Utilisation et programmation de la carte ESPLORA d'ARDUINO

www.arduino.cc

Table des matières

Utilisation et programmation de la carte ESPLORA d'ARDUINO

www.arduino.cc

1 - Introduction

Avantages :

- système libre, opensource et bon marché, nombreuses ressources sur internet.
- Langage de programmation basé sur le C.
- Platine riche de plusieurs capteurs (micro, potentiomètre linéaire, capteur de température, de lumière, accéléromètre x,y,z, 4 boutons, joystic xy+bouton) et composants (Leds, buzzer) déjà en place. Un écran TFT couleur incluant un lecteur carte SD peut se monter sur les connecteurs prévus pour cela.
- Communication avec le PC via le cable USB,
- Possibilité d'interfaçage avec Processing (programme graphique libre sur PC).
- 4 ports de connection TInkerkit (2 sorties PWM et 2 entrées digitale-analogiques)

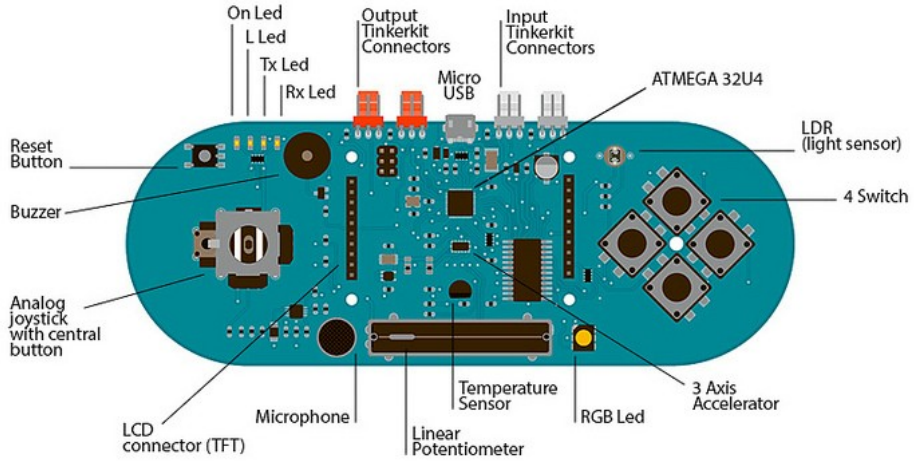
Inconvénients :

- Documentations et exemples simples, donc limités pour des utilisations pointues (projet complexes, interruption timer...).
- L'écran TFT couleur ou la librairie ne permettent pas une récupération de la couleur de chaque pixel. Il faut garder une "image" de son affichage dans le programme si on veut créer un affichage dynamique correct (effacement position précédente => affichage nouvelle position).
- Le type des images (Pimage) n'est pas documenté et il n'est pas possible de faire un traitement de l'image lue sur la carte SD rotation, réduction. Elle est lue et affichée à un endroit donné de l'écran, c'est tout..
- Les fonctions graphiques TFT sont très limitées (affichage de point, cercle, rectangle et ligne), en plus de l'affichage d'image BMP couleur 24bit. Les cercles et les rectangles peuvent être colorés automatiquement, comme le fond d'écran, mais c'est tout. Sinon il faut se calculer le coloriage.

1 - Caractéristiques

Microcontroller	ATmega32u4
Operating Voltage	5V
Flash Memory	32 KB of which 4 KB used by bootloader
SRAM	2.5 KB
EEPROM	1 KB
Clock Speed	16 MHz

2 - Équipements disponibles, en entrées et sorties



The design of the Esplora board recalls traditional gamepad design with an analog joystick on the left and four pushbuttons on the right.

The Esplora has the following on-board inputs and outputs :

- **Analog joystick with central push-button** two axis (X and Y) and a center pushbutton.
- **4 push-buttons** laid out in a diamond pattern.
- **Linear potentiometer** slider near the bottom of the board.
- **Microphone** for getting the loudness (amplitude) of the surrounding environment.
- **Light sensor** for getting the brightness.
- **Temperature sensor** reads the ambient temperature
- **Three-axis accelerometer** measures the board's relation to gravity on three axes (X, Y, and Z)
- **Buzzer** can produce square-waves.
- **RGB led** bright LED with Red Green and Blue elements for color mixing.
- **2 TinkerKit Inputs** to connect the TinkerKit sensor modules with the 3-pin connectors.
- **2 TinkerKit Outputs** to connect the TinkerKit actuator modules with the 3-pin connectors.
- **TFT display connector** connector for an optional color LCD screen, SD card, or other devices that use the SPI protocol.

Leds

- ON [green] indicates whether the board is receiving power supply
- L [yellow] connected directly to the microcontroller, accessible through pin 13
- RX and TX [yellow] indicates the data being transmitted or received over the USB communication

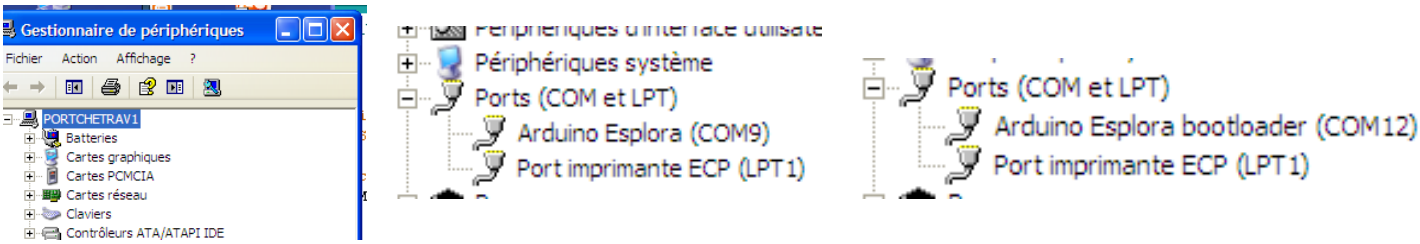
3 - installation

Les drivers sont intégrés au logiciel Arduino à partir de la version 1.0.2.
Voir http://arduino.cc/en/Main/Software et http://arduino.cc/en/Guide/ArduinoEsplora

Il est nécessaire de faire une installation des drivers 2 fois ! :

1. Lors du branchement de la carte
2. Lors du téléchargement du premier programme.

En effet, lors du téléchargement d'un programme, la carte est déconnectée, l'interface de téléchargement est connectée (bootloader), d'oû la deuxième installation de drivers. Ce dernier n'a pas le même USB-COM que la carte ESPLORA.



Après le téléchargement, le bootloader est déconnectée et la carte ESPLORA reconnectée. Cela se passe rapidement.

La Led L (jaune) à coté de la Led verte (ON) clignote 5 fois avant la déconnexion de la carte ESPLORA pour la connexion du Bootloader et le téléchargement du programme qui suit.

4 - Préliminaire Esplora à mettre dans tous les programmes selon le cas :

	au tout début :	dans le void setup() :
Pour la Carte ESPLORA :	**#include <Esplora.h>**	
Pour utiliser l'écran :	**#include <SPI.h>** **#include <TFT.h>**	**EsploraTFT.begin();**
Et pour la carte SD :	**#include <SD.h>**	**SD.begin(8);**

2 - Utilisation des éléments disponibles

1 - Led RGB (à droite du curseur linéaire) :

```
Esplora.writeRGB(red, green, blue); //  red, green, blue, valeurs comprises entre 0 et 255.
Esplora.writeBlue(255);             //  Led en bleu plus ou moins fort. Valeurs comprises entre 0 et 255.
Esplora.writeGreen(255);            //  Led en vert plus ou moins fort. Valeurs comprises entre 0 et 255.
Esplora.writeRed(255);              //  Led en rouge plus ou moins fort. Valeurs comprises entre 0 et 255.
```

2 - Buzzer

```
Esplora.noTone();        // Arrêt du buzzer
Esplora.tone(frequence, duree);      // buzzer à la fréquence f pendant un certain temps
```

Marche si une autre notre note n'est pas jouée avant la fin !
Pour jouer plusieurs notes successives, il faut mettre des pauses entre, par exemple (mauvais) avec `delay(duree)`.

3 - joystick

```
Esplora.readJoystickX(); // renvoie un entier de -512(à droite), à 512(à gauche), le neutre n'est pas à 0
Esplora.readJoystickY(); // renvoie un  entier de -512(en haut), à 512(en bas), le neutre n'est pas à 0
```
Il faut relever la position médiane au début du programme si on a besoin d'un neutre à zéro.

