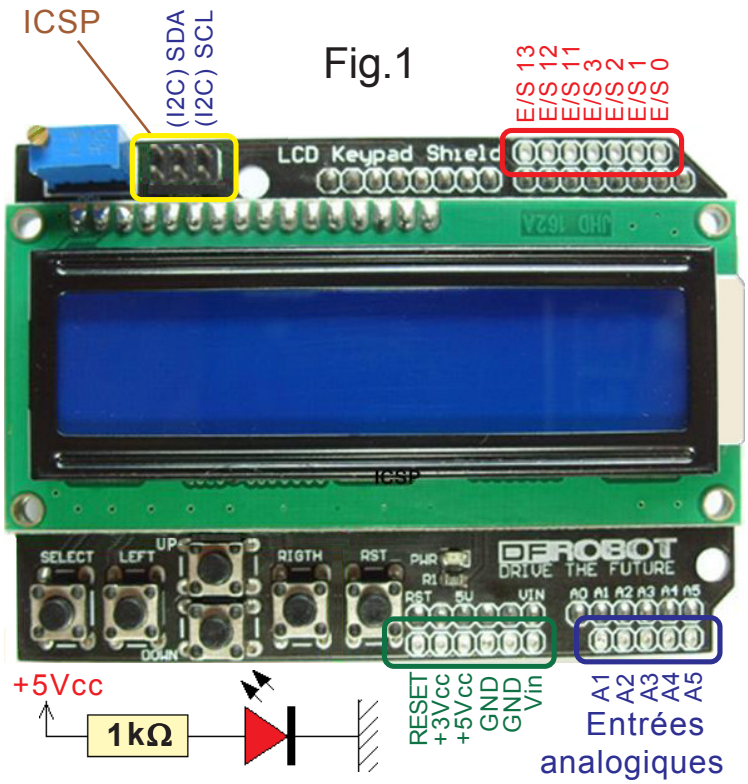



Fig.1

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**.

