Introduction à la programmation des micro-contrôleurs -TP 1

26 février 2015

Table des matières

1 Introduction **2**

2 Le micro-contrôleur Arduino **2**

3 La démarche de réalisation d'un programme **4**

4 Prise en main **4**
- 4.1 Structure du programme . . . 4
- 4.2 Le fameux "Hello Word" . . . 5
- 4.3 Les différents types et les précautions qui vont avec . . . 5
- 4.4 Elements de syntaxe . . . 6
  - 4.4.1 Boucle *for* . . . 6
  - 4.4.2 Boucle *while* . . . 6
  - 4.4.3 Structure conditionnelle *if* . . . 6
- 4.5 Affection des Entrées/Sorties . . . 7
- 4.6 Lire les entrées . . . 7
- 4.7 Imposer une tension en sortie . . . 7
- 4.8 Quelques programmes de base . . . 8
  - 4.8.1 Le "Led Blink" . . . 8
  - 4.8.2 Variation de luminosité d'une Led . . . 8

5 Mini-projet : Réalisation d'un radar de recul **9**
- 5.1 Introduction . . . 9
- 5.2 Objectif . . . 9
  - 5.2.1 Déroulement du mini-projet . . . 9
  - 5.2.2 Montage . . . 10
  - 5.2.3 Pour ceux qui sont en avance . . . 10

A Description du fonctionnement du module Ultrason "HCSR04" **11**

1 Introduction

De nombreux systèmes industriels sont automatiques, et à ce titre, ils exécutent des tâches plus ou moins complexes de manière autonome. Chacune de ces tâches peuvent très souvent être décomposées en tâches élémentaires que l'on peut décrire (comme nous l'avons vu en début d'année) par une chaîne fonctionnelle comme le montre la figure 1 :

