Introduction aux interruptions, les interruptions externes et les interruptions à intervalle régulier.

Ateliers Arduino

par X. HINAULT
www.mon-club-elec.fr

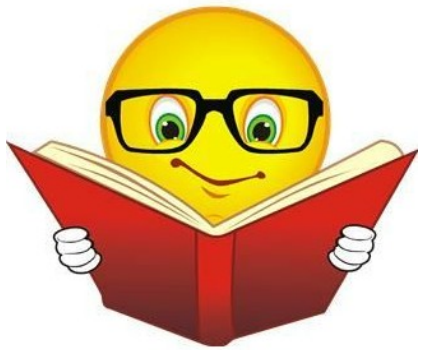
Truc d'utilisation : visualiser ce document en mode diaporama dans le visionneur PDF. Navigation avec les flèches HAUT / BAS ou la souris.
En mode fenêtre, activer le panneau latéral vous facilitera la navigation dans le document. Bonne lecture !
Lancer également le logiciel Arduino et connecter votre carte Arduino afin de pouvoir tester au fur et à mesure les codes d'exemples !

1. Intro

L'objectif ici est :

- de découvrir le concept d'interruption
- de comprendre comment fonctionne une interruption
- d'apprendre à utiliser les interruptions externes
- d'apprendre à utiliser les interruptions à intervalles réguliers

... afin d'être en mesure de créer des programmes plus efficients et optimisant les temps d'utilisation du microprocesseur de l'Arduino.



Prêt ? C'est parti !

Remarque :

Le domaine des interruptions est un peu plus compliqué à maîtriser que la programmation « classique », mais en apprenant à utiliser une interruption comme présenté ici, vous posez du même coup les bases de la compréhension des « signaux » ou « événements » qui sont très utilisés par les langages de « haut niveau » orientés objets tel que Java ou Python, notamment pour la mise en place d'interfaces graphiques côté PC.

Pratique :

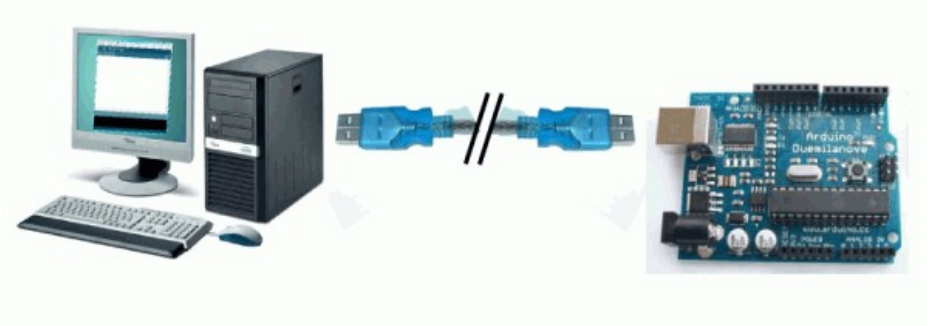
Les codes de cet atelier sont disponibles ici :

https://github.com/sensor56/255df4edf7460b622b76d2d4ebd89251

2. *Matériel nécessaire pour les ateliers Arduino*

Pour cet atelier, vous aurez besoin de tout ou partie des éléments suivants pour pouvoir réaliser les exemples proposés :

De l'espace de développement Arduino

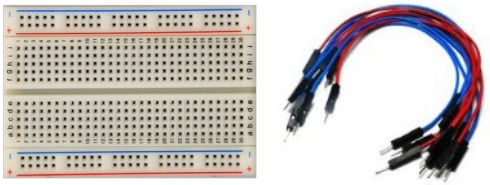


L'espace de développement Arduino associe :

- un ordinateur sous Windows, Mac Os X ou Gnu/Linux (Ubuntu)
- avec le logiciel Arduino installé (voir : http://www.arduino.cc/)
- un câble USB
- une carte Arduino UNO ou équivalente.

disponible chez : http://shop.snootlab.com/ ou http://www.gotronic.fr/

Du nécessaire pour réaliser des montages sans soudure

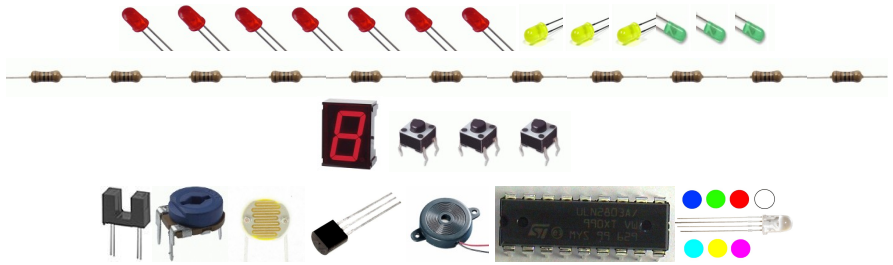


Pour réaliser des montages sans soudure, vous aurez besoin :

- d'une plaque d'essai ou breadboard moyenne (450 points)
- de quelques câbles souples (ou jumpers) mâle/mâle

disponible chez : http://www.gotronic.fr/

De quelques composants de base



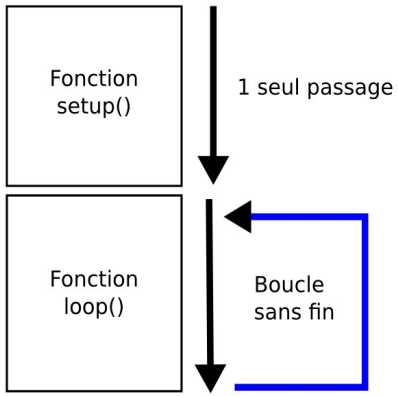
Pour les ateliers Arduino niveau débutant, vous devrez idéalement disposer des composants suivants :

- des LEDs 5mm Rouges(x20), Vertes (x5) et 3 Jaunes (x5)
- digit à cathode commune rouge 13mm (x1)
- Résistances (1/4w - 5%) de 270 Ohms (x20), 4,7K Ohms (x1), 1K Ohms (x1)
- mini bouton-poussoir (x3)
- Opto-fourche (x 1)
- Résistance variable linéaire 10K (x 1)
- Photo-résistance 7mm (x 1)
- Capteur de température LM35DZ (-55/+150°C - 10mV/°C) (x 1)
- Capsule son piézoélectrique (x 1)
- ULN 2803A (CI amplificateur 8 voies, 500mA/ voie) (x 1)
- LED 5mm multicolore RVB cathode commune (x 1)

3. Les limites de l'écriture « séquentielle » d'un programme

Le déroulement type d'un programme

- Jusqu'à présent vous avez écrit des programmes qui s'exécutaient de façon dite « séquentielle », autrement dit les instructions étaient exécutées dans l'ordre chronologique de leur enchaînement dans votre code.
- Rappelons le déroulement type d'un programme Arduino :
  - la première fonction qui est appelée est la fonction **setup()** :
    - elle n'est exécutée qu'une seule fois et en premier au début du programme,
    - on y place les instructions d'initialisation et de configuration du programme qui sont exécutées dans l'ordre.
  - ensuite, la fonction **loop()** :
    - est exécutée en boucle, se répétant indéfiniment tant que le programme n'est pas interrompu.
    - on y place les instructions à exécuter de façon répétée en boucle. Les instructions sont exécutées dans l'ordre.



- Cette structure de base du programme est complétée, comme vous le savez, au besoin par l'écriture de fonctions séparées qui sont appelées à la demande soit depuis loop() ou depuis setup().

Le problème

Cette structure de programme, même si elle performante et satisfaisante dans la plupart des situations, trouve ses limites dans au moins 2 situations type :

Lecture de l'état d'une broche en entrée

- lorsque l'on utilise des dispositif en entrée, notamment un bouton poussoir, un appui ne va être détecté uniquement si l'état de la broche du bouton poussoir est lue à ce moment précis.
- Si le programme est court, cela ne pose pas de problème, la vitesse du microprocesseur d'Arduino est telle que l'état de la broche est lue plusieurs milliers de fois par secondes.
- Mais si le programme s'allonge ou si certaines actions intermédiaires prennent du temps, l'état de la broche ne peut être lue que épisodiquement...
- Il serait bien de pouvoir détecter un changement de l'état de la broche exactement au moment où elle survient...

Temporisation et pause

- lorsque l'on souhaite répéter une action à intervalle régulier, on utilise comme nous l'avons vu une pause, avec l'instruction delay().
- lorsque l'on fait clignoter une LED par exemple, on utilise une pause entre chaque changement d'état de la LED. L'inconvénient de cette façon de faire est facile à comprendre : si on fait faire quoi que ce soit au microprocesseur, cela va modifier le délai de clignotement. Et à l'inverse le microprocesseur va passer la majorité de son temps à ne rien faire, si ce n'est attendre...
- Là encore, il serait pratique de pouvoir changer l'état de la LED uniquement au moment voulu, indépendamment des autres tâches que le microcontrôleur doit exécuter...

La solution : utiliser une interruption

- La solution à ces 2 types de situation passe parce que l'on appelle une interruption. **Une interruption, en bref, permet de stopper le programme à n'importe quel moment lorsqu'un événement déclencheur survient et à exécuter le code voulu, avant de reprendre l'exécution normalement.**
- Dans le premier cas, l'interruption sera déclenchée au changement d'état de la broche, dans le second cas, lorsque le délai voulu sera écoulé.

4. Principe d'une interruption et déroulement d'un programme utilisant une interruption

Une image pour comprendre

- Imaginez quelqu'un qui est en train de lire un livre (ou un atelier Arduino...) et soudain le téléphone sonne. Que va faire cette personne ?
  - elle va poser son livre en marquant l'endroit où elle est arrivée
  - elle va prendre son téléphone et répondre à l'appel
  - puis une fois la conversation terminée, elle va revenir s'asseoir et reprendre la lecture de son livre là où elle s'était arrivée.
- Une autre image : imaginez un enfant en train de jouer dans sa chambre à un jeu de construction. Sa mère l'appelle : « A table ! » Que va-t-il faire ?
  - il va laisser sa construction en l'état,
  - il va venir s'asseoir à la table familiale, va manger...
  - puis le repas terminé, il va retourner dans sa chambre et reprendre son jeu là où l'avait laissé.

Si vous avez compris ces 2 images, alors vous avez compris ce qu'est une interruption !

