http://www.didel.com/ info@didel.com

www.didel.com/diduino/CommandeLeds.pdf

Commande de Leds

Programmer des Leds est amusant, et permet d'apprendre à programmer en C de façon approfondie. Des aspects électroniques sont détaillées dans www.didel.com/kits/Leds.pdf
Les exemples de ce documents sont expliqués pour Arduino. Les programmes se trouvent sous www.didel.com/C/ComLedsArduino.zip
Avec Pinguino, il y a quelques différences mineures expliquées sous www.didel.com/C/ComLedsPinguino.pdf avec des programmes exemple.
Avec le PicStar et le compilateur MCC, et avec le MSP40, les primitives Arduino/Pinguino ne sont plus standard. On peut les ajouter en librairie. Des lignes directrices sont expliquées sous www.didel.com/C/ComLedsMSP.pdf avec des programmes exemple.

Table des matières

References

www.didel.com/kits/Leds.pdf
A lire en parallèle avec ce document. Les bases pour l'aspect électronique.
www.didel.com/kits/Af16Leds.pdf
Shields Arduino: voir SparcFun, Makershed, Adafruit

http://hypnocube.com/ plusieurs cubes et un simulateur
Comment préparer le câblage http://hypnocube.com/downloads/instructions/

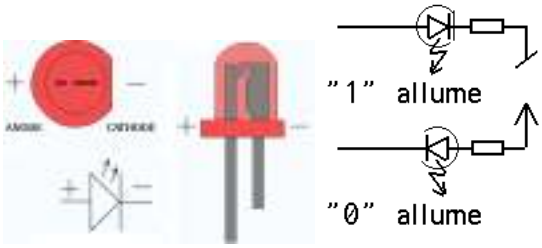
Chercher avec Google pour avoir les spécification de circuits série
Circuits simples HC4094, HC595 voir section 3.2
Circuits avec ampli intégrés TB62705 TB62706 CAT4016 A6276 STP16CP05 section 3.3
Circuits sophistiqués avec intensité CAT9532 TCA6507 TLC5940 MAX6974 MAX6960 LT8500 …

1. Quelques Leds

Les Leds sont polarisées et ont besoin d'une résistance pour limiter le courant. Leurs caractéristiques sont expliquées sous www.didel.com/kits/Leds.pdf.

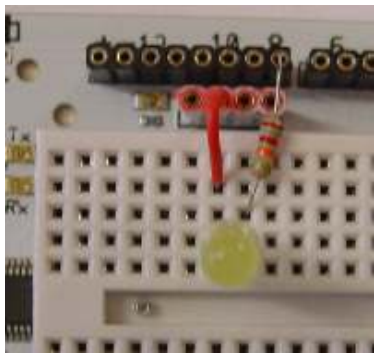
1.1 Clignoter stupide

Pour activer une Led, le courant doit passer de l'anode vers la cathode, avec en série, une résistance pour limiter le courant, d'un côté ou de l'autre.
Le microcontrôleur peut être programmé pour avoir un "1* (env 5V) ou un "0" (env 0V) sur une sortie. On a donc 2 façons de brancher une Led. Les électonicien prèfèrent mettre la cathode du côté du processeur, donc allumer avec un zéro , car l'ampli de sortie donne plus de courant, donc la Led sera plus lumineuse.



Avec Arduino, on doit déclarer la pin et l'activer-désactiver. C'est le programme Cligno0,ino dans lequel on règle le délai en millisecondes (ms).

//Cligno0.ino
```
void setup()
{
  pinMode(8,OUTPUT);
}
void loop()
{
  digitalWrite (8, LOW) ; //allume
  delay (500);
  digitalWrite (8, HIGH) ; //éteint
  delay (500);
}
```



Sur la pin 13, il y a une LED câblée avec l'anode contre le contrôleur. Elle s'allume si la pin 13 est à l'état "1". Pour le Blink, cela ne change rien, mais il faut corriger les commentaires.