4 - Bouton de joystick

```
Esplora.readJoystickSwitch();  // renvoi un entier 1023 (pas appuyé) ou 0 (appuyé).
Esplora.readJoystickButton(); // LOW when pressed, HIGH when not pressed
```

5 - un microphone,

```
Esplora.readMicrophone();  // mesure de l'intensité sonore de 0 à 1023.
```

6 - un capteur de température,

```
Esplora.readTemperature(DEGREES_C) ;
```
*ou "*DEGREES_F" renvoi en entier *de -40°C to 150°C*

7 - un accéléromètre 3 axes,

```
Esplora.readAccelerometer(X_AXIS);      // read the X axis      -149(-g), +170(+g)
Esplora.readAccelerometer(Y_AXIS);      // read the Y axis      -149(-g), +177(+g)
Esplora.readAccelerometer(Z_AXIS);      // read the Z axis      -185(-g), +138(+g)
```

8 - Quatre boutons poussoirs disposés en losange,

```
Esplora.readButton(1);        // renvoie la valeur 0 (appuyé) ou 1 (non appuyé)  pour le bouton du bas
Esplora.readButton(2);        // renvoie la valeur  0 (appuyé) ou 1 (non appuyé)  pour le bouton de gauche
Esplora.readButton(3);        // renvoie la valeur  0 (appuyé) ou 1 (non appuyé)  pour le bouton du haut
Esplora.readButton(4);        // renvoie la valeur  0 (appuyé) ou 1 (non appuyé)  pour le bouton de droite
```

Autres arguments aussi valables :
SWITCH_1 or SWITCH_DOWN SWITCH_2 or SWITCH_LEFT SWITCH_3 or SWITCH_UP SWITCH_4 or SWITCH_RIGHT

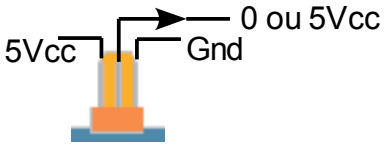
9 - Potentiomètre linéaire

```
Esplora.readSlider();    // renvoie un entier entre 1023 (à gauche) et 0 (à gauche)
```

10 - capteur de lumière

```
Esplora.readLightSensor(); // lecture du capteur de lumière (de 0 noir à 255 pleine lumière)
```

11 - Connectivité TinkerKit OUT (orange)

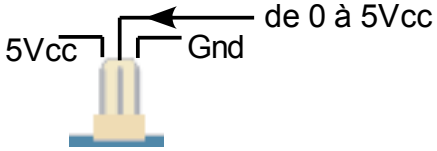


Ces deux connecteurs sont reliés directement (broche du milieu) à deux bornes du microcontrôleur. Elles sont référencées (IO 11 et 3 dans Arduino), ce sont des bornes d'entrée-sorties digitales.
On peut donc imposer une tension de 0V ou 5V sur ces broches configurées en sorties pour commander ou alimenter (puissance très faible) ce qu'on veut.
La commande de ces sorties peut être binaire avec la fonction `digitalWrite()` ou modulée en PWM cadencée à 976,5 Hz avec un certain rapport cyclique (voir chapitre 6) avec la fonction `analogWrite()`.
On peut aussi les utiliser en entrées digitales après reconfiguration de ces broches en entrée. On peut alors lire un niveau logique binaire avec digitalRead() ou mesurer la durée d'impulsion sur l'état haut ou bas par exemple avec `pulseIn()`.

Exemple de pilotage de ces sorties en tout ou rien, ou PWM (voir paragraphe 6.1 pour les explications) :

```
void setup() {
      pinMode(11, OUTPUT); //11 pour Tinkerkit outB, 3 pour TinkeKit outA
      pinMode(3, OUTPUT); //11 pour Tinkerkit outB, 3 pour TinkeKit outA
}

void loop()
{
analogWrite(11,200);     // envoi du PWM sur la sortie TinkerKit OutB,
                         // rapport cycle r = 200/255, fréquence par défaut à près de 1kHz
digitalWrite(3,HIGH);    // Mets la sortie TinkerKit OutA à l'état haut (+5Vcc)
delay(500);              // Pause syndicale pour vous laisser le temps d'apprécier
analogWrite(11,100);     // envoi du PWM sur la sortie TinkerKit OutB,
                         // rapport cycle r = 100/255, fréquence par défaut à près de 1kHz
digitalWrite(3,LOW);     // Mets la sortie TinkerKit OutA à l'état bas (0Vcc)
delay(500);              // Pause syndicale pour vous laisser le temps d'apprécier
}
```

12 - Connectivité TinkerKit IN (blanc)



Ces deux connecteurs sont reliés par leurs broches (celle du milieu seulement) à un multiplexeur qui sélectionne ce qui doit être branché à l'entrée analogique A4 du micro-contrôleur. Tous les capteurs et boutons passent par ce multiplexeur (voir dernière page). Les deux connecteurs sont reliés à la borne A4 si la valeur binaire décimale envoyé au multiplexeur est 8 (IN-A) ou 9 (IN-B),
Via l'entrée A4 et ces deux connecteurs, on peut mesurer une tension comprise entre 0 et 5 V avec une résolution de 10 bits (valeur numérique comprise entre 0 et 1023) avec la fonction `analogRead()`, ou bien un niveau logique binaire 0 (0v) ou 1 (5V) avec la fonction `digitalRead()`, ou mesurer la durée d'impulsion sur l'état haut ou bas par exemple avec `pulseIn()`.

Exemple de paramétrage pour lire une tension analogique sur le connecteur Tinkerkit in-A :

```
//paramétrage du multiplexeur pour lire sur l'entrée Tinkerkit IN-A ou IN-B
byte channel = 8 ; // 8 pour tinkerkit in-A,  9 pour tinkerkit IN-B
pinMode(A0, OUTPUT);
pinMode(A1, OUTPUT);
pinMode(A2, OUTPUT);
pinMode(A3, OUTPUT);
digitalWrite(A0, (channel & 1) ? HIGH : LOW);
digitalWrite(A1, (channel & 2) ? HIGH : LOW);
digitalWrite(A2, (channel & 4) ? HIGH : LOW);
digitalWrite(A3, (channel & 8) ? HIGH : LOW);

//lecture de la valeur sur l'entrée (A4) reliée au multiplexeur
int val = analogRead(A4); // mesure analogique (de 0 à 1023)
int val2 = digitalRead(A4); // mesure digitale (0 ou 1)
int val2 = pulse(A4,1);   // mesure la durée à l'état haut d'une pulsation en µs
```

3 - programmation

La structure des programmes Arduino est un peu particulière et s'éloigne, en apparence, des structures habituelles du langage C. La syntaxe est la même qu'en langage C.

Au début du programme, la déclaration des librairies utilisée par le programme et à compiler avec le programme.
Exemple :

```
#Nom_de_la_librairie1.h
#Nom_de_la_librairie2.h
#Nom_de_la_librairie3.h
```

Ensuite on déclare les variables et constantes communes à toutes les fonctions du programme.
Exemple :

```
int curseur;
String Mon_texte = "Vive la carte ESPLORA !";
char text_tab[Mon_texte.length()];
```

Ensuite, dans tout programme Arduino, au moins deux fonctions (même si elles restent vides) sont **indispensables** :

void **setup() {}**

La fonction **setup()** est appelée au démarrage du programme. Cette fonction est utilisée pour initialiser des composants (écran, carte SD, connexion série usb,...), définir le sens (entrée ou sortie) des broches (pin) digitales qui resteront fixes. **La fonction setup() n'est exécutée qu'une seule fois**, après chaque mise sous tension ou reset (réinitialisation) de la carte.

void **loop() {}**

La fonction loop() s'exécute en boucle sans fin. Après l'initialisation du système (fonction setup), elle permet d'exécuter la partie répétitive d'un programme, correspondant au fonctionnement d'un processus automatisé. C'est en quelque sorte le programme principal.

1 - Les types de variables disponibles avec Arduino IDE

Par défaut, les variables de type **char**, **int** et **long** contiennent des entiers relatifs (variables signées). En leur ajoutant le préfixe "**unsigned**", elles deviennent des variables entières positives (comprises entre 0 et 2.NMAX+1).

Type de variable	*Caractéristiques*	*Amplitude de variation*
boolean	Chiffre binaire (occupe quand même 1 octet de mémoire)	**0, 1**
char	Nombre entier relatif (signé*) de 8 bits (1 octet)	**–128** à **127**
byte (unsigned char)	Nombre entier positif 8 bits (1 octet)	**0 à 255**
int	Nombre entier relatif (signé*) de 16 bits (2 octets)	**–32768** à **32767**
word	Nombre entier positif 16 bits (2 octets)	**0 à 65535**
long	Nombre entier relatif (signé*) de 32 bits (4 octets)	**–2147483648** à **2147483647**
float (double)	Nombre réel à virgule flottante (4 octets) "float" ou "double", la précision n'est que de **6 à 7 chiffres** décimaux significatifs (mantisse), le type double n'apportant rien (alors qu'il est normalement codé sur 8 octets).	**–3,4x10^{38}** à **3,4x10^{38} (la mantisse et l'exposant sont signés)**
Tableaux de N éléments	Collection de variables ou de constantes accessibles à l'aide d'un numéro indiquant leur **rang** dans le tableau (indiquée entre crochets, de 0 à N-1). Exemple : tab[5]	
String (Chaîne de caractères)	Chaine de caractères (tableau de variables de type **char** se terminant par la valeur numérique **zéro**). Un caractère unique s'écrit **'c'** alors qu'un mot s'écrit **"mot"**.	

2 - Expressions numériques entières

Les expressions numériques entières sont des nombres entiers utilisés directement dans un programme (123 par exemple). Par défaut, ces nombres sont traités :

- en base 10,
- comme des variables de type **int** (entier relatif de 2 octets).