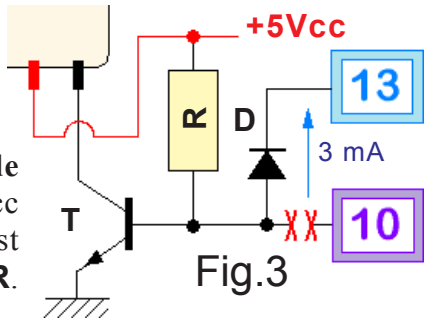



Fig.3

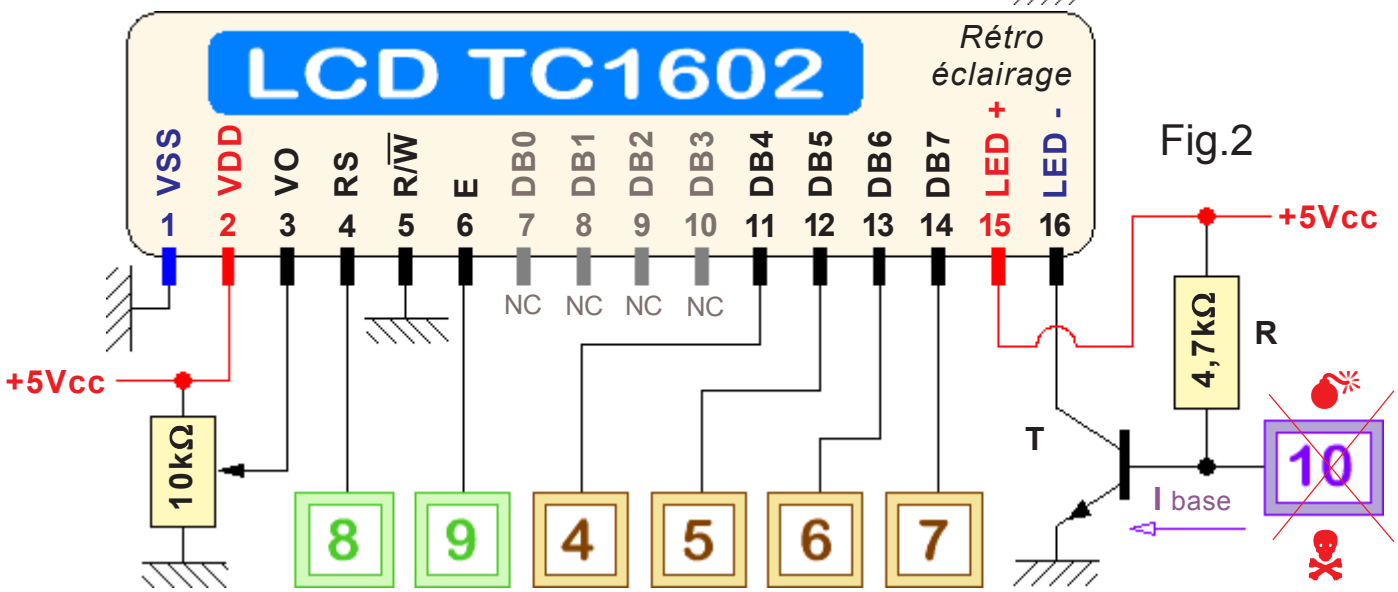



Fig.2

**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.

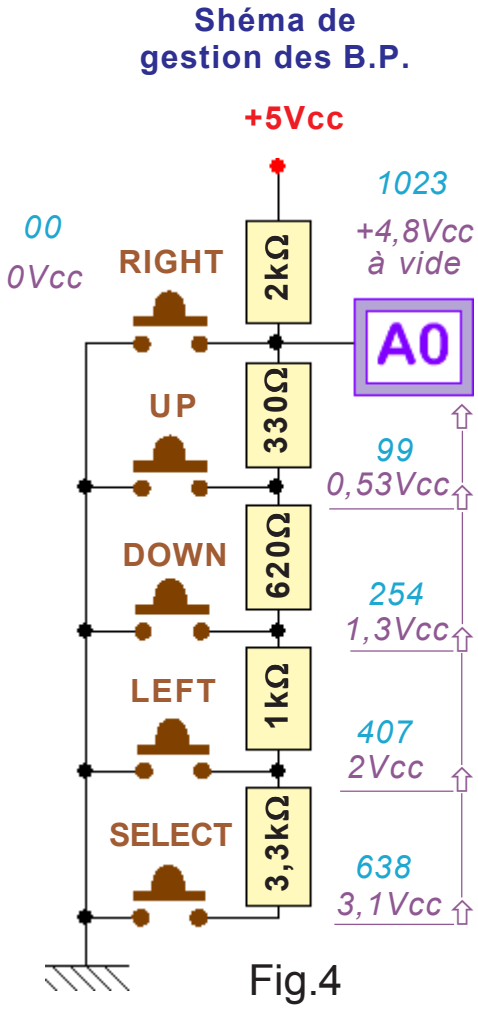
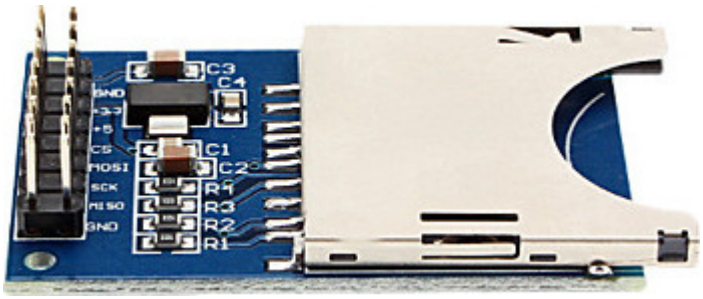


Fig.4

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

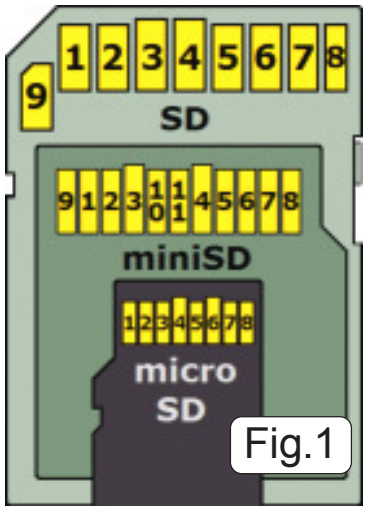


Fig.1

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.

