mise au point. Sans compter que l'éclairement de la LED bleue devient une source de pollution visuelle quand la fourche optique doit être ignorée. C'est la raison pour laquelle la LED interne peut être isolée en ouvrant le pont **S**. *(STRAP)* Le petit programme Test_Fourche_Optique.ino permet d'observer le comportement du composant TCST 2000 dont la tension en sortie varie proportionnellement à l'éclairement entre 0,6Vcc et 3,1Vcc. *(Un masque triangulaire obturant progressivement le faisceau infrarouge dans le passage de la fourche)* Les valeurs de numérisation qui en découlent et qui sont visualisées sur la ligne série USB sont comprises entre environ 484 et 642. Pour en déduire un comportement binaire fiable on situe donc le seuil de basculement à la valeur moyenne de 563.

Fig.1





NOTE : Dans le programme archivé l'entrée binaire D5 est utilisée pour valider ou suspendre l'affichage sur la ligne série USB qui ralentit considérablement la rapidité possible de comptage. Pour effectuer un test de cette performance il suffit d'interdire les affichages en ligne série, d'effectuer un RESET, de procéder à une utilisation rapide de la fourche avec une bande de type "code barre", puis un grand nombre d'interception étant effectués, valider l'affichage de la valeur du compteur.



Exemple de programme :

```
/* Exemple simple de mesure sur la sortie analogique de la Fourche Optique. À partir des
   mesures effectuées qui se situent entre 484 et 642, le seuil "logique" à été placé à la
   moyenne de 563. Un compteur est incrémenté à chaque front "descendant" */

const byte Entree_mesuree = 0; // Entrée analogique 0 utilisée.
const byte LED_Arduino = 13;  // Broche 13 utilisée.
int CNA; // Mesure analogique retournée par le CNA.
long COMPTEUR = 0;
boolean Etat1; boolean Active;

void setup() {Serial.begin(19200);
 Serial.println(">>> Test de la fourche optique <<<");
 Serial.println();
 pinMode(LED_Arduino, OUTPUT); }

void loop() {
 CNA = analogRead(Entree_mesuree);
 Serial.print("Valeur du CNA : "); Serial.print(CNA);
 Serial.print(" - COMPTEUR : "); Serial.print(COMPTEUR);
 if (CNA > 563) {digitalWrite(LED_Arduino, HIGH); Etat1=true; Active=true;}
   else {digitalWrite(LED_Arduino, LOW);Active=false;};
 // ========== Compter sur le front descendant. ==========
 if ((!Active) && (Etat1))
   {Etat1=false; COMPTEUR++;} ;  }
```

Ce programme allume et éteint la DEL disponible sur la carte ARDUINO en fonction de l'état de la fourche optique. Allumée si non masquée, éteinte si faisceaux intercepté. La ligne USB retourne en permanence la valeur "analogique" retournée par le CNA. Un compteur totalise le nombre de fronts descendants détectés depuis le RESET.

Test_Fourche_Optique.ino

Étant donné qu'un Barregraphe n'est rien d'autre qu'un ensemble de LED insérées dans un boitier unique, la programmation d'une rampe lumineuse n'est pas vraiment différente de celle de plusieurs LED individuelles. Tout au plus le choix d'un composant qui utiliserait des anodes ou des cathodes communes pourrait influencer le branchement électrique et la programmation. *(Par exemple obligation de provoquer l'éclairage par un état "0" dans le cas d'anodes communes)* Ce n'est pas le cas du LTA1000HR qui intègre dix LED strictement indépendantes. Le schéma électrique montré en Fig.1 n'appelle que peu de commentaires. Les résistances de limitation de courant brident le débit total à fournir par Arduino si toutes les DEL s'allument simultanément. Ce courant maximal est un impératif à vérifier si l'application génère une "bande lumineuse" où toutes les LED peuvent être allumées en même temps. Comme nous avons le choix, les sorties d'Arduino alimentent en fournissant le +5Vcc, donc allumer va se traduire par un état "1" sur la sortie binaire concernée. C'est toujours plus "parlant" sur un listage de programme qu'avoir à coder en logique négative.