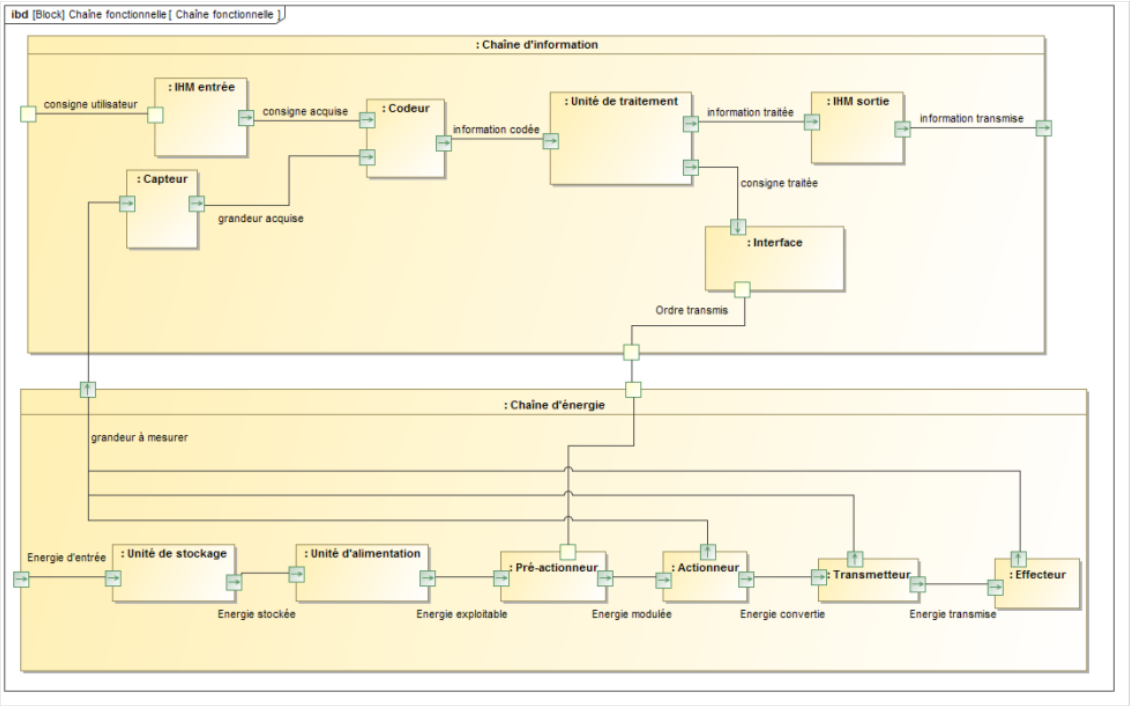


FIGURE 1 – Chaîne fonctionnelle (IBD)

Dans ce TP (et ceux qui vont suivre), on s'intéresse plus particulièrement aux composants de la chaîne d'information, et notamment à la manière dont l'unité de traitement gère les ordres et informations.

2 Le micro-contrôleur Arduino

Les unités de traitement existent sous différentes formes. Les micro-contrôleurs en font partie. Leurs coûts et les fonctionnalités qu'ils proposent dépendent du cahier des charge du système à concevoir. Pour des raisons de coûts et de simplicité de manipulation, nous utiliserons une carte de type **Arduino**. Le site officiel www.arduino.cc propose de multiples ressources sur les produits Arduino. La carte de "base" est la carte **Arduino Uno**. Cette carte est fondée sur un micro-contrôleur *ATMega328* cadencé à 16 MHz. Les connecteurs situés sur les bords extérieurs du circuit imprimé permettent d'enficher une série de modules complémentaires. Cette carte peut se programmer avec le logiciel Arduino, grâce à un cordon de connexion USB [1].

1. Attention, si vous désirez achetez une carte (pour les TIPE par exemple), il faut acheter également le cordon !

Les entrées/sorties de la carte Arduino sont détaillées sur la figure 2.

