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

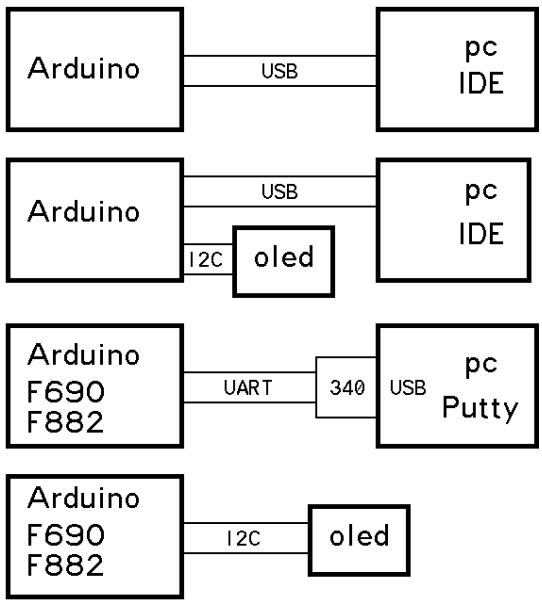
BuddyTerm et Terminal série Arduino

Oled I2C, affichage série sur PC séparé ou tablette

Introduction

Le Terminal série d'arduino est très utile pour afficher des valeurs de capteurs et pour mettre au point des programmes.
Ses défauts sont d'utiliser trop de mémoire, d'avoir des instructions assez lourdes pour le débutant, et longues à taper. Pour surveiller des capteurs, il n'est plus utilisable sur le terrain, avec une alimentation sur pile et plus de PC.
Un affichage Oled I2C est facile à mettre en œuvre. Il affiche 50 variables si nécessaire, et on peut afficher graphiquement des valeurs qui évoluent, avec peu de précision, mais quelle aide pour vérifier que tout se passe bien.

1
2
3
4



1 Arduino et le terminal série

Bien connu, voir aussi http://www.didel.com/C/Terminal.pdf, plus précis concernant la lecture de paramètres au clavier (fonction parse).
On peut améliorer le confort de ce terminal avec des macros plus rapides à écrire.
Ce qui est déclaré dans SerieNum.h, décrit plus loin, est

```
#define P Serial.print
#define Psp(X,Y) Serial.print(' '); Serial.print
#define Pln Serial.println
#define Sp() Serial.print(' ')
#define Enter() Serial.println()
```

`P` est vraiment l'équivalent de `Serial.print`. Il économise beaucoup de frappes et fautes de frappes. Il permet aussi mettre plusieurs actions sur la même ligne, donc avoir une meilleure vue d'ensemble sur une page.
`Psp` est pratique si plusieurs valeurs se suivent. Il ajoute (en fait avant, vous voyez pour quelle raison?) un espace pour séparer les nombres.
`Pln` est utulisé pour le dernier nombre de la ligne. `Sp();` insère un espace. Vous pouvez modifier en `Sp(n);` en ajoutant une boucle for, pour insérer n espaces.

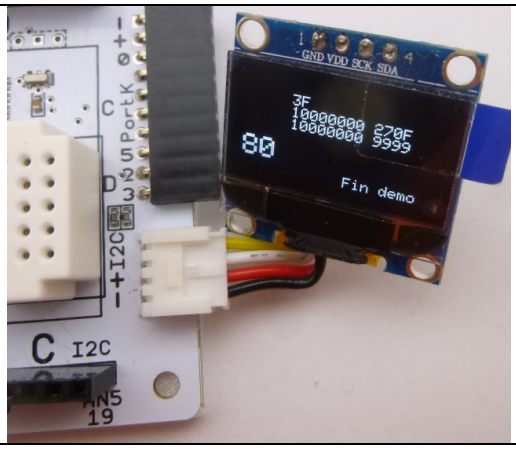
Pour être compatible avec ce qui suit, le fichier SerieNum.h contient, en plus de ces définitions, les fonctions `Dec8(v8); Hex8(v8); Bin8v8); Dec9999(v16); Hex16(v16); Bin16(v16);`

La librairie Serial d'Arduino est très lourde. Utiliser Ser.h vu plus loin pour communiquer directement via USB ou Rx Tx est possible en ne changeant que la référence à la librairie.