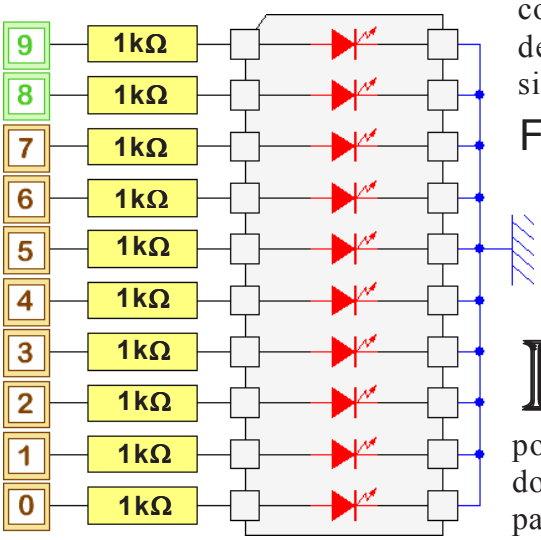


Fig.1

L'expérimentation de ce Barregraphe est l'occasion rêvée pour utiliser les instructions spécifiques à la programmation des ports d'Entrées/Sorties. Le programme Test_des_PORTS.ino donné en exemple ci-dessous démontre que si le logiciel n'utilise pas la ligne série, les broches D0 et D1 sont totalement disponibles pour servir d'Entrées ou de sorties banales. En outre, contrairement à ce qui semble affirmé sur http://arduino.cc/en/Reference/PortManipulation, l'utilisation du PORT B est totalement libre en E/S.

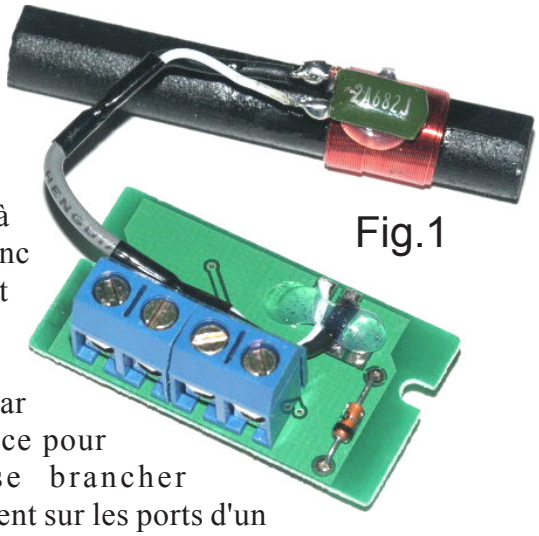
Exemple de programme :

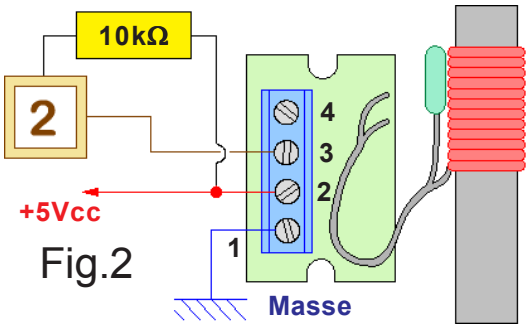
Test_des_PORTS.ino

```
/*  Programme de test des instructions travaillant sur les PORTS */

// Définition des E/S pour ce programme.
#define EntreeAnalogique 0 // Entrée analogique 0 utilisée.

// Définition des variables pour ce programme.
unsigned int Mesure; // Valeur mesurée sur l'entrée analogique.

void setup() {
DDRB = B11111111;    // Impose huit sorties sur 8 à 13.
DDRD = B11111111; }  // Impose huit sorties sur 0 à 7.

void loop() {
Mesure = analogRead(EntreeAnalogique); // Mesure analogique [0 à 1023].
Mesure = (uint32_t) Mesure * 10 / 1023; // Transpose entre 0 et 10.
switch (Mesure) {
  case 0 : PORTD = B00000000;PORTB = B00000000; break;
  case 1 : PORTD = B00000001;PORTB = B00000000; break;
  case 2 : PORTD = B00000011;PORTB = B00000000; break;
  case 3 : PORTD = B00000111;PORTB = B00000000; break;
  case 4 : PORTD = B00001111;PORTB = B00000000; break;
  case 5 : PORTD = B00011111;PORTB = B00000000; break;
  case 6 : PORTD = B00111111;PORTB = B00000000; break;
  case 7 : PORTD = B01111111;PORTB = B00000000; break;
  case 8 : PORTD = B11111111;PORTB = B00000000; break;
  case 9 : PORTD = B11111111;PORTB = B00000001; break;
  case 10 : PORTD = B11111111;PORTB = B00000011; break; } }
```

Ce programme augmente la "hauteur" de la rampe lumineuse à LED proportionnellement à la tension en entrée de la broche Analogique 0. Les états sur les orties binaires sont placés par OCTETS directement sur les registres de données des PORTS B et D.

