Apprendre le C avec le LearnCbot

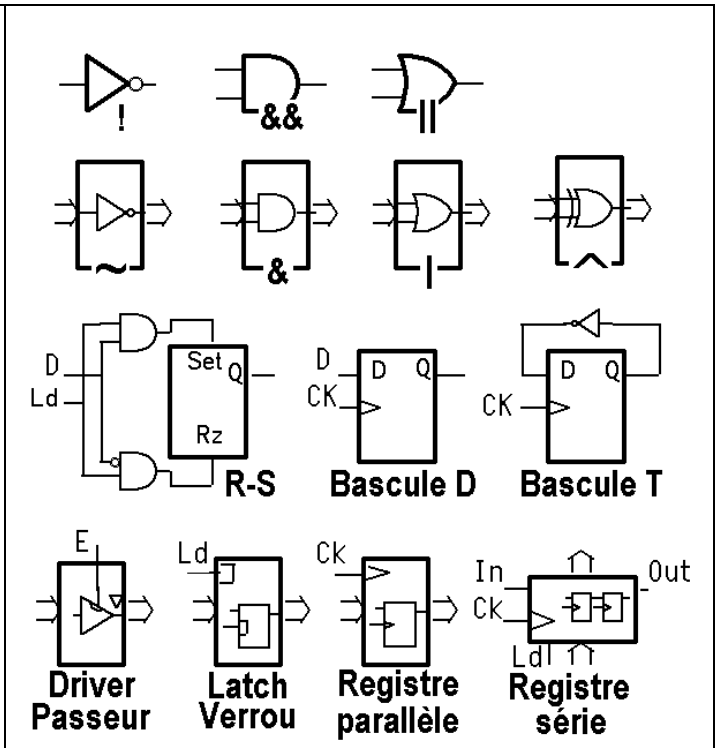
Le texte équivalent pour Energia/Msp430G se trouve sous www.pyr.ch/coursera/LC6-msp.pdf

Chap 6 – Logique, timers, SPI, I2C

Les sources des exemples sont à disposition sous www.didel.com/coursera/LC6ino.zip

6.1 Circuits logiques - rappel

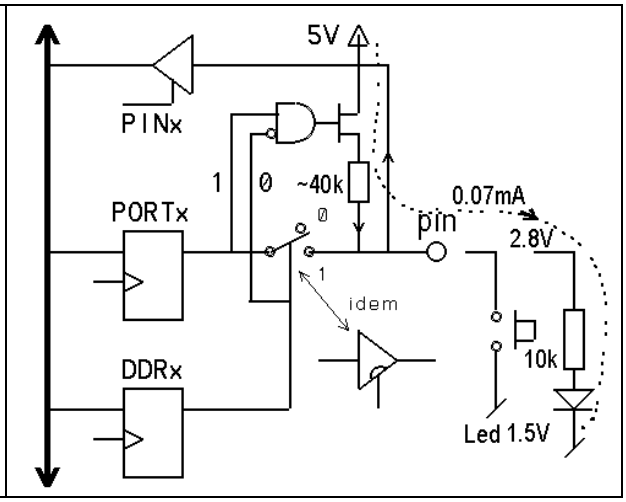
Les microcontrôleurs regroupent l'information par mots de 8 bits ou multiple. Une variable booléenne est un mot de 8 bits qui vaut 0 ou différent de zéro.
Il y a 3 types de bascules: RS, D et T (toggle), plus des types mélangés..
Il en est de même pour les registres.
La différence entre Latch statique et registre dynamique n'est souvent pas comprise et est importante:
Si Ld =1, le latch est transparent.
Ck lui n'est actif que dans la transition de 0 à 1. Si Ld est une impulsion brève, LD se comporte comme un Ck, d'où la confusion.
Un compteur a le même dessin qu'un registre. Si on peut le charger en parallèle, il faut savoir si ce chargement est statique ou dynamique. De même pour les registres à décalage.
Ces notions sont utiles pour bien comprendre des détails de l'architecture interne des microcontrôleurs, non essentielles ici.



