ARDUINO

Licence pro MICVI
2015-2016

Eric Simon
François Cabestaing
Ludovic Macaire

Objectif pédagogique
L'objectif est de prendre en main la carte ARDUINO et de donner les bases de la programmation sur cette carte. La progression se fera pas à pas en s'appuyant sur ce qui aura été déjà assimilé.

Les notions suivantes seront abordées:
- les entrées-sorties numériques et analogiques de l'ARDUINO ainsi que les fonctions qui leurs sont associées,
- les fonctions de gestion du temps,
- la gestion de signaux périodiques,
- les interruptions matérielles,
- la communication par liaison série.

Organisation des TPs
Ce document s'articule autour de l'excellent ebook de Simon Landrault (Eskimon) et de Hippolyte Weisslinger, « *Arduino : Premiers pas en informatique embarquée* », édition du 19 juin 2014. Cet ebook sera notre fil conducteur. Etant donné la taille de ce livre (450 pages), nous nous focaliserons uniquement sur les parties essentielles.
Pour chaque notion abordée, nous renverrons d'abord l'étudiant sur l'ebook pour assimiler les notions essentielles, puis un exercice servira de mise en pratique.
Ce document s'inspire également du livret « *Arduino, cahier de projet domotique* » rédigé par Jean-Marc Vannobel, enseignant-chercheur à Lille1.
Les renvois vers l'ebook seront indiqués en *italique,* ainsi que le *travail à faire*.

Matériel mis à disposition et caractéristiques
La mise en oeuvre du travail s'appuiera sur :
- l'IDE ARDUINO,
- un ARDUINO de type UNO,

Pour rappel, l'ARDUINO UNO dispose au niveau matériel de :
- 14 entrées-sorties numériques (0 à 13) dont :
- 6 sorties PWM (3, 5, 6, 9, 10 et 11),
- 2 entrées pour les interruptions externes (2, 3),
- 6 entrées analogiques (A0 à A5) qui peuvent être utilisées en :
- entrées-sorties numériques (14 à 19),
- communication I2C sur A4 (SDA) et A5 (SCL).

Les principales fonctions standard auxquelles nous ferons appel sont :
- pinMode(),
- digitalWrite(),
- digitalRead(),
- analogRead(),
- analogWrite(),
- tone(),
- noTone(),
- millis(),
- delay()

Premiers pas avec Arduino

Définition [wikipedia] : Arduino est un circuit imprimé en matériel libre (les plans de la carte elle-même sont publiés en licence libre, cependant, certains composants de la carte, comme le microcontrôleur par exemple, ne sont pas en licence libre) sur lequel se trouve un microcontrôleur qui peut être programmé pour analyser et produire des signaux électriques, de manière à effectuer des tâches très diverses comme la domotique (le contrôle des appareils domestiques - éclairage, chauffage…), le pilotage d'un robot, etc. C'est une plateforme basée sur une interface entrée/sortie simple. Il était destiné à l'origine principalement mais pas exclusivement à la programmation multimédia interactive en vue de spectacle ou d'animations artistiques.

Avant de réaliser vos premières expériences avec la carte Arduino, vous devrez apprendre à démarrer le circuit et à téléverser votre premier programme. Pour cela, suivez les étapes ci-dessous :

1) Interface du logiciel Arduino et fonctionnement d'un programme :
Lecture ebook pages 29 à 34

2) Présentation de la carte :
Lecture ebook pages 34 à 36

3) Tester son matériel :
Pour tester son matériel, vous allez ouvrir un programme à partir de la bibliothèque d'exemples et le téléverser pour s'assurer que tout fonctionne. Ce programme, appelé « blink », consiste à faire clignoter la LED intégrée de l'arduino qui se trouve sur la broche 13. Pour l'instant ne vous attardez pas sur les fonctions utilisées, elles seront étudiées dans la section 2 lorsque vous écrirez votre propre programme. Il s'agit juste ici de faire tourner un programme existant pour vérifier le bon fonctionnement de la carte. Le détail de la procédure est donné dans l'ebook :
Lecture ebook pages 37 à 45

4) Fonctionnement d'un programme :
Lecture ebook pages 46 à 49

Programme 1 : Fonctions allumages permanent et clignotant

Dans l'exemple « blink » de la bibliothèque Arduino que vous avez ouvert puis exécuté précédemment pour tester votre matériel, vous avez fait clignoter la LED intégrée de l'Arduino sur la broche 13.
Nous vous proposons maintenant de réaliser votre propre programme, qui va consister à faire un allumage permanent puis à faire clignoter une LED **externe** branchée sur la broche 2.
Mais avant de programmer, il faut dimensionner les composants.
Pour éviter d'avoir un courant trop fort qui traverse la LED, il faut ajouter une résistance en série.