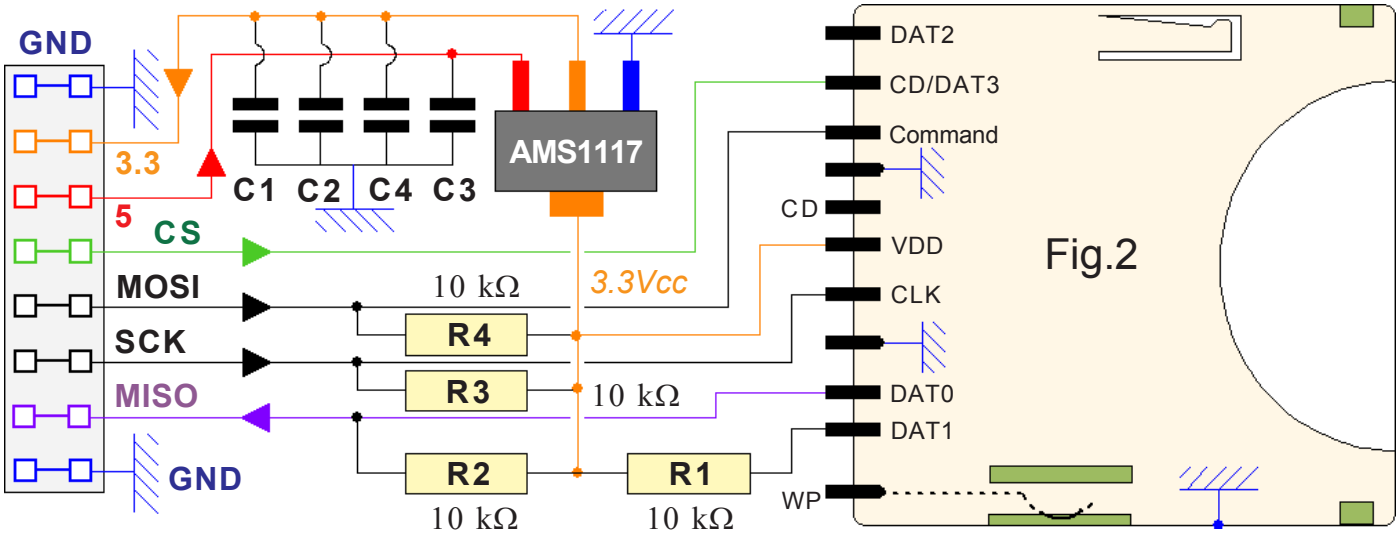


Fig.2

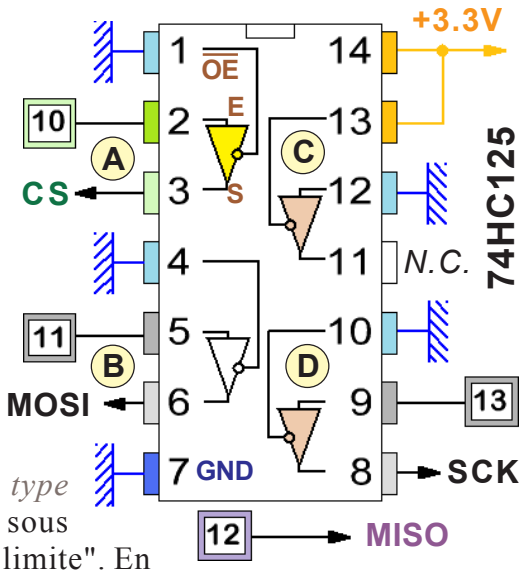
## Problème d'interfaçage avec la carte ARDUINO.

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.

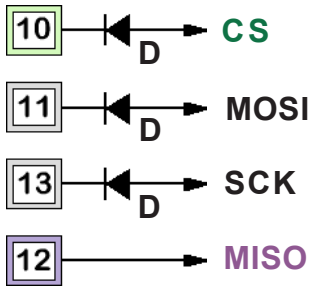
Fig.3




### *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

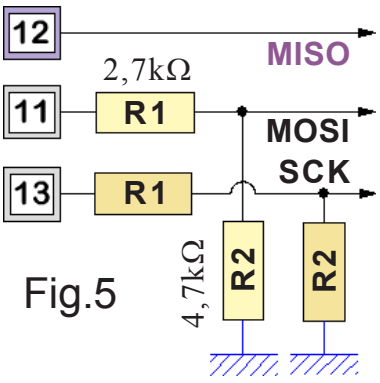
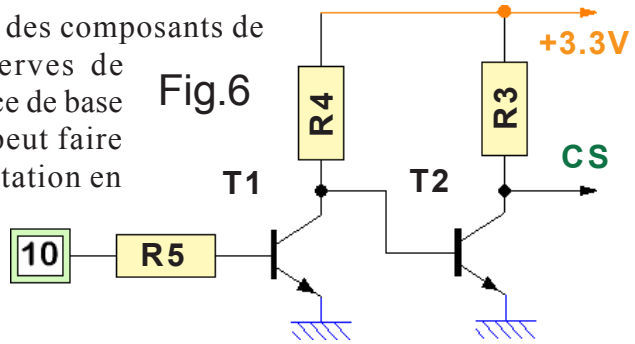



