

Stratégies de programmation : temporisations avec Arduino



Ateliers Arduino

par X. HINAULT

www.mon-club-elec.fr



Tous droits réservés – 2012.

Ce document légèrement payant est soumis au droit d'auteur et est réservé à l'usage personnel.

Afin d'encourager la production de supports didactiques de qualité, ce document est légèrement payant.

La licence d'utilisation est attribuée pour un usage personnel uniquement, dans le cercle familial. Mise en ligne et diffusion non autorisées.

Si vous n'êtes pas le détenteur de la licence attribuée pour l'usage de ce document, soyez sympa, merci d'acheter votre exemplaire personnel ici : <https://monclubelec.dpdcart.com/>

Pour tout problème lié à l'utilisation de ce document, veuillez envoyer une copie ici : support@mon-club-elec.fr

Pour obtenir tout autres types de licence d'utilisation (enseignement, commercial, etc...), veuillez contacter l'auteur ici : support@mon-club-elec.fr

Vous avez constaté une erreur ? une coquille ? N'hésitez pas à nous le signaler à cette adresse : support@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 !

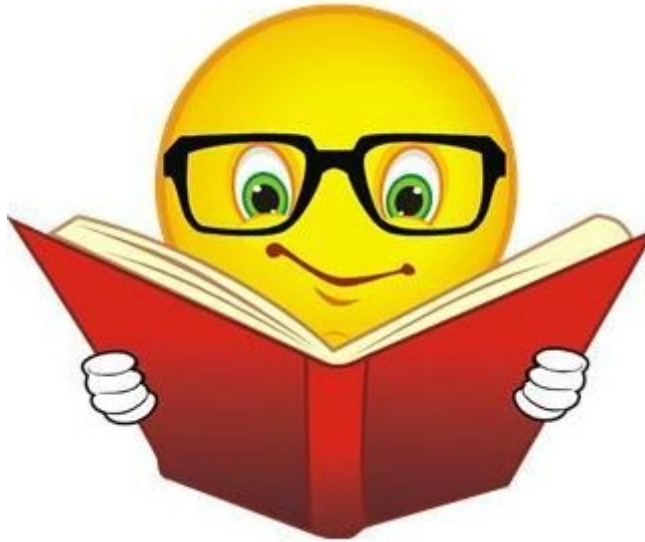
Lancer également le logiciel Arduino et connecter votre carte Arduino afin de pouvoir tester au fur et à mesure les codes d'exemples !

1. *Intro*

L'objectif ici est :

- d'apprendre les différentes façon de réaliser des « temporisations » avec Arduino.

... afin d'être en mesure de ...

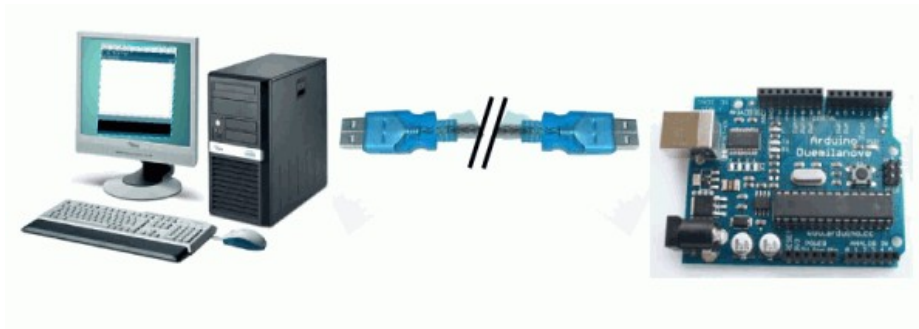


Prêt ? C'est parti !

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

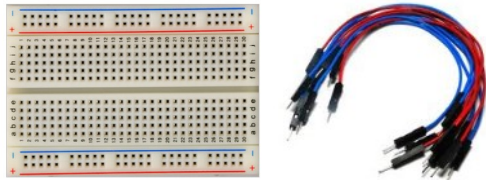


L'espace de développement Arduino associe :

- 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

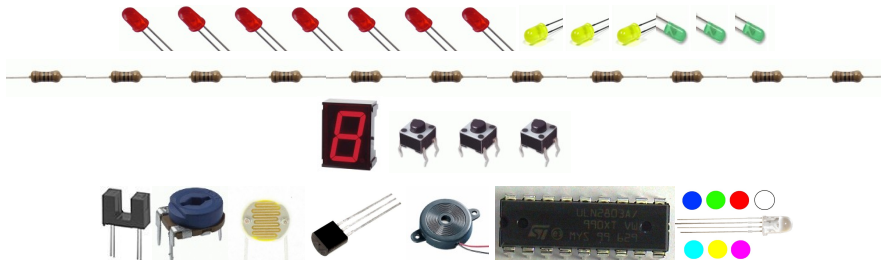


Pour réaliser des montages sans soudure, vous aurez besoin :

- 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 vous simplifier la vie, nous avons négocié ce kit pour vous !