Dans Arduino IDE, les expressions numériques entières peuvent être écrites de trois manières distinctes. Ces trois écritures pouvant coexister dans le même programme, choisissez celle qui vous convient le mieux :

Ecriture des nombres entiers, selon la base utilisée :

Base	*Syntaxe*	*Exemples*
Décimale	Le nombre est écrit directement	123
Binaire	Le nombre est précédé de **0B** (**zéro et B**) **Seuls les caractères 0 et 1 sont valides**	**0B10001110**
Octale	Le nombre est précédé de **0** (**zéro**) **Seuls les caractères 0 à 7 sont valides**	**0147**
Hexadécimale	Le nombre est précédé de 0x **Les caractères 0 à 9 et A à F (ou a à f) sont valides**	**0xB5 ou 0xb5**

3 - Déclarer de nouvelles constantes

La déclaration d'une nouvelle constante s'effectue par la directive (**sans** point-virgule final) :
#define *nom_de_la_constante valeur_numérique_associée*
Exemple : #define pi 3.14

4 - Fonctions basiques

- for (int i=0; i <= 255; i++){}
- if (condition) {}
- if (condition) {} else {}
- switch (var) {

```
case label:
  // statements
  break;
case label:
  // statements
  break;
default:
  // statements
}
```

- while(expression){}
- do {} while (test condition);
- **Goto label;** // sends program flow to the label

label: //etiquette dans le programme

5 - "Mappage"

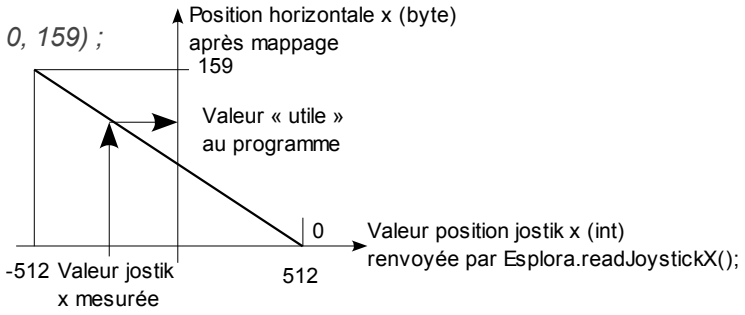
Le mappage est une conversion de la valeur d'une variable dans une échelle et un type donné, en une autre échelle, d'un autre type (si besoin). Cela revient à appliquer une loi affine de conversion à une donnée (le changement de type est une possibilité, il est utilisé si c'est nécessaire) pour obtenir sa valeur correspondante dans une nouvelle échelle.

syntaxe :

nouveau_type nouvelle_variable = map(ancienne_variable, mini_ancien, maxi_ancien, mini_nouveau, maxi_nouveau);

Exemple :
byte positionX = map(Esplora.readJoystickX(), 512, -512, 0, 159) ;

*Esplora.readJoystickX() renvoie un entier (**int**) que le mappage réchelonne et transforme en Byte.*

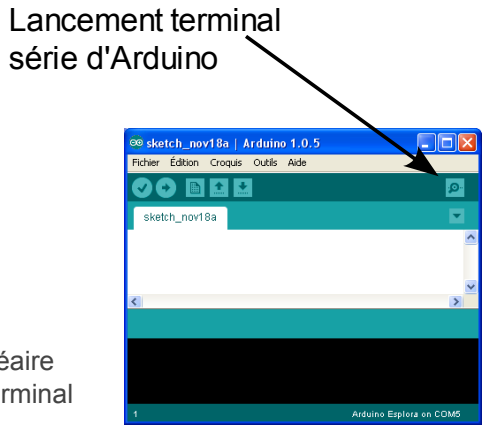


6 - Terminal Série

Pour débugger un programme, le terminal série est un atout précieux. Il permet de lire la valeur des variables, de vérifier des points de passage dans le programme.

La carte ESPLORA peut transmettre et recevoir des informations au/du PC via le câble série USB. Les informations envoyées par la carte ESPLORA peuvent être directement visualisées sur le terminal série du logiciel Arduino, ou un autre logiciel libre comme Processing.
Les informations sont transmises sous forme d'octets **en code ascii (pas d'accents**). La vitesse de transmission doit être paramétrée et configurée à l'identique des deux cotés.

Pour envoyer des informations :

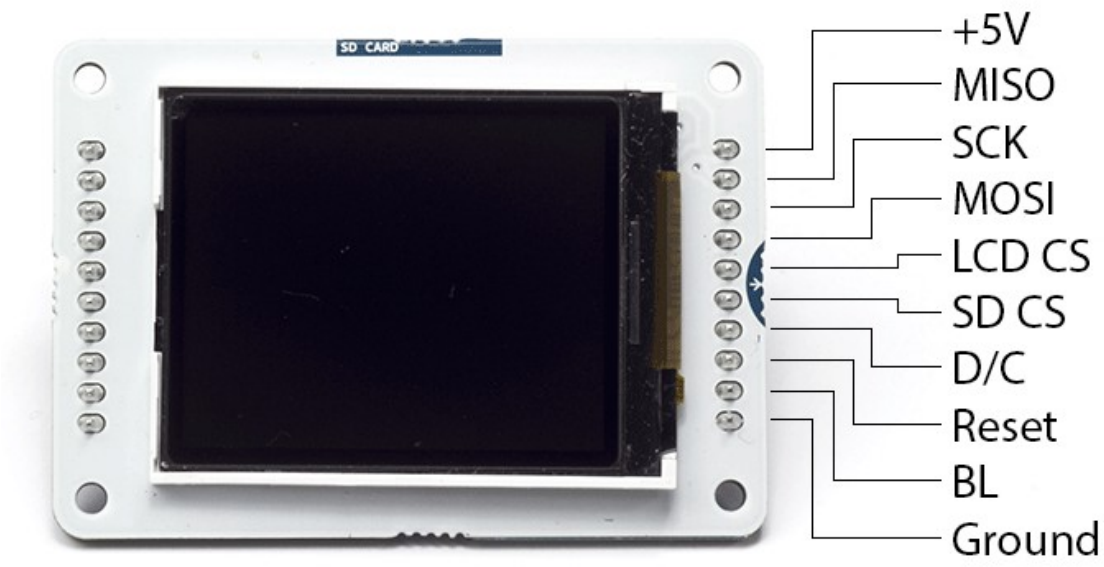



```
#include <Esplora.h>

void setup(){
  Serial.begin(9600);     // démarre et paramètre la communication série
}                         // avec l'ordinateur à 9600 bds

void loop() {
…
  int Potlin=Esplora.readSlider(); // lit la valeur du potentiomètre linéaire
  Serial.print("Potlin : ");  // Envoi des caractères du texte sur le terminal
série
  Serial.println(Potlin);       // Envoi des caractères du texte de la la valeur de Potlin sur le terminal
…
}
```

On notera qu'**on ne peut pas mettre plusieurs éléments en paramètre de la même fonction print() ou println()** dans Arduino.

L'écran LCD - Fonctions graphiques

http://arduino.cc/en/Main/GTFT
http://boutique.semageek.com/fr/204-ecran-tft-lcd-177-spi-arduino.html
http://www.gotronic.fr/art-ecran-lcd-1-77-arduino-20517.htm



Taille de l'écran : 28.032mm x 35.04 *mm*
The screen is 1.77" diagonal, with 160 x 128 pixel resolution. The TFT library interfaces with the screen's controller through SPI when using the TFT library. Refer to the screen's data sheet for complete details.

The screen runs on +5 VDC

The micro-SD slot is accessible through the SD card library.

Il semble que seuls les fichiers BMP au format 24bits/couleur fonctionnent.

Il ne semple pas à priori possible de rentrer dans l'image même, de lui appliquer des transformations (rotation, changement de couleurs), ni de lire les pixels affichés à l'écran.

Tout programme pour la carte Arduino Esplora avec l'écran TFT doit contenir les éléments suivants :

```
// Importation des bibliothèques
#include <Esplora.h> // Pour la carte ESPLORA
#include <SPI.h> // Pour le bus SPI (connexion de l'écran)
#include <TFT.h>  // Pour l'écran TFT

void setup(){
  EsploraTFT.begin();    // initialise l'écran
}
```

4 - Fonctions d'affichage sur l'écran TFT de la carte ESPLORA

Fonction	Résultat obtenu	Syntaxe
stroke()	Défini la couleur des traits et bordures et texte	EsploraTFT.stroke(red,green,blue); red, green, blue : de 0 à 255
noStroke()	Supprime les traits des entités à tracer et le texte	EsploraTFT.noStroke();
fill()	Défini la couleur de coloriage	EsploraTFT.fill(red,green,blue); red, green, blue : de 0 à 255
noFill()	supprime coloriage	EsploraTFT.noFill();
text()	affiche du texte à une postion donnée, sans curseur.	*EsploraTFT.text("Testing!", 0, 0);* *ou* *EsploraTFT.text(mon_text, xPos, yPos);*

		mon_text ***ne peut pas être*** *du type String, ça ne marche pas (dommage !)* *mon_text doit être du type char array (voir après le tableau pour plus de détails)*
println()	écriture sur l'écran TFT avec curseur et mise à la ligne auto	EsploraTFT.println("texte à écrire"); EsploraTFT.println(variable_string); Nota : incompatibilité avec text(), Lenteur provoquée au débordement de page. A EVITER
print()	écriture sur l'écran TFT, avec curseur	EsploraTFT.print("texte à écrire"); EsploraTFT.print(variable_string); Nota : incompatibilité avec text() Lenteur provoquée au débordement de page. A EVITER
setTextSize()	Défini la taille du texte	EsploraTFT.setTextSize(size); size : de 1(par défaut) à 5. Mais on peut mettre plus !! Ca marche.
point()	Trace un point	*EsploraTFT.point(xPos,yPos);*
line()	Tace une ligne	*EsploraTFT.*line(xStart, yStart, xEnd, yEnd);
circle()	trace un cercle	*EsploraTFT.*circle(xPos, yPos, radius);
rect()	Trace un rectangle	*EsploraTFT.*rect(xStart, yStart, xwidth, ywidth); Voir après le tableau pour plus de détails.
image()	trace a donnée de type PImage sur l'écran	EsploraTFT.image(Mon_image, xPos , yPos);
loadImage()	Charge une image dans une variable de type PImage.	Mon_image = EsploraTFT.loadImage("image.bmp"); image BMP couleurs 24 bits
PImage	Définition d'une variable de type PImage	PImage Mon_image; couleurs 24 bits
PImage.isValid()	Evalue la validité (couleur24 bit bmp) de la variable de type PImage	Mon_image.isValid() ; true ou false
background()	Met une couleur unie sur tout l'écran	EsploraTFT.background(255,255,255);
width()	Retourne la largeur de l'écran, ou de l'image à afficher	*EsploraTFT.width();* pas très utile car on la connaît déjà (160). Mon_image.width(); pour la largeur de l'image (type Pimage)
height()	Retourne la hauteur de l'écran, ou de l'image à afficher	EsploraTFT.height(); pas très utile car on la connaît déjà (128). Mon_image.height(); pour la hauteur de l'image (type Pimage)

1 - Affichage de texte sur l'écran TFT

On peut envoyer directement une chaîne de caractères (String) sur l'écran avec :

```
EsploraTFT.text("Bonjour !", xPos, yPos);
```

On ne peut pas envoyer directement une chaîne de caractères (String) sur l'écran avec :

```
String mon_text="Bonjour !";
EsploraTFT.text(mon_text, xPos, yPos);   //Ça ne marche pas !!!!!!!!!
```

En effet, **mon_text** doit être un tableau de caractères (char mon_text [longueur du texte +1]) et non un String.
Il faut convertir la chaine de caractère (String) en tableau de caractères avant avec :

```
char tableau_caracteres [ Ma_Chaine_String.length()+1];
Ma_Chaine_String.toCharArray(tableau_caracteres, Ma_Chaine_String.length()+1);
```

La première ligne pour la création du tableau de caractères pour l'envoi sur l'acran TFT (+1 pour éviter le dépassement

mémoire et le plantage). Ensuite, la conversion du texte en tableau de caractères, le +1 est nécessaire pour avoir le dernier caractère.