6.2 Pull-up programmable

Sur une pin en entrée, il faut souvent brancher une résistance "pull-up" pour garantir un niveau haut quand il n'y a pas de signal, par exemples si un poussoir est connecté avec une broche vers la masse et une broche vers la pin. Contact ouvert, la tension est de 5V, contact fermé, elle est de 0V.
La condition pour activer la pull-up (~4k) est que la pin soit en entrée et que l'on programme une "1" sur le port.

Que se passe-t-il si la pin d'une Led active à "1" est programmée en entrée avec la pull-up active? Le courant qui passe est suffisant pour allumer la Led. La tension et le courant indiqués sont-ils corrects?



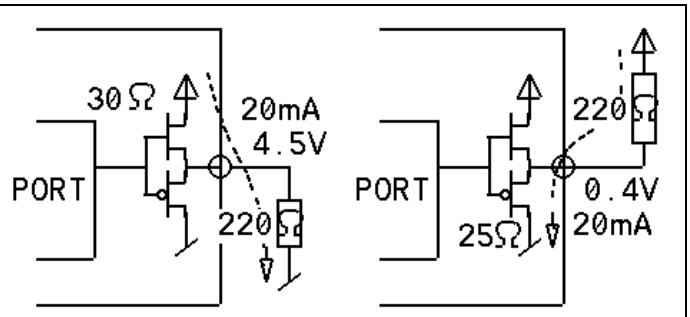
Exemple 6.21 - une curiosité AVR

On peut lire et écrire les registres `DDRx` et `PORTx`. Cela n'a pas de sens d'écrire sur `PINx`. Mais cela peut être utilisé pour autre chose. Si on écrit sur PINx, le bit du PORTx bascule (toggle)!
Probablement le truc AVR est plus rapide de 0.2 microseconde, mais utiliser un truc peu connu, même dans un fichier de définitions, entraîne une perte de lisibilité. Vérifions que cela marche.

```
//A621.ino   Toggle le bit 4 du Port D
#include "LcDef.h"
void setup () {
  LcSetup ();
}
void loop () {
   bitSet (PIND,bL1);
   delay (500);
}
```

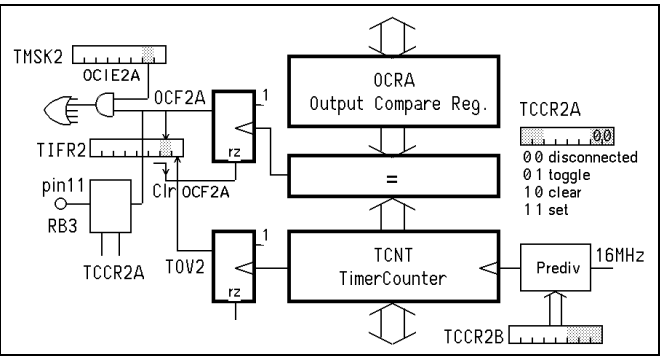
6.22 Courant de sortie

Profitons de rappeler ce qu'il faut savoir à propos des sorties. Les transistors de sortie ont une résistance de 25 et 30 Ohm.
Le courant max est documenté 20mA, mais on peut aller jusqu'à 40mA sur quelques pins (il faut penser à l'échauffement total).
Si on court-circuite une sortie au Gnd ou +5V, le courant est de 100-200 mA, ce qui est destructif au bout de quelques secondes.



6.3 Timer en "output compare"

Associé à un compteur-timer, on trouve en général un registre appelé "output compare", que l'on peut initialiser à une valeur quelconque. Un comparateur d'égalité active un signal quand il y a égalité entre le registre et le compteur timer. Ce signal active un flag.
En passant par un peu de logique on peut commander la pin 11 (RB3) qui passe à 1, à 0 ou change d'état quand il y a égalité. D'autres bits de commande permettent de générer du PWM.