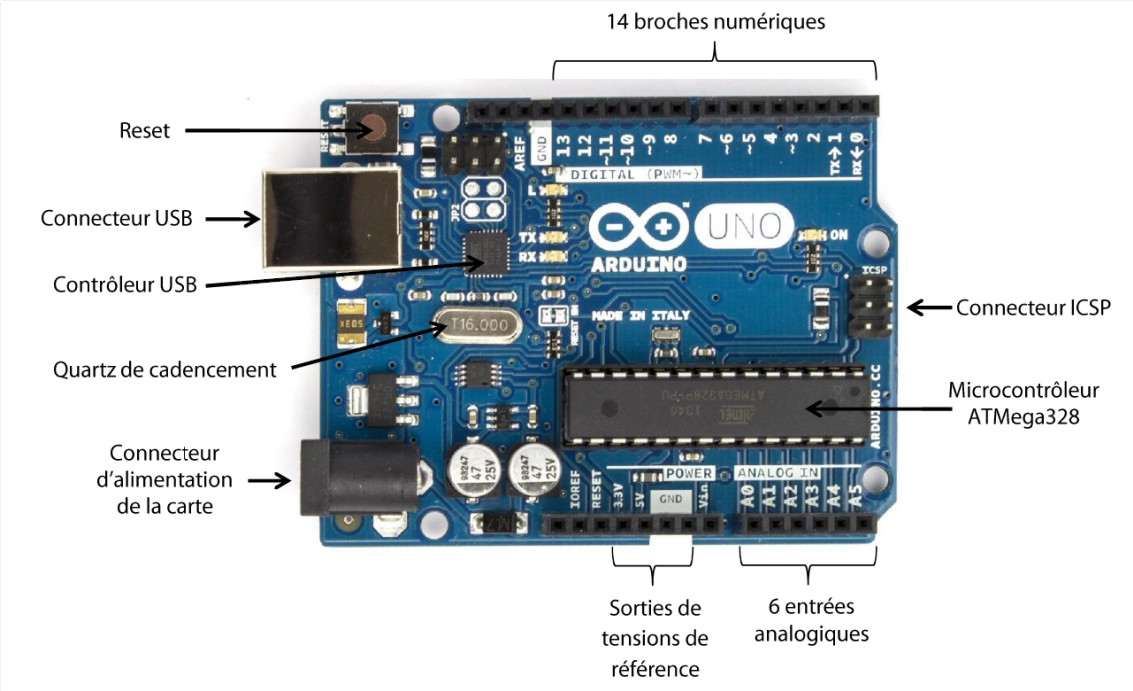


FIGURE 2 – Entrées-Sorties de la carte Arduino Uno

Cette carte dispose :

- d'un micro-contrôleur ATMega328,
- de 14 broches numériques d'entrées/sorties (dont 6 peuvent être utilisées en sorties PWM [2]),
- de 6 entrées analogiques (qui peuvent également être utilisées en broches entrées/sorties numériques),
- d'un quartz 16MHz (horloge de cadencement),
- d'une connexion USB (avec son contrôleur associé) qui permet la programmation du micro-contrôleur ainsi que l'alimentation de la carte,
- d'un connecteur d'alimentation jack (nécessaire si le cordon USB est déconnecté après programmation),
- d'un connecteur ICSP(programmation "in-circuit"),
- et d'un bouton de réinitialisation (reset).

2. Ce terme sera détaillé plus loin dans le sujet

3 La démarche de réalisation d'un programme

La carte Arduino est une carte programmable, à volonté, à cela prêt que le langage de base n'est malheureusement pas Python, mais le langage Arduino (un mix entre le langage C et C++). La syntaxe est bien plus lourde que celle offerte par Python, mais aussi bien plus rigoureuse. La démarche de conception d'un programme est donnée ci dessous :

1. On utilise le logiciel Arduino pour écrire son programme
2. On télécharge (télé-verse) le programme dans la carte (il y a une étape de compilation)
3. Le programme étant téléchargé, la carte fonctionne en autonomie si elle a sa propre source d'énergie, ou reste connectée au PC sinon, ou si une communication avec le PC est prévue (via la liaison série par exemple)

4 Prise en main

Activité 1. Brancher la carte Arduino à l'ordinateur puis démarrer le logiciel Arduino en cliquant sur l'icône suivante :

Une fois le logiciel démarré, aller dans *Outils*, *Type de carte* puis vérifier que *Arduino Uno* est bien coché (le cocher sinon). Ensuite, vérifier que la carte est bien reconnue en allant dans *Outils*, *Port*. La carte est reconnue si un port est affiché, par exemple "*Com 3*".

4.1 Structure du programme

A l'ouverture du logiciel, on peut remarquer qu'un bout de code est déjà fourni :

```
void setup() {
// put your setup code here, to run once:

}
```

setup() est une fonction de ***type*** *void*, c'est à dire que cette fonction ne renvoie rien. On dit que c'est une procédure. Tout le code compris entre les accolades ne sera exécuté qu'une **seule** fois.
Ensuite, on rencontre :

```
void loop() {
// put your main code here, to run repeatedly:

}
```

C'est aussi une procédure (typée *void*). Elle contiendra le code principal. Le corps de cette procédure sera exécutée en boucle.

4.2 Le fameux "Hello Word"

On désire afficher, sur l'écran de l'ordinateur, une seule fois "Hello Word". Dans ce cas, la carte communique avec le PC. On utilisera la communication série RS232 [3] Une nouvelle liaison série (plus rigoureusement un nouvel objet de communication série RS232) s'initialise dans le *setup* avec *Serial.begin(115200);*[4]. Ensuite, pour afficher un message, on utilise l'instruction *Serial.println("Hello World!");*