Exemple :

```
// Importation des bibliothèques
#include <Esplora.h> // Pour la carte ESPLORA
#include <SPI.h> // Pour le bus SPI (connexion de l'écran)
#include <TFT.h>  // Pour l'écran TFT

void setup(){
  EsploraTFT.begin();    // initialise l'écran
  EsploraTFT.background(0,0,0);
}
```

Dans le void setup() ou le void loop() :

```
String Mon_texte = "A Boire !";
char text_tab[Mon_texte.length()+1];  //  création du tableau de caractères pour l'envoi sur l'acran TFT,
                                      //+1 pour éviter le dépassement mémoire et le plantage
Mon_texte.toCharArray(text_tab, Mon_texte.length()+1); // conversion du texte en tableau de caractères
                                                       // le +1 est nécessaire pour avoir le dernier caractère.
EsploraTFT.stroke(255,255,255);       // Blanc pour le texte
EsploraTFT.text(text_tab, 0, 0);      // écriture sur l'écran TFT des caractères du tableau
```

2 - Affichage de valeurs entières

Pour envoyer la valeur d'un entier sur l'écran TFT, il faut convertir cet entier en tableau de caractères avant.

On peut utiliser la méthode précédente.

Exemple :

```
int mon_entier = -9645;                                  // Définition et affectation de l'entier
String mon_entier_texte = String(mon_entier);    // conversion de l'entier en texte défini à l'occasion
char tableau_caracteres [mon_entier_texte.length()+1];//  création du tableau de caractères
                                                          //+1 pour éviter le dépassement mémoire et le plantage
```

// conversion du texte en tableau de caractères, là aussi, le +1 est nécessaire mais pour avoir le dernier caractère :

```
String_Chaine.toCharArray(tableau_caracteres, mon_entier_texte.length()+1);
```

Ce n'est pas la meilleure manière de faire puisque l'on fait deux conversions (int=>String=>char). **Il est préférable d'utiliser une méthode de conversion directe** (int=>char) avec **itoa()**.

Exemple :

```
int mon_entier = -9645;                                                  // Définition et affectation de l'entier
char  tableau_caracteres [String (mon_entier).length()];//  création du tableau de caractères
                                                                  // le +1 n'est pas nécessaire contrairemenent
                                                                  // à la méthode précédente.
```

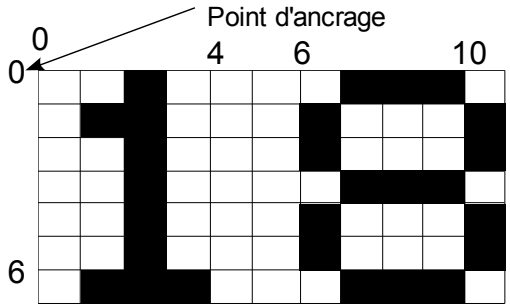
// conversion directe de l'entier en tableau de caractères au format décimal :

```
itoa(mon_entier, tableau_caracteres, DEC);
```

Il y a moins de ligne à écrire, c'est plus simple.

3 - Résolution des caractères :

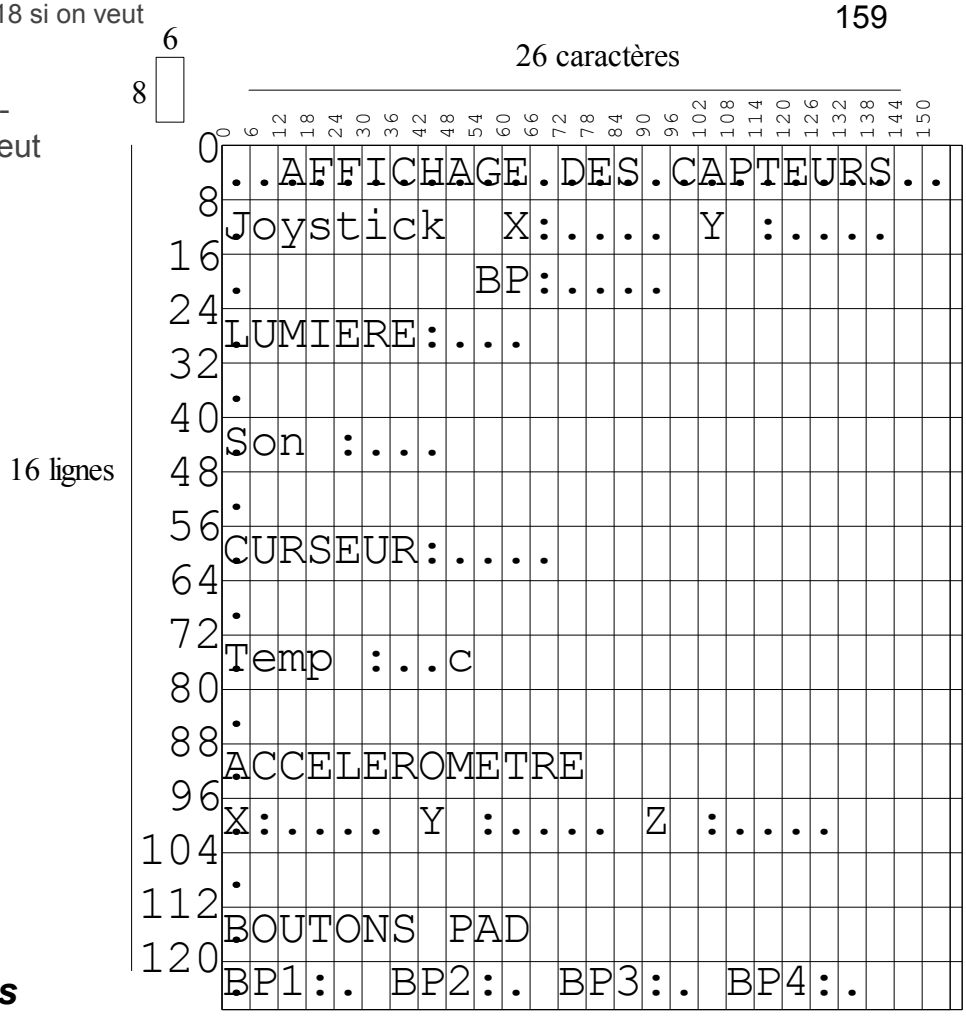
Taille 1: 5 x7 pixels (26 caractères par ligne, avant mise à la ligne automatique)



Taille 2: 10 x14 pixels (13 caractères par ligne, avant mise à la ligne automatique)

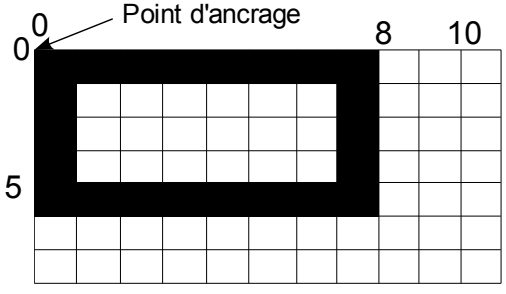
Et ainsi de suite... Tout est multiplié par 2, 3 , 4. La taille maxi du caractère affichable est 18 si on veut qu'il rentre dans l'écran en hauteur.

La grille ci-contre permet de pré-câler le texte (taille 1) que l'on veut afficher sur l'écran.



4 - Tracé des rectangles

EsploraTFT.rect(xStart, yStart, xwidth, ywidth);



Exemple pour le rectangle ci-dessous :
Coordonnées du rectangle : (0, 0)
Taille horizontale : 8 pixels
Taille verticale : 5 pixels

=> Codage : **`EsploraTFT.rect(0, 0, 8, 5);`**

Ne pas oublier de mettre avant :
`EsploraTFT.stroke(red,green,blue);`

Le rectangle peut être colorié avec : **`fill(red,green,blue);`**

Il n'est pas possible de faire un rectangle de taille xwidth=0 ou ywidth=0. Cela provoque le tracé d'un rectangle décalé. Le rectangle le plus petit à une taille de 1 pixel.

5 - Carte SD

Pour plus de détails, références sur http://arduino.cc/en/Reference/SD

Tout programme pour utiliser le lecteur de carte SD (sous l'écran TFT) doit contenir les éléments suivants :

```
#include <Esplora.h>    // bibli carte ESPLORA
#include <SD.h>         // bibli carte SD

void setup(){
  SD.begin(8);   // Initialise le lecteur SD
}
```

Il s'agit d'une micro carte SD de 2 Go, ou plus, montée dans un logement idoine sous l'écran TFT de la carte ESPLORA. Éviter de la sortir, sauf pour la monter dans son adaptateur SD normal pour mettre la carte sur un PC ou via un adaptateur USB. Ne la laisser pas traîner, remettez-la dans le logement de l'afficheur TFT de la carte ESPLORA. Le formatage de la carte doit être en FAT, Fat16 ou Fat32.