1.2 Déclarer intelligent

Si on teste le programme précédent avec une autre sortie, il faut corriger à trois endroits différents. Et si on lit le programme, c'est quoi cette pin 12? Plutôt que de mettre en commentaire, définissons un nom:

```
#define LED 12
```

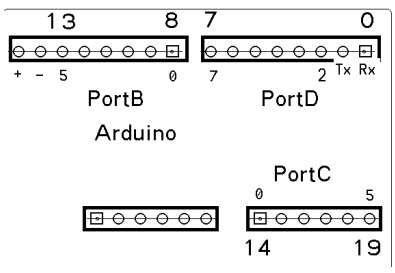
Arduino écrit `var LED 8 ;`
qui gaspille une position mémoire.
Puisque un zéro, état LOW, allume la LED, c'est clair de définir `#define ALLUME LOW`
Plus besoin du commentaire!

Si on veut mettre la LED sur une autre sortie, il suffit de changer la 1ere ligne.
Si on préfère câbles la LED en sens inverse, il faut modifier les lignes 2 et 3.
Bien déclarer permet de maîtriser des programmes complexe. Si on modifie le câblage, on ne doit pas intervenir dans le programme, uniquement dans les définitions et des fonctions spécifiques.

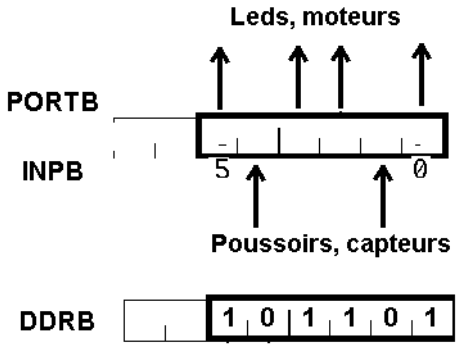
//Cligno1.ino
```
#define LED 8
#define ALLUME LOW
#define ETEINT HIGH

void setup()
{
  pinMode(LED,OUTPUT);
}
void loop()
{
  digitalWrite (LED, ALLUME) ;
  delay (500);
  digitalWrite (LED, ETEINT) ;
  delay (500);
}
```

1.3 Accéder facile

Si on a plusieurs LEDs à commander, le setup et le programmes s'alourdit d'instructions longues à écrire. Arduino facilite les premiers exercices avec ses numéros de pins, mais ce n'est pas comme cela que l'on programme une application industrielle en C. Le schéma ci-contre rappelle ces numéros de pins, avec la dispodition des connecteurs des cartes Arduino. Les connecteurs correspondent aux trois ports du processeur.



Le microcontrôleur du Diduino, un AVR Atmega-328, a 3 ports , B, C, D. Ces ports de 8 bits ont des entrées/sorties numérotées de 0 à 7, mais ils sont parfois incomplets.
Il faut comprendre que chaque ligne de ces ports peut être une entrée (lire un interrupteur) ou une sortie (allumer une LED). Chaque port est associé à un registre de direction appelé DDR. Donc le port B contient des aiguillages commandés par DDRB; s'il y a des "1" dans ce registre , le sorties correspondantes sont actives, la sortie est à 0 ou 1 selon le contenu du registre PORTB.



S'il y a des "0" dans DDRB, les pins correspondantes sont en entrée, sans état défini (on dit flottant), et on peut brancher un poussoir ou un capteur sans risque de court-circuit.
Pour lire ces entrées, on doit accéder le registre INPB, qui donne aussi la valeur des sorties que l'on a préparé.

Ecrivons le programme qui clignote les 2 Leds de droite et les deux Leds de gauche alternativement.

```
// Cligno2.ino   Clignote 5-3 et 2-0
#define DirB 0b00101101 // 1 sortie
#define Motif1 0b00000101 // allume 5-3
#define Motif2 0b00101000 // allume 2-0
// 0 allume, 1 éteint, seules les sorties sont concernées
void setup ()
{
      DDRB = DirB ;
}
void loop()
{
  PORTB = Motif1 ;
  delay (500);
  PORTB = Motif2 ;
  delay (500);
}
```