Vous pouvez commander ce kit complet directement en 1 clic chez notre partenaire

<http://www.gotronic.fr/> avec le code express **701710**

GO TRONIC
ROBOTIQUE ET COMPOSANTS ÉLECTRONIQUES

Pour plus de détails, voir : http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS

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)

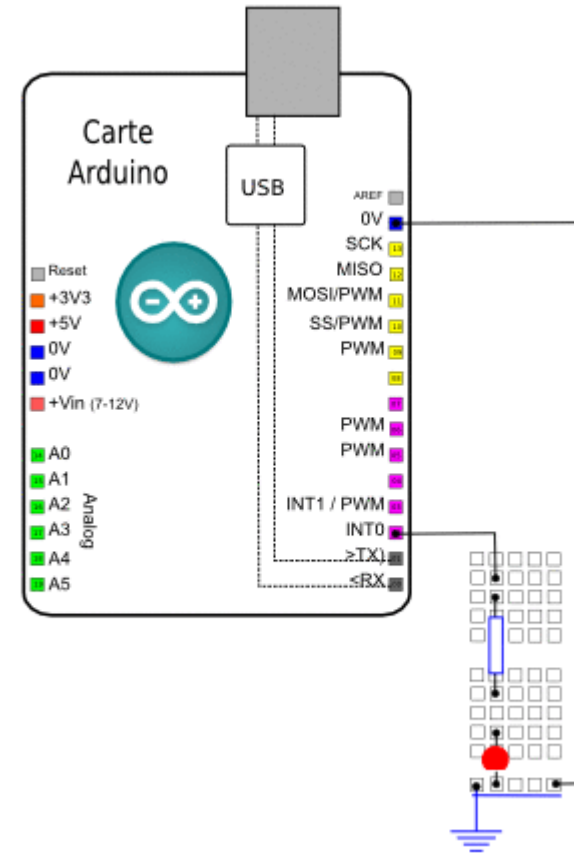
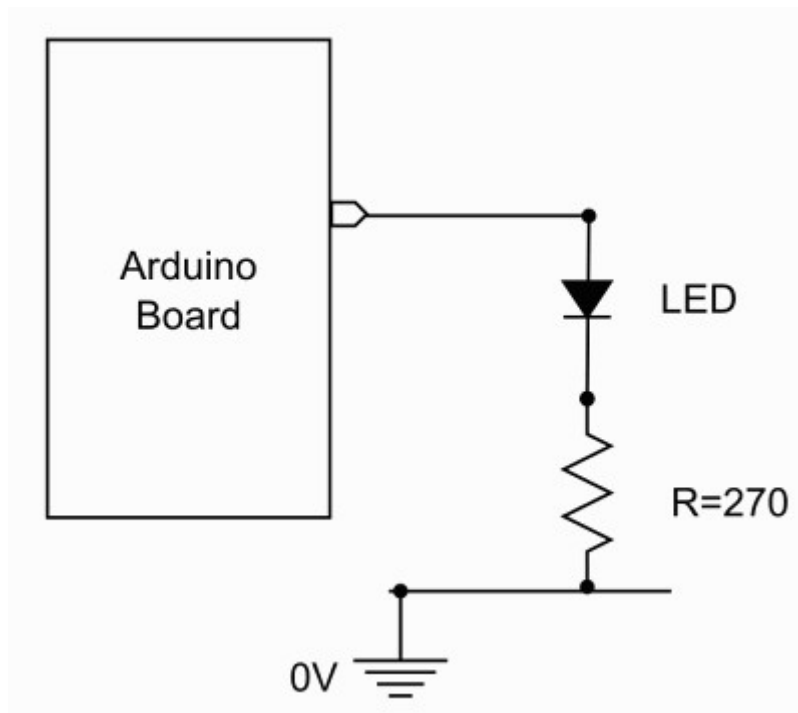
3. Faire clignoter une LED : Le montage.

Dans ce tuto, nous n'utiliserons pas de montages très compliqués : le but ici va surtout être de donner des principes de programmation utilisables dans différentes situations auxquelles on sera susceptible d'être confrontés.

Nous allons commencer par reprendre l'un des premiers montages que vous avez réalisé : connecter une LED sur une broche de la carte Arduino.

Pour faire clignoter une LED, on va connecter une LED en série avec une résistance sur une broche E/S de la carte Arduino en sortie. Le principe sera le suivant :

- lorsque la broche sera au niveau HAUT, la LED sera allumée,
- lorsque la broche sera au niveau BAS, la LED sera éteinte.



Comme vu précédemment, si on désire une intensité de 13mA dans la LED, on utilisera, d'après la loi d'ohm, une résistance de $R = U/I = 3,5V/0,013A = 270 \text{ Ohms}$.

4. Intervalle régulier : Faire clignoter une LED avec une pause : le programme.

Ce qu'on va faire ici...

- Ici, nous allons reprendre pour commencer un programme déjà vu : celui qui permet de faire clignoter une LED.
- Typiquement, pour ce faire :
- on allume la LED,
- on réalise une pause avec l'instruction `delay()`
- puis on éteint la LED et on réalise une nouvelle pause avec `delay()`, etc...

Entête déclarative

On déclare :

- une constante de type `int` pour désigner la broche 2, appelée LED
- une variable de type `int` pour fixer la vitesse, appelée vitesse

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

const int LED=2; // constante désignant la broche de la LED
int vitesse=250; // variable fixant la durée de la pause en ms
```

Fonction `setup()`

A ce niveau, on va :

- initialiser la broche utilisée en sortie (avec la constante LED) avec l'instruction `pinMode()`

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

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

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

Fonction `loop()`

A ce niveau on va :

- Allumer la LED = mettre la broche au niveau HAUT (5V) avec l'instruction `digitalWrite()`
- Attendre le temps voulu (= la valeur de la variable vitesse en millisecondes)
- Eteindre la LED = mettre la broche au niveau BAS (0V)
- Attendre le temps voulu (= la valeur de la variable vitesse en millisecondes) avec l'instruction `delay()`
- le code de la fonction loop se répète sans fin...

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

    digitalWrite(LED,HIGH); // allume la LED
    delay(vitesse); // pause de n millisecondes
    digitalWrite(LED,LOW); // éteint la LED
    delay(vitesse); // pause de n millisecondes

} // fin de la fonction loop()

// NB : les lignes précédées de // sont des commentaires
```

Fonctionnement du programme

- Une fois la carte Arduino programmée la LED clignote.