Dans le cas du Timer2 de l'AVR 328, il y a deux OCR, on utilisera `OCR2A`. On retrouve un flag `OCF2A` et un bit de masquage de l'interruption `OCIE2A` pour tenir compagnie à ceux que l'on a vu dans les registres `TIFR2` et `TMSK2`.

Exemple 6.31

Pour générer un son, on peut dans une boucle lire TCNT, ajouter une valeur selon une durée et charger OCRA. Un attend sur le flag `OCF2A` pour inverser la sortie HP et recommencer.
Clignoter serait plus embêtant, puisque le timer 2 tourne trop vite.

Exercice: faire la même chose par interruption (voir 5.71, le vecteur est `TIMER2_COMPA_vect`).

```
//A631 son avec output compare
#include "LcDef.h"
void setup () {
  LcSetup ();
  TCCR2A = 0x00;
  TCCR2B = 0b00000111;  // clk/1024 16kHz
  OCR2A = 64;  // pour 256 Hz
  sei();
}
byte duree=20;
void loop () {
   while (!(TIFR2 & (1<<OCF2A))) { }
   bitSet (TIFR2,OCF2A);
   OCR2A = TCNT2 + duree;
   HPToggle;
}
```

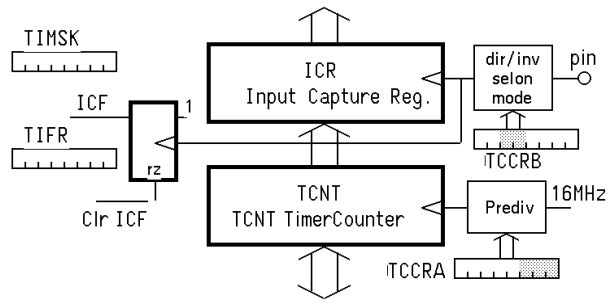
Exemple 6.32

Arduino a des fonctions pour générer des sons qui utilisent apparemment le même timer.
`tone(pin,frequence)` lance l'oscillation sur la pin à la fréquence donnée en kHz
`noTone(pin)` arrête l'oscillation
`tone(pin, frequence, durée)` lance l'oscillation pour la durée indiquée.
Ces appels ne sont pas bloquants. Le 3e doit utiliser deux interruptions différentes.
A l'oscilloscope, on voit des interruptions de 10 microsecondes, parfois distantes de 20-30 us.

```
//A632.ino Quelques sons
#include "LcDef.h"
#define HP 1 // pin (LcDef déclare un no de bit bHP)
void setup () {
  LcSetup ();
}
void loop () {
   tone (HP,432);
   delay (500);
   noTone (HP);
   delay (500);
   tone (HP,400,500);
   while (1) {}
}
```

6.4 Timer en "input capture"

Le register ICR "photographie" le timer-compteur quand la pin associée est activée.
Un flag est activé simultanément et le programme sait, en lisant le flag ICF, que la pin a été activée et que l'instant est mémorisé.
Sur AVR 328, cette fonctionnalité n'existe que pour le timer 1.



Mesurer l'instant où se produit une action, ou des durées d'événements est fréquemment utile. On verra à propos de capteurs ultrasons la fonction Arduino `Pulsin()` utile pour mesurer la durée jusqu'à la transition d'un signal.