Vous préférez la méthode Arduino?

```
//Cligno3.ino Clignote 13-11 et 10-8
void setup ()
{
      pinmode (13,OUTPUT) ;
      pinmode (11,OUTPUT) ;
      pinmode (10,OUTPUT) ;
      pinmode (8,OUTPUT) ;
// les lignes non déclarées sont en entrée
}
void loop()
{
      digitalWrite (13, LOW) ; //allume
      digitalWrite (11, LOW) ;
      digitalWrite (10, HIGH) ;
      digitalWrite (8, HIGH) ;
      delay (500);
      digitalWrite (13, HIGH) ; //eteint
      digitalWrite (11, HIGH) ;
      digitalWrite (10, LOW) ;
      digitalWrite (8, LOW) ;
      delay (500);
}
```

Les ports ne sont pas complets (8 bits) ce qui demande parfois des explications supplémentaire. La direction pour les deux lignes Rx Tx est importante

Remarque: une difficulté du C (des microcontôleurs en fait) est de bien savoir quand on raisonne avec des bits (pins) et avec des mots de 8 bits (ports), et d'utiliser les notations adaptées.

1.4 Pull-ups

On a vu que des pins en entrée sont flottantes. Pour lire un poussoir, qui court-circuite la ligne à "0", il faut mettre une résistance de tirage vers le haut (pull-up) qui définit l'état "1" si le poussoir est relâché.
AVR a un truc pour mettre en service des résistances pull-up interne de 100 kOhm environ sur des entrées choisies: il suffit d'écrire dans le PORTB la valeur "1" pour ces lignes, et "0" si on ne veut pas de pull-up.
Dans l'exemple précédent, il n'y a pas de pull-up. Si on en veut sur les 2 entrées, il faut changer les motifs.
En Arduino, on agit dans le set-up, puisque les instructions ne modifient que les pins mentionnés

```
#define Motif1 0b00010111
#define Motif2 0b00111010
  etat actif de façon interne   ^   ^
```

```
void setup ()
.....
digitalWrite (12, HIGH) ;
digitalWrite (9, HIGH) ;
}
```

1.5 Entrées analogiques

Mentionnons les entrées analogiques sur le portC. Le processeur contient des registres de configuration et de lecture pour 10 canaux analogique. C'est compliqué, et Arduino facilite bien le travail avec les instruction analogRead (pin) sur les pins 14 à 19 (PortC). Il n'y a rien à initialiser, un setup particulier se fait quand le compilateur voit que l'on utilise la pin en mode analogique.

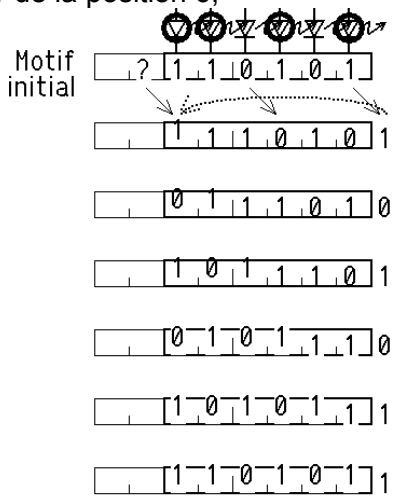
L'exemple ci-contre montre comment lire un potentiomètre et afficher son contenu avec la fonction terminal série. La valeur du pot varie la vitesse de clignotement. La valeur lue est entre 0 et 1023 (10 bits). La valeur 0 correspond à un délai de plus d'une heure (65536 millisecondes). L'affichage série ralentit naturellement; transférer la mesure et le CR prend environ 5 ms.

```
// Cligno4.ino  Vitesse selon pot
#define DirB 0b00101101
#define Motif1 0b00000101 // allume 5-3
#define Motif2 0b00101000
#define PinPot 14
int  Pot ;
void setup()
{
    DDRB = DirB ;
    Serial.begin(9600);
}
void loop()
{
  Pot = analogRead(PinPot);
  Serial.println(Pot);
  PORTB = Motif1 ;
  delay (Pot);
  PORTB = Motif2 ;
  delay (Pot);
}
```

1.6 Chenillards

Branchons 6 LEDS sur le PORTB, actives à 0. L'idée est de charger un motif, et ensuite de le décaler en rond. L'instruction `motif >>1 ;` permet de décaler d'un cran à droite, mais ce que l'on voudrait c'est que ce qui déborde à droite soit injecté à gauche (décalage circulaire). Pour cela, il faut tester le bit qui va se perdre et forcer le bit qui va arriver de la position 6,