1) Calculer la valeur de la résistance pour avoir un courant maximum de 20 mA.
Vous aurez besoin pour cela de connaître la tension de seuil de la LED. Consulter la data sheet de la LED.

2) Réaliser le montage hors tension, puis appeler le prof pour vérification.
Lecture ebook pages 85-86 pour le montage

3) Passons maintenant au programme. *Saisir dans la fenêtre d'édition le code ci-dessous :*

```
//définition de la broche 2 de la carte en tant que variable
byte led_rouge = 2;
//fonction d'initialisation de la carte
void setup()
{
//initialisation de la broche 2 comme étant une sortie
pinMode(led_rouge, OUTPUT);
// écriture en sortie (broche 2) d'un état BAS
digitalWrite(led_rouge, LOW);
}
//fonction principale, elle se répète (s'exécute) à l'infini
void loop()
{

}
```

Depuis les menus « Fichier » puis « Enregistrer sous... », enregistrez votre travail dans un répertoire « ARDUINO » que vous allez à cette occasion créer dans votre zone de travail sur H : Par la suite, chacun de vos travaux sera sauvegardé dans ce répertoire.
Ce code utilise les deux fonctions pinMode() et digitalWrite(). Aller voir les définitions des fonctions dans les annexes.

4) Cliquer sur le bouton « Téléverser » pour transférer le code de votre sketch dans l'Arduino et observer le résultat.

5) Modifier le sketch donné de façon à ce que la LED clignote indéfiniment avec une fréquence de 2 Hz.
Vous aurez besoin de la fonction delay().

Programme 2 : Fonction Interrupteur avec un bouton à deux positions

Notre but est de gérer l'éclairage de la LED par un bouton à deux positions. Nous allons utiliser pour cela le switch donné par le prof, qui comprend 4 boutons à deux positions :

Vous utiliserez un seul des 4 boutons. Le bouton ne peut être branché directement sur l'entrée de la carte. Il va nécessiter l'utilisation d'une résistance de pull-up

1) Lecture ebook pages 119 à 123 pour comprendre le rôle de la résistance de pull-up

2) Réaliser le montage et le programme pour allumer la LED lorsque le bouton est en position ON. Utiliser une résistance de pull-up ***externe****.*
Vous aurez besoin de la structure conditionnelle pour écrire le programme : *lecture ebook page 59-60.*

3) Utiliser maintenant la résistance de pull-up ***interne****.*
Pour activer la résistance interne, il vaut mieux procéder de la manière suivante, en utilisant le mot-clé INPUT_PULLUP dans l'instruction de déclaration de mode de la broche. INPUT_PULLUP permet simultanément de configurer la broche en mode entrée numérique avec la résistance interne:

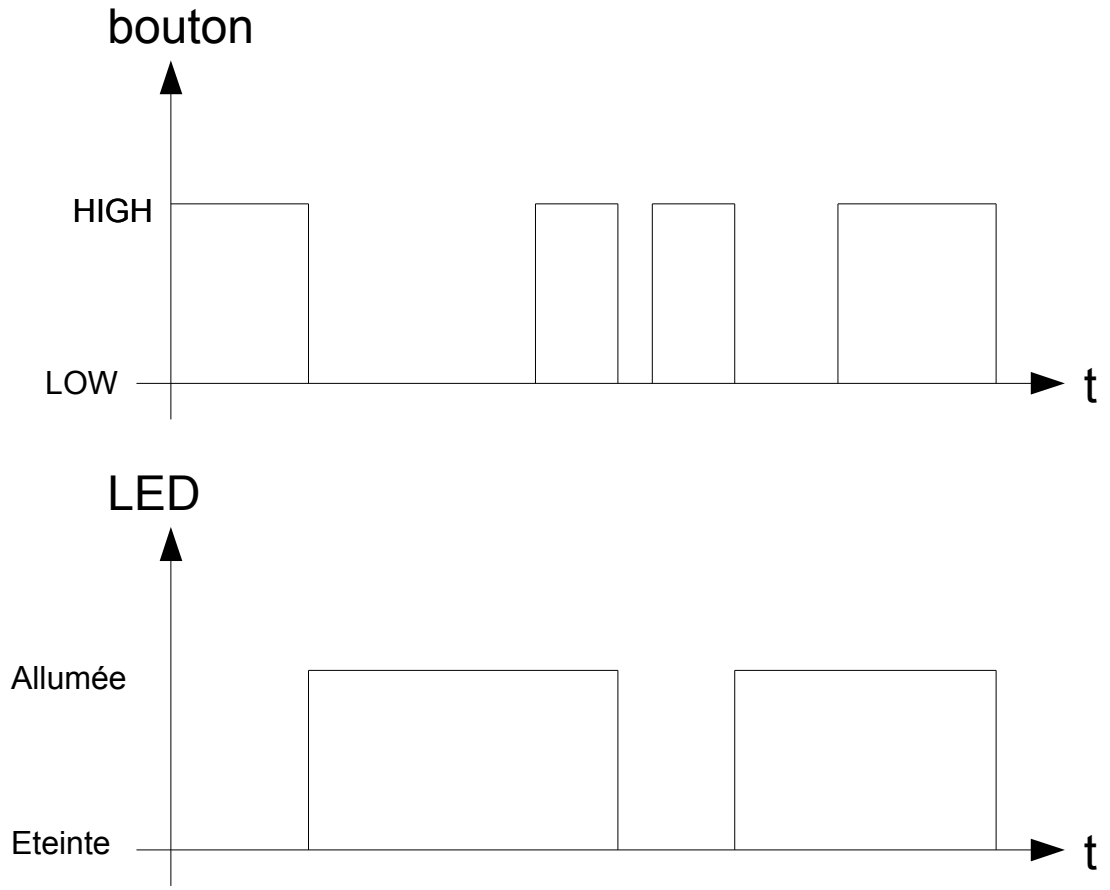
pinMode(numéro broche, INPUT_PULLUP);