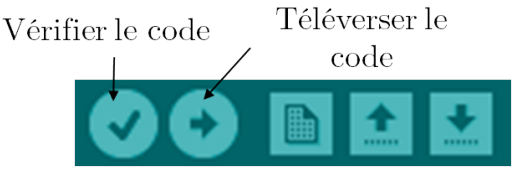
Activité 2. Créer le répertoire suivant Utilisateurs/Documents/Arduino/Nom_Prénom. Ensuite, sous Arduino, créer le programme suivant :

```
void setup() {
Serial.begin(115200);
Serial.println("Hello World!");
}

void loop() {
// put your main code here, to run repeatedly:

}
```

Une fois le code créé, cliquer sur l'icône de vérification du code :



Téléverer ensuite le programme dans la carte, puis sélectionner *Outils/Moniteur série*. Si tout se passe bien, le message apparaît.

4.3 Les différents types et les précautions qui vont avec

Contrairement à Python, Arduino requiert qu'on lui renseigne un type à chaque variable créée. Par exemple, si on veut créer une variable a, dont est sûr qu'elle est entière et comprise entre 0 et 255, alors on pourra écrire :

```
byte a=20;
```

Si on affecte la valeur 300 à a, on peut observer ce qu'il se passe :

```
void setup() {
Serial.begin(115200);
byte a=300;
Serial.println(a);
}

void loop() {
}
```

3. En fait, c'est une liaison série émulée, via le câble USB, pour tous renseignements concernant la liaison série, se référer à http ://fr.wikipedia.org/wiki/Communication_série

4. Le débit de communication (en bits par seconde) sera proportionnel à 115200. D'autres débits sont acceptés. Pour plus d'information, consulter http ://fr.wikipedia.org/wiki/Baud_(mesure)

Le programme renvoie 44 : il y a eu un dépassement de capacité de la variable. Il faut donc être très vigilant sur le type utilisé. Bien heureusement, on peut stocker dans une variable de très "grandes" valeurs, cependant elles prendront plus de place en mémoire [5]. Les types les plus utilisés sont regroupés dans le tableau suivant :

Type de variable	Intervalle	Commentaires
boolean	*True* ou *False*	Occupe 1 octet en mémoire.
char	Caractère ASCII	Occupe 1 octet en mémoire.
String	mots	Occupe 1 octet/caractère.
byte	[0 ;255]	Représentation partielle de $\mathbb{N}^+$ Occupe 1 octet en mémoire.
int	[-32,768 ;32,767]	Représentation partielle de $\mathbb{N}$ Occupe 2 octets en mémoire.
unsigned int	[0 ;65535]	Représentation partielle de $\mathbb{N}^+$ Occupe 2 octets en mémoire.
long	[-2 147 483 648 ;2 147 483 647]	Représentation partielle de $\mathbb{N}$ Occupe 4 octets en mémoire.
unsigned long	[0 ;4 294 967 295]	Représentation partielle de $\mathbb{N}^+$ Occupe 4 octets en mémoire.
float	$[-3,4028235.10^{38};3,4028235.10^{38}]$	Représentation partielle de $\mathbb{R}$ Occupe 4 octets en mémoire.

4.4 Elements de syntaxe

4.4.1 Boucle *for*

```
for(int i=0;i<=10;i++){
//corps de la boucle
}

for(int i=10;i>=0;i--){
//corps de la boucle
}
```

4.4.2 Boucle *while*

```
int i=10;
while(i>=0){
//corps de la boucle
i--;

}
```

4.4.3 Structure conditionnelle *if*

```
int a=0;
int b=2;
```

5. Il est fortement conseillé de consulter régulièrement la référence arduino située ici :http ://arduino.cc/en/Reference/HomePage

```
if(a==b){
Serial.println("a=b");
}
else if(a>b){
Serial.println("a>b");
}
else{
Serial.println("a<b");
}
```

On remarquera qu'il faut terminer chaque instruction par un " ;".