Fig.5

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



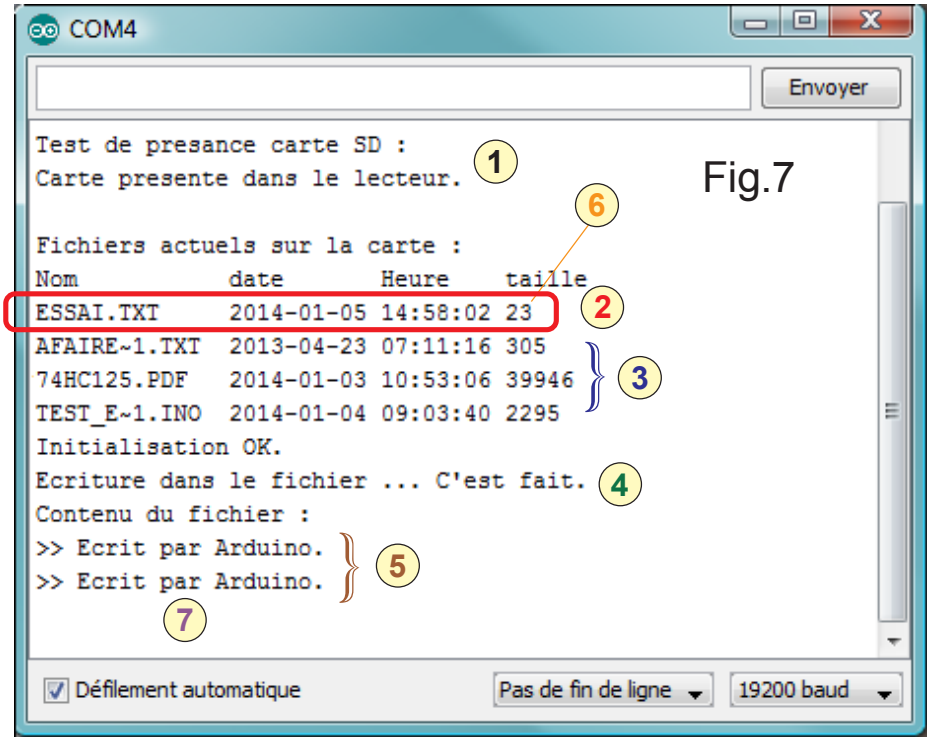

Le plus simple de tous est celui avec les trois diodes, mais s'il ne convient pas, il serait préférable d'utiliser la solution n°1. Ce n'est que par manque de disponibilité du 74HC125 qu'il serait envisageable de mettre alors en œuvre la solution avec deux transistors.

## 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.

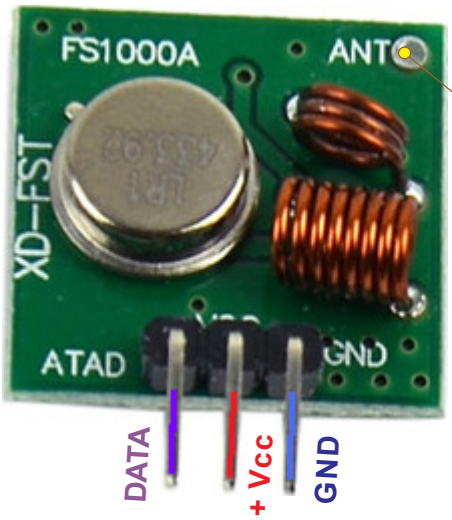
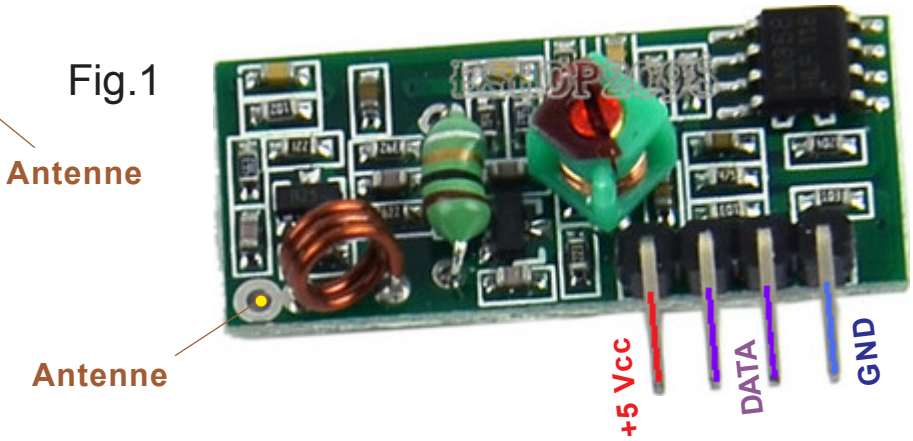



Fig.1




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.