A noter que l'on préfère travailler sur la variable et copier sur le port.




```
//Cligno5b.ino chenillard sur 6 leds portc
int motif = 0b00110101 ;  // "1" allumé
void setup()
{
  DDRC = 0b00111111 ; // "1" sortie
}
 void loop()  //cligno5b.ino
 {
   if  (motif & 1<<0)
     motif |= 1<<6 ;
   else
     motif &= ~(1<<6) ;
   motif >>=1 ;
   PORTC = motif ;
   delay (1000) ;
 }
```

```
void loop()   //cligno5.ino
{
  if (motif&0b00000001)
    motif |= 0b01000000 ;
  else
    motif &= 0b00111111 ;
  motif >>=1 ;
  PORTC = motif ;
  delay (1000) ;
}
```

Ce programme révise les opérations logiques! En écrivant `if (motif&0b00000001)` on obtient une valeur qui vaut 0 ou 1 selon le dernier bit, donc une valeur logique. Si elle vaut 1, donc vraie, le if va forcer le bit 6 à 1, le else va le forcer à 0.

Avec un chenillard sur 8 bits, il n'y aurait pas de problème puisque la variable a été déclarée **`int`**, donc 15 bit plus un signe. Mais le problème serait de copier les 8 bits de la variable vers les ports utilisés.

Arduino permet une autre façon de procéder, plus facile à comprendre dans les cas simples.

On déclare un tableau avec les no de pins, dans l'ordres du câblage. Les numéros de pins peuvent être dans un ordre quelconque et appartenir à des ports différents.

C'est facile d'allumer/éteindre dans l'ordre (boucle **`for`**), mais moins évident de décaler un motif.

```
//Cligno6.ino
int nLeds = 6 ;
int ledPins[] = {8,9,10,11,12,13} ;
int i ;
void setup()
{
    for (i=0; i< nLeds; i++)
    {
        pinMode (ledPins[i], OUTPUT) ;
    }
}
void loop()
{
    for (i=0; i< nLeds ; i++)
    {
        digitalWrite (ledPins[i],LOW ); // allume
        delay (1000) ;
    }
    for (i=nLeds-1; i >= 0 ; i--)
    {
        digitalWrite (ledPins[i],HIGH); // eteint
        delay (1000) ;
    }
       delay (2000) ;
}
```

1.7 PWM Arduino

Arduino a une instruction `analogWrite (pin,valeur) ;` qui en fait génère des impulsions de longueur variable à 1 kHz (PWM). Avec une Led ou un moteur, le signal apparaît comme ayant des niveaux entre "0" et "1". Pour une valeur de 0 la sortie est à "0". Pour une valeur de 128, le signal est actif 50% du temps. Pour une valeur de 255 (max en 8 bits) le signal est continu à "1". L'instruction lance l'oscillation, jusqu'à ce qu'elle soit modifiée. C'est une tâche par interruption qui bloque votre programme pour 8 microsecondes toutes les 1ms.
Seules les pins 3, 5, 6, 9, 10,11 acceptent l'instructions `analogWrite`. On verra plus loin comment faire du PWM efficace sur toutes les pins que l'on veut.
Un programme de démonstration est facile à écrire. Si on copie une valeur lue sur un pot, il faut diviser par 4. Si on fait des variations lentes, deux boucles `for` permettent d'augmenter et diminuer l'intensité en alternance.
Le programme ci-contre fait varier une led tricolore câblée sur les pins 3,5,6. Les trois couleurs varient toutes les 100 ms avec trois incréments premiers entre eux. L'effet n'es pas génial, en particulier parce que l'intensité passe du max à zéro.

```
// Cligno7.ino Effets RGB
#define Red 3
#define Green 5
#define Blue 6
byte vitRed ;
byte incRed = 3 ;
byte vitGreen ;
byte incGreen = 5 ;
byte vitBlue ;
byte incBlue = 7 ;
```

```
void setup()
{
  pinMode (Red, OUTPUT) ;
  pinMode (Green, OUTPUT) ;
  pinMode (Blue, OUTPUT) ;
  digitalWrite (Red,HIGH) ;
  digitalWrite (Green,HIGH) ;
  digitalWrite (Blue,HIGH) ;
}
void loop()
{
  analogWrite (Red,vitRed) ;
  vitRed -= incRed ;
  analogWrite (Green,vitGreen) ;
  vitGreen -= incGreen ;
  analogWrite (Blue,vitRed) ;
  vitRed -= incRed ;
  delay (100) ;
}
```