L'émetteur des signaux horaires DCF77 est situé en Allemagne à proximité de Francfort et piloté par une horloge atomique. La stabilité de l'onde porteuse d'une fréquence "immuable" de 77,5 kHz *(D'où le nom de l'émetteur est dérivé)* est générée à partir d'une horloge atomique. Les faibles fluctuations de cette fréquence sont principalement dues à la propagation des grandes ondes dans l'atmosphère. Elle peut donc servir à calibrer un oscillateur ou tout autre dispositif nécessitant une grande précision. Le petit module vendu dans le commerce intègre une antenne ferrite avec circuit accordé, un décoder/amplificateur piloté par quartz et une interface pour pouvoir se brancher directement sur les ports d'un microcontrôleur quelconque. La Fig.2 précise les branchements élémentaires à effectuer. Outre la masse et le +5Vcc on utilise directement la sortie **3**. Comme il s'agit d'une sortie "à collecteur ouvert", il importe de forcer un état "1" à l'aide d'une résistance de 10kΩ reliée au +5Vcc. La sortie **4** n'est pas utilisée. C'est également une sortie à collecteur ouvert, mais fournissant des signaux logiques inversés. Le module peut être alimenté entre 1,2 et 15Vcc. Les sorties à collecteur ouvert peuvent commuter jusqu'à 30Vcc avec un courant maximal de 1mA.





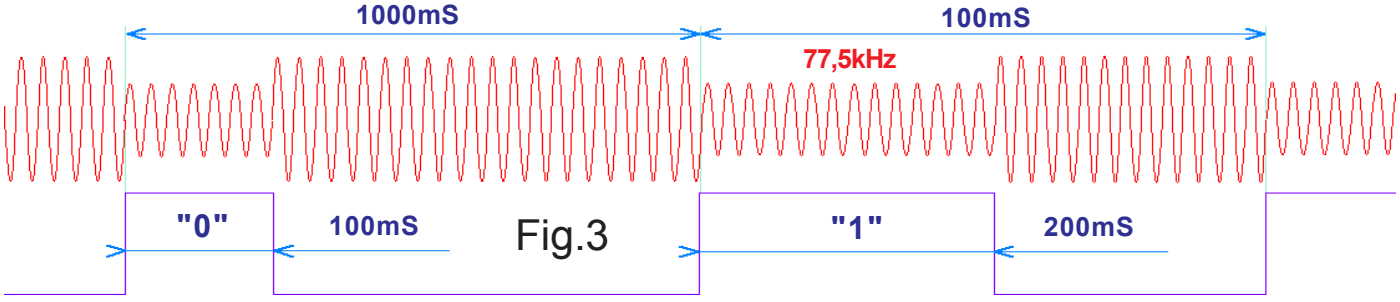
Comme toute antenne de type "cadre ferrite", l'axe longitudinal doit être placé perpendiculairement à la direction de la station à capter pour optimiser le niveau de réception. La Fig.2 ne le montre pas, mais la bibliothèque Funkuhr.h outre le décodage du signal binaire issu de DCF, recopie sur la LED d'Arduino en sortie D13 les impulsions à l'état "1" de l'entrée binaire D2. C'est d'autant plus nécessaire que **le transistor de sortie du module électronique ne peut pas drainer plus de 1mA**. Quand l'antenne est correctement orientée, on doit observer des impulsions lumineuses régulières une fois par seconde. La sortie du signal issu de DCF77 du module de réception de la Fig.1 est connectée sur l'entrée binaire n°2, étant également une entrée d'interruption. Sur http://www.idreammicro.com/post/dcf77-arduino on trouve un exemple d'utilisation et toutes les informations utiles avec des détails relatifs à la bibliothèque utilisée.

Le signal horaire.

La porteuse de 77,5 kHz est modulée en amplitude par les signaux horaires codés en BCD au moyens d'impulsions au rythme d'une par seconde. Ces impulsions Se traduisent chaque seconde par une diminution de 25% de l'amplitude du signal reçu. La durée d'une impulsion, détermine l'état du bit reçu : 100 ms correspondent à un bit "0" et 200 ms à un bit à l'état logique "1". *(Voir Fig.3)*

Les informations sont émises par cycles inclus dans une trame étalée sur une minute. Chaque trame est divisée en soixante impulsions, chacune d'entre elles débutant par le front de l'impulsion. Pour l'élément n°59 il n'y a pas d'impulsion afin de permettre au décodeur de repérer la fin de la trame en cours. L'impulsion suivante détermine le début de la trame suivante. ***Toute absence d'impulsion plus grande que 999 ms doit donc être considérée comme le début d'une nouvelle trame.***

Conformément aux lois internationales d'occupations des fréquences, un émetteur doit préciser régulièrement son identité. L'information "DCF77" a été retenue et est émise en code Morse trois fois chaque heure durant les minutes 19, 39 et 59. *(Durant les secondes 20 à 32)* L'identificateur est envoyé



entre deux impulsions, par abaissement de l'amplitude de 25 % au rythme de 250 Hz sans interrompre le signal normal. Cette procédures n'interfère pas sur la réception du signal horaire. Ce signal Morse est en général filtré par l'électronique du récepteur, on ne le retrouve pas sur l'information logique.

La trame à décoder.