Remarque : la résistance interne de pull-up d'une broche est automatiquement déconnectée dès que cette dernière est configurée en sortie numérique même si la broche avait auparavant été déclarée en entrée avec sa résistance de pull-up activée.

Programme 3 : Fonction Interrupteur avec un bouton poussoir

Le but de ce programme est d'utiliser maintenant un bouton poussoir pour changer l'état de la LED à chaque appui sur le bouton poussoir :

Il faut donc repérer un appui, c'est-à-dire le passage de l'état non appuyé à l'état appuyé. En d'autres termes, le passage du bouton de l'état haut (repos) à l'état bas (actif). Ce passage s'appelle un front descendant.
Représenter par une flèche dirigée vers le bas sur le chronogramme les fronts descendants du bouton.



Ce chronogramme montre que la LED, initialement éteinte s'allume lors d'un premier front descendant du bouton, puis s'éteint au suivant, s'allume au troisième, etc.
Réaliser le programme correspondant à ce fonctionnement.

Programme 4 : Fonction Gradateur

Le principe de la fonction gradateur est de moduler l'intensité de l'éclairage de la LED (de 0 % à 100 %). Nous aurons alors besoin de sortir une information analogique correspondant à l'intensité souhaitée. Or l'arduino ne dispose pas de vrais sorties analogiques. Par contre nous pouvons utiliser la sortie digitale en mode analogique avec une commande de type modulation de largeur d'impulsions (MLI) ou PWM en anglais. Le principe est de faire varier le rapport cyclique du signal d'amplitude 5 V. La fréquence de la PWM est de 490 Hz, ce qui est bien supérieur aux 50 Hz nécessaires pour observer aucun clignotement. En effet, au delà des 50 Hz, l'œil n'est plus capable de distinguer le clignotement. Cette opération est réalisée à l'aide de la fonction analogWrite(numéro broche, rapport cyclique). Tous les détails sont donnés dans l'ebook :

1) Lecture ebook pages 259 à 263.

2) Écrire un programme où la LED sera modulée avec un taux de 100 % durant une seconde, puis avec un taux de 90 % pendant 1 seconde, puis 80 %, et ce jusqu'à l'extinction de la LED.

3) La carte arduino dispose de convertisseurs analogiques numériques sur 10 bits. Ce sont les 6

entrées analogiques. L'objectif est maintenant d'utiliser un potentiomètre pour régler l'intensité de l'éclairage de la LED.
Pour cela, vous aurez besoin de comprendre comment marchent les entrées analogiques :
Lecture ebook pages 243 à 245 pour les entrées analogiques.
Lecture ebook pages 246 à 248 pour l'utilisation du potentiomètre.
Écrire le programme qui permet de régler l'intensité de l'éclairage avec le potentiomètre.

Attention : les rapports cycliques sont sur 256 niveaux alors que les valeurs analogiques sont sur 1024 niveaux (les 10 bits du convertisseur).

Programme 5 : Commande de feux automobiles

On dispose, sur une voiture, de 4 commandes indépendantes: Cv pour les veilleuses, Cc pour les feux de croisement, Cr pour les feux de route et Ca pour les phares anti-brouillard (valeur 1 au travail, 0 au repos).
On note les états des lumières V pour les veilleuses, C pour les feux de croisement, R pour les feux de route et A pour les feux antibrouillard (valeur 1 à l'allumage, 0 à l'extinction).
les veilleuses peuvent être allumées seules mais l'allumage des feux de croisement ou des feux de route ou des antibrouillard entraîne obligatoirement l'allumage des veilleuses. De plus, il est précisé que les 4 phares ne peuvent être allumés simultanément. On suit alors les règles suivantes :
- les feux de croisement ont priorité sur les feux de route et sur les antibrouillard ;
- les antibrouillard ont priorité sur les feux de route

Utilisez le switch à 4 boutons pour simuler les 4 commandes Cv, Cc, Cr, Ca, et 4 LED pour simuler les 4 feux.