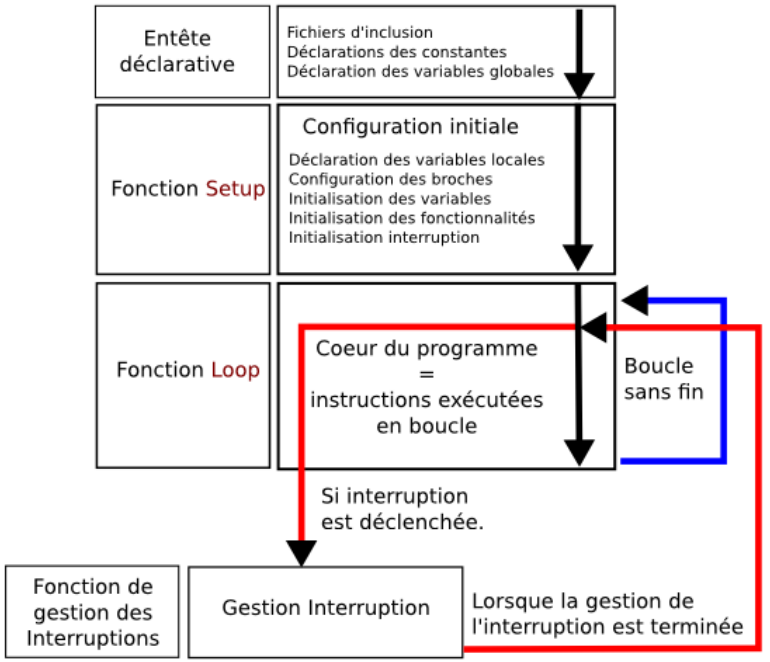
Principe général d'une interruption

- Le principe général d'une interruption est le suivant :
  - le microprocesseur est en train d'exécuter le programme dans l'ordre des instructions
  - soudain survient un événement déclencheur de l'interruption : le microprocesseur va alors interrompre ce qu'il est en train de faire tout en mémorisant l'endroit de l'arrêt
  - il va lire et exécuter le code présent dans une fonction spéciale que l'on appelle « routine de gestion de l'interruption »
  - puis, une fois terminé, il va reprendre l'exécution du programme là où il s'était arrêté.

Déroulement d'un programme utilisant une interruption

Le déroulement global d'un programme utilisant une interruption devient alors :

- exécution de la fonction setup()
- exécution de la fonction loop() en boucle
- à n'importe quel moment, lors de la survenue d'une interruption, la fonction de gestion de l'interruption est exécutée
- puis le programme reprend son cours...



L'avantage d'une interruption est triple :

Le code est exécuté immédiatement si l'événement déclencheur a lieu !

Le code est exécuté seulement si l'événement déclencheur a lieu !

Le microcontrôleur est libre de faire autre chose entre deux événements !

5. Technique : pour info : la machinerie interne d'une interruption

Le microcontrôleur, vu de l'intérieur...

- Même si l'utilisation est interruptions est simplifiée avec Arduino, il est intéressant de savoir comment çà marche au niveau matériel au sein du microcontrôleur.
- Un microcontrôleur, en interne, c'est un peu comme un cockpit d'avion :
  - il y a pleins d'interrupteurs On/Off qui permettent d'activer/désactiver des fonctions : les bits d'activation.
  - il y a plein de voyants qui signalent tel ou tel événement : les bits de drapeau.

Principe du contrôle d'une interruption

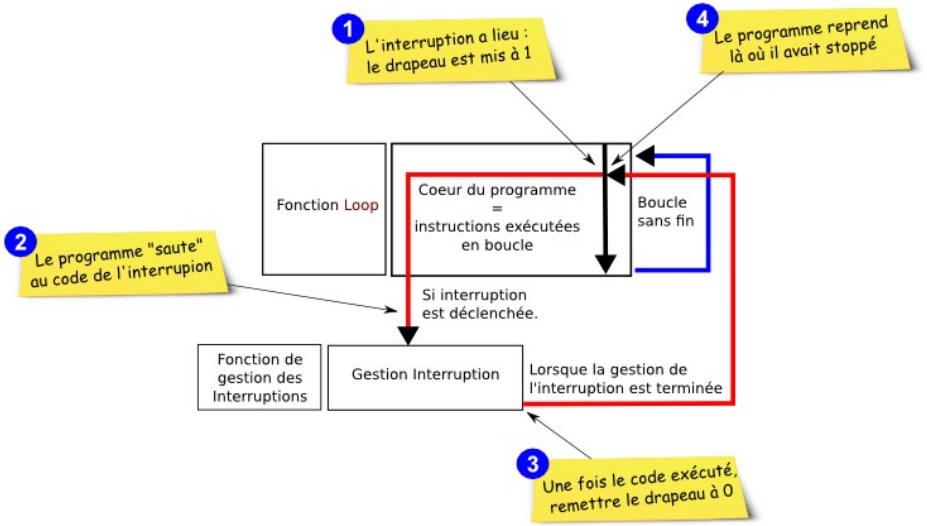
- Le contrôle d'une interruption va nécessiter typiquement :
  - un bit d'activation de l'interruption : c'est l'équivalent d'un interrupteur On/Off qui doit être mis à On...
  - un bit de drapeau flag, témoin que l'interruption a eu lieu... : c'est l'équivalent d'un témoin qui s'allume lorsque l'interruption a eu lieu...

Activation d'une interruption

- Pour qu'une interruption soit prise en compte, c'est à dire capturée lorsque l'événement voulu surviendra :
  - le bit général d'activation des interruptions doit être à 1
  - le bit d'activation de l'interruption voulue doit être à 1

Déroulement d'une interruption

- En interne, lorsqu'une interruption survient, le microprocesseur :
  - met le drapeau à 1
  - saute à une zone spéciale de la mémoire programme correspondant à la routine d'interruption (parfois appelée aussi ISR pour Interruption Service Routine) :
    - à ce niveau, au sein du code, on peut tester les « flags » pour savoir quelle interruption à été déclenchée
    - le code voulu est exécuté
  - le drapeau doit être remis à 0
  - puis le microcontrôleur quitte la routine d'interruption et reprend l'exécution du programme là où elle s'était interrompue.



Remarque

Encore une fois, les choses seront beaucoup plus simples avec Arduino, mais je prends le temps de vous détailler toute cette machinerie sous-jacente pour que vous ayez conscience de ce qui se passe lorsqu'une interruption survient.

6. Pour info : les interruptions du microcontrôleur ATMega 328 (celui des cartes Arduino UNO)

Vue d'ensemble

Pas de panique : je donne ces détails juste pour info !

- Elles sont très nombreuses, jugés plutôt :

RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
INT0	External Interrupt Request 0
INT1	External Interrupt Request 1
PCINT0	Pin Change Interrupt Request 0
PCINT1	Pin Change Interrupt Request 1
PCINT2	Pin Change Interrupt Request 2
WDT	Watchdog Time-out Interrupt
TIMER2 COMPA	Timer/Counter2 Compare Match A
TIMER2 COMPB	Timer/Counter2 Compare Match B
TIMER2 OVF	Timer/Counter2 Overflow
TIMER1 CAPT	Timer/Counter1 Capture Event
TIMER1 COMPA	Timer/Counter1 Compare Match A
TIMER1 COMPB	Timer/Coutner1 Compare Match B
TIMER1 OVF	Timer/Counter1 Overflow
TIMER0 COMPA	Timer/Counter0 Compare Match A
TIMER0 COMPB	Timer/Counter0 Compare Match B
TIMER0 OVF	Timer/Counter0 Overflow
SPI, STC	SPI Serial Transfer Complete
USART, RX	USART Rx Complete
USART, UDRE	USART, Data Register Empty
USART, TX	USART, Tx Complete
ADC	ADC Conversion Complete
EE READY	EEPROM Ready
ANALOG COMP	Analog Comparator
TWI	2-wire Serial Interface
SPM READY	Store Program Memory Ready

Classification des interruptions

- On peut distinguer notamment :
  - les interruptions sur broches E/S
  - les interruptions des Timers (0, 1 et 2)
  - les interruptions de communication série USART
  - les interruptions de communication SPI et I2C
  - l'interruption de conversion analogique-numérique
  - l'interruption d'EEprom

Interruptions « invisibles » utilisées par le langage Arduino

- Sans que vous le sachiez forcément, le langage Arduino utilise en interne certaines interruptions pour son propre fonctionnement.
- Les fonctions qui utilisent des interruptions sont :
  - les fonctions millis(), delay() et micros() utilisent le timer 0
  - la fonction tone() utilise le timer 2
  - la fonction analogWrite() utilise les timer 1 et 2
  - ...
- Les librairies qui utilisent (à priori) des interruptions :
  - la librairie Serial
  - la librairie Servo utilise le timer 1
  - la librairie Wire utiliser l'interruption I2C
  - la librairie SPI utilise l'interruption SPI
  - ...

Interruptions accessibles à partir du langage Arduino

- Ce sont ces interruptions les plus intéressantes : celles que l'on va pouvoir utiliser !
- La première chose possible avec le langage Arduino : activer ou désactiver l'utilisation de l'ensemble des interruptions.
- Les interruptions accessibles sont par ailleurs :
  - des interruptions dites externes, c'est à dire les interruptions déclenchées lors d'un changement d'état d'une broche.
  - mais aussi des interruptions des timers qui restent accessibles à l'aide de librairies dédiées... (usage « avancé »)

7. Arduino : Les instructions d'activation / désactivation générale des interruptions

Au démarrage

- La première chose importante à savoir : **les interruptions sont activées par défaut lorsque l'on exécute un programme Arduino** et certaines sont utilisées par les fonctions Arduino, comme nous l'avons précisé précédemment.

Pour info : le principe d'activation des interruptions

- Comme nous l'avons vu précédemment, pour qu'une interruption voulue soit active, il est nécessaire que son « bit d'activation » soit égal à 1
- ... autrement dit, il faut que son « interrupteur d'activation » soit sur ON, pour reprendre l'image du cockpit d'avion.
- Ceci est vrai pour chaque interruption individuelle... **mais il existe un bit d'activation général des interruptions** : c'est ce bit d'activation qui est mis à 1 ou 0 par les instructions que nous allons voir ici. **Ce bit permet d'activer / désactiver toutes les interruptions à la fois.**

Les instructions d'activation/désactivation des interruptions

- Il est donc possible, comme nous venons de le dire, à tout moment, d'activer/désactiver toutes les interruptions, ceci grâce à 2 instructions du langage Arduino :
  - l'instruction noInterrupts() : désactive toutes les interruptions
  - l'instruction interrupts() : active toutes les interruptions
- A savoir : ces instructions sont en fait une réimplémentation de deux instructions du langage C natif, qui font la même chose :
  - sei() : activation générale des interruptions
  - cli() : désactivation générale des interruptions
- Pour preuve, voici la définition des fonction interrupts() et noInterrupts() dans le fichier Arduino.h :

```
#define interrupts() sei()
#define noInterrupts() cli()
```

- La forme C sei() et cli() est directement utilisable dans un programme Arduino : il pourra d'ailleurs vous arriver de la rencontrer.