Première façon de réaliser une temporisation : utiliser les fonctions `delay(duree)` ou `delayMicroseconds(duree)`

(NB : L'instruction `delayMicroseconds()` ne bloque pas les interruptions)

5. Intervalle régulier : Exécuter une action à intervalle régulier... sans bloquer le programme.

Ce qu'on va faire ici...

- Rien de très compliqué dans le programme précédent.... mais un très très gros défaut qui se manifestera rapidement dès que les programmes vont s'étoffer : les pauses utilisées bloquent le programme qui ne fait rien d'autre pendant ce temps là... ce qui est peu efficace en pratique !
- A présent, je vous présente une solution simple et facile à utiliser pour contourner ce problème : utiliser la fonction `millis()` qui renvoie le nombre de millisecondes écoulées depuis le lancement du programme. Le truc va consister :
 - d'une part à mémoriser la valeur de `millis()` lors de la dernière prise en compte,
 - et d'autre part à utiliser une variable de délai pour permettre de tester si la durée voulue s'est écoulée ou non depuis la dernière prise en compte.
- La technique présentée ici est assez commune et permet facilement de réaliser des comptages de durée sans bloquer un programme.

Entête déclarative

On déclare :

- une constante de type `int` pour désigner la broche 2, appelée LED
- une variable de type `long` pour mémoriser la dernière valeur de `millis()` : on utilise un type long car les valeurs vont rapidement dépasser 32768, valeur maximale pour une variable `int`
- une variable de type `int` pour fixer le délai à mesurer
- une variable de type `boolean` pour mémoriser l'état de la LED : noter que l'on peut indifféremment utiliser `HIGH`, `true`, 1 ou `LOW`, `false`, 0.

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

const int LED=2; // constante désignant la broche de la LED

long millis0=0; // variable pour mémoriser dernier millis() pris en
compte
int delai=1000; // intervalle à utiliser en millisecondes