La synchronisation des récepteurs se fait sur le premier bit. (BITn° 0 : M) L'apparition de la première impulsion marque le début d'une nouvelle minute. Les informations transmises pendant une minute correspondent à l'heure qu'il sera au moment du prochain "top départ".

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
M	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
R	A1	Z1	Z2	A2	S	1	2	4	8	10	20	40	P1	1
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
2	4	8	10	20	P2	1	2	4	8	10	20	1	2	4
45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
1	2	4	8	10	1	2	4	8	10	20	40	80	P3	---

0 : (**M**) Début d'une trame. Toujours à l'état "0".
1 à **14** : Utilisation diverses : Alertes civiles, données météo etc.
15 : (**R**) L'émetteur de réserve est actif lorsque ce bit est à "1". *(Émetteur de secours)*
• En cas de maintenance ou de dysfonctionnement, un émetteur de secours est mis en service.
16 : (**A1**) Annonce de l'heure d'hiver.
17 et **18** : (**Z1** , **Z2**) : Fuseau horaire actuel : **0-1** : CET = UTC + 1h **1-0** : CEST = UTC + 2h.
19 : (**A2**) Une seconde va être supprimée pour corriger les irrégularités de la rotation de la Terre.
20 : (**S**) Bit de début de codage des informations horaires. Toujours à l'état "1".
21 à **27** : Minutes codées en BCD, bits de poids faibles en premier.
28 : (**P1**) Bit de parité pour les minutes. *(Parité PAIRE pour les bits de 21 à 27)*
29 à **34** : Heures codées en BCD, bits de poids faibles en premier.
35 : (**P2**) Bit de parité pour les heures. *(Parité PAIRE pour les bits de 29 à 34)*
36 à **41** : Jours codées en BCD, bits de poids faibles en premier.
42 à **44** : Jour de la semaine codé en BCD, bits de poids faibles en premier.
45 à **49** : Mois codées en BCD, bits de poids faibles en premier.
50 à **57** : Année *(Sur deux chiffres)* codée en BCD, bits de poids faibles en premier.
58 : (**P3**) Bit de parité pour la date. *(Parité PAIRE pour les bits de 36 à 57)*
59 : Pas d'impulsion pour marquer la fin de trame. *(Marque de la minute)*

Exemple de décodage :

La date et l'heure sont en Décimal Codé Binaire avec les poids binaires 1, 2 , 4, 8, 10, 20, 40, 80.
1 0 0 1 0 1 0 : Minutes = 1 + 8 + 20 = 29.
Les bits 21 à 27 à l'état "1" sont impairs. Le bit de parité **P1** doit être à "1" pour rétablir la PARITÉ.
0 1 0 0 0 1 : Heures = 2 + 20 = 22.
Les bits 29 à 34 à l'état "1" sont pairs. Le bit de parité **P2** doit être à "0" pour maintenir la PARITÉ.
Il n'est pas possible de faire fonctionner entièrement une horloge avec le signal radio, car la réception peut être interrompue ou parasitée. Une horloge radio-pilotée classique est régulée par quartz, avec mise à l'heure ou corrections éventuelles à l'aide du signal radio DCF77.
La parité est indispensable pour vérifier s'il n'y a pas eu d'erreur pendant la réception et ainsi ne pas afficher une valeur incorrecte. Il est aussi recommandé de vérifier que le bit 20 est bien à "1".
En cas de problème de réception il est fréquent de se retrouver avec des dates et heure du genre : 45/25/2165 à 45h85, ce qui arrive quand sur parasite les bits reçus sont tous plus ou moins à "1".

EXEMPLE DE PROGRAMME : Decodage_DCF.ino

Un petit projet complet avec affichage sur LCD.

Assez proche du projet avec détermination de la date et de l'heure utilisant un circuit DS13-02, ce programme est plus simple car il n'y a pas à gérer le clavier pour la remise à l'heure. En revanche, comme la bibliothèque Funkuhr.h ne fournit pas une variable indiquant le jour de la semaine, la date est indiquée de façon plus élémentaire. Le programme **RECEPTION_DCF77.ino** assure le décodage des signaux de la station radio et indique la date sur la ligne du haut de l'afficheur LCD et l'heure sur la ligne du bas. Si le programme est en mode "analyse" l'affichage est celui de la Fig.4 avec sur la ligne du bas le chronométrage depuis la mise sous tension ou depuis un RESET. Quand les signaux sont correctement reçus, l'affichage prend l'allure montrée sur la Fig.5 avec formatage des données sur deux lignes. Les zéros en tête sont remplacés par des espaces. Si la réception est correcte, la synchronisation prend en général moins de trois minutes.

Fig.4

Fig.5

IMPORTANT : La perturbation du récepteur DCF77 par les parasites est infiniment moins importante si on relie la masse d'Arduino à la prise de terre de la fiche du secteur 220V.