ATTENTION

Je vous donne l'information sur ces instructions d'activation / désactivation générale des interruptions uniquement pour que vous sachiez qu'elles existent.

En pratique, il est déconseillé de désactiver la totalité des interruptions car on s'expose alors à des comportements inattendus et compliqués à déboguer :

> des fonctions du langage Arduino utilisant les interruptions (delay(), PWM, etc...) qui ne fonctionneront plus,

> des librairies utilisant des interruptions qui ne fonctionneront plus, (notamment la librairie Wire, qui de plus utilise une interruption « bloquante » tant qu'une communication n'est pas terminée... ou encore la librairie Servo... etc...)

A retenir : en pratique, ne pas utiliser cli() ou noInterrupts() : çà vous évitera de perdre trop vite vos cheveux !

sauf éventuellement au sein de la routine d'interruption,
afin d'éviter un re-déclenchement intempestif de l'interruption.

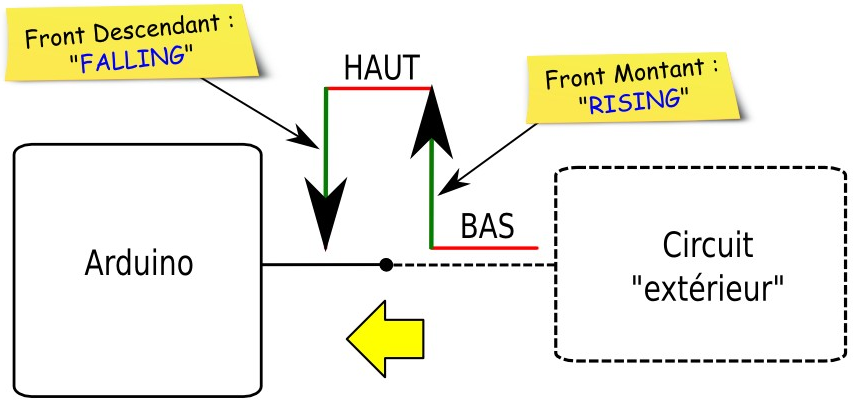
8. Arduino : Présentation des interruptions externes et des instructions associées

Les interruptions externes: principe

- Le microcontrôleur utilisé sur la carte Arduino dispose au niveau matériel de plusieurs interruptions attachées à des broches : lorsque l'état de la broche sera modifié, l'interruption sera déclenchée.
- C'est ce que l'on appelle une interruption « externe » : ceci est potentiellement très pratique pour prendre en compte un événement dès qu'il survient.
- La bonne nouvelle, c'est que le langage Arduino dispose de plusieurs instructions qui permettent d'utiliser et paramétrer simplement ces interruptions dites « externes ».

Les différents événements déclencheurs possibles

- Comble de la sophistication, l'interruption externe va pouvoir être déclenchée lors de la survenue d'une modification précise de l'état de la broche et il va logiquement être possible de choisir la modification qui devra déclencher l'interruption.
- Une broche pourra se trouver dans 4 états différents :
  - 2 états fondamental :
    - soit niveau HAUT (ou HIGH)
    - soit niveau BAS (ou LOW)
  - et 2 transitions :
    - soit front descendant (ou « FALLING »)
    - soit front montant (ou « RISING »)



Les broches utilisables

- La plupart des cartes Arduino ont deux interruptions externes :
  - interruption externe n°0 sur la broche numérique 2
  - et interruption externe n°1 sur la broche numérique 3.
- La carte Arduino Mega en possède quatre de plus : interruption externe n°2 sur la broche 21, n°3 sur la broche 20, n°4 sur la broche 19 et n°5 sur la broche 18.

Les instructions Arduino utilisables

- Les instructions de configuration des interruptions externes sont les suivantes :
- Tout d'abord, pour configurer l'interruption externe, l'instruction attachInterrupt (interruption, fonction, mode) où :
  - **interruption** : le numéro de l'interruption (type int) :
    - 0 pour la broche 2
    - 1 pour la broche 3
  - **fonction**: la fonction à appeler quand l'interruption survient; la fonction ne doit recevoir aucun paramètres et ne renvoie rien. Cette fonction est également appelée une routine de service d'interruption (ou ISR).
  - **mode** : définit la façon dont l'interruption externe doit être prise en compte. Quatre constantes ont des valeurs prédéfinies valables :
    - LOW : pour déclenchement de l'interruption lorsque la broche est au niveau BAS
    - CHANGE : pour déclenchement de l'interruption lorsque la broche change d'état BAS/HAUT, quelque soit le sens
    - RISING : pour déclenchement de l'interruption lorsque la broche passe de l'état BAS vers HAUT (front montant)
    - FALLING : pour déclenchement de l'interruption lorsque la broche passe de l'état HAUT vers l'état BAS (front descendant)
- Pour désactiver l'interruption externe, l'instruction detachInterrupt(interruption) où :
  - **interruption** : le numéro de l'interruption (type int)

9. Arduino : Variables et interruptions : utiliser le qualificateur de variable volatile

Le stockage des variables « classique »

- Typiquement, lorsque l'on utilise une variable au sein d'un programme Arduino celle-ci est stockée dans un registre, espace de mémoire temporaire.
- Dans le cas de l'exécution d'un code classique, cela ne pose aucun problème. Par contre, dès lors que l'on va utiliser les interruptions, dans certaines situations, les variables « classiques » peuvent ne plus être accessibles à partir du code l'interruption, ce qui pose problème.

La solution ? Utiliser le qualificateur de variable « volatile »

- Il est possible avec le langage Arduino d'utiliser ce que l'on appelle un « qualificateur de variable », qui va indiquer un comportement particulier d'une variable donnée.
- Vous connaissez déjà le qualificateur de variable const qui transforme une variable en une constante, c'est à dire que sa valeur ne sera pas changée au cours du programme (voir l'atelier consacré aux variables).
- Dans notre cas, nous allons pouvoir utiliser le qualificateur de variable volatile
- Comme son nom ne l'indique pas, ce qualificateur va indiquer à Arduino (au compilateur pour être précis), de stocker la variable en RAM de façon à ce qu'elle soit accessible en tout point du code, notamment aussi bien dans la routine d'interruption que dans les fonctions loop() et setup()

Principe d'utilisation

```
volatile int state = LOW; // déclaration variable stockée en RAM
```

Code d'exemple

```
// inverse l'état de la LED quand une interruption par changement d'état d'une broche survient

int pin = 13;
volatile int state = LOW; // déclaration variable volatile = stockée en RAM

void setup()
{
  pinMode(pin, OUTPUT);
  attachInterrupt(0, blink, CHANGE); // Attache l'interruption à la fonction blink
}

void loop()
{
  digitalWrite(pin, state); // met la broche dans l'état voulu
}

void blink() // la fonction appelée lorsque l'interruption survient
{
  state = !state; // inverse l'état de la variable
}
```

Voilà, à ce stade, vous êtes parés pour passer à l'action avec les interruptions externes, ce que nous allons faire à présent !

10. Rappel : Fiche composant : découvrir le transistor et le photo-transistor

Description

En électronique, le transistor est un composant semi-conducteur (il en existe 2 types dits PNP ou NPN) dans un petit boitier qui dispose de 3 broches :

- la base B qui reçoit une intensité de déclenchement Ib
- le collecteur C qui laisse entrer une intensité Ic porportionnelle à Ib
- l'émetteur E qui laisse sortir une intensité valant Ie=Ic+Ib

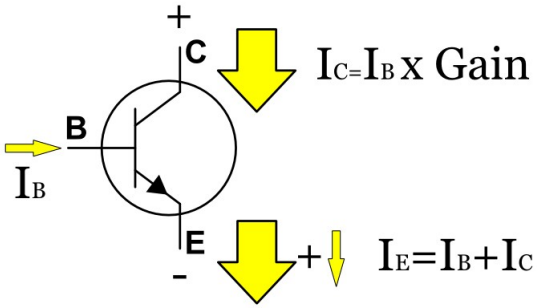
A savoir :

Le transistor est un composant essentiel, qui date des années 40, et qui a révolutionné l'électronique et permis l'apparition de l'électronique numérique et des ordinateurs. Les processeurs des ordinateurs actuels possèdent des millions de transistors miniaturisés !!

Le micro-contrôleur de votre carte Arduino lui-même intègre environ 500 000 transistors !

Principe de fonctionnement

Le principe fondamental de fonctionnement d'un transistor est le suivant : une petite intensité circulant sur la broche de la base va provoquer la circulation d'une intensité importante proportionnelle entre le collecteur et l'émetteur.



Le Transistor NPN

Pour faire simple, on peut dire qu'un transistor est un « multiplicateur » d'intensité : il multiplie l'intensité de la base et l'intensité résultante circule entre le collecteur et l'émetteur. Le coefficient multiplicateur est appelé **gain.**

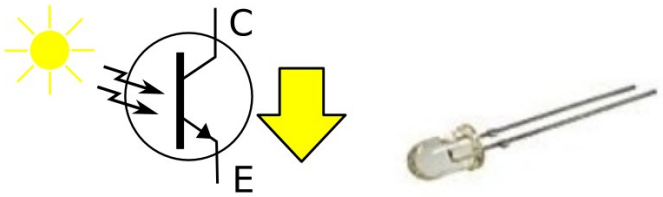
Modes de fonctionnement d'un transistor

Le transistor est un composant qui peut être utilisé aussi bien en mode analogique que « numérique » :

- **en mode analogique,** la variation d'intensité sur la base se répercute immédiatement en variation d'intensité du collecteur. C'est ce principe qui est à la base des amplificateurs audio et autres appareils de radio (dont lui vient d'ailleurs le nom de transistor).
- **en mode « numérique » ou « ON/OFF » appelé également mode saturé** : dès qu'une intensité est présente sur la base, le courant de collecteur est d'emblée maximal. L'absence de courant sur la base ne laisse passer aucun courant de collecteur. C'est une sorte d'interrupteur à commande électrique. C'est ce mode de fonctionnement qui est à la base de tous les circuits logiques et numériques.

Une variante du transistor : le photo-transistor

Dans ce composant, la broche de la base est remplacée par une zone sensible à la lumière infra-rouge. Le photo-transistor n'a donc que 2 broches :



Le principe de fonctionnement est le suivant : une intensité lumineuse présente sur la zone photo-sensible va provoquer la circulation d'un courant de collecteur qui sera proportionnel à l'intensité lumineuse reçue.

Le photo-transistor pourra être utilisé soit en mode analogique ou saturé.

Les transistors avec Arduino en pratique

Afin de ne pas compliquer inutilement les montages, en pratique, on n'utilisera quasiment pas les transistors « bruts » avec Arduino, mais plutôt des circuits les utilisant tels que le circuit intégré ULN 2803 qui intègre 8 étages d'amplification ON/OFF et ne nécessite aucun composant externe.