boolean etatLED=HIGH; // LED allumée au départ
// remarque HIGH = true = 1 et LOW = false = 0
```

Fonction `setup()`

A ce niveau, on va :

- initialiser la broche utilisée en sortie (avec la constante LED) avec l'instruction `pinMode()`
- on initialise la LED
- on mémorise la valeur courante de `millis()`

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

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

    digitalWrite(LED,etatLED); // met la LED dans l'état voulu
    millis0=millis(); // mémorise millis()

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

Fonction `loop()`

- A chaque passage, on va tester si le délai est écoulé en vérifiant simplement si la différence entre la valeur courante de `millis()` et la dernière valeur prise en compte est supérieure ou égale au délai :
 - si c'est le cas, l'état de la LED est inversé en conséquence
 - la nouvelle valeur de `millis()` est mémorisée
- Cette condition ne sera au final exécutée que lorsque cela sera nécessaire et le reste du code sera exécuté sans être bloqué le reste du temps.

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

    if (millis()-millis0>=delai) { // si le délai est écoulé

        millis0=millis(); // mémorise millis()
        etatLED=!etatLED; // inverse l'état de la variable
        digitalWrite(LED,etatLED);

    } // fin si délai écoulé

    // mettre les autres instructions ici
    // elles seront exécutées meme si le délai n'est pas écoulé

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

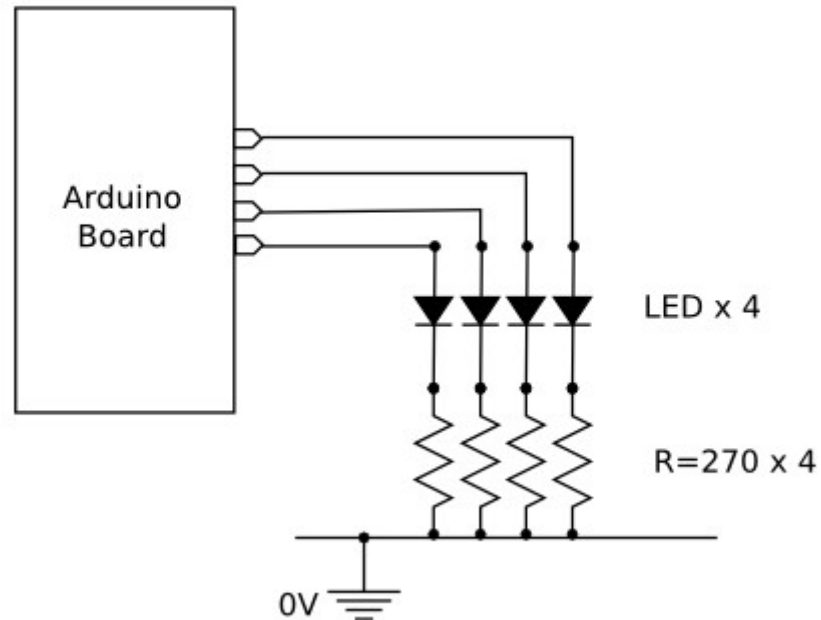
Fonctionnement du programme

- Une fois la carte Arduino programmée la LED clignote... la différence ici étant dans la façon de le faire au niveau du code.

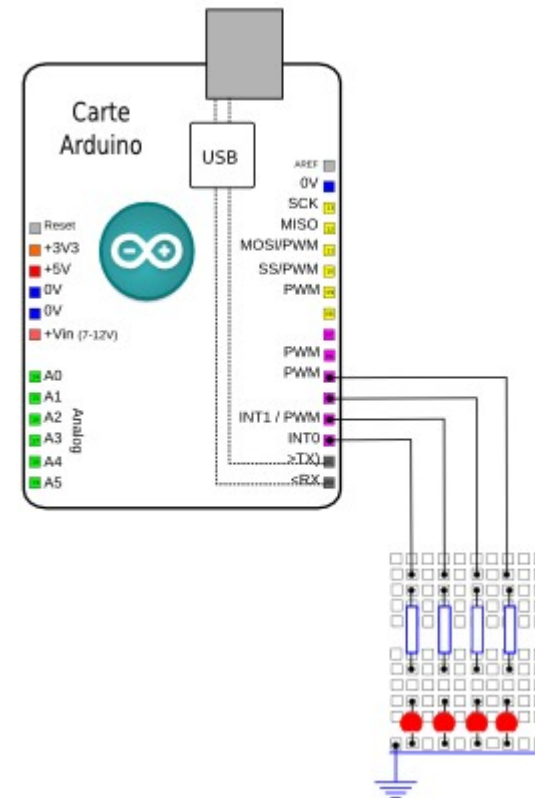
6. Intervalle régulier : Exécuter plusieurs actions asynchrones à intervalle régulier : le montage

Nous allons ici utiliser 4 LEDs représentant chacune un dispositif à contrôler. On va connecter 4 LEDs chacune en série avec une résistance sur 4 broche E/S de la carte Arduino configurées en sortie. Le principe sera le suivant :

- lorsque la broche sera au niveau HAUT, la LED sera allumée,
- lorsque la broche sera au niveau BAS, la LED sera éteinte.



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.



7. Intervalle régulier : Exécuter plusieurs actions asynchrones à intervalle régulier : le programme.

Ce qu'on va faire ici...

- Imaginons que l'on veuille faire la même chose mais pour plusieurs actions différentes : par exemple, faire clignoter une LED toutes les secondes, une autre toutes les 2 secondes et une autre toutes les 3 secondes, etc... Le principe utilisé va nous permettre de le faire assez simplement.
- Ici, on se contente d'allumer/éteindre des LEDs, mais ça pourra être à peu près ce que l'on veut... : mesurer la sortie d'un capteur, allumer des dispositifs indépendants entre eux tout en continuant de lire l'état de boutons poussoir ou autre, etc...

Entête déclarative

- On déclare avant tout une constante fixant le nombre de délais de comptage à utiliser. On pourra ainsi facilement adapter le code.

On déclare plusieurs tableaux correspondant aux paramètres voulus :

- de constantes de type **int** pour désigner la broche 2, appelée LED
- de variables de type **long** pour mémoriser la dernière valeur de **millis()** : on utilise un type long car les valeurs vont rapidement dépasser 32768, valeur maximale pour une variable **int**
- de variables de type **int** pour fixer le délai à mesurer
- de variables de type **boolean** pour mémoriser l'état de la LED : noter que l'on peut indifféremment utiliser **HIGH**, **true**, 1 ou **LOW**, **false**, 0.

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

const int nombreDelaies=4; // nombre de délais de comptage à utiliser

const int LED[nombreDelaies]={2,3,4,5}; // tableau de constantes
désignant la broche de la LED

long millis0[nombreDelaies]={0,0,0,0}; // tableau de variables pour
mémoriser dernier millis() pris en compte

int delai[nombreDelaies]={500, 1000, 2000, 4000}; // tableau
d'intervalles à utiliser en millisecondes

boolean etatLED[nombreDelaies]={HIGH, HIGH, HIGH, HIGH}; // LEDs allumées
au départ
// remarque HIGH = true = 1 et LOW = false = 0
```

Fonction **setup()**

A ce niveau, à l'aide d'une boucle **for**, pour chaque tableau, on va :

- initialiser la broche utilisée en sortie (avec la constante LED) avec l'instruction **pinMode()**
- on initialise la LED
- on mémorise la valeur courante de **millis()**

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

    for (int i=0; i<nombreDelaies; i++) {
        pinMode(LED[i], OUTPUT); // met la broche en sortie

        digitalWrite(LED[i],etatLED[i]); // met la LED dans l'état voulu
        millis0[i]=millis(); // mémorise millis()
    }

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

Fonction `loop()`

- A chaque passage, on va tester successivement si chaque délai est écoulé en vérifiant simplement si la différence entre la valeur courante de `millis()` et la dernière valeur prise en compte est supérieure ou égale au délai :
 - si c'est le cas, l'état de la LED est inversé en conséquence
 - la nouvelle valeur de `millis()` est mémorisée
- Cette condition ne sera au final exécutée que lorsque cela sera nécessaire et le reste du code sera exécuté sans être bloqué le reste du temps.

Remarque

On aurait pu condenser ici le code à l'aide d'une boucle `for`, mais en pratique, le code à exécuter pourra être différent pour chaque délai écoulé et par conséquent, il est préférable de les tester un à un successivement.