La carte SD sert à stoker des fichiers images qui peuvent être lues et affichées sur l'écran. Mais on peut aussi enregistrer des données (températures mesurées à différents moments de la journée, les accélérations avec l'heure et le jour. On peut enregistrer les scores de différents joueurs d'un jeu. Les utilisations possibles sont nombreuses.
Le temps d'accès et d'écriture dans un fichier est assez long. La gestion du temps peut être problématique si l'on a beaucoup de choses à écrire en peu de temps. On peut travailler sur plusieurs fichiers en même temps.

Les fichiers doivent être au format 8.3 : xxxxxxxx . yyy (**ne pas dépasser 8 caractères pour le nom du fichie**r).

Un fichier numérique est une suite d'octets qui peuvent être directement les octets du code ascii des caractères d'un texte ou de nombre en format byte.

La construction d'un fichier numérique et l'accès à des données stockées dans ce fichier pose toute la problématique de la structuration du fichier. Comment ranger et organiser les données à ranger dans le fichier pour savoir quel octet correspond à quoi.

1 - Bibliothèque carte SD

Voici les bibliothèques à mettre (respecter l'ordre) au début du programme :

```
#include <Esplora.h>     // bibli carte ESPLORA
#include <SD.h>          // bibli carte SD


void setup() {
  SD.begin(8);           // activation du lecteur SD via la broche 8 sur la carte ESPLORA
    }
```

2 - Les fonctions disponibles

Tout fichier est un ensemble d'octets mis bout à bout. Les fonctions disponibles dans la bibliothèque SD permettent de lire chaque octet séparément n'importe où dans le fichier. Il n'est pas possible d'écrire un octet au milieu ou en remplacement d'un octet existant. Il n'est pas possible d'effacer un octet existant sauf à effacer tout le fichier et le réécrire. Lorsque l'on écrit un octet dans un fichier il est placé automatiquement derrière le dernier octet du fichier.

Au préalable, on déclare une donnée de classe « File » qui va permettre de travailler sur les données à lire ou écrire sur la carte avec par exemple : **`File mon_fichier;`** et ensuite ouvrir le fichier : **`mon_fichier =`** SD.open("Le_fichier.abc", FILE_READ); pour lire le fichier par exemple.

Lire la taille du fichier :

`mon_fichier.size();` renvoie la valeur de la taille du fichier en nombre d'octet (valeur décimale)

Position du curseur courant dans le fichier :

`mon_fichier.position();` // donne la position en cours (n° d'octet) dans le fichier. A l'ouverture du fichier cette position vaut 0 (devant le premier octet)

Modifier la position du curseur dans le fichier :

mon_fichier.seek(n_octet) ; // Redéfini la position du curseur dans le fichier à une nouvelle position (**n_octet** est un nombre entier compris entre 0 et le nombre total d'octet dans le fichier).

Lecture d'un octet :

mon_fichier.peek(); // Renvoie l'octet à la position en cours sans incrémenter la position en cours.
mon_fichier.read(); // Renvoie l'octet à la position en cours et incrémente la position en cours de un (octet // suivant) pour après.

Ecriture dans un fichier

mon_fichier.Write(data); data peut être une donnée de type byte, char. Cette fonction écrit un octet à la fin du fichier (après le dernier octet déjà présent).

mon_fichier.write(tab); // tab peut être un tableau de char. La taille du tableau de char doit être supérieur d'un point par rapport au nombre de valeur contenue dans le tableau.
Exemple :
char char_int1[5]={69,70,71,72};
mon_fichier.write(char_int1);

mon_fichier.write(tab, len); // Non testé avec succés
où tab : tableau de "char" ou "bytes"
et len : le nombre d'éléments de tab

mon_fichier.print(data); écrit data en format texte (code ascii) dans le fichier. "data" peut être du type char, tableau de char, byte, int, long, or string. Ce type d'écriture n'est pas économique en terme d'octet à écrire dans le fichier. Exemple le nombre 32767 qui est un entier de 2 octets en binaire décimal sera codé sur 5 octets (un pour chaque chiffre en code ASCII).
La conversion nombre -> texte avec print() ou itoa(), puis en retour, texte -> nombre avec la fonction atoi() par exemple est sans doute plus aisée que décomposer le nombre, en n octets correspondant exactement à son type de données.

mon_fichier.print(data, base); Même chose qu'au-dessus mais avec la posibilité de formater l'écriture du nombre.
Choix pour **base** : BIN, DEC, OCT, HEX.

Exemple :

mon_fichier.print(un_octet, BIN); // Utilise 8 octets pour écrire 1 octet (11111111) avec un_octet=255, un octet pour chaque chiffre binaire.
mon_fichier.print(un_octet, DEC); // Utilise 3 octets pour écrire '2', '5', '5', la valeur décimale 255
mon_fichier.print(un_octet, OCT);// Utilise 3 octets pour écrire '3', '7', '7', la valeur Octale de 255
mon_fichier.print(un_octet, HEX);// Utilise 2 octets pour écrire 'F', 'F' la valeur Hexadécimale de 255

mon_fichier.println(data, base);
La même chose que précédemment mais avec 2 octets en plus pour LF (0xOD) et CR (0xOA) à la fin de chaque écriture.

La débauche et le faible coût du gigaoctet rend dérisoire la recherche d'optimisation du nombre d'octet à écrire pour économiser la mémoire, surtout sur une carte mémoire SD.

Enregistrement dans la carte SD

mon_fichier.flush(); // Enregistre les données de "**mon_fichier"** sur la carte SD (mise à jour) sans fermeture du fichier.

Enregistrement dans la carte SD et fermeture du fichier :

mon_fichier.close(); // enregistre les données de "**mon_fichier"** sur la carte SD et ferme le fichier (obligatoire quand on a fini).

3 - Lire dans un fichier

minimum nécessaire :

```
#include <Esplora.h>     // bibli carte ESPLORA
#include <SD.h>          // bibli carte SD

File monfichier;         // variable monfichier de type File.
String contenu_fichier; // définition d'une variable de type String pour lire et afficher le contenu du fichier

void setup()
{
....
Serial.begin(9600);      // pour l'exemple à suivre et visualiser ce qu'il se passe sur le terminal série
SD.begin(8);       // activation du lecteur SD via la broche 8 sur la carte ESPLORA
....
}
```

dans setup() ou void loop() :

```
monfichier = SD.open("test.txt");

while (monfichier.available()) {  // pour balayer tout le fichier sinon on ne prend que le premier code
contenu_fichier =contenu_fichier+ monfichier.read()    // extrait chaque octet du fichier et
                                                       //  le copie dans contenu_fichier.
}                                                      // On pourrait aussi transformer
monfichier.close();                                    // directement les codes en caractères
                                                       // ascii.
// Autre visualisation du fichier sur le terminal série
    monfichier = SD.open("test.txt", FILE_READ );  // ré-ouverture du fichier.
    while (monfichier.available()) {  // lecture de toutes les valeurs du fichier.
    Serial.write(monfichier.read()); // Envoi et affiche les caractères via leur code Ascii sur
                                     // le terminal série
  }
   monfichier.close();               // Enregistre sur la carte SD et ferme le fichier
```

Voir l'exemple : http://arduino.cc/en/Tutorial/ReadWrite

4 - Ecrire dans un fichier

minimum nécessaire :

```
#include <Esplora.h>     // bibli carte ESPLORA
#include <SD.h>          // bibli carte SD

File monfichier;                    // création de la variable de type File (fichier dans Arduino)
String A_stocker="important !";     // défini une donnée en format texte.
float a=12.354;                     // défini une valeur float

void setup()
{
....
SD.begin(8); // activation du lecteur SD via la broche 8 sur la carte ESPLORA
....
}
```

dans setup() ou void loop() :

```
monfichier = SD.open("exemple.txt", FILE_WRITE);      // ouvre le fichier en écriture
monfichier.println(A_stocker);                        // enregistre la chaine dans le fichier.
monfichier.println(123456);                           // écrit une valeur dans le fichier
```

```
monfichier.println(a);                    // écrit une valeur float dans le fichier
monfichier.close();                       // Enregistre sur la carte SD et ferme le fichier
```

Voir l'exemple http://arduino.cc/en/Tutorial/ReadWrite

5 - Créer un fichier

minimum nécessaire :

```
#include <Esplora.h>     // bibli carte ESPLORA
#include <SD.h>          // bibli carte SD

File monfichier;         // création de la variable de type File (fichier dans Arduino)

void setup()
{
......
SD.begin(8); // activation du lecteur SD via la broche 8 sur la carte ESPLORA
......
}
```

dans setup() ou void loop() :

```
monfichier  = SD.open("exemple.txt", FILE_WRITE);      // il n'y a pas de différence entre créer
                                                       // et ouvrir un fichier. S'il n'existe pas, le
                                                       // fichier à ouvrir est créé.
                                                       // respecter le format 8x3 !!!!!!!!!!
monfichier.close();                                    // Enregistre sur la carte SD et ferme le fichier
```

Voir l'exemple http://arduino.cc/en/Tutorial/Files

6 - Supprimer un fichier

minimum nécessaire :

```
#include <Esplora.h>     // bibli carte ESPLORA
#include <SD.h>          // bibli carte SD

void setup()
{
......
SD.begin(8);   // activation du lecteur SD via la broche 8 sur la carte ESPLORA
....
}
```

dans setup() ou void loop() :

```
SD.remove("exemple.txt");        // supprime le fichier
```

Voir l'exemple http://arduino.cc/en/Tutorial/Files

7 - Datalogger

Cela consiste à enregistrer une série de données mesurées ou autres. Les données vont être enregistrées dans un fichiers texte, avec un séparateur (virgule, point ou point-virgule) entre chaque donnée. Lorsque la saisie est terminée, le fichier texte est envoyé dans le fichier où les données doivent être stokées.