2 Arduino et Oled Adafruit, option librairie LibOled.h

Les Oled type SSD1306 sont disponibles et bien documentés en version SPI et I2C. Si la librairie GFX est utilisée, l'adaptation de ce qui suit est facile. Le document http://www.didel.com/diduino/OledI2C.pdf a été écrit en 2015 pour simplifier et compléter la librairie Adafruit. Comme pour le Terminal série, les zéros non significatifs sont supprimés, ce qui est absurde pour les nombres hexa et binaires. Pour les nombres décimaux, il faut mettre

espaces pour avoir toujours 3 signes. De cette façon, les mesures, résultats de calculs et tableaux restent alignés dans l'écran.
Le librairie LibOled.h, appelée par un #include "LibOled.h" ajoute les fonctions suivantes:

`REFR;` identique a `display.display();`
 	
`Pdec()` affiche une variable 8 ou 16 bits en décimal 4 digits (max 9999)
 	
`P8Hex()` affiche une variable 8 bits en hexadécimal
 	
`P16Hex()` affiche une variable 16 bits en hexadécimal
 	
`P8Bin()` affiche une variable 8 bits en binaire 8 chiffres 0/1
 	
`Pos(x,y)positionne le pointeur en x(colonne 0à127) et y(ligne 0à7)`
	

La documentation donne des exemples de programmes disponibles sur http://www.didel.com/diduino/OledI2C.zip.

Les noms utilisés en 2015 ne sont pas toujours cohérents. Les fonctions développées en 2017 avec une compatibilité UART et Oled, utilisent des nouveaux noms.
Modifier les #define pour utiliser les noms que vous préférez est trivial.

Les fichiers inclus, peu utilisés dans le monde Arduino, nécessite un effort d'adaptation des utilisateurs habitués à quelques librairies ou il suffit de copier des exemples. Voir http://www.didel.com/diduino/ArduinoInclude.pdf

2 Arduino ou autre processeur et simulateur de terminal sur PC

Ce que l'on transmet sur le Terminal série est accessible sur les pins Rx Tx de la carte Arduino, mais la connexion gêne la programmation (voir http://www.didel.com/USBtoSerial.pdf qui explique aussi que pour se connecter sur l'entrée USB d'un PC ou d'une tablette, il faut utiliser un convertisseur UART (série) – USB. Ce circuit a un entrée Rx, à relier avec la sortie Tx du microcontrôleur, Arduino ou autre. De même, Tx est relié à Rx.	

Sur Arduino, il existe une librairie série. Nous documentons l'équivalent (fichier à inclure Uart.h) qui fait ce que l'on veut et rien de plus (xx bytes). Les 2 fonctions nécessaire sont AfCar() qui affiche un caractère ASCII et AfCR qui force le retour à la ligne (CR LF, parfois seulement CR ou seulement LF, facile à éditer dans le fichier Uart.h).
L'affichage de nombres est dans un autre fichier inclus, car il dépend si c'est un terminal "imprimante" ou un affichage qui offre des possibilités de mise en page.
TerNum.h regroupe les fonctions fonctions suivantes et n'utilise que 300 bytes.

Fonction	Exemple	Equivalence Terminal série Arduino

`AfHex8() byte ou char`
`AfHex16() int ou uint`
`AfDec8()`
`AfDec16()`
`AfBin8()`
`AfText("txt");`
	`AfHex8(32);AfHex8(0x32);`
	`AfHex16(0x1234);`
	`AfDec8(255); AfDec8(0xFF);`
	`etc`
		`Serial.print (32,HEX);`

Rappelons que la fonction travaille sur des valeurs binaires en mémoire. Une donnée en décimal est convertie en binaire pour être mise en mémoire, et recalculée en décimal si on veut l'afficher en décimal.

Programmes de test

Le premier test est de s'assurer que la communication série se fait. Cela dépend des connecteurs Arduino, du câble, de l'adaptateur série-USB et de l'installation et configuration du simulateur de terminal. Putty est une solution, sélectionner Serie et trouver le no de port via le gestionnaire de périphérique, avant connexion de la carte Arduino.
Le programme de test le plus simple envoie une lettre tous les 0.1s. Facile de vérifier que l'écran du terminal se remplit avec cette lettre. Un testeur logique (Lopen) ou oscilloscope permet de suivre le signal depuis sa source si cela ne passe pas.