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

    //--- 0
    if (millis()-millis0[0]>=delai[0]) { // si le délai est écoulé
        millis0[0]=millis(); // mémorise millis()
        etatLED[0]=!etatLED[0]; // inverse l'état de la variable
        digitalWrite(LED[0],etatLED[0]);
    } // fin si délai écoulé

    //--- 1
    if (millis()-millis0[1]>=delai[1]) { // si le délai est écoulé
        millis0[1]=millis(); // mémorise millis()
        etatLED[1]=!etatLED[1]; // inverse l'état de la variable
        digitalWrite(LED[1],etatLED[1]);
    } // fin si délai écoulé

    //--- 2
    if (millis()-millis0[2]>=delai[2]) { // si le délai est écoulé
        millis0[2]=millis(); // mémorise millis()
        etatLED[2]=!etatLED[2]; // inverse l'état de la variable
        digitalWrite(LED[2],etatLED[2]);
    } // fin si délai écoulé

    //--- 3
    if (millis()-millis0[3]>=delai[3]) { // si le délai est écoulé
        millis0[3]=millis(); // mémorise millis()
        etatLED[3]=!etatLED[3]; // inverse l'état de la variable
        digitalWrite(LED[3],etatLED[3]);
    } // fin si délai écoulé

    // les autres instructions ici
    // elles seront exécutées meme si le délai n'est pas écoulé

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

Fonctionnement du programme

- Une fois la carte Arduino programmée les LEDs clignotent... chacune à son rythme et indépendamment les unes des autres.

La limite de cette solution...

Bien que très intéressante, cette technique consistant à tester l'écoulement d'un délai pour exécuter un code peut manquer de précision : si une action à exécuter est un peu longue, cela entraîne un décalage pour l'action suivante. Pour contourner ce problème, dans les cas où la précision est cruciale, il est possible d'utiliser une interruption temporelle comme nous allons le voir à présent.

Pour plus de détails sur les interruptions, voir le tuto dédié.

8. 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

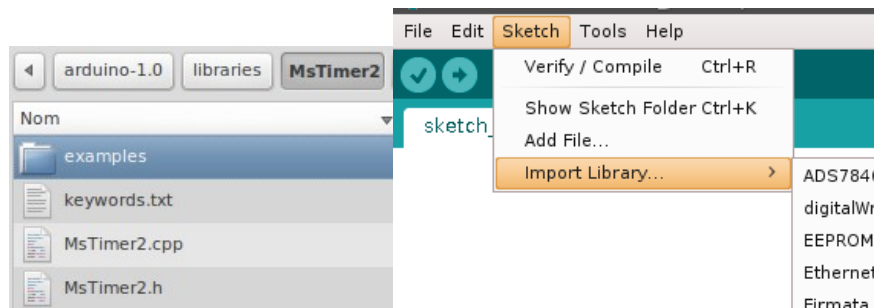
- Site officiel : http://www.pjrc.com/teensy/td_libs_MsTimer2.html

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** (= accessible 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 :

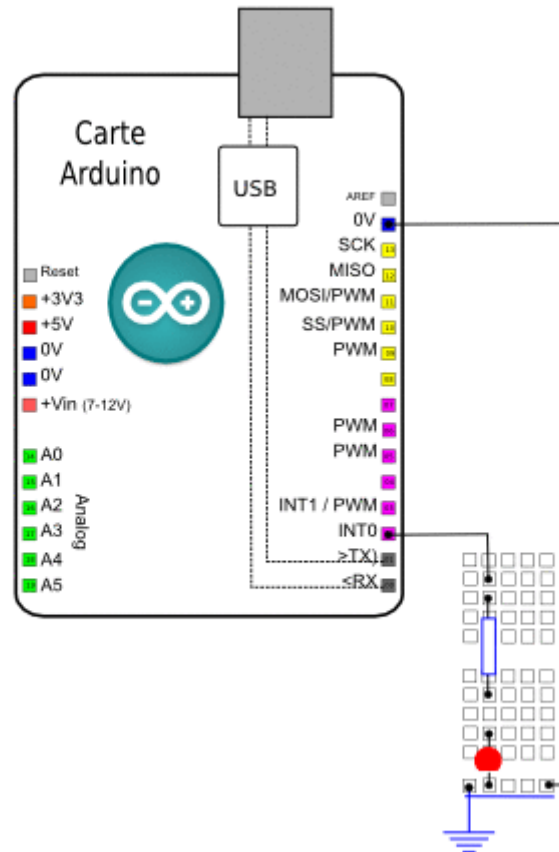
Classe.fonction()

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.

9. *Faire clignoter une LED en utilisant une interruption temporelle : le montage*

- On va reprendre ici le premier montage : une simple LED en série avec sa résistance sur la broche 2 :



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

Ce qu'on va faire ici...

- Dans ce programme, nous allons à nouveau ... faire clignoter une LED !!
- Bon, je sais, c'est pas extraordinaire... mais ça 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()` ni à tester l'écoulement d'un intervalle : on va utiliser ici une interruption « temporelle » ! Aller, c'est parti... rien de bien sorcier, vous allez voir !

Attention : ce tuto utilise une librairie et une interruption : si vous êtes dérouté et ne comprenez pas tout, vous pouvez passer à la suite.

Vous pourrez approfondir ces sujets dans les tutos dédiés.

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 faites-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 ça sur vos tablettes, ça vous servira à l'occasion !



11. Temps réel : Emuler une horloge « temps-réel » avec **millis()**

Ce qu'on va faire ici...

- A présent nous allons aborder le « temps-réel » avec Arduino : c'est le comptage de l'heure « vraie », au format heure:minute:seconde
- La première façon de le faire assez simplement est de se baser sur la fonction **millis()** qui, je le rappelle une nouvelle fois, renvoie une valeur de type **long** correspondant au nombre de millisecondes écoulées depuis la mise sous tension de l'Arduino.
- Le principe que nous allons utiliser ici est le même que précédemment : une variable de mémorisation de la dernière valeur de **millis()** prise en compte et une variable de délai. Sur cette base, nous allons gérer des variables de comptage des minutes et des heures :
 - les minutes seront incrémentées toutes les 60 secondes...
 - les heures seront incrémentées toutes les 60 minutes...
 - etc...