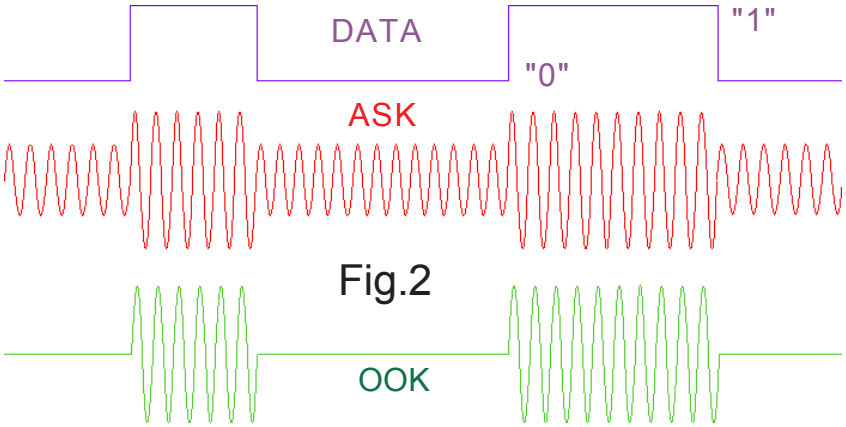



Fig.2

## 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.
```

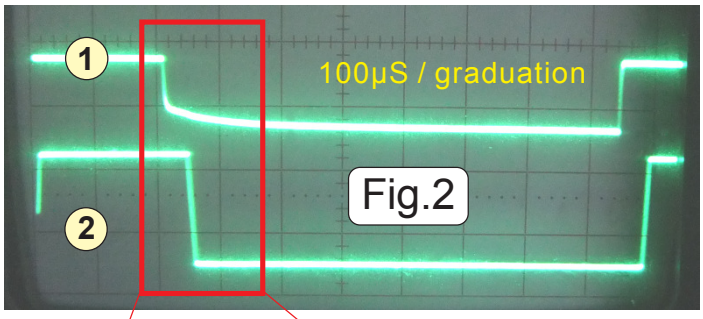
Ce petit programme génère comme montré sur la Fig.2 des impulsions de 250 µs suivies d'un état logique zéro pour aboutir à un cycle d'une milliseconde. Il y a génération de ces créneaux courts durant environ une seconde. Puis, toujours sur une seconde il y a création des impulsions longues de la Fig.3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

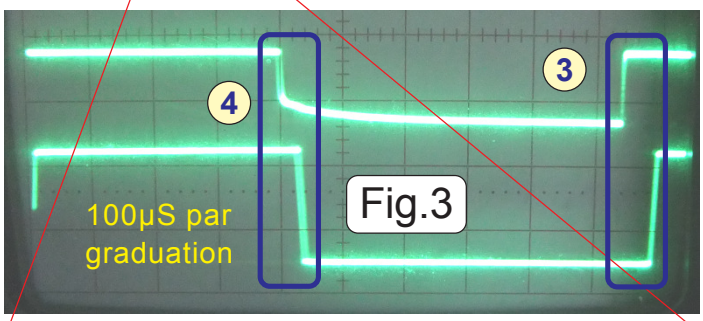
### Programme sur l'émetteur pour utiliser l'oscilloscope :

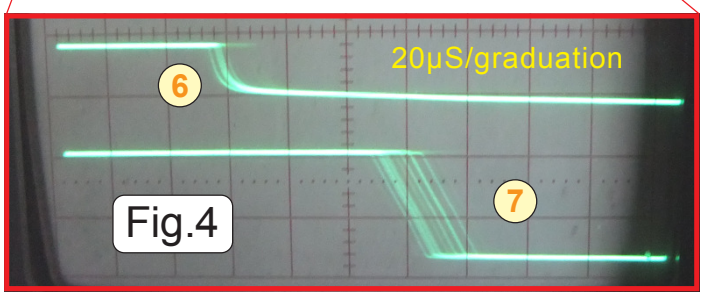
TEST_TX_433MHz_oscilloscope.ino

```
/* 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.

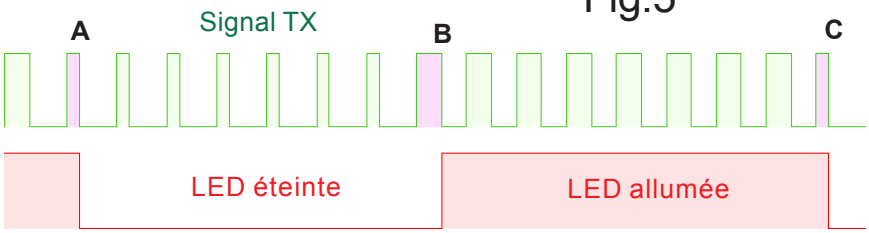
## Programme pour tester le récepteur :

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.***

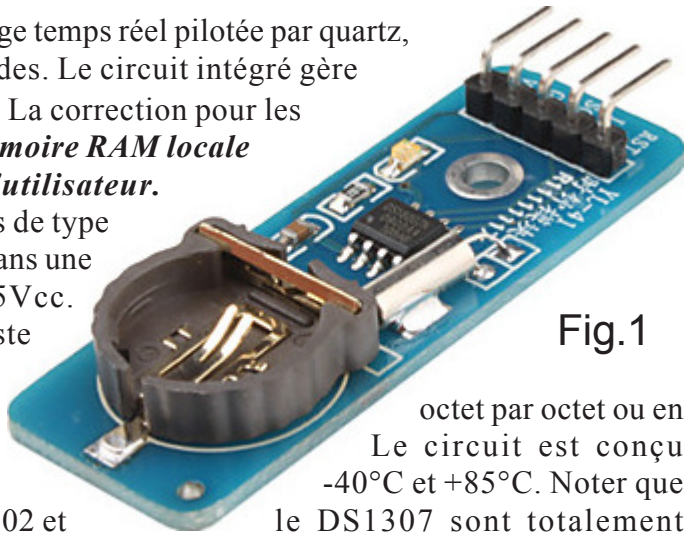


Fig.1

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.