Comme expliqué en page 20 de ce document et reprécisé sur la Fig.6, le "Sheild" LCD a été modifié pour piloter le rétro-éclairage avec la broche binaire 13 qui sur le circuit imprimé était la mieux disposée. Mais la bibliothèque Funkuhr.h utilise aussi cette sortie pour allumer une LED au rythme des impulsions issues du récepteur de démodulation et ainsi évaluer la qualité de réception. On pourrait en déduire qu'il va en résulter une interférence, et que le rétro-éclairage va clignoter à la cadence des impulsions détectées. Dans la pratique on constate qu'il n'en est rien. Quand la sortie 13 d'Arduino fournit un état "1" pour allumer le LED, ce dernier est sans effet puisque bloqué par la diode **D**. L'état "0" pour éteindre la LED devrait aussi éteindre l'arrière écran,

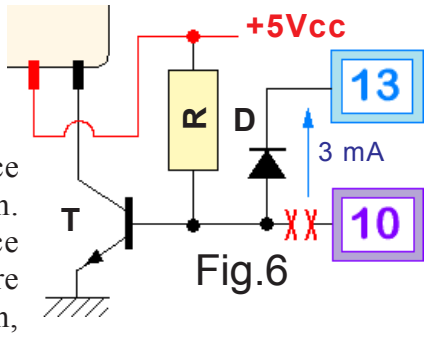


Fig.6

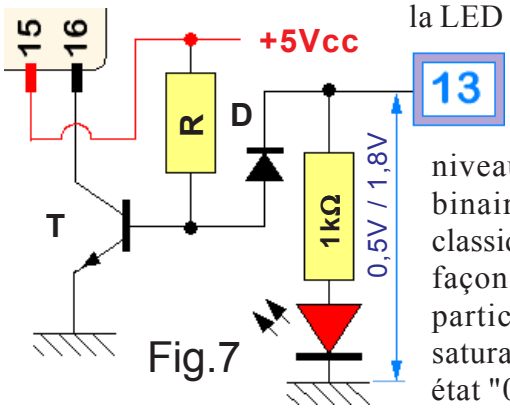


Fig.7

mais on constate qu'il n'est rien. La Fig.7 représente la branche électronique relative au pilotage de la sortie 13 avec en bleu clair la tension mesurée sur cette broche pour les états "0" et "1". Ces niveaux électriques ne sont pas du tout conformes à ceux d'une broche binaire programmée en sortie. Du reste si on utilise les instructions classiques le rétro-éclairage clignote effectivement et la LED s'illumine de façon plus intensive. La gestion de cette sortie par Funkuhr.h est donc particulière, car ces tensions mesurées ne peuvent s'expliquer par une saturation de la sortie du µP qui dans le pire des cas fournit 3mA pour un état "0" et un courant du même ordre de grandeur pour allumer la LED.

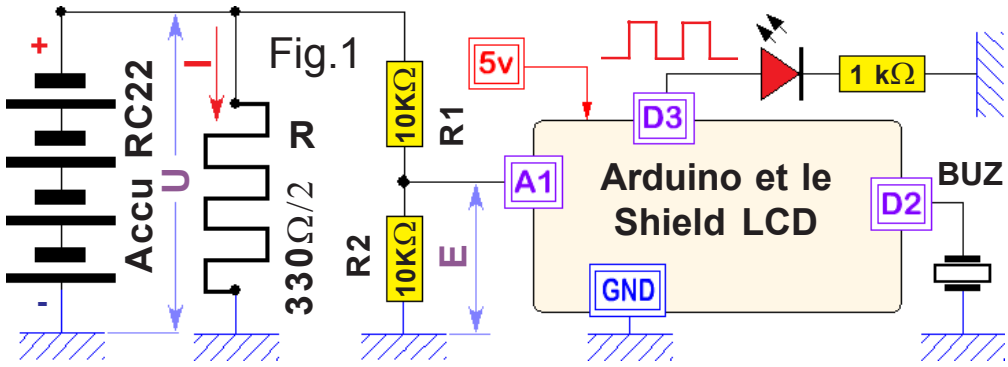
Globalement le programme d'application **RECEPTION_DCF77.ino** se contente de réunir les fonctions d'affichage LCD de DS1302_et_affichage_LCD.ino et d'ajouter les routines de gestion du décodage de la station DCF77. Toutefois, par rapport au petit programme expérimental Decodage_DCF.ino on se sert de la faculté de pouvoir utiliser les résistances PULL UP internes de l'ATMEGA328 pour faire l'économie de la résistance de forçage de 10kΩ sur l'entrée D2 :

```
void setup(void) {dcf.init();
PORTD = PORTD | B00000100; } // Validation de la PULL UP interne au µP.
```

On peut remarquer au passage que la sélection du texte pour le mois est effectuée avec un switch au lieu d'une suite de if comme c'était le cas dans LCD de DS1302_et_affichage_LCD.ino ce qui contribue à un programme peut être plus lisible et fait gagner huit octets. En outre l'information void confirme le non passage de paramètre dans quelques procédures.