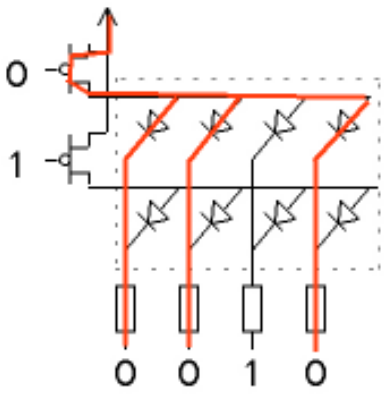
2. Plus de Leds en commande directe

2.1 En ligne

Arduino a 20 pins qui peuvent être toutes programmées en sorties pour commander de Leds.
On peut les programmer de différentes façon, avec un vecteur de pins comme on a vu, ou comme des ports dans lesquels on écrit d'un coup une information préparée en mémoire. Par exemple, si on a 20 diodes en ligne pour faire un chenillard. le plus efficace est probablement de voir ce chenillard comme 20 bits consécutifs dans une variable long (32 bits). On décale cette variable en s'occupant des extrémités et on copie dans les ports suite à des masquages et décalages.

2.2 En x-y

En câblant les leds en x-y, avec 2N lignes on commande N^2 Leds, Mais on ne sélectionne qu'une ligne à la fois, donc il faut balayer (refraîchir) sans cesse avec une fréquence supérieure à 50 HZ pour que le clignotement ne soit pas visible.
Des réseaux de 5x7, 8x8, 16x16 Leds existent à bon prix; il faut faire attention que l'on ait bien les anodes communes.
L'intensité lumineuse n'est pas réduite, car on peut augmenter le courants dans les Leds, puisqu'elles sont pulsées un bref instant.
Notons envore que l'on a tout avantage à utiliser un décodeur (HC138) pour sélectionner les lignes. Il faut alors le minimum de $\log_2 N+N$ lignes (3+8 =11 pour 64 Leds).
Pour plus de détails, consulter

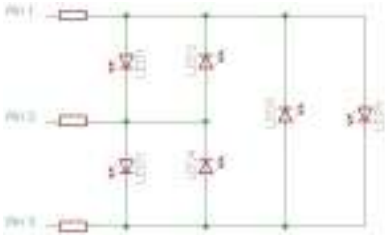


2.3 En cube

Un cube de 4x4x4 de fait avec 4 plans de 16 diodes en commandes directes, 4 lignes sélectionnant les plans. Il faut donc 20 lignes compatibles avec Arduino, et plusieurs variantes sont documentéessur la toile. La commande directe ne permet pas d'aller plus loin. On verra dans la section 3 comment faire mieux.

2.4 CharlieMultiplexing

En utilisant le fait qu'une pin du processeur peut être flottante (en entrée, elle ne peut pas fournir de courant) et que l'on peut programmer des "1" et "0", avec seulement 3 pins on peut commander 6 Leds., Amusant de chercher les combinaisons!
http://en.wikipedia.org/wiki/Charlieplexing



2.5 Bicolore 2 et 3 fils

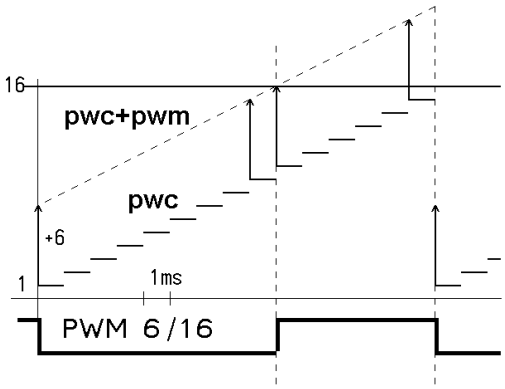
Ne répétons pas ce qui est dit dans www.didel.com/kits/Leds.pdf Il faut 2 fois plus de bits pour la commande.