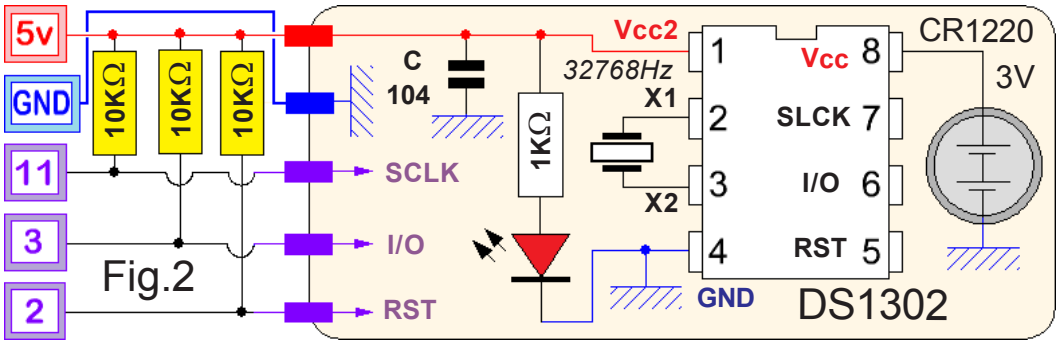



Fig.2

( 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.*

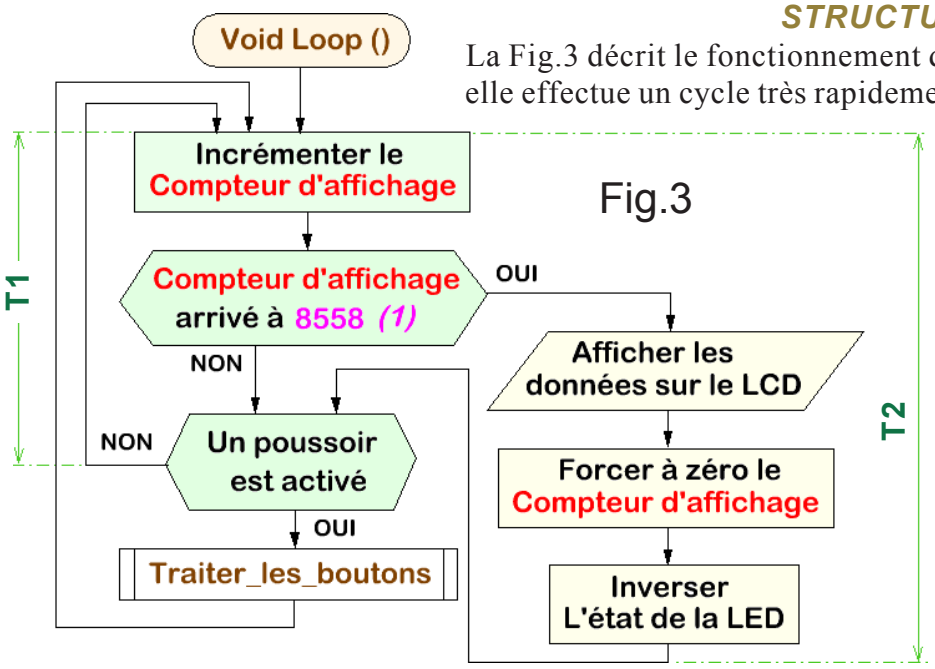


Fig.3

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.

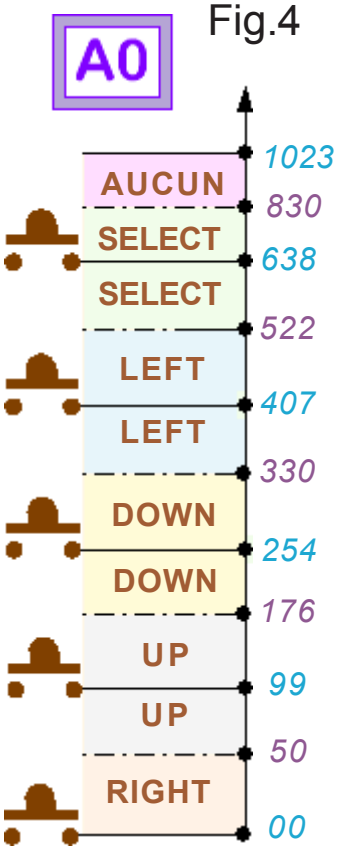


Fig.4

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

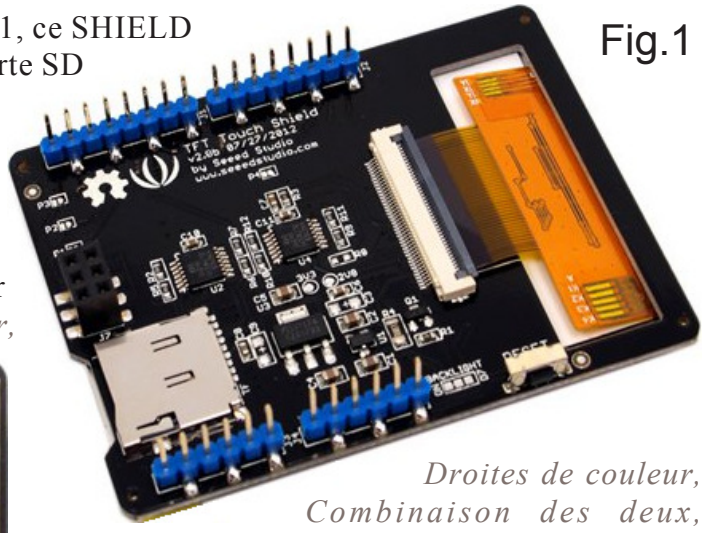
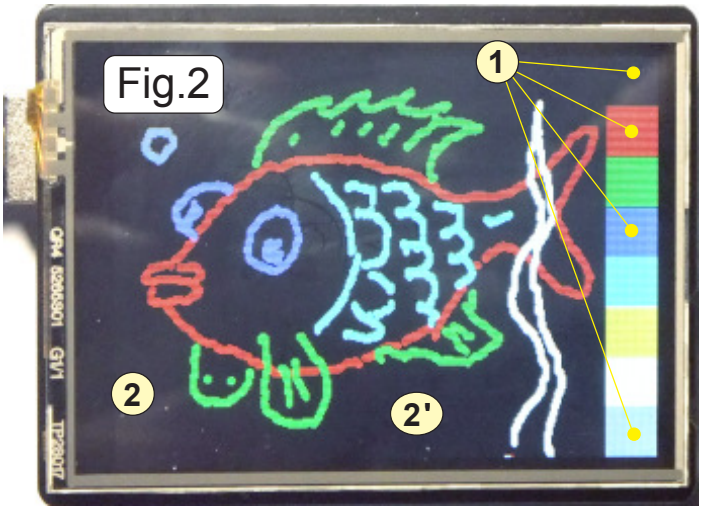