4.5 Affection des Entrées/Sorties

Les affectations des entrées/sorties sont à placer dans la procédure *setup*. Pour affecter une entrée sur une broche :

```
pinMode(8,INPUT) //Affectation de la broche 8 (logique) en tant qu'entrée.
pinMode(A0,INPUT) //Affectation de la broche A0 (analogique) en tant qu'entrée.
pinMode(8,OUTPUT) //Affectation de la broche 8 (logique) en tant que sortie.
pinMode(9,OUTPUT) //Affectation de la broche 9 (sortie PWM car marquée par le symbole
  // ~ sur la carte) en tant que sortie.
```

4.6 Lire les entrées

Lorsqu'une broche "digitale" ou logique est configurée en entrée, il est possible de récupérer la valeur correspondante en entrée de cette broche via :

```
digitalRead(numero_de_broche)
```

Cette fonction renvoie 1 si la tension en entrée de la broche est supérieur à 3V, et 0 si la tension est inférieure.

Il est possible de lire une tension sur une broche analogique, via un convertisseur analogique/numérique, en appelant la fonction :

```
analogRead(numero_de_broche)
```

La tension lue doit varier entre 0 et 5V. La valeur renvoyée est un entier contenu entre 0 et 1023.

4.7 Imposer une tension en sortie

Sur une broche logique configurée en sortie, on peut imposer soit 0V soit 5V via :

```
digitalWrite(LOW) //impose 5V
digitaleWrite(HIGH)// impose 0V
```

Certaines broches spéciales configurées en sortie, et suivies du symbole ~, peuvent délivrer une tension **moyenne** variable grâce à la fonction :

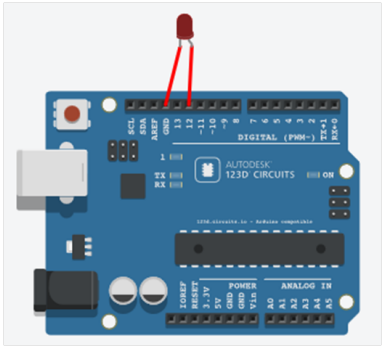
```
analogWrite(val)
```

avec *val* une valeur entière comprise entre 0 et 255. Si val=0, la valeur moyenne vaut 0V, si val=255, la valeur moyenne vaut 5V. La sortie est obtenue par modulation de largeur d'impulsion (MLI ou PWM en anglais). Son étude sera réalisée plus tard.

4.8 Quelques programmes de base

4.8.1 Le "Led Blink"

On souhaite réaliser un programme, associé à un montage, qui fait clignoter une LED à intervalles de temps réguliers. Le circuit est donné ci-contre [6].



Activité 3. Réaliser le montage suivant avec le matériel à disposition, puis proposer un programme qui permet de faire clignoter la LED en permanence toutes les 500 ms (LED allumée pendant 250 ms, LED éteinte pendant 250 ms.) On pourra utiliser la fonction *delay()* (consulter la référence arduino [7]) . Améliorer ensuite le programme pour qu'il affiche sur le terminal l'état de la LED (allumée ou éteinte).

4.8.2 Variation de luminosité d'une Led

On souhaite réaliser un programme, associé à un montage, qui fait varier la luminosité d'une LED, à partir d'un potentiomètre, dont le fonctionnement est donné figure 3.

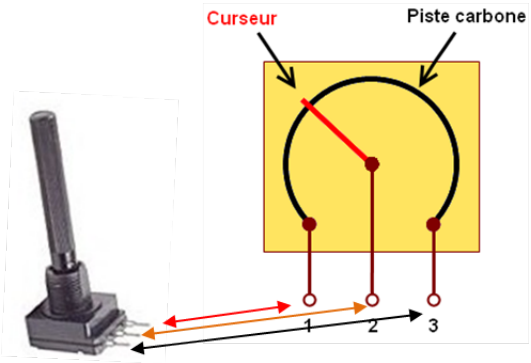
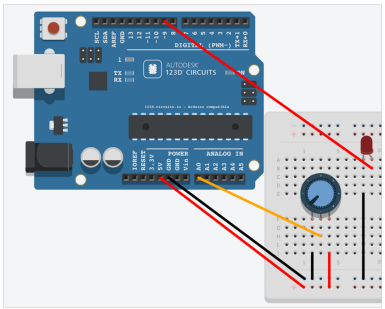