Par contre, on utilisera le photo-transistor, utilisé au sein des opto-coupleurs, comme nous allons le voir par la suite.

Remarque : l'étude des transistors et de leur utilisation est un domaine passionnant et qui peut faire l'objet de livres entiers. Ici, nous en parlons uniquement pour introduire le photo-transistor. Si vous voulez approfondir, voir notamment : http://fr.wikipedia.org/wiki/Transistor

11. Rappel : Fiche composant : découvrir l'opto-coupleur en fourche

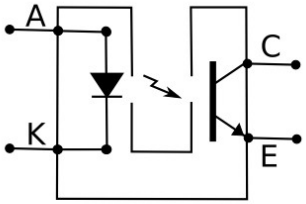
Description

L'opto-coupleur en fourche est un composant qui associe en fait 2 composants différents qui sont positionnés face à face dans un même boitier (2 broches par composant soit 4 broches en tout) :

- d'une part une photo-diode ou LED infra-rouge qui fonctionne comme une LED classique et émet une lumière invisible dite infra-rouge,
- d'autre part un photo-transistor infra-rouge utilisé ici pour détecter la présence de la lumière infra-rouge (patte courte = Emetteur).

Schéma interne

Le schéma théorique de l'optocoupleur est le suivant :



Opto-coupleur fourche
(Ex: LTH301-7)

- On retrouve d'une part la LED infra-rouge signalée par les lettres A et K sur le boitier de l'opto-coupleur correspondant à l'anode (A = +) et la cathode (K = - = patte courte)
- On retrouve d'autre part le photo-transistor signalé par les lettres C et E sur le boitier de l'opto-coupleur correspondant au collecteur (C = +) et à l'émetteur (E = - = patte courte).

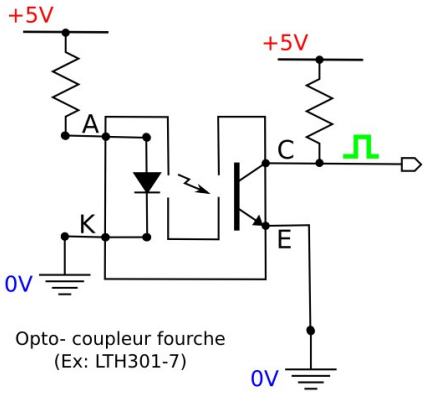
Principe de fonctionnement

- Lorsque la LED infra-rouge est allumée, la base du photo-transistor est éclairée et le photo-transistor laisse passer le courant.
- Lorsque la LED infra-rouge est éteinte, ou si un objet se trouve dans la fente, la base du photo-transistor ne laisse passer aucun courant.

Le montage type

L'utilisation de ce composant nécessite en fait la réalisation de 2 circuits :

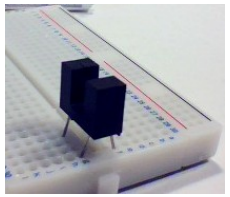
- tout d'abord, le circuit de la LED infra-rouge, qui s'utilise comme une LED standard. On pourra donc se contenter de mettre une résistance en série avec LED pour qu'elle soit allumée. Comme vu précédemment, si on désire une intensité de 13mA dans la LED, on utilisera, d'après la loi d'ohm, une résistance de R=U/I = 3,5V/0,013A= **270 Ohms**.
- le circuit du photo-transistor qui sera ici utilisé en mode saturé, autrement dit :
  - si pas d'objet dans la fente = lumière IR présente, alors la tension du collecteur vaudra 0V
  - si objet présent dans la fente = pas de lumière IR, alors la tension du collecteur vaudra 5V
  - pour obtenir ce résultat, on se contente d'utiliser une résistance de quelques milliers d'Ohms entre le collecteur et le +5V. En pratique, on utilisera **4,7KOhms** avec un LTH301-7.



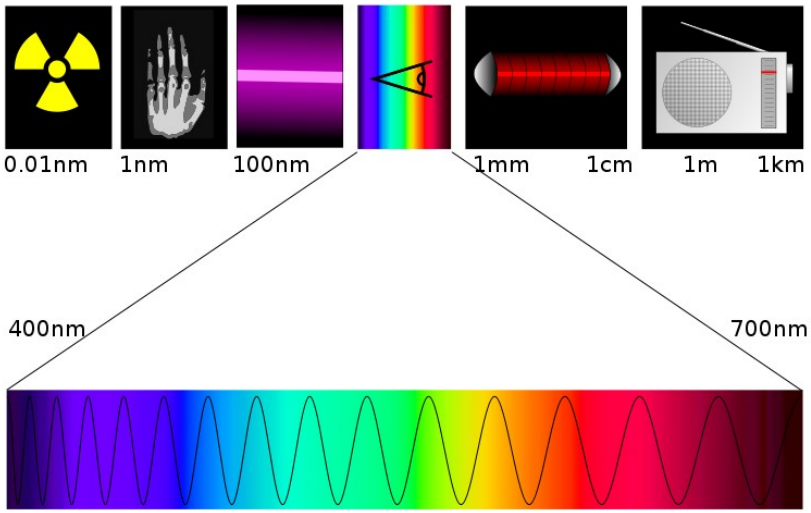
Opto- coupleur fourche
(Ex: LTH301-7)

Principe d'utilisation sur une plaque d'essai

L'opto-coupleur s'utilise simplement, à cheval sur le rail central :

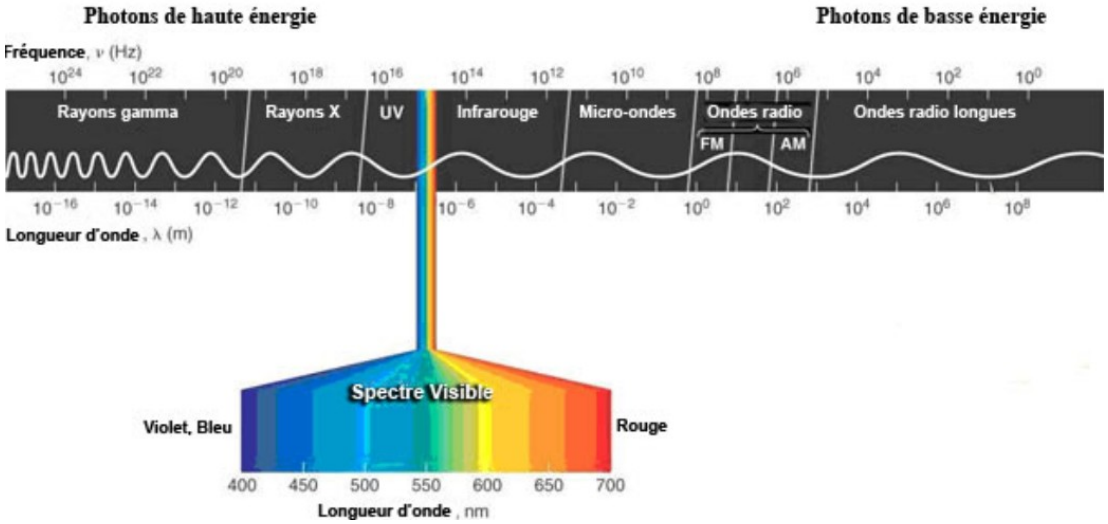


12. Pour info : le spectre des ondes électro-magnétiques et de la lumière visible



http://fr.wikipedia.org/wiki/Fichier:Spectre.svg

La lumière, tout comme les ondes radio ou les micro-ondes sont des ondes dites « électro-magnétiques »



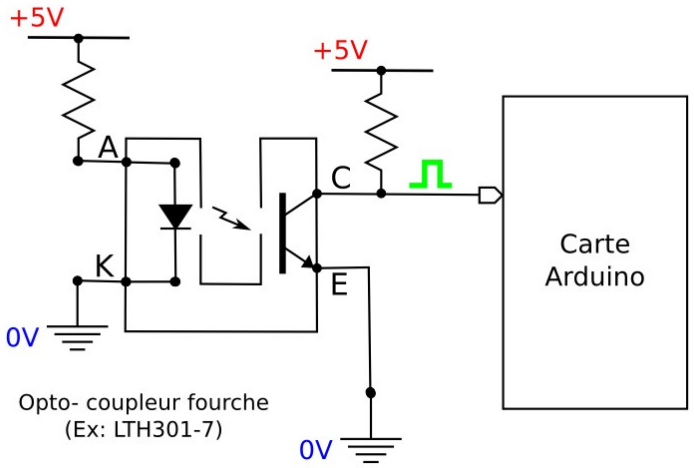
source : http://www.lampexpress.fr/images/ampoules-fiche-technique/spectre-lumiere.jpg

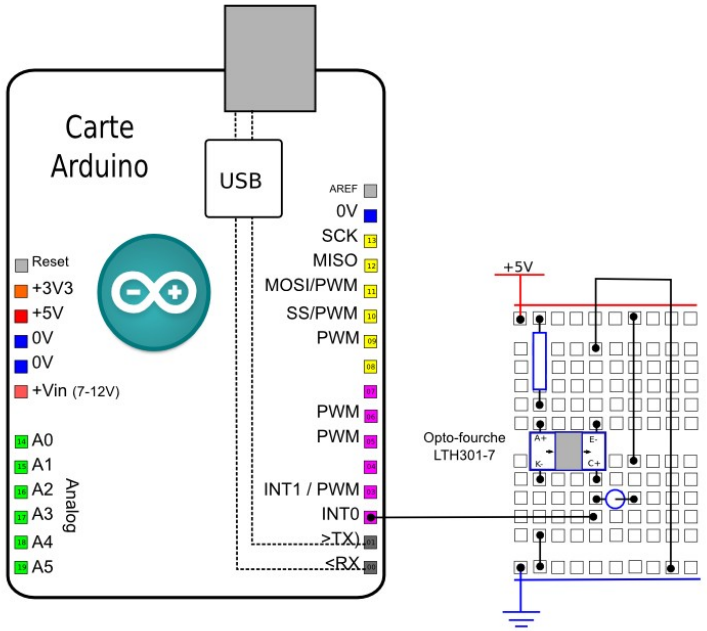
La lumière visible ne représente qu'une toute petite partie de l'ensemble des ondes électro-magnétiques.

La lumière infra-rouge, à laquelle est sensible le photo-transistor, est une lumière invisible à l'oeil nu, de même que la lumière ultra-violette.