2.6 Tricolore

Les 3 Leds à l'intérieur d'une Led tricolore n'ont pas la même efficacité lumineuse et il faut subjectivement adapter les résistances en série. Aussi, ces 3 Leds dans le boitier sont à une distance d'un mm ou plus et le mélange de couleurs ne trompe pas bien l'œil.
A part cela, c'est plus complexe mais mes mécanismes de commande sont les mêmes.

2.7 PWM programmé

Le processeur des timers et compteur qui permenttent de générer du PWM sur un nombre limité de sorties. Il faut bien connaître le processeur, ou utiliser des librairies.
Expliquons le programme qui génère du PWM par "logiciel". Il est rapide et il peut commander autant de pins que nécessaire. Mais il faut accepter l'idées que 256 niveaux de PWM est inutile pour nos applications. 16 niveaux est bien suffisant.
Le principe pour 16 niveaux est d'avoir un compteur pwmCount (abbrégé **pwc**) qui compte de 1 à 15. Ce compteur avance toutes les 1ms par exemple et on décide si la ou les sorties doivent être activées ou non.
Dans une application simple ou il n'y a que les leds à faire clignoter, un délai de 1ms entre les décisions PWM est ce que l'on va faire.
S'il y a d'autres tâches, cette opération pwm peut être lancées toutes les 1ms par une interruption timer, et on peut en faire une librairie facile à utiliser comme les autres librairies pwm.



Le programme est simple.et a différentes variantes
Toutes les ms on répète l'intérieur de la boucle for pour préparer **pwc**. On fait ensuite la calcul qui décide si on doit activer la Led.
Il y a plusieurs façons de préparer LedOn LedOff. Arduino écrirait digitalWrite (pin,low);
La durée de ces intructions qu'il faut appeler toutes les 1ma est de 40 microsecondes, 4% du temps processeur.

// extrait de Pwm1.ino

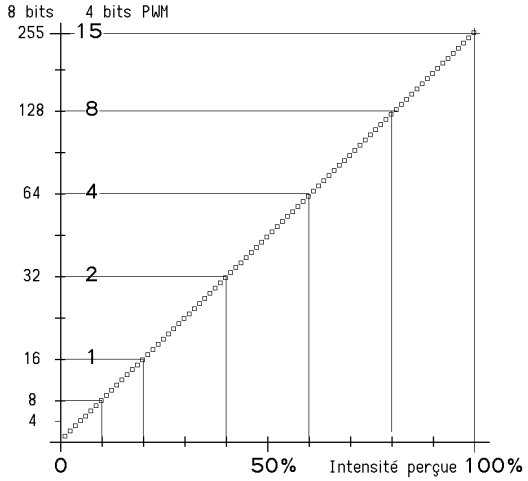
```
void loop()
{
  for ( pwc=1 ; pwc<16 ; pwc++ )
  {
      if ((pwm0+pwc)>15)  LedOn ;
      else                LedOff ;
      delay (1);
  }
}
```

Le grand intérêt de cet algorithme est que l'on peut ajouter autant de Leds en pwm que l'on veut en multipliant les `if .. else`.
L'exemple ci-contre agit sur les bits 0-5 du portB (pins 8-13) montre encore que pour les leds d'un port donné, on travaille sur un alias du port prepB. Toutes les pins sont mises à jour d'un sel coup.

```
// extrait de Pwm2.ino
void loop()
{
  for ( pwc=1 ; pwc<16 ; pwc++ )
  {
     prepB = 0 ;
     for (i=0 ; i<5 ; i++)
     {
       if ((pwm[i]+pwc)>15) prepB |= 1<<i ;
     }
     PORTB = prepB ;
    delay (1);
  }
}
```