Programme de test complet

```
//EnvoiSer.ino  698 bytes
#include "Ser.h"
void setup() {
  SetupUart();
  SndSer('O');SndSer('K');
}
byte car='d';
void loop () {
  SndSer(car);
  delay (100);
}
      - - - -
//Ser.h
#include <Arduino.h>
#define RX_READY (UCSR0A&0x80)//
check bit7
#define TX_READY (UCSR0A&0x20)//
check bit5

void SndSer( byte dd) {  // write
bloquant  Ok initial
  while(!TX_READY); // wait
  UDR0=dd;  // OK, send it now!
}
byte GetSer() {  // read bloquant
  while (!RX_READY); // wait
  return UDR0;
}
void SetupUart() {
  UBRR0=F_CPU/(9600*16L)-1; // set
speed
  UCSR0B=0x18; // - - - rxe txe - - -
 UCSR0C=0x06; // set mode: 8 data bits,
                // no parity, 1 stop bit
}
```

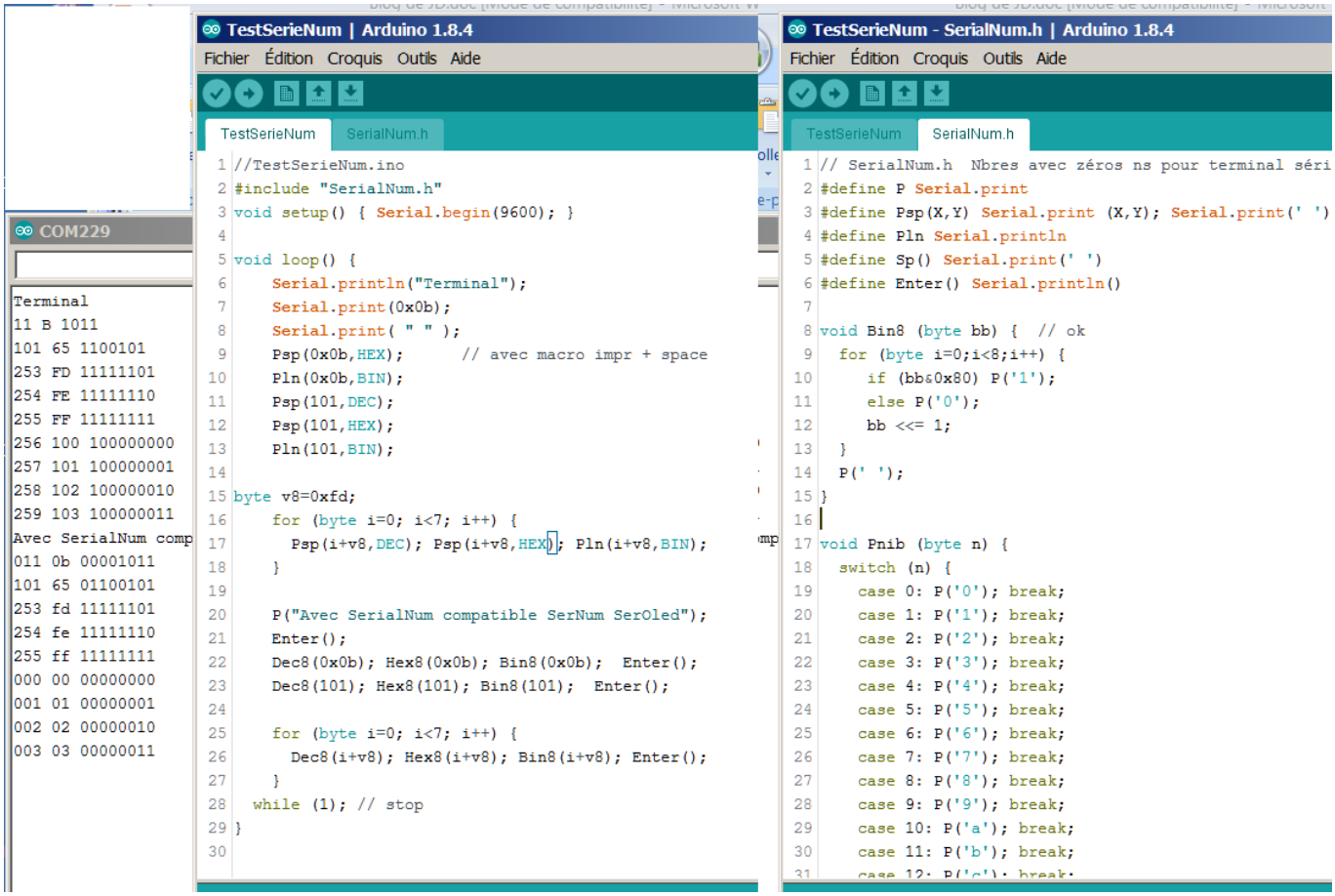
Exemple de séquence d'affichage

```
//ExempleSer.ino
//On a 3 variables à afficher On crée la fonction
Liby
// qui sera appelée par le programme à tester
// On a appelé un fichier vide.h que l'on devra
renommer
// plus tard dans l'explorateur ou avec Notepad.
// (voir
http://www.didel.com/coursera/FichiersInclus.pdf)

// Variables globales utilisées dans des
firchiers inclus
// et dans le programme principal
byte liby0,liby1,liby2;
#include "Base.h"
#include "Ser.h"
#include "SerNum.h"
#include "vide.h" // sera renommé en AfLiby.h
void setup() {
  SetupUart();
  AfCar('O'); AfCar('K');
}

void loop () {
  // appel algorithme calcule liby0 1 2
  Dot24(); // affichage des valeurs calculées
  delay (100);
}
// pourrait faire partie du prog principal
// AfLiby.h
byte Txtliby[]={"Liby 0 1 2 "};
void Dot24 () { // Envoie sur terminal
  AfText(Txtliby); AfDec8(liby0);
  AfDec8(liby1); AfDec8(liby2);
  AfCR();
}
```