C'est une application purement "utilitaire" qui, outre un SHIELD LCD, n'utilise que quelques composants externes pour être opérationnelle. Elle consiste à mesurer par décharge "complète" la capacité d'une pile rechargeable de type RC22 de tension nominale 8,4v. Naturellement, avant de procéder à la mesure, l'échantillon en test doit avoir été totalement rechargé. La mesure s'achève quand la tension aux bornes passe en dessous d'une valeur prédéterminée de 6,9v. L'accumulateur reste en décharge, mais un bruiteur prévient qu'il faut venir le débrancher. La décharge se fait sous un courant moyen de 50mA. L'accumulateur peut être au Ni-Cd ou de technologie MH. Tout composant dont la tension aux bornes ne dépassera pas 10v peut être testé quelle que soit sa capacité, mais si le test dépasse 9 heures, l'affichage sera perturbé sur le LCD. Il est conseillé d'alimenter la carte Arduino en autonome, pour ne pas perdre le résultat des mesures quand la tension sur l'échantillon s'effondre en fin de décharge.

Méthode retenue :



La Fig.1 ci-contre présente le schéma électrique retenu pour mettre en œuvre cette petite application. Noter que le bruiteur **BUZ** est de type actif, un signal logique d'état "0" le fait fonctionner et il produit une tonalité assez pénible. Comme précisé en page 6, utilisé directement il consomme 20 mA. Pour en atténuer la puissance une LED verte est placée en série. C'est la résistance **R** de sollicitation qui dose le courant **I** de décharge. Elle est choisie à 165Ω pour engendrer un courant moyen d'environ 50mA. Ce courant est fonction de la tension que présente l'accumulateur au cours du processus. En début d'expérience la tension avoisine 9,2v. Le courant qui en résulte fait 56mA. Mais en fin de décharge elle tombe à 7v pour un courant de 42 mA. La mesure s'achève quand la tension passe en dessous du seuil choisi à 6,9v. Pour que la tension aux bornes de la batterie soit compatible avec les limites imposées par l'entrée analogique A1, on divise par deux la tension avec les résistances **R1** et **R2**. Pour afficher la valeur réelle sur le LCD il suffit par calcul de multiplier la valeur déduite de la mesure par deux. Par définition la capacité d'une batterie ou d'un accumulateur est :

Capacité *(En Ampèreheures)* = **Intensité** *(En Ampères)* x **Temps** *(En heures)*

Mais la capacité totale résulte d'un calcul intégral durant toute la décharge, car l'intensité n'est pas constante au cours du temps. La méthode consiste à calculer Intensité x Temps à chaque seconde écoulée et à cumuler dans **Capacité**. Pour une seconde de temps écoulé, la capacité devient :