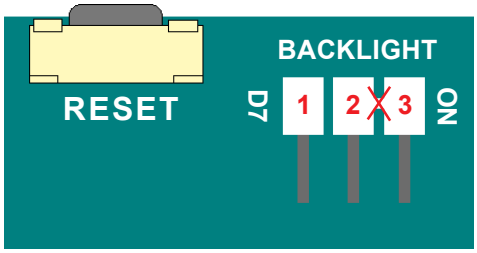

Fig.2




## 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.*

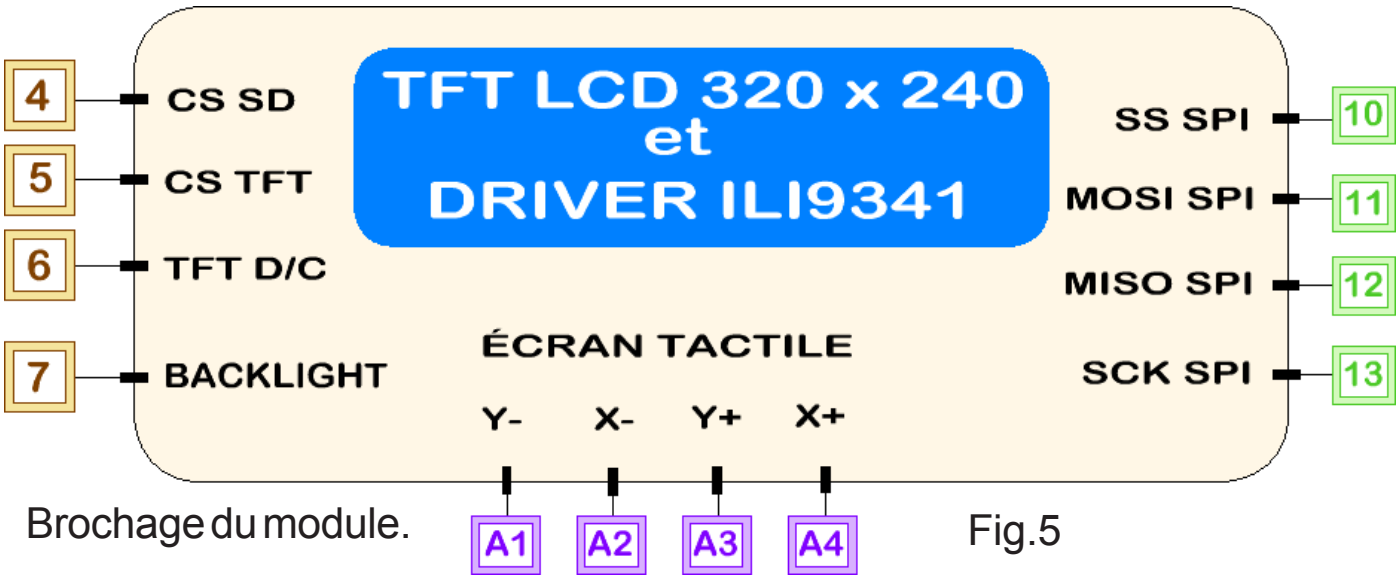
## Pilotage du rétro éclairage.

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.