6.42 Utilisation des timers par Arduino

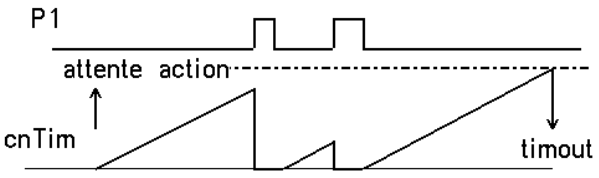
Le microcontrôleur AtMega328 a 3 timers utilisés par les librairies Arduino.
Le **Timer0** est 8 bits et est utilisé pour les fonctions delay(), millis() et micros().
Il commande le PWM sur les pins 5 et 6, apparemment sans conflits.
Le **Timer1** est 16 bits et est utilisé pour la librairie Servo
Il commande le PWM sur les pins 9 et 10 si la librairie Servo n'est pas utilisée
Le **Timer 2** est 8 bits et est utilisé pour la fonction Tone().
Il commande le PWM sur les pins 3 et 11 si Tone n'est pas utilisé.

6.5 Timouts

Dans une application on attend souvent sur un événement. Si cet événement ne vient pas dans un certain délai, il faut exécuter une autre partie du programme. Un timout (foreclos) peut se programmer au bas niveau dans le processeur avec le "watchdog timer". Si ce compteur déborde parce qu'il n'a plus été remis à zéro, il génère un reset sur le processeur et tout redémarre. C'est essentiel pour une station météo qui peut dérailler dans un orage!
Un timout logiciel est facile à programmer, avec une variable compteur ou un timer.

Exemple 6.51

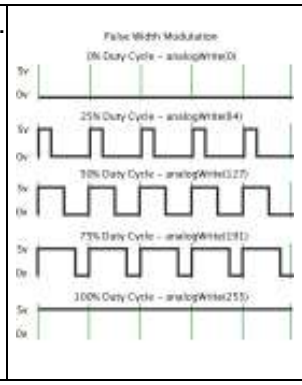
On doit presser au moins toutes les 2 secondes sur P1. Autrement, le programme se bloque avec L2 allumé.
Le compteur `cntTim` avance à chaque boucle de 20ms, dans la boucle d'attente du poussoir. Il est remis é zéro quand on presse. S'il dépasse 20*100 (2 secondes) on sort de l'attente.

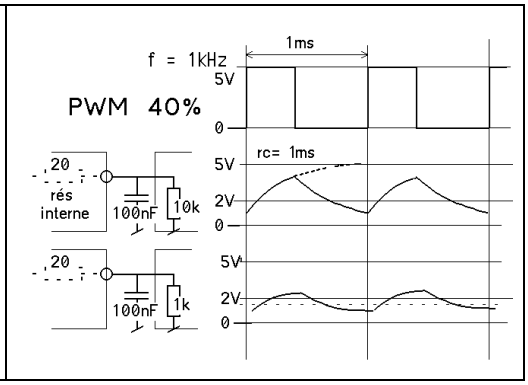



```
//A651 Timout logiciel
#include "LcDef.h"
void setup () {
  LcSetup ();
}
int cntTim=0 ;
#define MaxTim 2000/20  // 2s  20ms
void loop () {
  Led1On;
  while (cntTim++ < MaxTim) {
     delay (20);
     if (Pous1On) {
       cntTim=0;
     }
  }
  Led2On;
  while (1) ;
}
```

6.6 PWM et analogWrite

La figure explique clairement le PWM.
Arduino le génère avec la fonctions `analogWrite (pin,valeur)` sur les pins 3,5,6,8,9,11
La fréquence du PWM est de 1 kHz et il y a 256 pas.
On peut utiliser analogWrite () pour générer une tension entre 0 et 5V.
Un filtrage est nécessaire, dont l'efficacité dépend du condensateur de filtrage et de la charge.





Exercice 6.61

Pour modifier l'intensité d'une led, le PWM est parfait. Les deux leds vertes du LearnCbot (L2 et L3) sont sur des sorties PWM.
Programmons une variation d'intensité. 256 pas d'intensité n'ont pas de sens dans la plupart des applications. Pour mieux voir les différence d'intensité, programmons pour la Led2 8 niveaux linéaires.
Notre oeil a une sensibilité logarithmique, corrigeons l'intensité pour la Led3 en passant par une table.

Modifiez la longueur de la table et son contenu. Remarquez que tous les paramètres suivent - c'est programmé correctement.