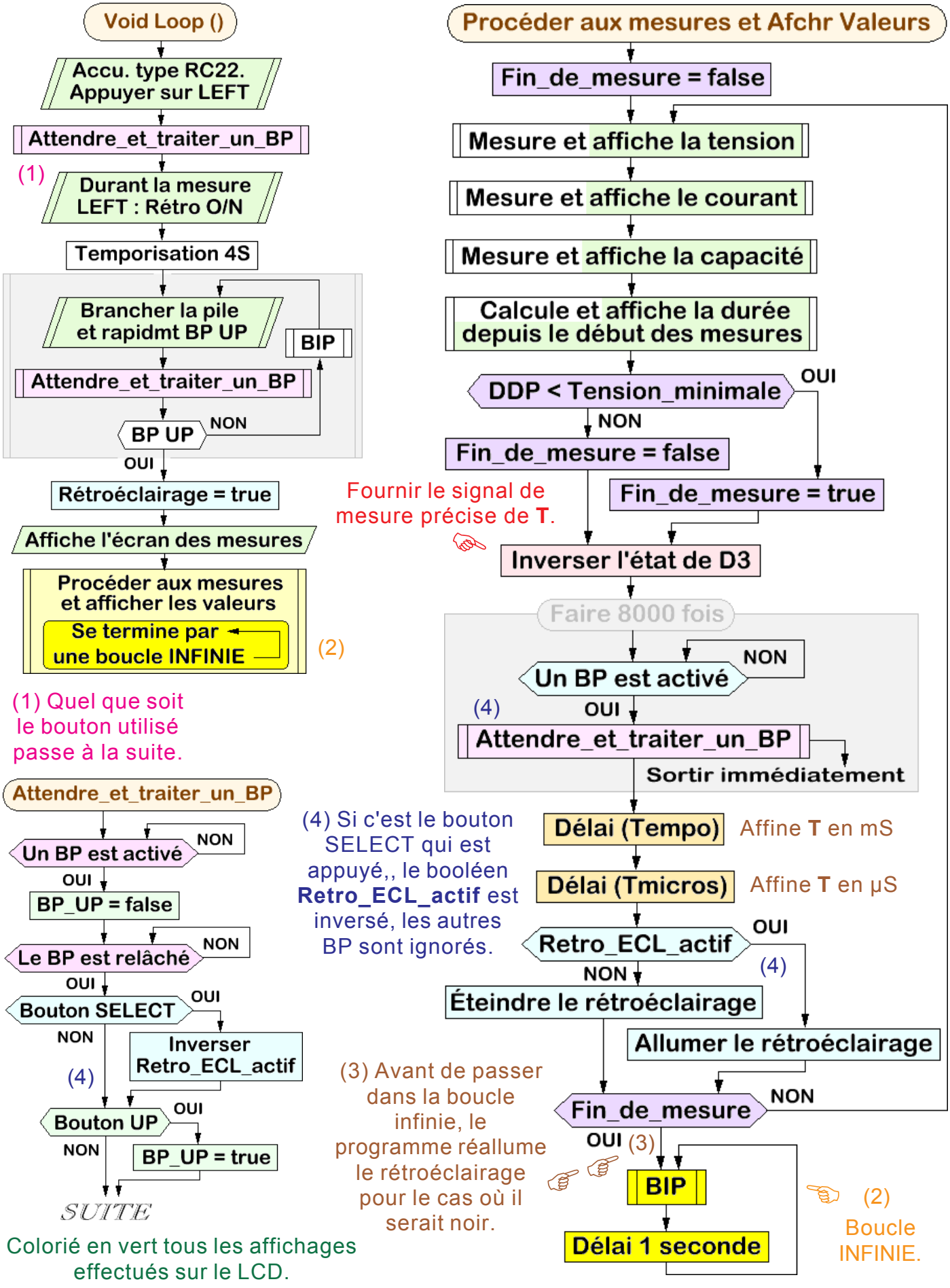
```
Capacite = Capacite + (Courant_traversant_R / 3600);
```

Le résultat ne sera fiable que si la période **T** entre deux cumuls soit exactement d'une seconde. Le logiciel Mesure_Capacite_Acc_Rechargeables.ino utilise deux variables pour ajuster le plus précisément possible la valeur de **T**. Avec les valeurs utilisées dans le programme la période **T** varie entre 1,000036S et 0,999983S. *(Soit entre +36µS et -17µS)* Comme c'est la valeur la plus grande qui s'impose en grande proportion, la valeur affichée pour le résultat sera au moins égale à celle de la réalité. La résistance de décharge **R** doit dissiper sous forme de chaleur une puissance moyenne de 8,2V x 0,05A soit environ 0,5W. Choisir un modèle en conséquence. Pour répartir la chaleur deux résistances de 330Ω sont placées en parallèle. Inutile de mesurer la valeur de l'intensité débitée par l'accumulateur. Elle se calcule facilement avec **I** = **E** / **R** sachant que **E** = **U** / 2. On en déduit l'instruction qui va calculer l'intensité :

```
Courant_traversant_R = (DDP / Resistance_Accu_MH) * 1000;
```

Dans le programme, Resistance_Accu_MH est défini comme constante et vaut 165. *(Valeur de **R**)* Le facteur de multiplication 1000 permet d'exprimer la faible capacité de ces piles rechargeables en mAh et non en Ah. DDP pour Différence De Potentiel représente la tension aux bornes de l'accumulateur notée conventionnellement **U** dans les formules. Le programme est résumé par l'organigramme de la page 54.

Comportement global du programme Mesure_Capacite_Acc_Rechargeables.ino :