FIGURE 3 – Principe de fonctionnement d'un potentiomètre rotatif



Activité 4. Réaliser le montage suivant avec le matériel à disposition, puis proposer un programme qui affiche la valeur lue en sortie du potentiomètre.

Activité 5. Compléter le programme pour que la luminosité de la led varie avec la rotation du potentiomètre. On rappelle que la fonction *analogRead* renvoie des valeurs comprises entre 0 et 1023, alors que la fonction *analogWrite* ne prend que les valeurs comprises entre 0 et 255 en argument. On pourra utiliser la fonction *map* pour régler le problème.

6. se réfèrer à http ://fr.wikipedia.org/wiki/Diode_%C3%A9lectroluminescente pour la polarité d'une LED
7. ici : http ://arduino.cc/en/Reference/HomePage

5 Mini-projet : Réalisation d'un radar de recul

5.1 Introduction

Les radars de recul pour les véhicules automobiles sont des systèmes d'assistance qui avertissent de la présence d'un obstacle à l'arrière du véhicule, lors d'une opération de marche arrière. Les principaux constituants de ce type de système sont :

- Les capteurs à ultrasons, composés d'émetteurs et de récepteurs
- Le calculateur, qui estime la distance entre l'obstacle et l'arrière du véhicule, à partir des informations délivrées par les capteurs à ultrasons
- un système d'émission sonore qui avertit le conducteur.

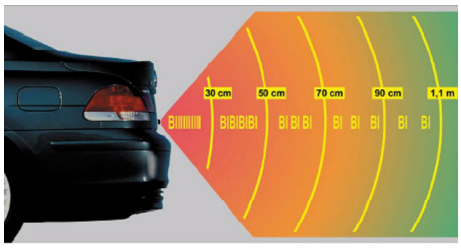


FIGURE 4 – Radar de recul

Habituellement, un bip sonore est émis lorsque la distance obstacle/véhicule devient inférieure à une distance limite, puis, plus l'obstacle se rapproche, plus la fréquence des bips est élevée, jusqu'à devenir une émission sonore continue lorsque le véhicule est très proche de l'obstacle.

5.2 Objectif

L'objectif de ce mini-projet est de reconstituer un radar de recul automobile. Le matériel mis à disposition est le suivant :

- Une carte Arduino-Uno et son câble USB
- Un capteur à ultrason de référence hc-sr04
- Un buzzer
- Une plaque d'expérimentation (ou breadboard)
- Des câbles de type jumper

Le cahier des charges est le suivant :

Le système conçu devra émettre un son, fonction de la distance entre un obstacle et le capteur à ultrasons. Ce son sera constitué d'une succession de bips d'une durée de 30 ms), puis d'un silence d'une durée T. Cette durée T diminue avec la distance. on note d la distance entre le capteur et l'objet et on impose :

- T=800 ms si 40 $cm < d \leq 50$ cm
- T=600 ms si 30 $cm < d \leq 40$ cm
- T=400 ms si 20 $cm < d \leq 30$ cm
- T=200 ms si 10 $cm < d \leq 20$ cm
- T=0 ms si $d \leq 10$ cm

5.2.1 Déroulement du mini-projet

- La description du capteur à ultrason est donnée en annexe : la première chose à faire est de bien comprendre son fonctionnement.
- Réfléchir à un programme qui permette d'afficher, sur le moniteur série, la distance en cm entre l'objet et le capteur (on testera avec une surface "suffisamment" plane (type cahier, feuille, etc ...)).
- Faire un autre programme, distinct du précédent, pour faire fonctionner le buzzer fourni. Il est conseillé d'utiliser la fonction *tone* et *noTone* (voir la référence Arduino).