13. Rappel : Utiliser un opto-coupleur en fourche en tant que capteur numérique : le montage

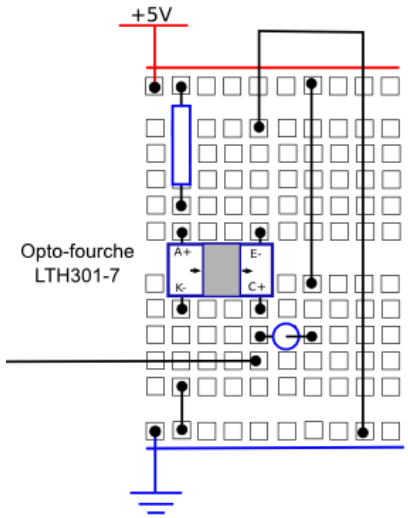
On reprend ici simplement le montage type de l'opto-coupleur vu précédemment. On connecte l'émetteur sur une broche analogique de la carte Arduino :





Comprendre comment çà marche

- Lorsqu'un objet est présent dans la fente, aucune lumière n'est détectée par le photo-transistor et donc aucune intensité ne circule dans le collecteur. La tension de la résistance en série vaut donc U = R x I = 0 V
- Lorsqu'aucun objet n'est présent dans la fente, la lumière est détectée par le photo-transistor et donc le courant circule dans le collecteur. La tension de la résistance en série vaut donc U = R x I ~ 5 V



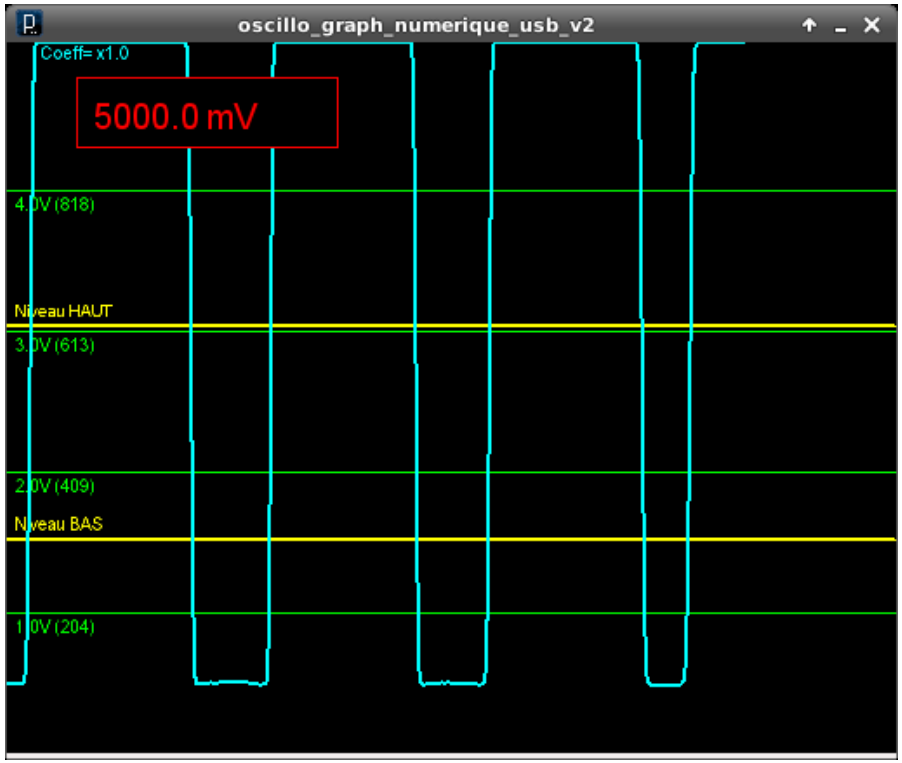
Truc de repérage :

Pour la LED, la broche courte est la cathode et la longue l'anode,
pour le photo-transistor, la broche courte est l'émetteur et la longue le collecteur.

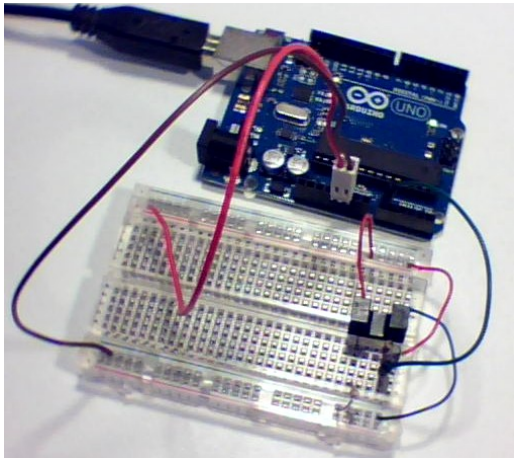
Truc pratique : pour vérifier que la LED s'allume bien, enlever l'opto-coupleur et remplacez-le par une LED normale. Si elle s'allume, tout est bien connecté. Ensuite, remettre l'opto-coupleur.

14. Pour info : Visualisation de la sortie de l'opto-coupleur.

Pour info, voici la visualisation dans une interface Processing de la sortie collecteur de l'opto-coupleur du montage précédent :



A chaque passage à 5V = présence d'un objet dans la fente !



15. Comptage d'événements à l'aide d'une interruption externe : le programme

Ce qu'on va faire ici...

- Dans ce programme, nous allons réaliser la même chose qu'un code présenté dans le tuto dédié aux capteurs ON/OFF numériques, mais ici en utilisant une interruption.
- Nous allons compter le nombre de passages d'un objet dans la fente d'un opto-coupleur. Les situations où l'utilisation d'une interruption est essentielle sont nombreuses, notamment :
  - si la fréquence de passage d'un objet dans la fente est élevée (l'interruption permet de ne rater aucun passage)
  - pour libérer le temps utile du microcontrôleur si le programme est conséquent et utilise d'autres fonctionnalités (communication série entre autre)

Entête déclarative

Variables utiles

- On déclare :
  - une constante de broche désignant la broche utilisée pour déclencher l'interruption, soit broche 2 (interruption n°0) ou 3 (interruption n°1),
  - une variable volatile utilisée pour le comptage d'événements,

```
//--- entête déclarative = déclarer ici variables et constantes globales

const int OPTO=2; // broche de l'optocoupleur

volatile int compt=0; // variable de comptage volatile
```

Fonction setup()

Initialisation série

- On initialise le port série à 115200 bauds, à l'aide de l'instruction begin()

Configuration broche utilisée

- On configure en entrée la broche utilisée pour l'interruption

Configuration interruption utilisée

- On configure l'interruption externe n°0, à l'aide de l'instruction attachInterrupt() en précisant :
  - le numéro d'interruption (0 = broche 2 ou 1 = broche 3)
  - le nom (sans les ( )) de la fonction à appeler, ici la fonction comptage(),
  - l'évènement déclencheur, ici le front montant (mot-clé RISING)

Code initial

- On affiche également un simple message d'accueil

```
//--- la fonction setup() : exécutée au début et 1 seule fois
void setup() {

  Serial.begin(115200); // initialise communication série

  pinMode(OPTO, INPUT); // broche en entrée

  attachInterrupt(0, comptage, RISING); // Attache l'interruption 0 (broche 2) à la fonction
  // RISING : détection sur front montant

  Serial.println("Arduino OK !") ; // message initial

} // fin de la fonction setup()
```

Fonction loop() ()

- Très simple ici : la fonction reste vide. On pourra y mettre le code à exécuter si nécessaire. Mais dans notre cas, tout va se passer dans la fonction comptage() appelée lors de la survenue de l'interruption :

```
//--- la fonction loop() : exécutée ensuite en boucle sans fin
void loop() {

  // mettre ici le code à exécuter


} // fin de la fonction loop()
```

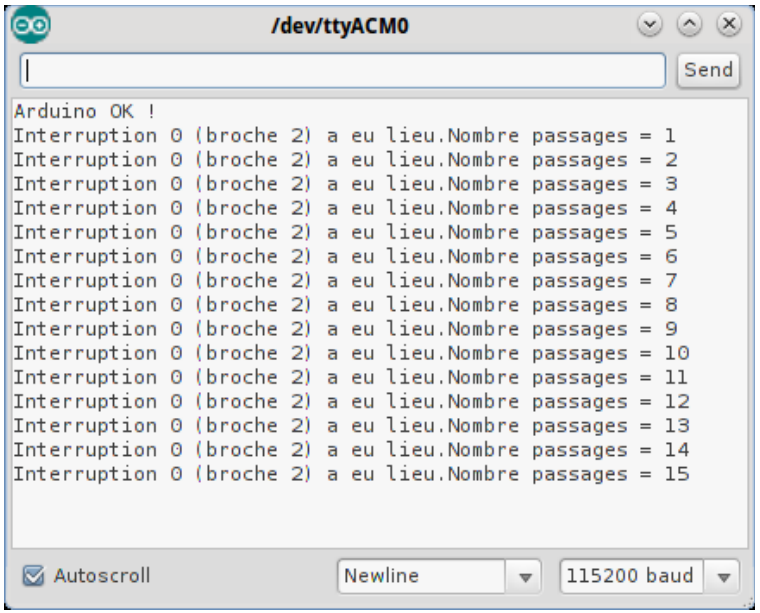
Fonction de gestion de l'interruption externe

- La fonction qui va être appelée lors de la survenue de l'interruption s'appelle ici comptage() :
  - cette fonction ne renvoie rien : elle est donc de type void
  - elle ne reçoit aucun paramètre : les parenthèses sont laissées vides
  - à ce niveau, on va tout simplement incrémenter la variable de comptage et afficher simplement un message signalant la survenue de l'interruption ainsi que la valeur de la variable de comptage.

```
// fonction appelée lors interruption n°0 (broche 2)
void comptage() {

  compt=compt+1; // incrémente variable comptage

  Serial.print("Interruption 0 (broche 2) a eu lieu.");
  Serial.print("Nombre passages = ");
  Serial.println(compt);

} // fin gestion interruption
```

Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 »,
- puis passer un objet suffisamment opaque dans la fente de l'opto-coupleur : chaque nouveau passage déclenche l'interruption 1 seule fois,
- ce qui donne ... :



Tout fonctionne ? Alors bravo, vous savez utiliser une interruption externe !

16. Remarque : utilisation des instructions utilisant des interruptions au sein de la routine d'interruption.

Intro

- Comme on l'a dit dans l'introduction, certaines instructions Arduino utilisent une interruption pour leur fonctionnement interne, notamment :
  - delay(), millis()
  - tone(), analogWrite()
  - mais aussi les librairies Serial, Servo, Wire, etc...
- En pratique, il s'avère que l'utilisation de ces fonctions au sein de la routine des interruptions est susceptible de poser des problèmes et de provoquer des comportements inattendus !

Mon conseil d'ami :

Eviter en pratique d'utiliser des fonctions utilisant des interruptions au sein de la routine de gestion des interruptions (= la fonction appelée par l'interruption)

Euh, mais c'est pourtant ce qu'on vient de faire !?