Remarques

La fonction **millis()** a une limite importante : elle ne pourra pas compter au delà de 2 147 483 647 ms (type **long**) soit 24 jours environ... Au-delà, la remise à zéro de **millis()** entraînera une erreur de comptage. Donc, cette solution n'est utilisable que pour des durées limitées, pas pour du « vrai » temps réel permanent.

D'autre part, **millis()** glissera au fil du temps, avec un décalage de quelques secondes possible au bout d'un certain délai, et d'autre part, cette « horloge » sera remise à zéro si Arduino est mis hors tension...

Ses réserves étant posées, utiliser **millis()** comme base « temps réel » est très pratique et suffisant dans de nombreuses situations.

Entête déclarative

Variables utiles

- On déclare une variable de mémorisation de **millis()** et une variable de délai
- On déclare 3 variables pour la gestion de l'heure, en utilisant l'heure courante.

```
//-- variables pour base temps
long millis0=0; // variable pour mémoriser dernier millis() pris en compte
int delai=1000; // intervalle à utiliser en millisecondes

//-- variables de mémorisation de l'heure ==> indiquer l'heure de programmation
int secondes=0;
int minutes=35;
int heures=13;
```

Fonction **setup()**

Initialisation série

- On initialise la fonction série à 115200 bauds

Initialisation du comptage de l'heure

- On mémorise **millis()** courant :

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

    Serial.begin(115200); // initialise port série
    millis0=millis(); // mémorise millis()

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

Fonction loop()

- On commence par tester si le délai de 1 seconde est initialisé :
 - ensuite par un jeu de condition, on assure l'incrémentation des variables des heures, minutes et secondes,
 - ainsi que la remise à zéro des variables lorsque les valeurs maximales sont atteintes.
- On affiche ensuite l'heure au format HH:MM:SS : remarquer la façon d'enchaîner plusieurs `Serial.print()` sur la même ligne...

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

    if (millis()-millis0>=delai) { // si le délai est écoulé
        millis0=millis(); // mémorise millis()

        //----- gestion variables de l'heure ---
        secondes=secondes+1; // incrémente secondes
        if (secondes>59) {
            secondes=0; // RAZ secondes
            minutes=minutes+1; // incrémente minutes
        } // fin if secondes

        if (minutes>59) {
            minutes=0; // RAZ minutes
            heures=heures+1; // incrémente heures
        } // fin if minutes

        if (heures>23) heures=0; // RAZ heures

        //--- affiche heure ---
        if (heures<10)Serial.print("0"),Serial.print(heures), Serial.print(":"); else Serial.print(heures), Serial.print(":");
        if (minutes<10)Serial.print("0"),Serial.print(minutes), Serial.print(":"); else Serial.print(minutes), Serial.print(":");
        if (secondes<10)Serial.print("0"),Serial.print(secondes), Serial.println(); else Serial.print(secondes), Serial.println();

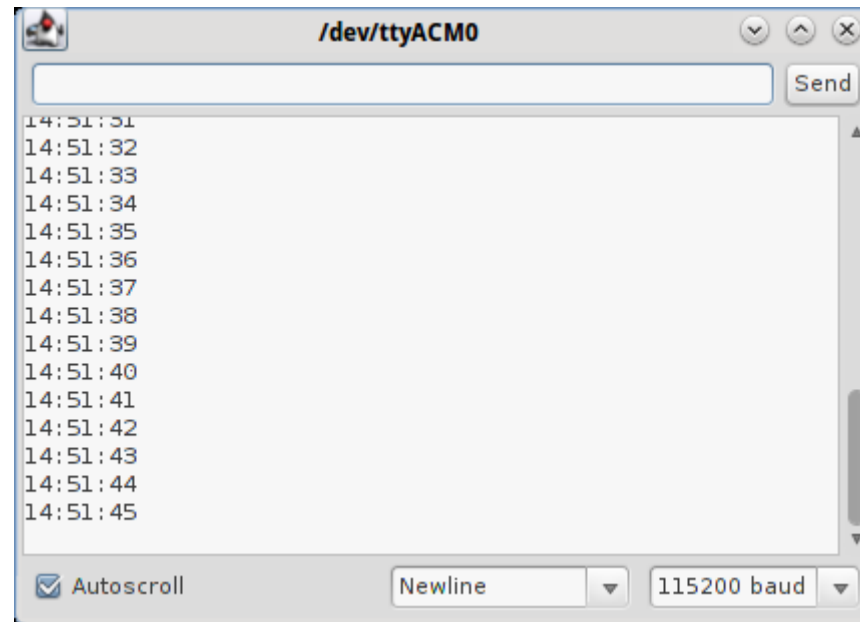
    } // fin si délai écoulé

    // les autres instructions ici
    // elles seront exécutées meme si le délai n'est pas écoulé

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

Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne ...



Quelques trucs utiles...

Truc n°1 : pour s'assurer du bon comptage des minutes et des heures, utiliser un délai court (100ms, 10ms et même 1ms) pour tester...

Truc n°2 : il est possible de mettre à l'heure automatiquement lors de la programmation à l'aide de la variable `__TIME__` (voir tuto dédié au « temps réel »)...

A savoir :

Il est possible et assez simple d'utiliser avec Arduino des composants dit « RTC » (Real Time Clock = horloge temps réel) : cette solution permet d'utiliser le temps réel sans limite de durée et de façon non volatile (la mise hors tension n'affecte pas l'heure...)