Variable de stockage des donnnées :

```
String Mes_donnees = ""; // création d'un fichier texte pour stocker les valeurs
```

Enregistrement des données dans une boucle :

```
int sensor = analogRead(analogPin); // lecture d'une valeur
```

```
Mes_donnees += String(sensor);      // ajout de la dernière valeur mesurée
Mes_donnees += ",";                 // ajout du séparateur (virgule)
```

Enregistrement des données dans un fichier :

```
File Mon_fichier= SD.open("datalog.txt", FILE_WRITE); // ouverture du fichier en écriture
Mon_fichier.println(Mes_donnees);                     // enregistrement des données (fichier texte)
Mon_fichier.close();                                  // Enregistre sur la carte SD et ferme le fichier
```

http://arduino.cc/en/Tutorial/Datalogger

8 - Lire une image sur la carte SD et l'afficher sur l'écran TFT

Les images doivent être au format BMP, 24bits par couleurs.

```
// this variable represents the image to be drawn on screen
PImage Mon_image; // définie la variable Mon-image de type Pimage

void setup() {
  EsploraTFT.begin();    // pour Initialiser l'écran TFT
  SD.begin(8);           // activation du lecteur SD via la broche 8 sur la carte ESPLORA
}


void loop (){
// Lecture de l'image "image.bmp" et affectation à la variable "Mon_image"
  mon_image = EsploraTFT.loadImage("image.bmp");
//  Affichage de la variable "Mon_image" sur l'écran TFT aux coordonnées x, y
  EsploraTFT.image(Mon_image, x, y);
}
```

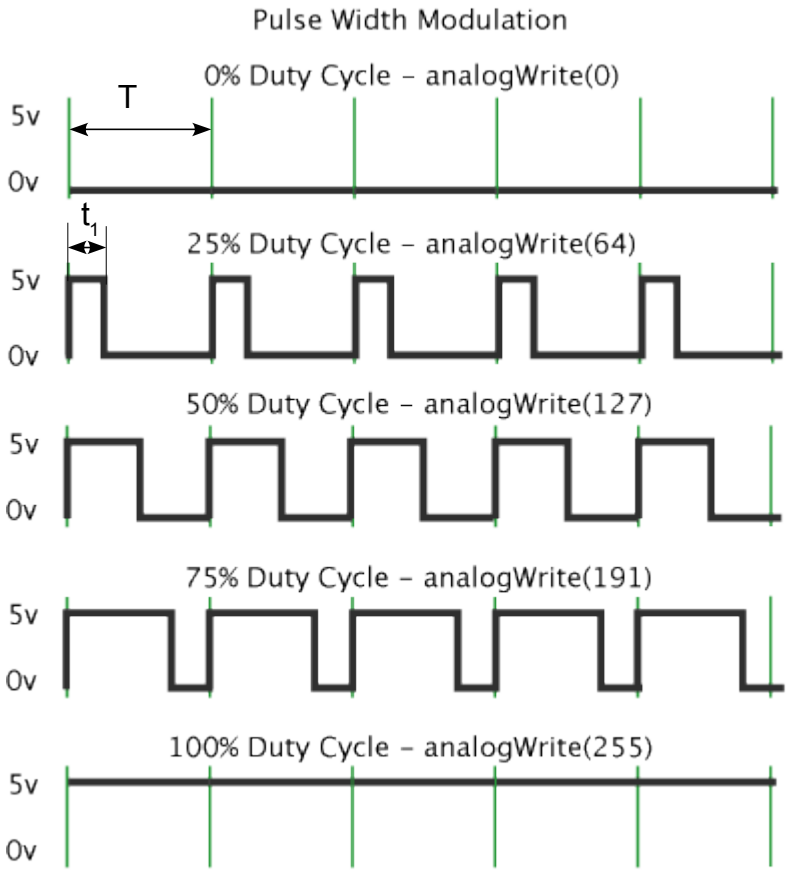
Voir l'exemple http://arduino.cc/en/Tutorial/EsploraTFTBitmapLogo

6 La carte Esplora pour les experts (de Nantes)

1 - PWM (simple)

Le **PWM** (Pulse Wave Modulation) consiste en un signal rectangulaire dont le rapport cyclique (r = t_1 / T) est compris entre 0 et 1. La résolution du rapport cyclique est de 8bits (de 0 pour r=0 à 255 pour r=1) ou 16 bits (de 0 pour r=0 à 65535 pour r=1).

Le PWM est un moyen de produire en sortie une tension **moyenne** variable comprise entre 0V et 5V. C'est un moyen peut onéreux de s'approcher d'une sortie analogique puisque le microcontroleur ATMEGA32U4 et la carte ESPLORA ne sont pas équipés d'un Convertisseur Numérique Analogique.



On ne peut donc pas l'utiliser produire un son, mais on peut ainsi donner l'impression qu'une LED est plus ou moins allumée, ou faire tourner un moteur à une vitesse plus ou moins forte. C'est déjà beaucoup !

La tension moyenne peut se calculer avec :

$$U_{moy} = U_{max} \cdot \frac{t_1}{T}$$

avec une résolution PWM de 8 bit on aura donc :

$$U_{moy} = U_{max} \cdot \frac{val.rapp.cycl}{255}$$

avec Umax = 5V
et 0 ≤ val_rapp_cycl ≤ 255

La période T n'est pas forcément modifiable. The frequency of the PWM signal is approximately **976,5 Hz** sur la carte ESPLORA. (rappel : T = 1/f).

Il est possible de faire un PWM sur mesure. On peut alors travailler sur le Timer 1 et l'interruption timer1.

Programmation du PWM

Sur Arduino, le PWM consiste, après avoir configuré l'une des broches PWM en sortie, à utiliser la fonction **analogWrite()**. Contrairement à son nom cette fonction n'a rien à voir avec une vraie sortie analogique.

exemple :

```
void setup() {
    pinMode(broche_PWM, OUTPUT);
}

void loop{
      analogWrite(broche_PWM , valeur_rapport_cyclique);
}
```

valeur_rapport_cyclique est compris entre 0 et 255 (8bits), ou entre 0 et 65537(16bits)

Les broches IO 11 (Tinkerkit outB ou Out3) et IO 3 (Tinkerkit outA ou out2) peuvent être utilisées en sortie PWM 8bits).

2 - Interruptions

Les interruptions sont des parties de programmes qui seront exécutées toutes affaires cessantes. Le programme normal est repris là où il a été laissé à la fin du traitement de l'interruption, sauf si une autre interruption s'est déclenchée entre temps.

Interruption sur état ou changement d'état de Broches (pin)

Sur les broches physique 1, 8, 19, 20 (IO 7, 3, 2, 0,1 du micro-contrôleur de la carte ESPLORA sont prises :
par l'afficheur (IO 0, 1, 7),
configurée en sortie (IO 3 : pwm tinkerkit outA),
ou non connectée (IO 2).
Il n'y a donc pas de possibilité à priori d'utiliser les interruptions sur les changements d'état de ces broches.
Eventuellement sur la 3 (celle prévue pour le Tinkerkit OutA). Sinon, il faut enlever l'écran TFT pour utiliser les broches IO 0, 1, 2, 7 comme déclencheur d'interruption.

Les interruptions broches pouvant déclencher une interruption :

Syntaxe : attachInterrupt(interrupt, function, mode)

choix de interrupt =>	**int.0**	**int.1**	**int.2**	**int.3**	**int.4**
Broche utilisée pour le déclenchement =>	3	2	0	1	7

mode : **LOW** , **CHANGE, RISING, FALLING**
LOW (état logique 0 sur l'entrée), **CHANGE** (changemenent d'état 0=>1 ou 1>0)**, RISING** (de 0 à 1)**, FALLING** (de 1 à 0)

exemple :

```
setup(){
.....
attachInterrupt(0, sous_prog_ca_change, CHANGE); // C'est la broche 3 qui déclenche l'interruption 0
....
}


// Cette fonction est exécutée lors d'un déclenchement de l'interruption 0 (broche3).
sous_prog_ca_change (){
....
....
}
```

Interruption temporelle (Timer : horloge interne du microcontroleur)

Il est possible d'utiliser le timer 1 de la carte ESPLORA pour déclencher des interruptions temporelles et exécuter des actions à un moment précis, unique ou répété, ou pour faire du PWM sur mesure. Le paramétrage du timer 1 est complexe et nous ne rentrerons pas dans l'explication de celui-ci. Il faut connaître le fonctionnement intime d'un micro contrôleur pour cela et choisir le bon paramétrage des registres à modifier du micro-contrôleur à l'aide de la sa documentation (ATmega32u4 de chez ATMEL).