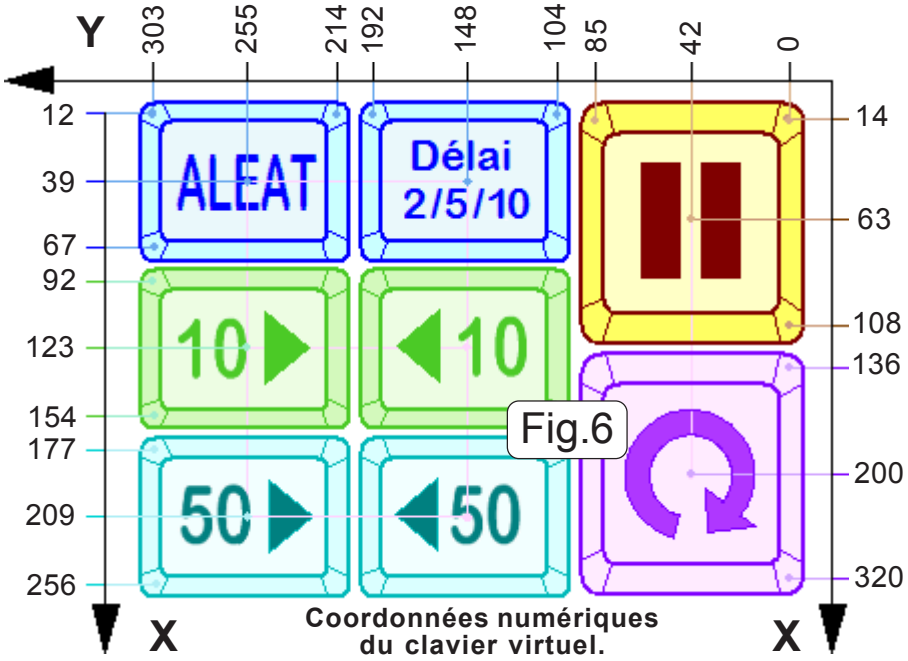

Brochage du module. Fig.5

**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.

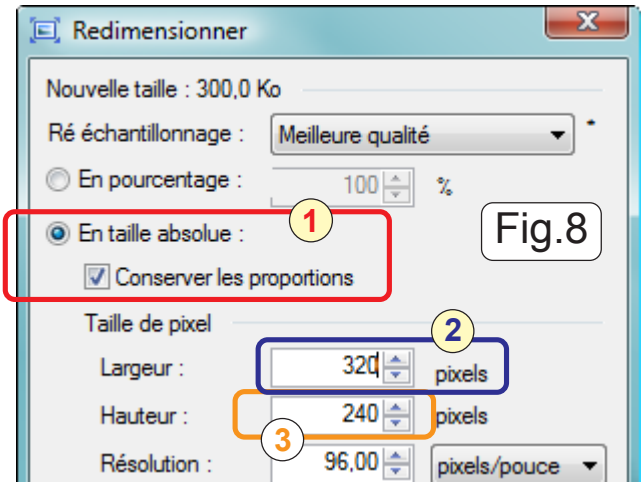


## Traiter une image pour la visionner sur le petit écran LCD.

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.

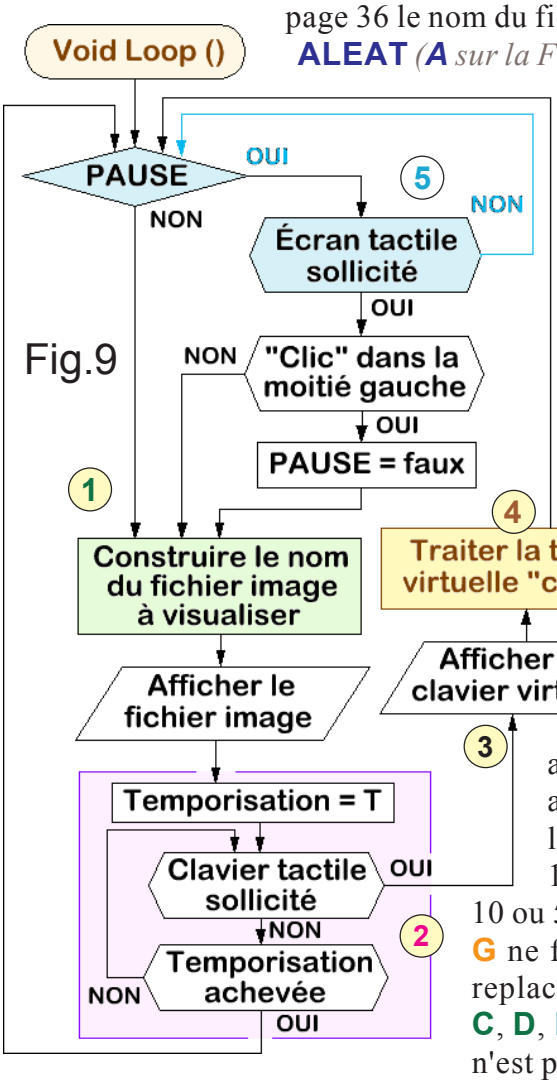


Fig.9

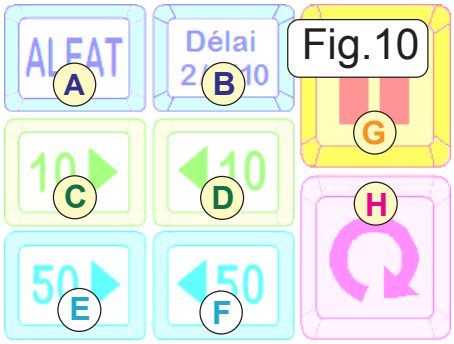


Fig.10

**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.