2.8 Perception de l'intensité

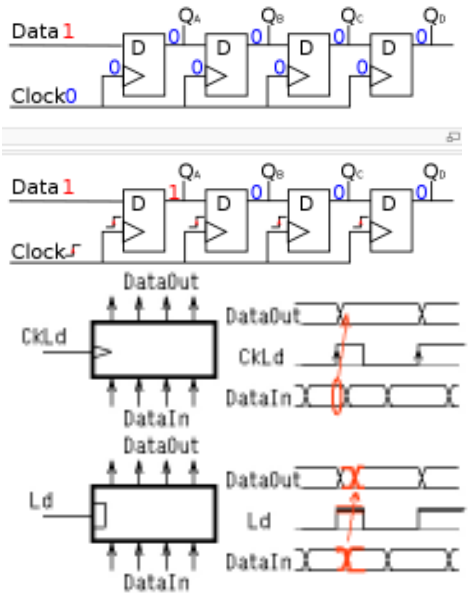
Le PWM agit de façon linéaire sur l'intensité de la Led, mais notre oeil a une réponse logarithmique. Comme le montre la figure suivante, 50% d'intensité correspond à un pwm de 30%, 10% d'intensité perçue à un pwm de 3%. Varier le pwm entre 13 et 14 change peu. Il suffit de travailler avec 5 niveaux de pwm:
0 (étent) 1 2 4 8 15 (allumé en continu)
0% ~20% 40% 60% 80% 100%
On travaille alors avec 5 niveaux d'intensité "physiologiques"



3. La simplicité du registre à décalage

3.1 Registre série et latch

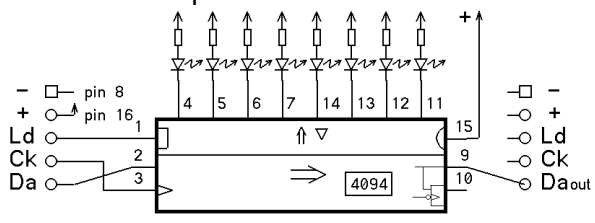
Une bascule (mémoire 1 bit) enregistre l'information au front montant d'un signal dit horloge (ck).
Plusieurs bascules en série forment un registre à décalage (sift register). A chaque coup d'horloge (ck=1 ck=0) l'information de chaque bascule passe dans la bascule suivante.

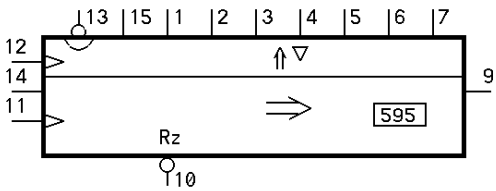


Un registre parallèle a autant d'entrées que de sorties, en un coup de clock, les entrées sont mémorisées.
Le verrou (latch) ressemble, mais le signal s'appelle load (Ld) et tant que Ld=1, les entrées sont copiées sur les sorties, la dernière information est bloquée quand Ld passe de 1 à 0. Si Ld=1, le registre est transparent, comme un ampli.

3.2 Circuits simples

Une solution simple pour débutant est d'utiliser des registres TTL et HCmos d'il y a 30 ans en boitier DIL compatible avec les blocs d'expérimentation. Attention aux différences, les explications sont souvent superficielles. Observons les 2 circuits suivants





Ck pin 2 le décalage se fait au front montant
Da pin 3 data en entrée
DaOut pin 9 data en sortie (retardé de 8 ck)
Ld pin 1 si Ld= 1 (~5V) le registre est transparent. Si Ld=0 le contenu ne change plus
Oe pin 15 Output enable doit être à 1 pour que l'information sorte du registre

Ck pin 11 idem
Da pin 14
DaOut pin 9
LdCk pin 12 L'information passe dans le registre parallèle quand LdCk passe de 0 à 1
/Oe pin 13 Output enable inversé doit être à 0 pour que l'information sorte du registre

C'est au début du programme que les définitions tiennent compte de ces différences fonctionnelles.

3.3 Transfert série

Arduino a une primitive shiftOut qui transfère 8 bits sans faire le transfert parallèle. La procédure que nous proposons va 10 fois plus vite et peut transférer 8,16 ou 32 bits.SPI est aussi utilisable, plus rapide pendant le transfert, mais il faut préparer les registres de configuratiuon SPI, surveiller

des bits. Cela peut être caché dans les interruptions, mais il faut communiquer avec la routine d'interruption!