Exemple pour obtenir une pulsation d'une seconde sur ARDUINO :

```
// service d'interruption sur débordement du timer1 : c'est la fonction executée au débordement du timer qui déclenche
l'interruption
ISR(TIMER1_OVF_vect)
        {                           // le démasquage de l'interruption de débordement du timer1 est automatique
          TCNT1 =3135;              // initialise le timer à 48576µs : il reste 1000000µs avant le débordement du timer1
          /* ICI tout ce que je veux effectuer dans le service de l'interruption du timer1*/
        }

setup(){
......
 // paramètre le timer1 en compteur 16 bits à débordement (RAZ quand TMR1 = 65536) de base de temps interne (mode 0)
 TCCR1A = B00000000;                // COM1A = COM1B =00 : sorties OC1A et OC1B en mode normal, WGM =0000 : mode 0
 // l'horloge du timer1 est dérivée de l'horloge système prédivisée (fosc/64)
 TCCR1B = B00000100;                // CS1 =100 : prédivision par 256, soit base de temps de 16 microsecondes
 TIMSK1 = TIMSK1 |(1 <<TOIE1);      // démasque l'interruption de débordement du timer1
 TCNT1 =3135;                       // initialise le timer à 48576µs  : il reste 1000000µs avant le débordement du timer1
```

```
.....
}
```

Pour modifier le paramétrage de ce timer, il faut consulter la documentation du microcontroleur (ATmega32U4)

3 - La mesure du temps

Une méthode plus simple que l'utilisation d'interruption sur les timers (page précédente) pour mesurer une durée consiste à utiliser les fonctions millis() ou micros() qui renvoient le temps écoulé depuis le démarrage du programme.

exemple pour compter des secondes :

```
// Importation des bibliothèques
#include <Esplora.h>     // Pour la carte ESPLORA

long time, seconde ;

void setup(){
  EsploraTFT.begin();              // démarrage de l'écran TFT
  EsploraTFT.background(0,0,0);    // mise en place du fond noir
}

void loop(){
  if (time+1000<millis())       // Si dépassement, 1 seconde s'est écoulé, 1000 ms = 1 s  !!
  {
    seconde +=1;       // une seconde de plus !!
    time=time+1000;    // incrémentation de time
  }
}
```

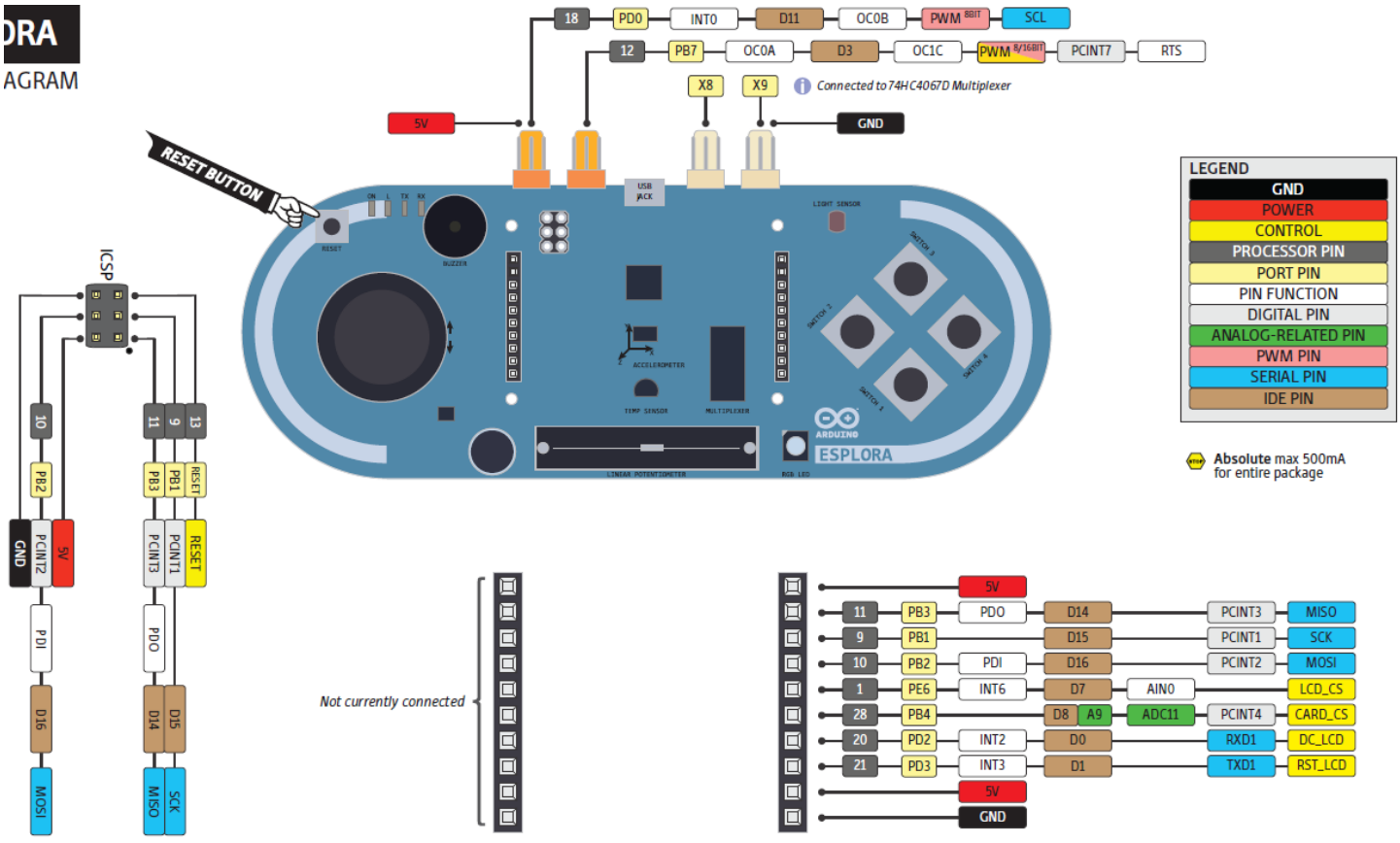
Il est à proscrire l'utilisation de "pause" comme delay() ou delayMicroseconds(), même si c'est très pratique, on ne paie pas un microcontrolleur à rien faire !