Tous les détails et plusieurs exemples d'utilisation sont présentés dans le tuto dédié au « temps réel » (ou RTC)

12. Attente de durée indéterminée : Attendre la survenue d'un événement

Ce qu'on va faire ici...

- Poursuivons notre petite exploration des stratégies de temporisation en langage Arduino. Une situation qui peut parfois se présenter : attendre une durée indéterminée qu'un événement survienne... Cette situation potentiellement bloquante d'ailleurs, peut parfois se présenter et je vous montre ici comment le faire.
- Une application possible sera notamment d'exécuter un programme pas à pas par clic sur le bouton « envoi » du terminal série notamment, ce que nous allons faire ici.

Entête déclarative

- Laissée vide ici.

Fonction **setup()**

Initialisation série

- On initialise la communication série à la vitesse voulue :

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

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

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

Fonction **loop()** ()

- Ici, on va se contenter de simuler 10 « pas » successifs par une boucle
- Entre chaque pas, on testera à l'aide d'une condition **while()** si le caractère ascii 10 (autrement dit un saut de ligne « \n ») a été reçu en réception sur le port série. Tant que ça ne sera pas le cas, le programme restera bloqué. On obtient ainsi une façon simple de réaliser une exécution pas à pas d'un programme en répartissant à bon escient la ligne de « blocage » :

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

    for (int i=0; i<10; i++) { // simule 10 "pas"

        Serial.print("Pas "), Serial.println(i);
        while(Serial.read()!=10); // attend tant que ne reçoit pas saut de ligne (\n = ascii 10)
        // en réglant le Terminal série sur newline : un clic sur envoi émet un saut de ligne

    } // fin for

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

Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne ...



A chaque clic sur « envoi », le « pas » suivant est exécuté... Pratique !

Un autre truc à connaître

Il est possible très simplement de bloquer la fonction `loop()` à l'endroit de son choix, à l'aide de l'instruction : `while(1) ;`

La condition étant toujours vraie, le programme se bloque à ce niveau !

Ceci est très pratique potentiellement pour mettre au point un code comportant un bug inexpliqué : on pourra exécuter tout ou partie de la fonction `loop()` à sa convenance pour voir si tout se passe bien jusqu'à l'endroit où l'on aura inséré cette ligne.

13. Attente de durée indéterminée : Sortir d'un code s'exécutant sans fin lors d'un événement voulu

Ce qu'on va faire ici...

- Dans la même lignée de ce que nous venons de voir, voici une petite variante : imaginer que l'on veuille exécuter en boucle un code et en sortir seulement lorsque un événement survient.

Entête déclarative

- Laissée vide ici.

Fonction **setup()**

Initialisation série

- On initialise la communication série à la vitesse voulue :

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

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

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

Fonction **loop()** ()

- A l'aide d'une condition while(true), on crée une boucle sans fin au sein de la fonction loop()
 - on y place les instructions à exécuter en boucle
 - une condition if permet de tester l'arrivée d'un saut de ligne : dans ce cas on quitte la boucle à l'aide de l'instruction **break** (sans ()) qui permet de sortir de la boucle **while()**

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

    while(true) { // le code contenu dans la condition est exécuté en boucle

        Serial.print(">");
        Serial.print(".");
        delay(1000);
        if (Serial.read()==10) break; // sort du while sur réception d'un saut de ligne

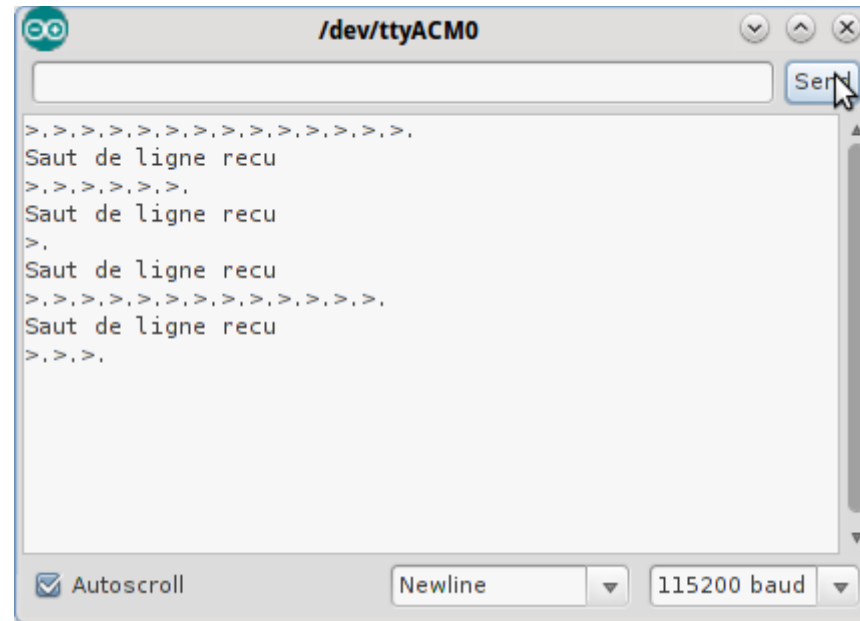
    } // fin while true

    Serial.println("\nSaut de ligne reçu");

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


Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne ...



A chaque clic sur « envoi », on sort de la boucle `while()`...

Mise en garde

ATTENTION : L'utilisation d'une instruction `while()` dans un programme est potentiellement **bloquante** !

Exemple : « Tant que le bouton pas appuyé... » : si aucun appui ne survient, le programme est bloqué !

En pratique, utiliser **while**(condition) avec parcimonie !

14. Mesure de durée courte ou ultra courte : par exemple mesurer le temps d'exécution d'une fonction

Ce qu'on va faire ici...