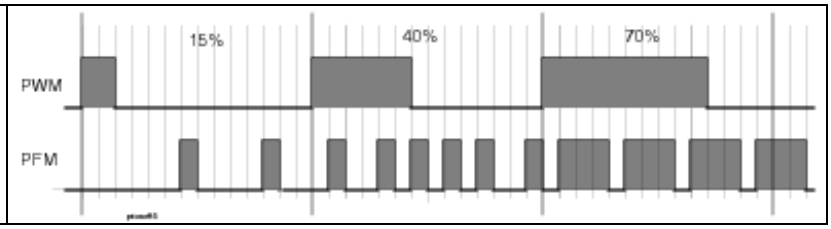
```
//A661.ino //LedVarie test linéarité
#include "LcDef.h"
#define L2 5   // no pin arduino, vert en haut
#define L3 6   // vert en bas
void setup () {
  LcSetup ();
}
const byte iled []= {0,4,7,15,30,60,120,255};

void loop () {
  // on change le PWM toutes les 0.8s
  for (int i=0 ; i<sizeof(iled) ; i++) {
     analogWrite (L2, i*(256/sizeof(iled)));
     analogWrite (L3, iled[i]);
     delay (800) ;
  }
  analogWrite (L2, 0);
  analogWrite (L3, 0);
  delay (100) ;

  // plus rapide, répété 50 fois up down
  for (int j=0; j<50; j++) {
    for (int i=0 ; i<sizeof(iled) ; i++) {
       analogWrite (L2, i*(256/sizeof(iled)));
       analogWrite (L3, iled[i]);
       delay (50) ;
    }
    for (int i=sizeof(iled)-1 ; i>=0 ; i--) {
       analogWrite (L2, i*(256/sizeof(iled)));
       analogWrite (L3, iled[i]);
       delay (50) ;
    }
  }
}
```

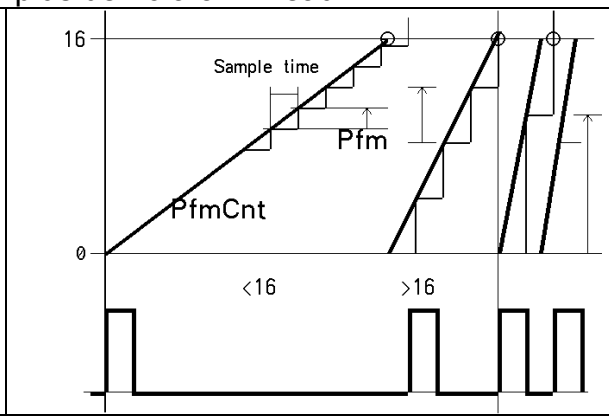
6.7 PFM

Le PFM est trop peu connu en robotique, où il permet des mouvements très lents (voir LC7).
Le principe est de rapprocher des impulsions de largeur constante selon le % voulu.



Le PFM est facile à programmer, on peut commander n'importe quelle pin en PFM, plusieurs à la fois si nécessaire. On choisit la résolution selon l'application; comme on vient de le voir pour les leds, il ne sert à rien d'avoir plus de 16 à 32 niveaux.

Prenons l'exemple avec 16 niveaux. La valeur pfm est de 0 à 15.
Une variable `pfmCnt` est augmenté à chaque cycle de la valeur `pfm`. Si le résultat dépasse 15, on soustrait 16 et on active la sortie. Autrement on désactive.