Attention : une fois le principe de fonctionnement du buzzer compris, arrêter les tests, et ne pas jouer avec ...

— Enfin, réunir les deux codes précédents pour la réalisation finale.

5.2.2 Montage

Afin de sauvegarder le matériel, seul le montage est imposé. Il est fourni sur la figure ci-dessous :

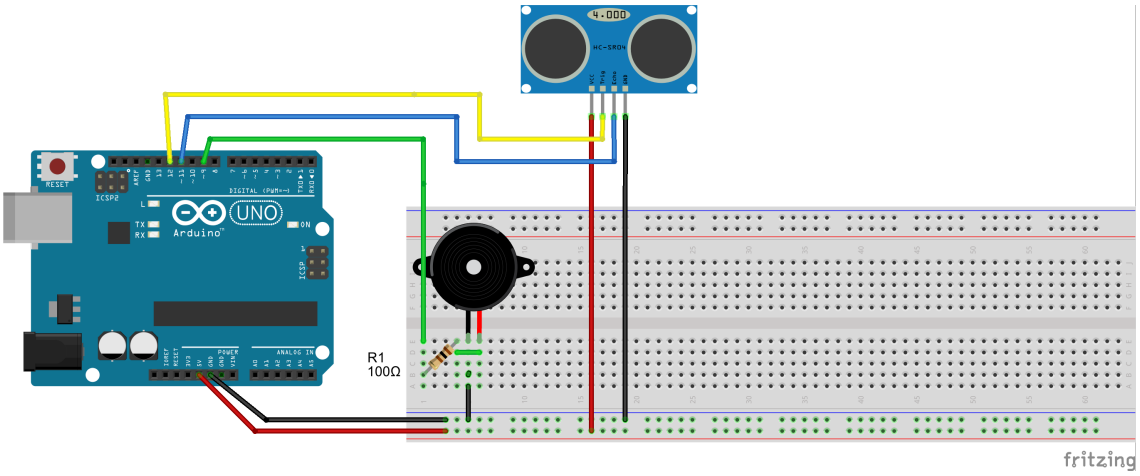


FIGURE 5 – Montage retenu pour le mini-projet

5.2.3 Pour ceux qui sont en avance

Améliorer le système précédent en y ajoutant une LED RGB : la couleur de la LED varie en fonction de la distance mesurée. On cherchera sur internet les informations liées au fonctionnement de ce type de LED, et on utilisera le montage suivant :