Réaliser le programme et le montage correspondant.

Note : Vous allez utiliser le type boolean pour faire cet exercice :
Lecture ebook pages 54-55 pour la définition du type boolean

Vous aurez également besoin d'utiliser les opérateurs logiques :
Lecture ebook pages 62-63

Annexes

pinMode()

Description :

Configures the specified pin to behave either as an input or an output. See the description of digital pins for details on the functionality of the pins.

Syntax :

pinMode(pin, mode)

Parameters :

pin: the number of the pin whose mode you wish to set

mode: INPUT, OUTPUT

Returns :

None

---

digitalWrite()

Description :

Write a HIGH or a LOW value to a digital pin.

If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.

Syntax :

digitalWrite(pin, value)

Parameters :

pin: the pin number

value: HIGH or LOW

Returns :

none

---

delay()

Description :

Pauses the program for the amount of time (in miliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

Syntax :

delay(ms)

Parameters :

ms: the number of milliseconds to pause (unsigned long)

Returns : nothing

analogWrite()

Description :

Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to analogWrite(), the pin will generate a steady square wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or digitalWrite() on the same pin). The frequency of the PWM signal on most pins is approximately 490 Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz. Pins 3 and 11 on the Leonardo also run at 980 Hz.

On most Arduino boards (those with the ATmega168 or ATmega328), this function works on pins 3, 5, 6, 9, 10, and 11. On the Arduino Mega, it works on pins 2 - 13 and 44 - 46. Older Arduino boards with an ATmega8 only support analogWrite() on pins 9, 10, and 11.

The Arduino Due supports analogWrite() on pins 2 through 13, plus pins DAC0 and DAC1. Unlike the PWM pins, DAC0 and DAC1 are Digital to Analog converters, and act as true analog outputs.

You do not need to call pinMode() to set the pin as an output before calling analogWrite().

The analogWrite function has nothing to do with the analog pins or the analogRead function.

Syntax

analogWrite(pin, value)

Parameters

pin: the pin to write to.

value: the duty cycle: between 0 (always off) and 255 (always on).

Returns

nothing

Notes and Known Issues

The PWM outputs generated on pins 5 and 6 will have higher-than-expected duty cycles. This is because of interactions with the millis() and delay() functions, which share the same internal timer used to generate those PWM outputs. This will be noticed mostly on low duty-cycle settings (e.g 0 - 10) and may result in a value of 0 not fully turning off the output on pins 5 and 6.

analogRead()

Description

Reads the value from the specified analog pin. The Arduino board contains a 6 channel (8 channels on the Mini and Nano, 16 on the Mega), 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit. The input range and resolution can be changed using analogReference().

It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

Syntax

analogRead(pin)

Parameters

pin: the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)

Returns

int (0 to 1023)

Note

If the analog input pin is not connected to anything, the value returned by analogRead() will fluctuate based on a number of factors (e.g. the values of the other analog inputs, how close your hand is to the board, etc.).

type boolean

A boolean holds one of two values, true or false. (Each boolean variable occupies one byte of memory.)

Example

```
int LEDpin = 5;       // LED on pin 5
int switchPin = 13;   // momentary switch on 13, other side connected to ground

boolean running = false;

void setup()
{
 pinMode(LEDpin, OUTPUT);
 pinMode(switchPin, INPUT);
 digitalWrite(switchPin, HIGH);      // turn on pullup resistor
}

void loop()
{
 if (digitalRead(switchPin) == LOW)
 {  // switch is pressed - pullup keeps pin high normally
  delay(100);                        // delay to debounce switch
  running = !running;                // toggle running variable
  digitalWrite(LEDpin, running)      // indicate via LED
 }
}
```

Boolean Operators

These can be used inside the condition of an if statement.

&& (logical and)
True only if both operands are true, e.g.

```
if (digitalRead(2) == HIGH  && digitalRead(3) == HIGH) { // read two switches
  // ...
}
```

is true only if both inputs are high.

|| (logical or)
True if either operand is true, e.g.

```
if (x > 0 || y > 0) {
  // ...
}
```

is true if either x or y is greater than 0.

! (not)
True if the operand is false, e.g.

```
if (!x) {
  // ...
}
```
is true if x is false (i.e. if x equals 0).