Il y a évidemment plusieurs contraintes du C à connaître, que la programmation simple "à la Arduino" évite en général.

1) <Arduino.h> soit être inclus dans le premier fichier inséré. Si on ne le fait pas, le type byte et les PORTS, entre autres, ne sont pas reconnus. En C, d'autres fichiers qui dépendent de ce que l'on utilise du microcontrôleur doivent être utilisées.
2) Une variable, une fonction doit être déclarée avant d'être utilisée. Le message " . . .was not declared in this scope" apparaît.
3) Les fichiers inclus donnent une très grande flexibilité et adaptabilité, par rapport a une librairie en C++ avec quelques fonctions et pas d'évolution. L'avantage est de pouvoir se concentre sur des modules indépendants et réutilisables. Le danger est que des modules de même nom peuvent avoir des petites différences, apportées à la mise au point.
4) Mettre les modules qui ne doivent pas changer avec les autre librairies Arduino est possible.

A 9600 bits/s, la librairie Ser.h, qui utilise les registres UBRR/UCSR du processeur, peut être émulée (bit-bang), et les pins Rx Tx assignées sur n'importe quelle sortie.
(voir www.didel.com/Serie.pdf).



4 Arduino ou autre processeur et affichage Oled I2C

Les fonctions sont les mêmes que pour TermNum.h, mais **TermOled.h** ajoute le positionnement en x/y et une possibilité graphique. A noter que ce n'est pas la libraire Adafruit qui est utilisée, mais OledPix.h (https://github.com/nicoud/Oled) qui travaille sans buffer, chaque ordre est immédiatement exécuté. On peut donc voir un compteur compter à 10 kHz (max 20Hz avec les librairies Arduino).

Si le Oled n'a que 128x32 pixels, il faut ajouter dans le setup la fonction `DoubleH();`
qui a pour effet sur un Oled de 128x64 de dédoubler toutes les lignes, donc aussi les pixels créés par Dot(x,y);.
On a donc les possibilités suivantes d'affichage:
Mode 0 Oled32 Pour SSD1306 de 128x32 pixels, 4 lignes de 21 caractères
Mode 1 Oled644 Pour SSD1306 de 128x64 pixels, 4 lignes de 21 caractères double hauteur
Mode 2 Oled648 Pour SSD1306 de 128x64 pixels, 8 lignes de 21 caractères
Mode 3 Oled646g Pour SSD1306 de 128x64 pixels, 6 lignes de 21 car. et graphique 128x16

Fonction	Description	photo
Licol(y,x);	Positionne le pointeur ligne y, colonne x	

Le librairies complémentaires OledBig.h et OledsGra.h ajoutent 5 autres fonctions.

jdn 170812/171009/171116