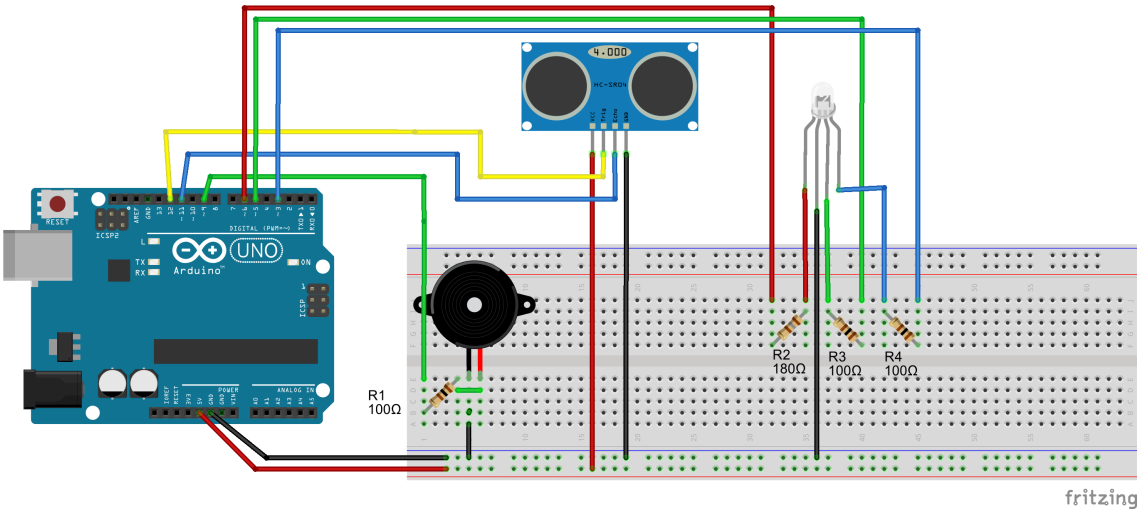


FIGURE 6 – Montage du radar avec LED RGB

Annexe1

A Description du fonctionnement du module Ultrason "HCSR04"

Les caractéristiques techniques du module ultrason sont :

— Tension d'alimentation : 5V CC.
— Consommation en utilisation 15mA.
— Gamme de distance : 2 cm à 5 m.
— Résolution 0,3 cm.
— Angle de mesure $< 15^\circ$.

Il faut envoyer une impulsion niveau haut (à + 5V) pendant au moins $10\mu s$ sur la broche "**Trig Input**". Cela déclenche la mesure. En retour, la sortie "**Echo**" (ou "**Output**") va fournir une impulsion de + 5V dont la durée correspond au temps mis par le son, pour faire un aller et retour, entre le transmetteur (repéré T sur le capteur) et le récepteur (repéré R sur le capteur). La figure suivante montre la séquence de fonctionnement du capteur.

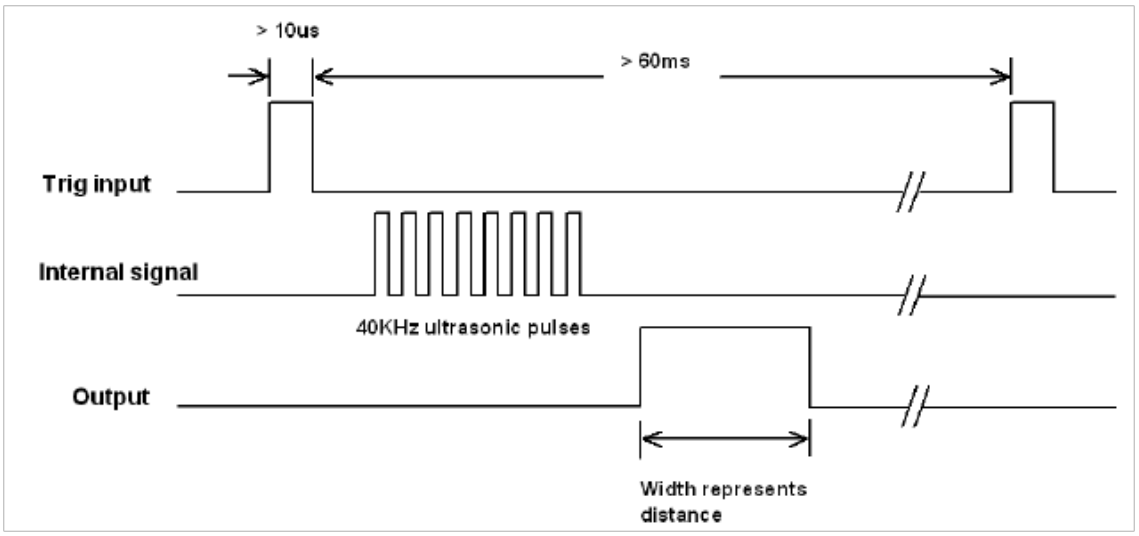


FIGURE 7 – Séquence de fonctionnement du capteur hc-sr04

En langage arduino, il est possible de mesurer la largeur d'une impulsion grâce à la fonction *pulsIn* (consulter la référence).