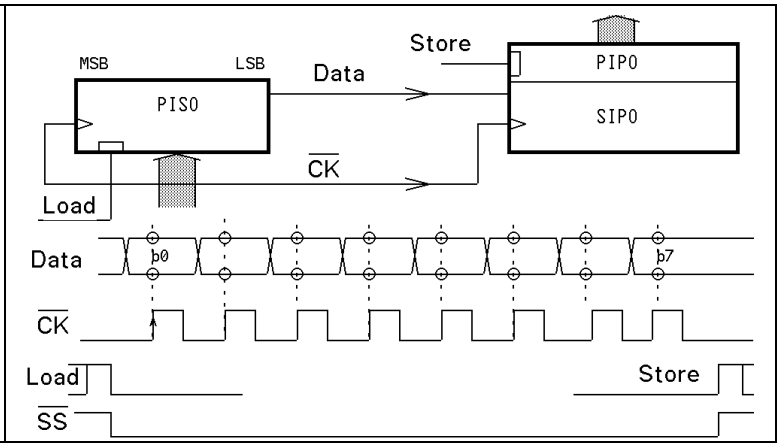
Exemple 6.71

Le programme agit sur l'intensité de la Led1. Avec un délai de 100, on voit les impulsions, et on remarque qu'elles sont parfois irrégulières, ce qui ne gêne pas à une fréquence de 1 kHz.
Changez la valeur `pfm` et le délai.
Pour une Led, le pwm est préférable. Pour un moteur, le pfm est essentiel pour tourner lentement, comme on le verra.

```
//A671 intensité selon PFM
#include "LcDef.h"
void setup () {
  LcSetup ();
}
byte pfm= 7, pfmCnt=0;  // 16 pas
void loop () {
  delay (100);
   if ((pfmCnt += pfm) > 16) {
      pfmCnt -= 16;
      Led1On;
   }
   else { Led1Off; }
}
```

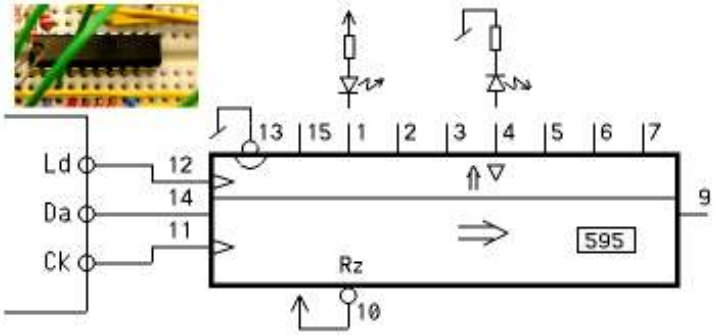
6.8 Transferts série et SPI

Le transfert d'information entre deux registres peut se faire en série. Les bits sont transférés en synchronisme avec une horloge. Un signal Load charge le registre PISO (Parallel In Serial Out). A chaque front montant de l'horloge Ck, un bit est transféré dans le SIPO (Serial In Parallel Out) et après 8 impulsions d'horloge, l'information est transférée par une impulsions Store dans le PIPO (Parallel In Parallel Out).



Exemple 6.81

Le registre 74HC595/74F595 est souvent utilisé pour piloter des Leds On peut en brancher autant que l'on veut en cascade, la broche 9 allant sur la broche 14 du circuit suivant.
Le 74F595 peut tirer un courant de 50mA par broche, il consomme 50 mA à vide.
Le 74HC595 ne tire ou pousse que 10mA, mais il ne consomme rien pour lui-même.
La fréquence maximale de l'horloge est supérieure à 5 MHz.
Le programme doit définir les signaux CkOn CkOff comme on l'a vu avec Led1On Led1Off.
Pour tenir compte du câblage des leds, actives à 1 ou à 0 (c'est mieux), il suffit de permuter DaOn DaOff.
La durée du décalage est 10 fois supérieure avec des digitalWrite, des exemples de programme sont donnés dans.
www.didel.com/diduino/SerieSPI.pdf