- Il peut souvent être utile de pouvoir mesurer la durée d'exécution d'une fonction, d'une instruction... Voici une petite stratégie qui vous montre comment le faire assez simplement. Le principe va consister à mémoriser `millis()` ou `micros()` avant le code dont il faut évaluer la durée, puis après : ensuite on fera la différence entre les 2 et on affichera le résultat. CQFD...
- Le code est disponible ici : <https://gist.github.com/sensor56/9edb583246ebdcf2389e>

Entête déclarative

- On déclare 3 variables de type long utilisées pour la mesure de durée :

```
//--- entete déclarative
long debut=0;
long fin=0;
long duree=0;
```

Fonction `setup()`

Initialisation série

- On initialise la communication série à la vitesse voulue :

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

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

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

Fonction `loop()` ()

- Ici nous allons mesurer la durée de l'instruction `analogRead()` qui réalise une conversion analogique-numérique : on mémorise `micros()` avant et après puis on calcule la durée que l'on affiche, ce qui donne :

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

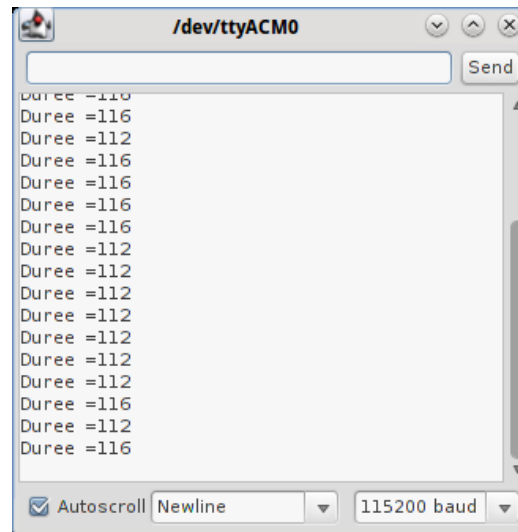
  debut=micros(); // mémorise la valeur de la fonction micros() avant
  analogRead(A0); // instruction dont on veut mesurer la durée d'exécution
  fin=micros(); // mémorise la valeur de la fonction micros() après
  duree=fin-debut;

  Serial.print("Duree =");
  Serial.println(duree);

  delay(500) ; // entre 2 mesure de duree..
} // fin de la fonction loop()
```

Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne ...



La fonction `analogRead()` dure 110 microsecondes environ.. soit jusqu'à 9000 mesures/secondes !

15. Déclencher un événement après un laps de temps court une fois un événement déclencheur survenu

Ce qu'on va faire ici...

- Une dernière situation dans laquelle on peut se trouver : par exemple, on veut déclencher un dispositif après un très court laps de temps après la survenue d'un événement déclencheur (photo d'une goutte d'eau qui tombe par exemple) : l'instruction `delay()` est ici trop « longue » et il va falloir utiliser l'instructions `delayMicroseconds()`... Tout bête, mais encore faut-il le savoir...
- Le code est disponible ici : <https://gist.github.com/sensor56/1a6a3aecab13fe231104>

Entête déclarative

- On déclare une variable pour la broche d'un bouton poussoir et une constante d'appui :

```
//--- entete déclarative
int BP=2;

const APPUI=0; // valeur correspondant à l'appui
```

Fonction `setup()`

Initialisation série

- On initialise la communication série à la vitesse voulue, on met la broche utilisée en sortie et on active le rappel au plus (voir tuto sur les boutons poussoir pour les détails) :

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

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

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

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

Fonction **loop()** ()

- Ici nous allons ici montrer le principe :
 - on teste l'état du bouton poussoir :
 - lorsqu'il est appuyé, on appelle la fonction **delayMicroseconds()** avec la durée voulue avant d'appeler le code voulu.

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

  if (digitalRead(BP)==APPUI) {

    delayMicroseconds(10); // pause très courte (10 microsecondes )

    // action à exécuter

  } // fin if

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

Un code tout bête mais qui pourra vous dépanner à l'occasion.

Au terme de ce tuto, vous aurez essentiellement découvert quelques stratégies de programmation utiles au fil des programmes Arduino : rien de très compliqué, mais toujours bon à connaître.

16. Les éléments du langage Arduino étudiés dans cet atelier

Structure

Variables et constantes

Fonctions

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

17. A présent, vous devriez être capable :

Table des matières

Stratégies de programmation : temporisations avec Arduino

Intro |

Matériel nécessaire pour les ateliers Arduino |

Faire clignoter une LED : Le montage. |

Intervalle régulier : Faire clignoter une LED avec une pause : le programme. |

Intervalle régulier : Exécuter une action à intervalle régulier... sans bloquer le programme. |

Intervalle régulier : Exécuter plusieurs actions asynchrones à intervalle régulier : le montage |

Intervalle régulier : Exécuter plusieurs actions asynchrones à intervalle régulier : le programme. |

Arduino : La librairie MsTimer2 |

Faire clignoter une LED en utilisant une interruption temporelle : le montage |

Faire clignoter une LED en utilisant une interruption temporelle : le programme |

Temps réel : Emuler une horloge « temps-réel » avec millis() |

Attente de durée indéterminée : Attendre la survenue d'un événement |

Attente de durée indéterminée : Sortir d'un code s'exécutant sans fin lors d'un événement voulu |

Mesure de durée courte ou ultra courte : par exemple mesurer le temps d'exécution d'une fonction |

Déclencher un événement après un laps de temps court une fois un événement déclencheur survenu |

Les éléments du langage Arduino étudiés dans cet atelier |

A présent, vous devriez être capable : |

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