On a une variable motif 16 bits à transférer dans un registre 16 bits. La variable peut être décalée avec l'instruction motif >>1 pour faire apparaître successivement tous les bits de la variable en position 0. Un coup de clock pousse ces bits dans le registre, et après 16 clocks, l'information est complète dans le registre série. Une impulsion ld transfère cette information interne dans le registre de sortie.

Programmes Decale0.ino …1.ino …2.ino

cœur de la procédure de décalage

```
for (i=0;i<8;i++)
{
 if (motif & 0x01)   DaOn ;
 else                DaOff ;
 CkOn ;
 CkOff ;  // durée 0.25 us
 motif >>= 1;   // décale
}
LdOn ;
LdOff ;
```

Un transfert 16 bits dure 35 us. On peut donc agir sur l'intensité globale de l'affichage en envoyant une fois le motif, puis n fois une motif éteint pour avoir une intensité de 1/n (programme Decale3.ino).

3.4 Drivers de Leds

Les circuit command de Leds les plus pratiques incorporent des amplis de sortie à courant constant. Avec une seule résistance on impose le même courant, donc la même luminosité sur 8 ou 16 Leds.

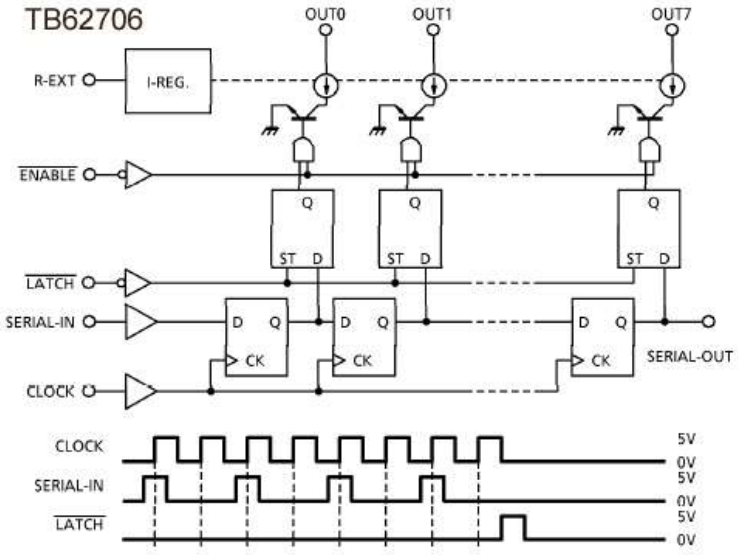
/Enable est à 0 pour activer les sorties.

/Latch est transparent à 1, donc il faut mettre un 0 pendant que cela décale

Clocl décale au front montant.

Attention /Latch = Ld de nos explications précédentes. Latch veut dire bloqué, Load ouvert, on charge!

La longueur maximale des registres que l'on peut imaginer dépends de la durée du transfert. Notre routine prend 20 us pour 16 bits. Connecter une centaine de registres laisse encore le temps pour modifier les motifs.



3.5 PWM couplé au décalage

Modifier l'intensité de chaque Led pose un gros problème de gestion des valeurs pwm que de la mise à jour allumé/éteint au niveau de chaque Led. Pour le transfert, notre technique est très efficace; elle transmet en série les bits pwm au fur et à mesure de leur calcul. Avec un pwm 4 bits, on ralenti 16 fois environ, ce qui limite à quelqes dizaines de registres 16 bits, ou 64 leds tricolores ayant chacune 15x15x15 couleurs possibles.

Le programme n'est pas si difficile à comprendre. A chaque incrément du compteur pwc, on envoie 16 bits en série, en ayant décidé selon la valeur du pwm associé à ce bit s'il faut envoyer un 1 ou un 0.

dans après les déf et setup

```
byte allPwm [16] = {15,0,15,0, 0,0,0,0,
1,2,4,8, 15,15,15,15,} ;
int i, pwc ;

void loop()
{
  for ( pwc=1 ; pwc<16 ; pwc++ )
  {
    LdOff ;
    for ( i=0 ; i<16 ; i++ )
    {
      if ((pwc + allPwm [i])> 15) { DaOn ; }
      else                        { DaOff ;}
      CkOn ;
      CkOff ;
    }
    LdOn ;
  }
}
```