- « C'est bien, il y en a au moins un qui suit !! » Blague à part, si vous êtes attentif, vous voyez que dans le code précédent, on utilise les fonctions **Serial**.println() et **Serial**.print() au sein de la fonction appelée lorsque l'interruption survient... exactement le contraire de ce que je viens de vous conseiller !
- En fait, si on utilise une instruction Arduino utilisant une interruption au sein de la fonction de gestion de l'interruption, je n'ai pas dit que çà ne marchait pas... mais que çà pouvait engendrer des comportements inattendus, typiquement un blocage du programme !
- C'est exactement ce qui se passe avec le code précédent, si vous faîtes passer un objet dans la fente de l'opto-coupleur 20 ou 50 fois... : à un moment, çà va bloquer.

On fait comment alors ?

- La solution consiste à mémoriser l'état d'une variable qui sera modifiée lors du passage dans l'interruption et à tester au niveau de la fonction loop() si elle a été modifiée : si c'est le cas, c'est que l'interruption a eu lieu et on exécute alors les instructions voulues.

Exemple

- Voici le programme précédent modifié de façon à ce que les instructions de la librairie **Serial** soient exécutées au sein de la fonction loop() :

```
//--- entete déclarative = déclarer ici variables et constantes globales

const int OPTO=2; // broche de l'optocoupleur

volatile int compt=0; // variable de comptage volatile
int compt0=0; // variable pour mémoriser dernière valeur

//--- la fonction setup() : exécutée au début et 1 seule fois
void setup() {

  Serial.begin(115200); // initialise communication série

  pinMode(OPTO, INPUT); // broche en entrée

  attachInterrupt(0, comptage, RISING); // Attache l'interruption 0 (broche 2) à
la fonction
  // RISING : détection sur front montant

  Serial.println("Arduino OK !") ; // message initial

} // fin de la fonction setup()


//--- la fonction loop() : exécutée ensuite en boucle sans fin
void loop() {

  // mettre ici le code à exécuter

  if (compt0!=compt) { // test si la variabla a changé
    Serial.print("Interruption 0 (broche 2) a eu lieu.");
    Serial.print("Nombre passages = ");
    Serial.println(compt);
    compt0=compt; // mémorise nouvelle valeur
  } // fin if

} // fin de la fonction loop()


// fonction appelée lors interruption n°0 (broche 2)
void comptage() {

  compt=compt+1; // incrémente variable comptage

  //Serial.print("Interruption 0 (broche 2) a eu lieu.");
  //Serial.print("Nombre passages = ");
  //Serial.println(compt);

} // fin gestion interruption
```

17. Technique : Remarque sur les capteurs ON/OFF « mécaniques » et les interruptions externes

Intro

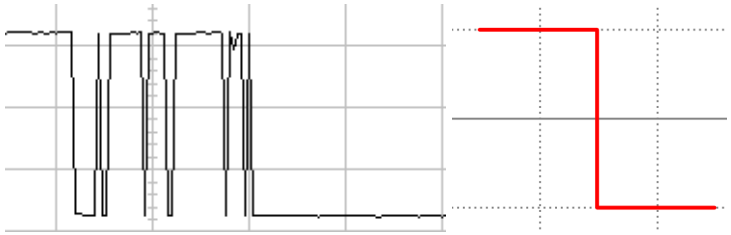
- Comme vous l'avez constaté, ici, nous avons utilisé un opto-coupleur en tant que capteur « numérique » non-mécanique. Mais dans bon nombre de situations, on pourrait avoir besoin de déclencher une interruption sur à l'aide d'un capteur mécanique.
- Par exemple, sur un robot, un micro-rupteur sera par exemple appuyé en cas de choc et déclenchera l'interruption.

Rappel : Comparatif bouton poussoir (« mécanique ») et capteur numérique ON/OFF (« non-mécanique »)

Un bouton poussoir est un capteur ON/OFF de type « mécanique », ce qui entraîne un certain nombre de problématiques spécifiques qui ont été abordées dans l'atelier consacré aux boutons poussoirs :

- nécessité d'un « rappel au plus » (ou « au moins ») de la broche numérique laissée non connectée,
- nécessité d'une pause « anti-rebond » lors de la lecture de l'état du bouton poussoir.

Un capteur numérique ON/OFF ne présente pas ces problèmes et a une transition HAUT/BAS nette et franche :



A gauche : bouton poussoir, à droite : capteur numérique

Le problème

- Le phénomène de rebond est particulièrement pernicieux dans le cas de l'utilisation d'une interruption :
  - en effet, le déclenchement de l'interruption est particulièrement sensible,
  - et on comprend très bien dès lors, au vu du schéma ci-dessus, que l'interruption sera déclenchée plusieurs fois lors d'un seul appui sur le bouton poussoir,
- La solution passera par l'utilisation d'une pause anti-rebond au sein de la fonction de gestion de l'interruption (et donc perte de réactivité).

Exemple

- Voici le code à utiliser si l'on souhaite déclencher l'interruption lors de l'appui sur un bouton poussoir sur la broche 2 : remarquer la pause anti-rebond dans la fonction d'interruption.

```
//--- entete déclarative = déclarer ici variables et constantes globales

const int BP=2; // broche de l'optocoupleur

volatile int compt=0; // variable de comptage volatile
int compt0=0; // variable pour mémoriser dernière valeur

//--- la fonction setup() : exécutée au début et 1 seule fois
void setup() {

  Serial.begin(115200); // initialise communication série

  pinMode(BP, INPUT); // broche en entrée
  digitalWrite(BP,HIGH); // active le rappel au plus

  attachInterrupt(0, comptage, FALLING); // Attache l'interruption 0 (broche 2) à la fonction
  // RISING : détection sur front montant
  // FALLING : détection sur front descendant

  Serial.println("Arduino OK !") ; // message initial

} // fin de la fonction setup()


//--- la fonction loop() : exécutée ensuite en boucle sans fin
void loop() {

  // mettre ici le code à exécuter

  if (compt0!=compt) { // test si la variabla a changé
    Serial.print("Interruption 0 (broche 2) a eu lieu.");
    Serial.print("Nombre passages = ");
    Serial.println(compt);
    compt0=compt; // mémorise nouvelle valeur
  } // fin if

} // fin de la fonction loop()

// fonction appelée lors interruption n°0 (broche 2)
void comptage() {

  //noInterrupts(); // +/- désactive interruption

  compt=compt+1; // incrémente variable comptage

  //Serial.print("Interruption 0 (broche 2) a eu lieu.");
  //Serial.print("Nombre passages = ");
  //Serial.println(compt);

  delay(500); // pause anti-rebond pour éviter double prise en compte...

  // interrupts(); // +/- réactive interruption

} // fin gestion interruption
```

A retenir : les capteurs ON/OFF « mécaniques » sont capricieux avec les interruptions externes : éviter de les utiliser si possible !

18. Rappel : Stratégie de programmation : comptage de fréquence

Notion de fréquence

Avant de passer à la suite, prenons le temps de réfléchir à la notion de fréquence, plus exactement au comptage de la fréquence de survenue d'un évènement.

Par exemple, imaginons que l'on veuille compter le nombre de tours par seconde d'un axe en rotation. Pour réaliser cette mesure, on va avoir besoin de 2 choses :

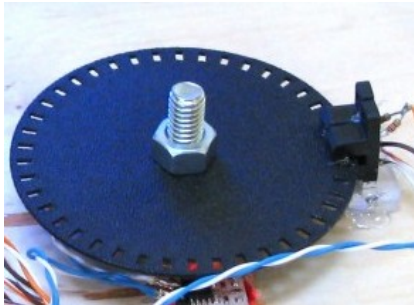
- d'une base temps fixe, autrement dit une durée précise pendant laquelle on va compter la survenue d'un événement
- d'un « compteur » qui va permettre de comptabiliser tous les évènements qui sont survenus pendant la durée du comptage.

La fréquence de survenue de l'évènement vaudra :

fréquence = nombres d'évènements / durée de comptage

Comptage des événements

Imaginons que l'on veuille compter le nombre fois où un événement survient dans un certain délai. Par exemple, si l'on veut compter la vitesse de rotation d'un moteur ou d'un axe, on pourra compter le nombre fois où l'objet en rotation est détecté dans un opto-coupleur. A ce stade, on sait faire comme on l'a vu dans un programme précédent : il suffit d'incrémenter une variable.



Exemple de comptage en rotation par opto-coupleur

Fixer un délai de comptage

Pour fixer un délai de comptage, on va se baser sur l'instruction Arduino millis() qui renvoie à tout moment le nombre de millisecondes écoulées depuis la mise sous tension de l'Arduino.

Pour fixer un délai de comptage fixe, on va utiliser 2 variables :

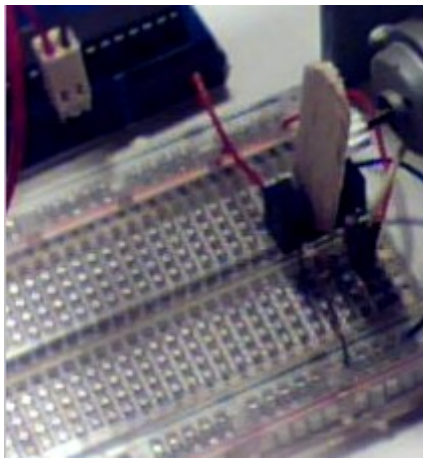
- une pour mémoriser la dernière valeur de millis() prise en compte
- une pour fixer le délai de comptage et permettre d'évaluer si le délai voulu s'est écoulé.

La stratégie de programmation va consister à :

- mémoriser la valeur de millis()
- tester à chaque passage de loop() si le délai de comptage est écoulé
- si oui :
  - exécuter les instructions voulues
  - remettre à zéro les variables de comptages
  - mémoriser la nouvelle valeur de millis()
- et ainsi de suite...

19. Mesurer la vitesse de rotation d'un axe : la mécanique

- Côté électronique, on va utiliser le même montage que vu précédemment (optocoupleur seul connecté sur la broche 2).
- Côté mécanique, on va ici utiliser un simple axe sur lequel on va fixer une languette de bois. On positionne l'axe de façon à ce que la languette de bois passe dans l'opto-coupleur à chaque rotation, ce qui va permettre de compter la vitesse de rotation de l'axe.



L'axe est fixé de manière à ce qu'une languette de bois fixée sur l'axe
passe dans la fente de l'opto-coupleur...
Chaque rotation entraînera 2 transitions « HAUT-BAS » : le nombre de tours sera nombre transition / 2

Applications possibles

Le comptage du nombre de tours à l'aide d'une interruption externe pourra être utilisé pour toutes sortes de comptage de vitesse en rotation, notamment pour un anémomètre, un compte tour de vélo, etc...

20. Comptage de la fréquence de rotation d'un axe à l'aide d'une interruption externe : le programme

Ce qu'on va faire ici...