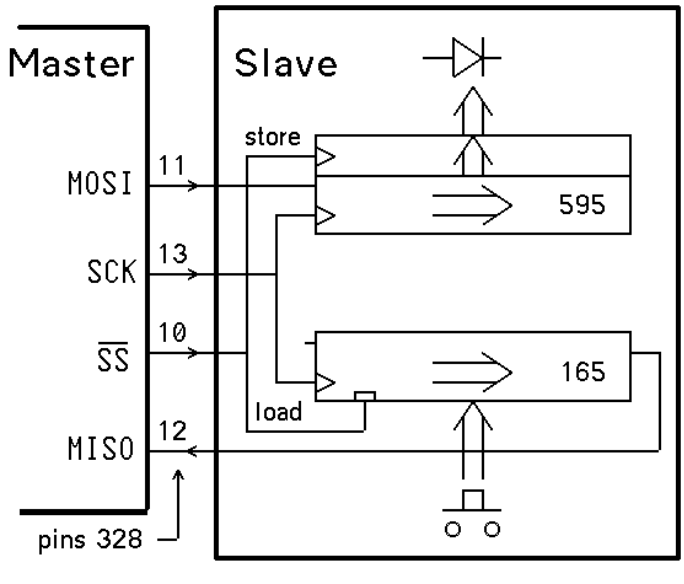
//A681.ino Transfert simple

```
void loop()  {
  int motif = 0x24 ; //
  SSOff ;
  for (byte i=0; i<8; i++) {
    if (motif & 0x01) { DaOn ; }
    else              { DaOff ;}
    CkOn ; CkOff ; // durée 0.25 us
    motif >>= 1;  // détruit motif
  }
  SSOn ;
  delay (1);
}
```

6.82 SPI

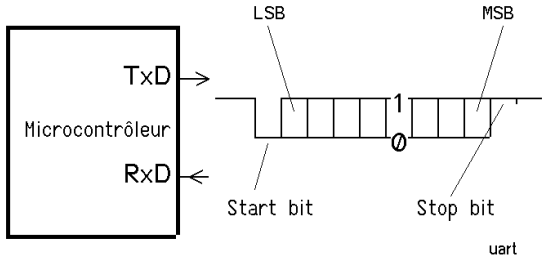
SPI documente 4 signaux, et offre des options pour la polarité de l'horloge. La fonction Arduino `shiftOut(Da,Ck,dir,val8)` est lente et limitée à 8 bits. Si on connaît le circuit à contrôler, la fonction de transfert est facile à écrire.

Le Atmega 328 a des registres interne et des flags pour gérer les transferts SPI par interruption. Les pins utilisées sont imposées par le microcontrôleur. Des librairies Arduino existent, mais si elles ne sont pas associées à un périphérique spécial et bien documenté, il n'y a pas intérêt à les utiliser. Elles forcent le câblage et ajoutent beaucoup de code sans gain de temps. On trouve facilement sur le web la documentations SPI.



6.9 Série UART

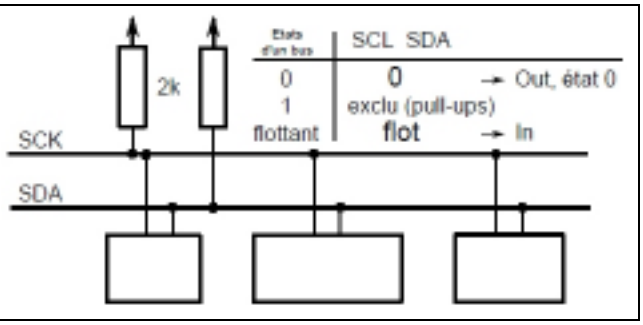
Pour éviter de transmettre le signal d'horloge, on utilise une horloge locale de même fréquence dans la source et la destination. Une transition initiale synchronise les instants d'échantillonnages. Un stop bit équilibre les délais et permet de reconnaître le prochain start bit.
La durée des bits est standardisée. Le plus souvent 9600 bits/sec (ne parlez pas de ‘’Baud rate’’ devant un spécialiste).