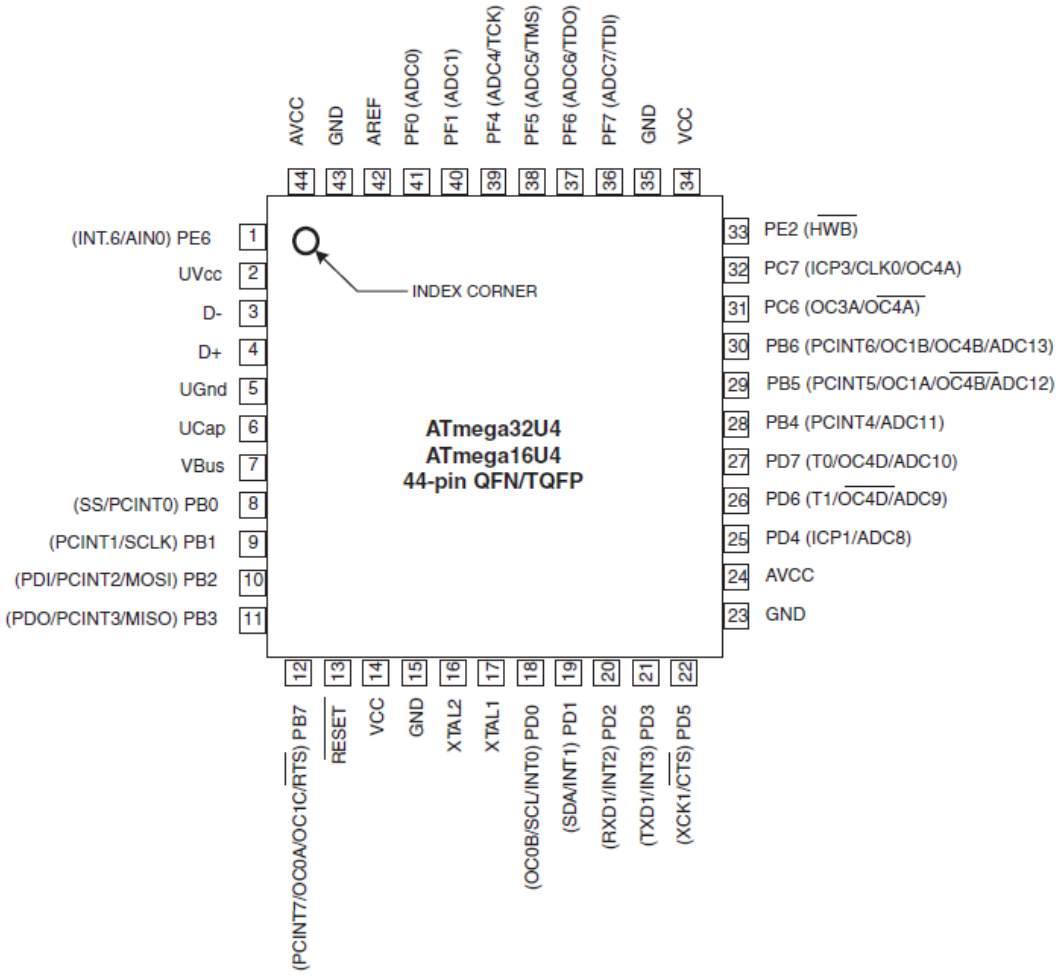
Exemple de programme

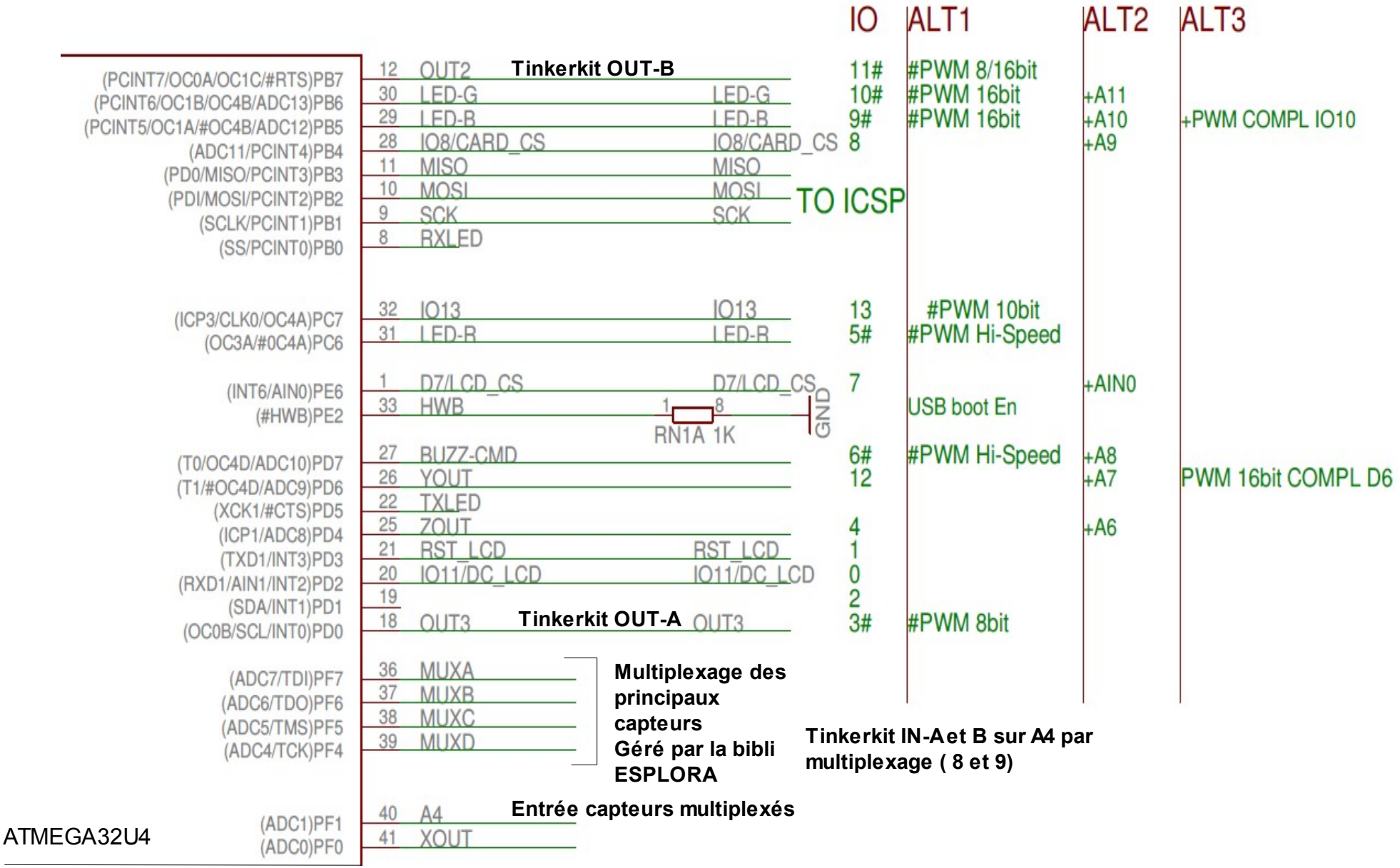
Le programme suivant test les différents éléments de la carte et renvoi des valeurs de test à l'ordinateur via le câble série. Ouvrir la console série pour voir l'affichage des valeurs.

```
#include <Esplora.h>

void setup()
{
  Serial.begin(9600);        // initialize serial communications with your computer
}

void loop()
{
  // lecture des capteurs
  int xAxis = Esplora.readAccelerometer(X_AXIS);    // read the X axis
  int yAxis = Esplora.readAccelerometer(Y_AXIS);    // read the Y axis
  int zAxis = Esplora.readAccelerometer(Z_AXIS);    // read the Z axis
  int xValue = Esplora.readJoystickX();        // read the joystick's X position
  int yValue = Esplora.readJoystickY();        // read the joystick's Y position
  int button = Esplora.readJoystickSwitch();   // read the joystick pushbutton
  int Potlin = Esplora.readSlider();// lecture du potentiomètre linéaire
  int light = Esplora.readLightSensor(); //lecture du capteur de lumière
  int bp1 = Esplora.readButton(1);  // en bas
  int bp2 = Esplora.readButton(2); //à gauche
  int bp3 = Esplora.readButton(3);//en haut
  int bp4 = Esplora.readButton(4); //àdroite
  int Temp = Esplora.readTemperature(DEGREES_C);
  int loudness = Esplora.readMicrophone();

  //  affichage des valeurs lues
  Serial.print("x: ");      // print the label for X
  Serial.print(xAxis);      // print the value for the acceleration X axis
  Serial.print("\ty: ");
  Serial.print(yAxis);      // print the value for the acceleration Y axis
  Serial.print("\tz: ");
  Serial.print(zAxis);    // print the value for the acceleration Z axis
  Serial.print("\tlight: ");
  Serial.println(light);        // print the light sensor value
  Serial.print("Joystick X: ");
  Serial.print(xValue);         // print the joystick X value
  Serial.print("\tY: ");
  Serial.print(yValue);         // print the joystick Y value
  Serial.print("\tButton: ");
  Serial.print(button);         // print the joystick button value
  Serial.print("\tPotlin: ");
  Serial.println(Potlin);       // print the line sensor value
  Serial.print("\tBP1: ");
  Serial.print(bp1);            // print the bp1 value (1 OFF, 0 ON)
  Serial.print("\tBP2: ");
  Serial.print(bp2);            // print the bp2 value (1 OFF, 0 ON)
  Serial.print("\tBP3: ");
  Serial.print(bp3);            // print the bp3 value (1 OFF, 0 ON)
  Serial.print("\tBP4: ");
  Serial.println(bp4);          // print the bp4 value (1 OFF, 0 ON)
  Serial.print("\tson: ");
  Serial.print(loudness);       // print the loudness
  Serial.print("\ttemp: ");
  Serial.print(Temp);           // print the Temperature
  Serial.println("'C");          // in degree
  Serial.println();          // in degree

  delay(500);              // wait half a second (500 milliseconds)
}
```

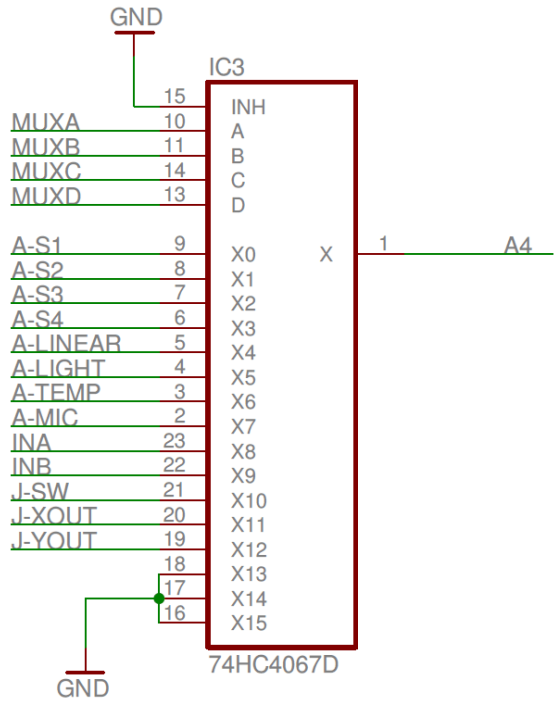
7 - Définition et fonctions des broches (pin) sur la carte ESPLORA






IO
ALT1
ALT2
ALT3
(PCINT7/OC0A/OC1C/#RTS)PB7
(PCINT6/OC1B/OC4B/ADC13)PB6
(PCINT5/OC1A/#OC4B/ADC12)PB5
(ADC11/PCINT4)PB4
(PD0/MISO/PCINT3)PB3
(PDI/MOSI/PCINT2)PB2
(SCLK/PCINT1)PB1
(SS/PCINT0)PB0
12 OUT2
Tinkerkit OUT-B
30 LED-G
29 LED-B
28 IO8/CARD_CS
11 MISO
10 MOSI
9 SCK
8 RXLED
LED-G
LED-B
IO8/CARD_CS
MISO
MOSI
SCK
TO ICSP
11#
#PWM 8/16bit
10#
#PWM 16bit
+A11
9#
#PWM 16bit
+A10
+PWM COMPL IO10
8
+A9
(ICP3/CLK0/OC4A)PC7
(OC3A/#0C4A)PC6
32 IO13
31 LED-R
IO13
LED-R
13
#PWM 10bit
5#
#PWM Hi-Speed
(INT6/AIN0)PE6
(#HWB)PE2
1 D7/LCD_CS
33 HWB
D7/LCD_CS
1
8
RN1A 1K
GND
7
+AIN0
USB boot En
(T0/OC4D/ADC10)PD7
(T1/#OC4D/ADC9)PD6
(XCK1/#CTS)PD5
(ICP1/ADC8)PD4
(TXD1/INT3)PD3
(RXD1/AIN1/INT2)PD2
(SDA/INT1)PD1
(OC0B/SCL/INT0)PD0
27 BUZZ-CMD
26 YOUT
22 TXLED
25 ZOUT
21 RST_LCD
20 IO11/DC_LCD
19
18 OUT3
Tinkerkit OUT-A
RST_LCD
IO11/DC_LCD
OUT3
6#
#PWM Hi-Speed
+A8
12
+A7
PWM 16bit COMPL D6
4
+A6
1
0
2
3#
#PWM 8bit
(ADC7/TDI)PF7
(ADC6/TDO)PF6
(ADC5/TMS)PF5
(ADC4/TCK)PF4
36 MUXA
37 MUXB
38 MUXC
39 MUXD
Multiplexage des principaux capteurs
Géré par la bibli ESPLORA
Tinkerkit IN-A et B sur A4 par multiplexage (8 et 9)
Entrée capteurs multiplexés
(ADC1)PF1
(ADC0)PF0
40 A4
41 XOUT
ATMEGA32U4

Multiplexage des capteurs sur l'entrée A4




```
const byte JOYSTICK_BASE  = 16; // it's a "virtual"
channel: its ID won't conflict with real ones

const byte MAX_CHANNELS   = 13;

const byte CH_SWITCH_1    = 0;
const byte CH_SWITCH_2    = 1;
const byte CH_SWITCH_3    = 2;
const byte CH_SWITCH_4    = 3;
const byte CH_SLIDER      = 4;
const byte CH_LIGHT       = 5;
const byte CH_TEMPERATURE = 6;
const byte CH_MIC         = 7;
const byte CH_JOYSTICK_SW = 10;
const byte CH_JOYSTICK_X  = 11;
const byte CH_JOYSTICK_Y  = 12;
```

On notera que les entrées externes Tinkerkit IN-A et IN-B (connecteurs à 3 broches blancs) sont accessibles sur la broche A4 avec les fonctions analogRead(), digitalRead() ou encore pulseIn() avec les codes binaires 8 et 9 sur le multiplexeur.