- Dans ce programme, nous allons réaliser la même chose qu'un code présenté dans le tuto dédié aux capteurs ON/OFF numériques, mais ici en utilisant une interruption.
- Nous allons appliquer ce que nous venons de voir en comptant le nombre de tours par 10 secondes d'un axe tournant à vitesse variable. Cette application est plutôt applicable à des mesures de vitesse peu élevées, de moins de 25 tours/seconde.

Entête déclarative

Variables utiles

- On déclare :
  - une constante de broche désignant la broche utilisée pour déclencher l'interruption, soit broche 2 (interruption n°0) ou 3 (interruption n°1),
  - une variable volatile utilisée pour le comptage d'événements, et une variable pour mémoriser la dernière valeur,
  - une variable pour mémoriser le dernier millis() pris en compte et une variable fixant le délai à prendre en compte en millisecondes.

```
//--- entete déclarative = déclarer ici variables et constantes globales

const int OPTO=2; // broche de l'optocoupleur

volatile int compt=0; // variable de comptage volatile
int compt0=0; // variable pour mémoriser dernière valeur

long millis0=0; // variable mémorisation valeur millis()
int delai=10000; // délai de comptage en millisecondes
```

Fonction setup()

Initialisation série

- On initialise le port série à 115200 bauds, à l'aide de l'instruction begin()

Configuration broche utilisée

- On configure en entrée la broche utilisée pour l'interruption

Configuration interruption utilisée

- On configure l'interruption externe n°0, à l'aide de l'instruction attachInterrupt() en précisant :
  - le numéro d'interruption (0 = broche 2 ou 1 = broche 3)
  - le nom (sans les ( )) de la fonction à appeler, ici la fonction comptage(),
  - l'évènement déclencheur, ici le front montant (mot-clé RISING) ou descendant (mot-clé FALLING)

Code initial

- On mémorise la valeur initiale de millis()
- On affiche également un simple message d'accueil

```
//--- la fonction setup() : exécutée au début et 1 seule fois
void setup() {

  Serial.begin(115200); // initialise communication série

  pinMode(OPTO, INPUT); // broche en entrée
  //digitalWrite(OPTO, HIGH); // rappel au plus

  attachInterrupt(0, comptage,FALLING); // Attache l'interruption 0 (broche 2) à la fonction
  // RISING : détection sur front montant
  // FALLING : détection sur front descendant

  millis0=millis(); // initialise la valeur de millis

  Serial.println("Arduino OK !") ; // message initial

} // fin de la fonction setup()
```

Fonction loop() ()

- A chaque passage on teste si le délai voulu est écoulé depuis la dernière mémorisation de millis(). Si c'est le cas :
  - on remet à jour la variable volatile de comptage et la variable de mémorisation de millis()
  - on affiche le nombre d'impulsion, de tours par seconde et de tour par minute,
- ce qui donne :

```
//--- la fonction loop() : exécutée ensuite en boucle sans fin
void loop() {

  // mettre ici le code à exécuter

  if (millis()-millis0>delai) { // si le delai s'est écoulé
    compt0=compt; // mémorise comptage actuel
    millis0=millis() ; // réinitialise millis0
    compt=0; // réinitialise comptage

    Serial.print("Delai ecoule.Comptage=");
    Serial.print(compt0);
    Serial.print(" soit ");
    Serial.print(compt0/2);
    Serial.print(" tours en 10 secondes");
    Serial.print(" soit ");
    Serial.print(compt0*3);
    Serial.println(" tours par minute");

  } // fin si delai écoulé


} // fin de la fonction loop()
```

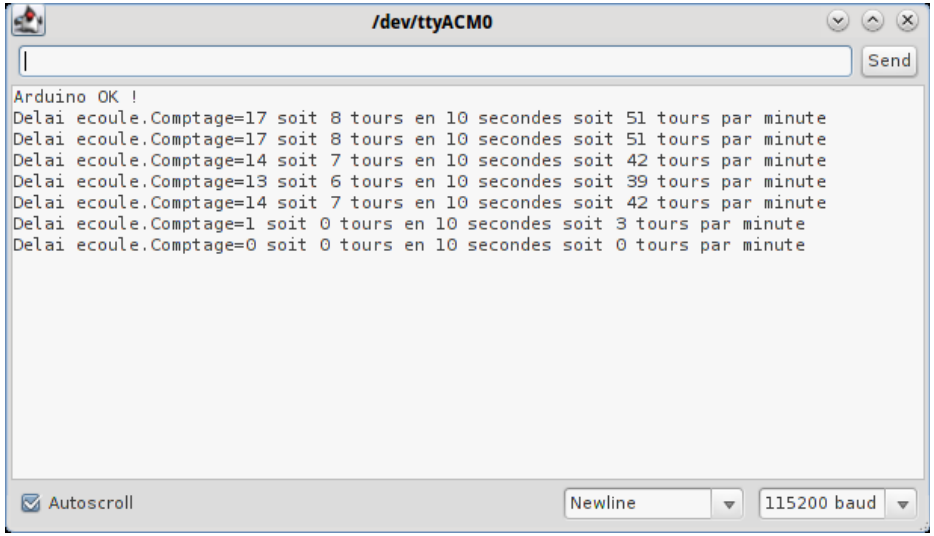
Fonction de gestion de l'interruption externe

- La fonction qui va être appelée lors de la survenue de l'interruption s'appelle ici comptage() :
  - cette fonction ne renvoie rien : elle est donc de type void
  - elle ne reçoit aucun paramètre : les parenthèses sont laissées vides
  - à ce niveau, on va tout simplement incrémenter la variable de comptage.

```
// fonction appelée lors interruption n°0 (broche 2)
void comptage() {

  compt=compt+1; // incrémente variable comptage

} // fin gestion interruption
```

Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 »,
- puis passer faire tourner l'axe avec languette de bois passant dans la fente de l'opto-coupleur : chaque nouveau passage déclenche l'interruption 1 seule fois, et donc 2 fois par tour :
- ce qui donne ... :



NOTE :

Sur le même principe, il est possible de mesurer des vitesses rapides, genre moteur en rotation. En fait, il s'avère que **l'interruption externe est très capricieuse** et sensible au champ électromagnétique, probablement en raison d'un défaut de découplage (utiliser un bon condensateur)...

Tout çà pour dire qu'il faut savoir que les interruptions externes sont plutôt capricieuses et se déclenchent vite de façon intempestive !!!

Et que çà peut vite devenir prise de tête !

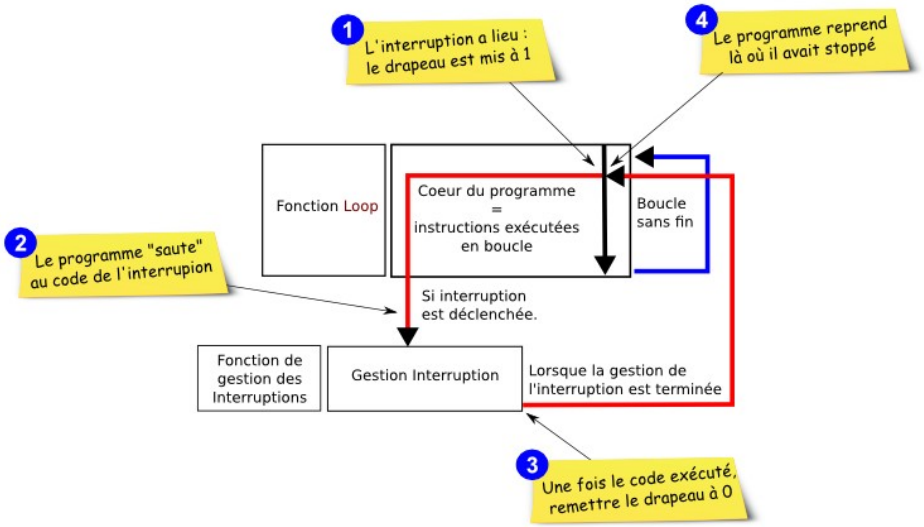
21. Technique : utiliser une interruption à intervalle régulier

Intro

- Jusqu'à présent, nous avons vu l'interruption dite « externe » intéressante pour prendre en compte des événements issus de capteurs et nécessitant une grande réactivité.
- Un autre type d'interruption est potentiellement très utile en pratique : l'interruption temporelle, autrement dit une interruption qui survient à intervalle régulier.
- Une telle interruption permettra de libérer le processeur pendant le délai entre 2 événements, au lieu de bloquer le programme avec une pause de type delay() qui bloque le programme.

Rappel : déroulement d'une interruption

- En interne, lorsqu'une interruption survient, le microprocesseur :
  - met le drapeau à 1
  - saute à une zone spéciale de la mémoire programme correspondant à la routine d'interruption (parfois appelée aussi ISR pour Interruption Service Routine) :
    - à ce niveau, au sein du code, on peut tester les « flags » pour savoir quelle interruption à été déclenchée
    - le code voulu est exécuté
  - le drapeau doit être remis à 0
  - puis le microcontrôleur quitte la routine d'interruption et reprend l'exécution du programme là où elle s'était interrompue.



Les timers de l'Arduino

- Comme nous avons déjà eu l'occasion de le dire, l'Arduino dispose (version UNO) de 3 timers (ou horloges internes) :
  - le timer 0 utilisé par millis() et analogWrite() sur broche 5 et 6
  - le timer 1 utilisé par analogWrite() sur broche 9 et 10
  - le timer 2 utilisé par analogWrite() sur broche 3 et 11
- On voit ici que tous les timers de l'Arduino sont utilisés, mais il va néanmoins être possible de les utiliser, en perdant cependant la génération PWM sur certaines broches.

Générer une interruption temporelle avec une librairie

- A la différence des interruptions externes, les interruptions « temporelles » ne sont pas directement implémentées dans le langage Arduino. Pour le faire, on va devoir utiliser une librairie externe. Heureusement pour nous il en existe plusieurs, notamment :
- Librairie TimerOne
  - Site officiel : http://code.google.com/p/arduino-timerone/
  - Doc sur le playground ARduino : http://arduino.cc/playground/Code/Timer1
- Librairie MSTimer2 :
  - simple à utiliser – configurable en millisecondes
  - Site officiel : http://www.pjrc.com/teensy/td_libs_MsTimer2.html
  - Doc sur le playground Arduino : http://arduino.cc/playground/Main/MsTimer2
- Librairie FlexTimer2 :
  - simple aussi, configurable en unités de temps de son choix simplement
  - Site Officiel : https://github.com/wimleers/flexitimer2
  - Doc sur le playground Arduino : http://www.arduino.cc/playground/Main/FlexiTimer2

Remarque

Je signale ici quelques possibilités : il en existe sûrement d'autres...