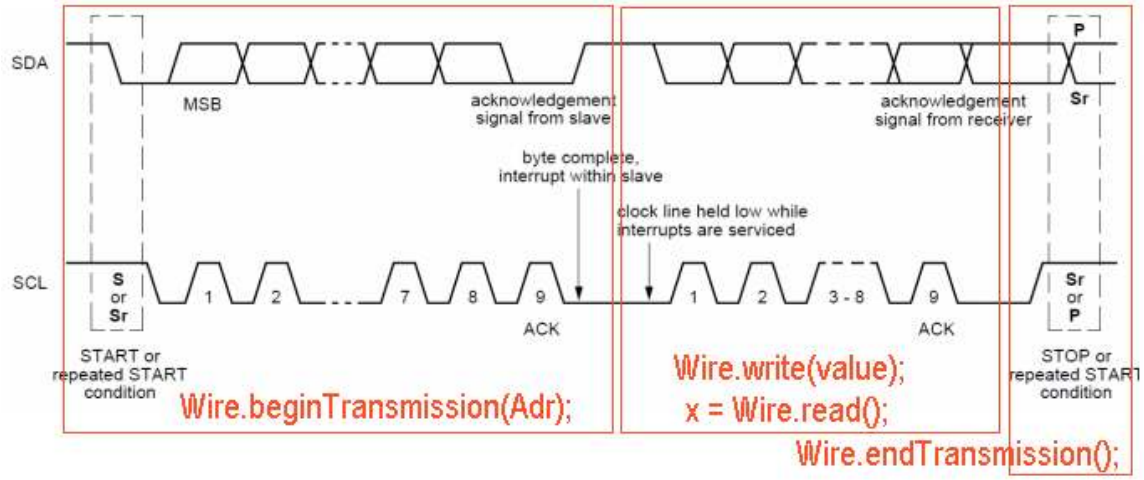
Arduino passe par USB pour retrouver sur le PC une émulation de l'entrée série RS232 que l'on trouvait au siècle dernier sur les PC. Ce sont les COM ports, un héritage souvent pénible à gérer dans les applications qui relient un Arduino à un PC, via Bluetooth, Xbee ou autre.

6.10 I2C / TWI / SMBus

I2C est un bus sur lequel plusieurs unités peuvent se connecter. Le maître envoie une adresse, l'esclave quittance, et les données peuvent être transférées par bloc.
Pour écrire sur le bus, si c'est un 1, le port est en entrée et une pull-up garanti l'état "1". Si c'est un 0, le port est initialisé à l'état "0" (et reste toujours dans cet état); il suffit de le commuter en sortie pour que le bus soit à l'état "0".



La grande idée de l'I2C est d'avoir décidé que le data ne doit pas changer quand le clock est à "1". Ceci permet de coder un start et un stop. Les mots de 8 bits avec un acknowledge se succèdent entre ces marqueurs, le premier mot étant une adresse 7 bits complétée pas un bit qui indique si les mots suivants sont en lecture ou en écriture.
La librairie `#include <wire.h>` est facile à utiliser. Le set-up doit contenir `Wire.begin();` L'adresse 7 bits est décalée à gauche puisque le bit de poids faible donne la direction. Il ne faut pas se laisser troubler par les habitudes de chacun. Pour le circuit intégré PCF 8574 par exemple, le fabricant documente l'adresse `0b0100000 = 0x40` en écriture. `Wire` traite la direction séparément, et décale, donc demande `0b00100000 = 0x20` et l'exprime en décimal: `32`. Voir www.didel.com/diduino/I2C.pdf pour une application de ce circuit qui gère 8 entrées-sorties avec Wire.h et en C et voir www.didel.com/kidule/CkiClock.pdf pour une application du circuit horloge DS1307.



Si on veut lire plusieurs mots, cela passe par une mémoire tampon.
`Wire.requestFrom(address, quantity);` demande des bytes et les mets dans le tampon
`while (Wire.available() { . .` est utilisé pour vider le tampon avec
`byte x = Wire.read();`