Cette application est strictement analogue à celle de la page 53 et s'apparente à un clone mis à part quelques petits détails relatifs au courant de décharge et à quelques valeurs dans les constantes du programme. Examinons les différences concernant le schéma électrique ainsi que les données électriques qui en résultent. Le premier changement impératif consiste à diviser la tension batterie par quatre, comme montré sur la Fig.1 et non par deux, car il ne faut pas dépasser 5V pour **E** sur l'entrée A1, sachant qu'une batterie au plomb entièrement chargée peut présenter une tension dépassant les 16V. Comme on décide utiliser des résistances identiques pour **R1** à **R4**, il faut descendre à ce rapport de division. Pour afficher la valeur réelle sur le LCD il suffit par calcul de multiplier la valeur déduite de la mesure analogique par quatre. Comme pour les piles rechargeables, avant de procéder à la mesure, l'échantillon en test doit avoir été totalement rechargé. La mesure s'achève quand la tension aux bornes passe en dessous d'une valeur prédéterminée de 10V. La batterie reste en décharge, et il faut la libérer rapidement et la remettre en charge quand le bruiteur prévient qu'il faut venir la débrancher. En effet, en dessous de 11V aux bornes une batterie au plomb se dégrade lentement. La décharge pour être significative doit se faire sous un courant de plusieurs ampères. Le courant prévu était initialement de 6 ampères, compte tenu de la combinaison possible de résistances de forte puissance disponibles dans les réserves de composants électriques. La combinatoire montrée sur la Fig.2 conduit théoriquement à une résistance de 2Ω. Mais le montage "volant" utilise des fils un peu longs et surtout des branchements provisoires à l'aide de "pinces crocodiles". Les test avec une alimentation de laboratoire permettent d'évaluer à 2,4Ω la valeur réelle quand les résistances parasites sont prises en compte. La tension nominale faisant approximativement 12,5V, la majorité de la décharge se fait sous 5,2A. Toute batterie peut être testée quelle que soit sa capacité, mais si le test dépasse 9 heures, l'affichage sera perturbé sur le LCD. Pour cette durée on aboutit à une capacité de 52Ah ce qui couvre largement les besoins courants. Il est conseillé d'alimenter la carte Arduino en autonome, pour ne pas perdre le résultat des mesures quand la tension sur l'échantillon s'effondre en fin de décharge. Les formules de calcul restent identiques mis à part le coefficient multiplicateur 1000 qui ne s'impose plus puisque l'on travaille ici avec les unités fondamentales.

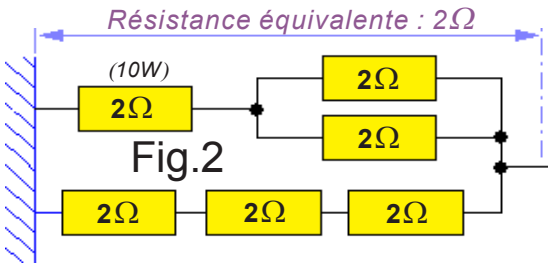


Fig.2

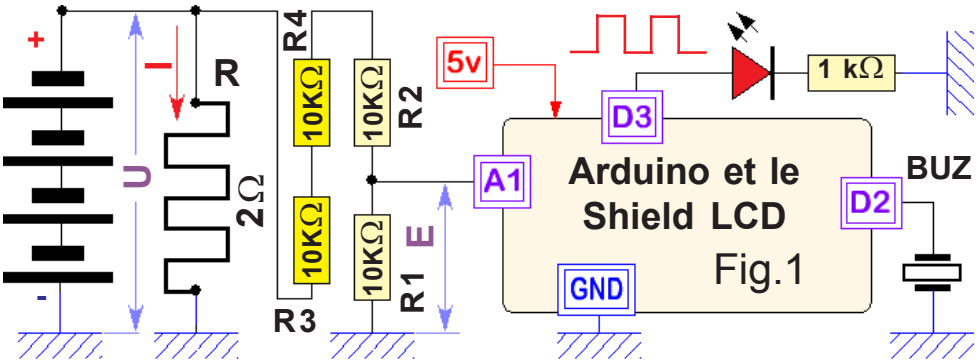


Fig.1

Capacité *(En Ampèreheures)* = **Intensité** *(En Ampères)* x **Temps** *(En heures)*

La méthode consiste toujours à déterminer Intensité x Temps à chaque seconde écoulée et à cumuler cet incrément dans la variable **Capacité**. Pour une seconde de temps écoulé, la capacité devient :

```
Capacite = Capacite + (Courant_traversant_R / 3600);
```

Comme pour les batteries rechargeables, le programme Mesure_Capacite_BAT_Pb_12v.ino utilise deux variables pour ajuster le plus précisément possible la valeur de **T**. Les constantes sont légèrement modifiées de celle du logiciel homologue car les temps de calcul et d'affichage sont légèrement différents. Avec les valeurs utilisées dans le programme la période **T** se stabilise globalement à 1,000042S *(Soit +42µS)* La valeur affichée pour le résultat sera par conséquent au moins égale à celle de la réalité. La résistance de décharge **R** doit dissiper sous forme de chaleur une puissance moyenne de 12,5V x 5,2A soit la puissance non dérisoire de 65W. Non seulement il faut choisir des modèles en conséquence, mais la température des ces derniers va s'avérer importante avec tous les risque que cela induit. Il importe donc de prendre toutes les précautions qui s'imposent. Inutile de mesurer la valeur de l'intensité débitée par la batterie. Elle se calcule avec **I** = **E** / **R** sachant que **E** = **U** / 4 et que pour **R** on tient compte des chutes de tension en ligne :

```
Courant_traversant_R = (DDP / Resistance_decharge_BAT);
```