22. Arduino : La librairie MsTimer2

La librairie MSTimer2

- La librairie MSTimer2 permet de générer très simplement une interruption à intervalle régulier en se basant sur le Timer 2 (on perd la génération PWM sur les broches 3 et 11)

Télécharger la librairie

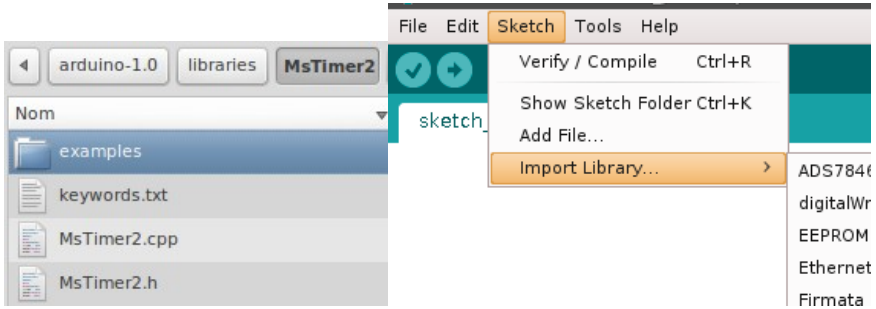
- Site officiel : http://www.pjrc.com/teensy/td_libs_MsTimer2.html

Documentation de la librairie

- Doc sur le playground Arduino : http://arduino.cc/playground/Main/MsTimer2

Installation

- Télécharger l'archive. au format zip ou autre. L'extraire
- Vérifier que le nom du répertoire de la librairie est strictement le même que le nom du fichier *.h ou *.cpp principal. Corriger au besoin. Ici le nom est **MsTimer2**
- Copier/coller le répertoire de la librairie dans le répertoire libraries de votre répertoire Arduino
- Relancer Arduino et vérifier que la librairie est présente dans le menu **Sketch > ImportLibrary.**



Inclusion

- On inclut la librairie dans un programme avec l'instruction #include (sans ; en fin de ligne +++) suivi du nom de la librairie :

```
#include <MsTimer2.h> // inclusion de la librairie Timer2
```

Le constructeur principal

- Le constructeur est implicite (= acessible directement = pas besoin de le déclarer, comme Serial) se nomme MsTimer2 :

```
MsTimer2
```

Fonctions de la librairie

- Les fonctions de la librairie sont au nombre de 3, très simple :
  - set(duree, fonction) : configure l'interruption où
    - duree est le délai entre 2 appels de l'interruption en ms
    - fonction est le nom de la fonction à appeler sans les ()
  - start() : démarre l'interruption
  - stop() : désactive l'interruption

- Les fonctions sont accessibles sous la forme C++ suivante :

MsTimer2::fonction()

- à la différence de la forme classique Arduino :

Classe.fonction()

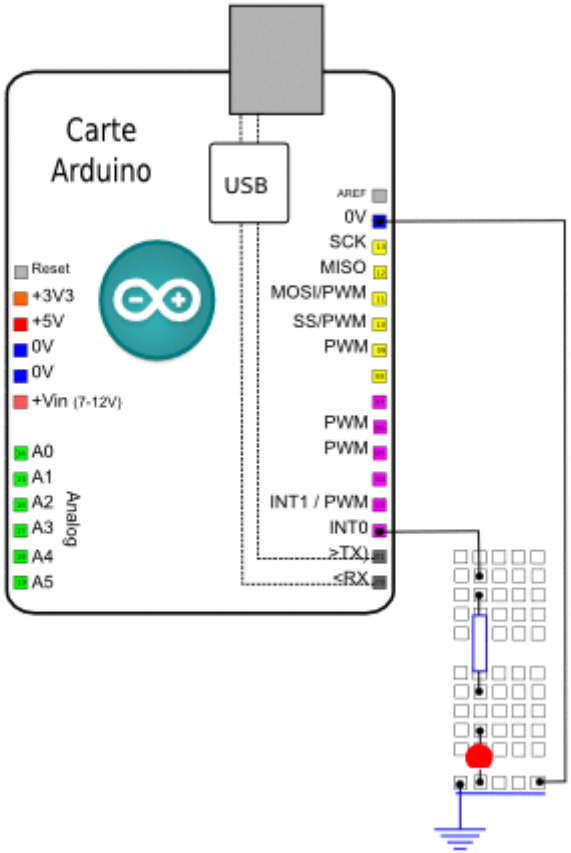
Note :

On retrouve ici les mêmes fonctions qu'un objet dit « Timer » qui existe dans plusieurs langages de haut niveau, notamment Python.

Encore une fois, le langage Arduino prépare le terrain pour l'apprentissage de langages plus élaborés.

23. Exemple d'utilisation : Faire clignoter une LED en utilisant une interruption temporelle : le montage

- On va reprendre ici le premier montage que vous avez dû réalisé lorsque vous appris Arduino : une simple LED en série avec sa résistance sur la broche 2 :



24. Faire clignoter une LED en utilisant une interruption temporelle : le programme

Ce qu'on va faire ici...

- Dans ce programme, nous allons ... faire clignoter une LED !! Non, non, ce n'est pas une blague... Bon, vous allez me dire que je vous fais régresser... retour à la case départ... tout çà pour çà ??? Je vous rassure, je ne me moque pas de vous... !
- Bon, je sais, c'est pas extraordinaire... mais çà va vous montrer comment faire clignoter une LED.... en libérant le micro-contrôleur qui ne sera pas obligé de passer son temps à exécuter la fonction delay() : on va utiliser ici une interruption « temporelle » ! Aller, c'est parti... rien de bien sorcier, vous allez voir !

Entête déclarative

Inclusion des librairies

- On commence par inclure la librairie MsTimer2 que vous avez dû installer précédemment (enfin, si vous avez fait ce que je vous ai dit... sinon, et bien faîtes-le !),

Variables utiles

- On déclare :
  - une constante de broche désignant la broche utilisée pour la LED

```
//--- inclusion des librairies

#include <MsTimer2.h> // inclusion de la librairie Timer2

//--- entete déclarative = déclarer ici variables et constantes globales

const int LED=2; //declaration constante de broche
```

Fonction setup()

Configuration broche utilisée

- On configure en entrée la broche utilisée pour l'interruption

Configuration interruption utilisée

- On configure l'interruption du Timer 2 :
  - tout d'abord on initialise l'interruption avec la fonction set() en fixant
    - le délai entre 2 appels de l'interruption en ms, ici 1000 ms
    - le nom de la fonction à appeler sans les () : ici interruptTimer2
  - puis on démarre l'interruption avec la fonction start() : ceci a pour effet de déclencher l'interruption toutes les 1000ms et donc d'appeler la fonction interruptTimer2 toutes les 1000ms.

```
//--- la fonction setup() : exécutée au début et 1 seule fois
void setup() {

  pinMode(LED, OUTPUT); //met la broche en sortie

  // initialisation interruption Timer 2
  MsTimer2::set(1000, interruptTimer2); // période 1000ms
  MsTimer2::start(); // active Timer 2


} // fin de la fonction setup()
```

Fonction loop() ()

- On laisse la fonction loop () vide : tout se passe dans la routine de gestion de l'interruption :

```
//--- la fonction loop() : exécutée ensuite en boucle sans fin
void loop() {

  // laissée vide


} // fin de la fonction loop()
```

Fonction de gestion de l'interruption externe

- La fonction qui va être appelée lors de la survenue de l'interruption s'appelle ici interruptTimer2() (on aurait pu donner tout autre nom...) :
  - cette fonction ne renvoie rien : elle est donc de type void
  - elle ne reçoit aucun paramètre : les parenthèses sont laissées vides
  - à ce niveau, on va :
    - déclarer une variable static, c'est à dire une variable dont la valeur sera mémorisée entre 2 appel de la fonction. Cette variable, de type booléen (= binaire = 0 ou 1 = HIGH ou LOW) sera initialisée à HIGH. Malgré les apparences, bien comprendre que cette variable est initialisée à HIGH SEULEMENT LORS DU PREMIER APPEL DE LA FONCTION. Lors des autres appels, la valeur courante sera utilisée.
    - ensuite, on met la broche dans l'état fixé par la variable static déclarée,
    - puis on inverse son état à l'aide d'une notation typique du C : le **!** devant la variable booléenne, ce qui la rend HIGH si elle est LOW, LOW si elle est HIGH, etc... vous avez compris ?
- Si on se résume :
  - au premier passage :
    - la variable static est déclarée et mise à HIGH
    - la broche est donc mise à HIGH : la LED s'allume
    - puis la variable est mise à LOW
  - au passage suivant :
    - la broche est mise dans l'état de la variable static qui a été mémorisée : la LED est donc allumée si était éteinte et inversement
    - puis la variable static est à nouveau inversée
  - etc... : au final, la LED clignote toutes les secondes !

```
// fonction appelée lors interruption Timer2
void interruptTimer2() { // debut de la fonction d'interruption Timer2

  static boolean etatLED=HIGH; // variable statique initialisée à HIGH

  digitalWrite(LED, etatLED);
  etatLED=!etatLED; // inverse la variable


} // fin InterruptTimer2()
```

Fonctionnement du programme

- Une fois la carte Arduino programmée, la LED clignote !

Remarque :

Derrière une apparence assez simple à première vue, ce petit code vous apprend au passage plusieurs choses importantes :
comment utiliser une interruption générée à intervalle régulier, permettant de libérer le temps « utile » pour faire autre chose,
comment mémoriser une variable entre 2 appels au sein d'une fonction à l'aide du qualificateur static,
comment inverser une variable booléenne (ou binaire) en la faisant précéder du sigle !

Sympa non ?
Notez tout çà sur vos tablettes, çà vous servira à l'occasion !

25. Les éléments du langage Arduino étudiés dans cet atelier

Structure

Variables et constantes

- static
- volatile

Fonctions

Interruptions Externes

- attachInterrupt(interruption, fonction, mode)
- detachInterrupt(interruption)

Interruptions

- interrupts()
- noInterrupts()

La documentation complète du langage Arduino en français est disponible ici :
http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.ReferenceMaxi

Pour aller plus loin avec les interruptions :
Si vous avez des besoins précis plus avancés utilisant les interruptions, vous pourrez utilement consulter la documentation de la librairie native interrupt.h ici :
http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html

26. A présent, vous devriez être capable :

- d'expliquer le concept d'interruption
- de comprendre comment fonctionne une interruption
- d'utiliser les interruptions externes
- d'utiliser les interruptions à intervalles réguliers

... afin d'être en mesure de créer des programmes plus efficients et optimisant les temps d'utilisation du microprocesseur de l'Arduino.

Table des matières

Bravo !

vous avez terminé cet atelier Arduino !

Prêt pour la suite ? Retrouvez de nombreux autres thèmes d'ateliers Arduino ici :

http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS