ARDUINO DU PROGRAMMEUR DEBUTANT AU PROGRAMMEUR AVANCE (CREATEUR DES LIBRAIRIES)

Par : Blaise Letakamba

Déclaration de l'auteur Blaise Letakamba

Aucune phrase se trouvant dans ce livre n'a été copié. C'est le livre est le fruit de mon expérience dans le développement des systèmes embarqués.

Un peu sur l'auteur

Blaise Letakamba ingénieur constructeur des machines et développeur embarqué & Android et IOS œuvrant en contrôle des systèmes et intelligence artificielle. Il est PDG et DT de la startup MALKIAH MACHINAH.

Pour plus de contact : +243 971 828 749

CHAPITRE0. CONSTITUTION DE LA CARTE ARDUINO

INTRODUCTION

Dans cette présente partie nous allons parler sur la constitution de la carte ARDUINO

DEVELOPPEMENT

La carte ARDUINO est constitué par :

1. **Un port USB** de type B permettant la connexion de la carte ARDUINO à l'ordinateur ce via ce port qu'on connecte la carte ARDUINO à l'ordinateur pour le programmer et il sert aussi à l'alimentation 5V de la carte ARDUINO

2. **Un bouton de RESET** le bouton de RESET permet la réinitialisation du déroulement du programme. C'est-à-dire lorsque vous appuyez sur ce bouton le programme recommence au début de son exécution.

3. **Les ports numériques et analogiques** ces derniers permettant la communication avec le monde extérieur

4. **Le microcontrôleur** il s'agit d'un système à microprocesseur auquel on a ajouté les périphériques d'entrées et de sorties c'est le cœur de notre de la carte ARDUINO c'est en son sein que loge le programme de fonctionnement de la carte.

5. **Circuit programmateur** il s'agit du circuit qui permet la programmation du microcontrôleur il permet la conversion de la communication USB en communication série

6. **Le port d'alimentation (7V à 20V)** ce port permet l'alimentation de la carte ARDUINO directement via une alimentation externe on conseille que l'alimentation soit dans une marge de 7V à 20V.

7. **Un circuit d'alimentation régulant la tension à 5V** lorsqu'on alimente la carte ARDUINO avec une tension se trouvant entre 7V et 20V on doit toujours gardé à l'esprit que le microcontrôleur utilise une tension de 5V et ce le rôle de cette circuiterie d'alimentation il permet de réguler la tension à 5V

8. **Le QUARTZ** il permet de cadencer la vitesse d'exécution des instructions

9. **Le connecteur ISCP** il permet la programmation du microcontrôleur de la carte ARDUINO via un programmateur externe.

10. **Des leds**
    Ici on a une led de TX qui s'allume lorsque la carte ARDUINO envoie une information sur la voix série
    On a la led de RX qui s'allume lorsque la carte ARDUINO reçoit une information sur la voix série
    On a la led se trouvant à la borne numérique 13 cette led facilite les essais rapides
    On a la led qui s'allume lorsque la carte ARUINO est branchée.

11. **Des résistances et des condensateurs**
    Les résistances sont placées en différents endroits du montage plus généralement pour limiter le passage du courage dans le circuit et les condensateurs pour le filtrage et pour éliminer les harmoniques.

CHAPITRE1. LA GESTION NUMERIQUES DES PORTS DE LA CARTE ARDUINO

fig.1 CARTE ARDUINO UNO

Une carte arduino est constituée par des broches numériques et des broches analogiques les broches numériques sont numérotés de 0 à 13 les broches analogiques commencent par une lettre A (A0,…, A5) dans cette partie nous allons nous intéresser sur la manipulation des broches numériques de la carte arduino.

Pourquoi ces broches sont appelés numériques : ces broches sont dites numériques car elles ne peuvent prendre ou traiter que deux états 0(signifiant l'absence de la tension : 0V) ou 1 (signifiant la présence de 5V).

Fonction d'usage des broches : ces broches peuvent être utilisées :

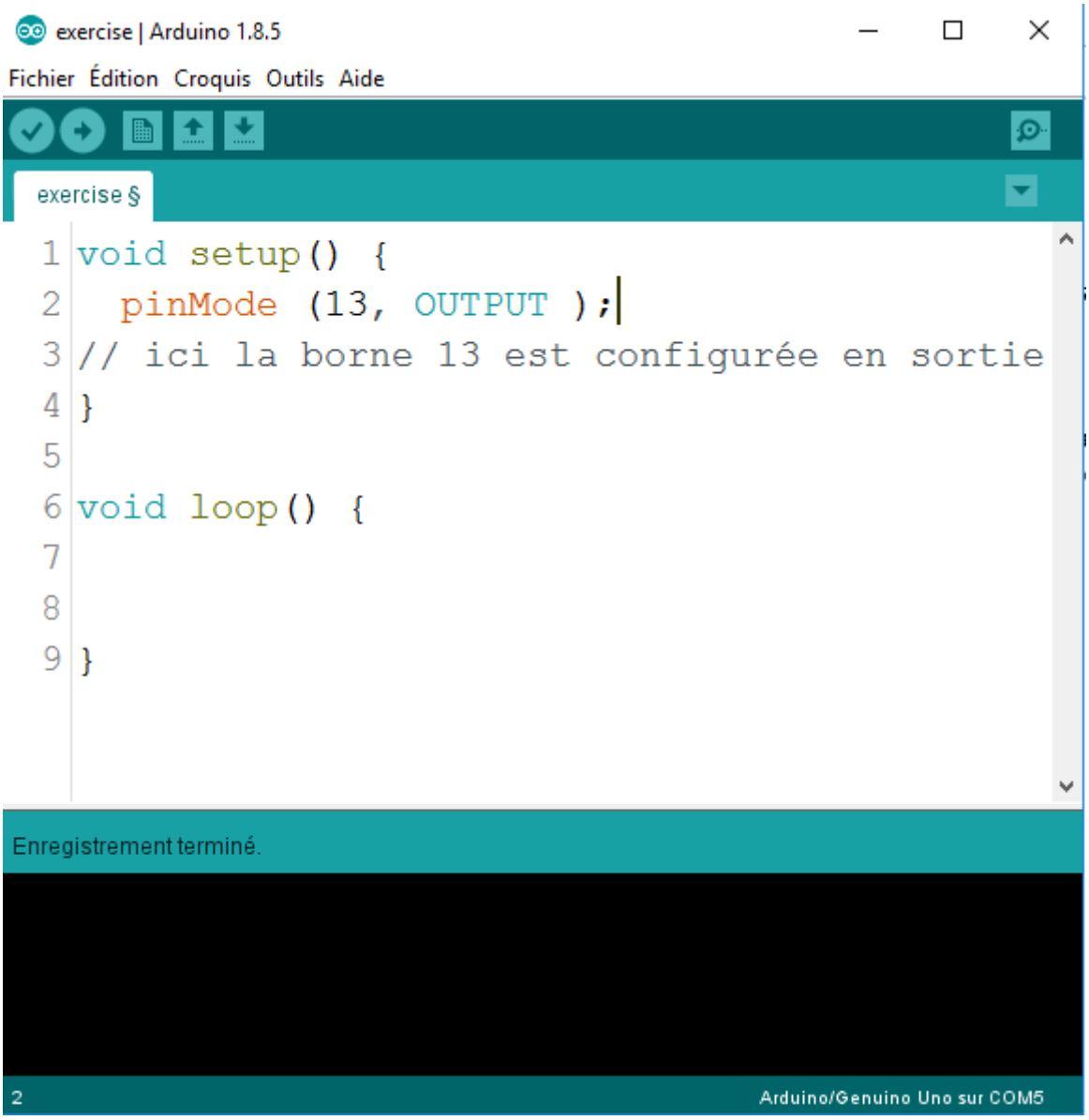
- **En entrée** : ici la carte arduino reçoit les informations du monde extérieur et les traite selon le programme se trouvant en son sein ses informations peuvent être : appui sur un bouton poussoir, impulsion venant d'un détecteur, appui sur un interrupteur, …
- **En sortie :** ici la carte arduino donne les informations au monde extérieur ces informations peuvent être (allumage d'une led se trouvant à une borne quelconque de la carte, commande d'un transistor commandant à son tour un autre dispositif,…)

GESTION NUMERIQUES DE BROCHES DE L'ARDUINO EN SORTIE

Comme dit précédemment dans la gestion numérique de broches de l'arduino en sortie c'est la carte arduino qui donne les informations au monde extérieur (allumage d'une led ou commande d'un transistor).

- Procédure pour faire la gestion numérique de sorties :

1. Configurer la broche que l'on veut utiliser en sortie dans le void setup () : pour cette configuration on utilise la fonction : ***pinMode (nom de la broche, OUTPUT) ;***
   Cette instruction permet de configure la broche en sortie



2. Effectuer l'action que vous voulez sur la broche (ici vous faites passer la broche à 5V (1) ou à 0V(0) selon votre programme) et cette action vous pouvez le faire dans le void loop () ou le void setup () selon le choix pour réaliser cette tache on utilise la fonction :
   digitalWrite (nom de la broche, HIGH) ; dans le cas où on veut que la broche puisse passer à un état logique haut (1) soit 5V ou ***digitalWrite (nom de la broche, LOW)*** dans le cas où on veut que la broche puisse passer à un état logique bas (0) soit 0V.

exercise | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

Nouveau

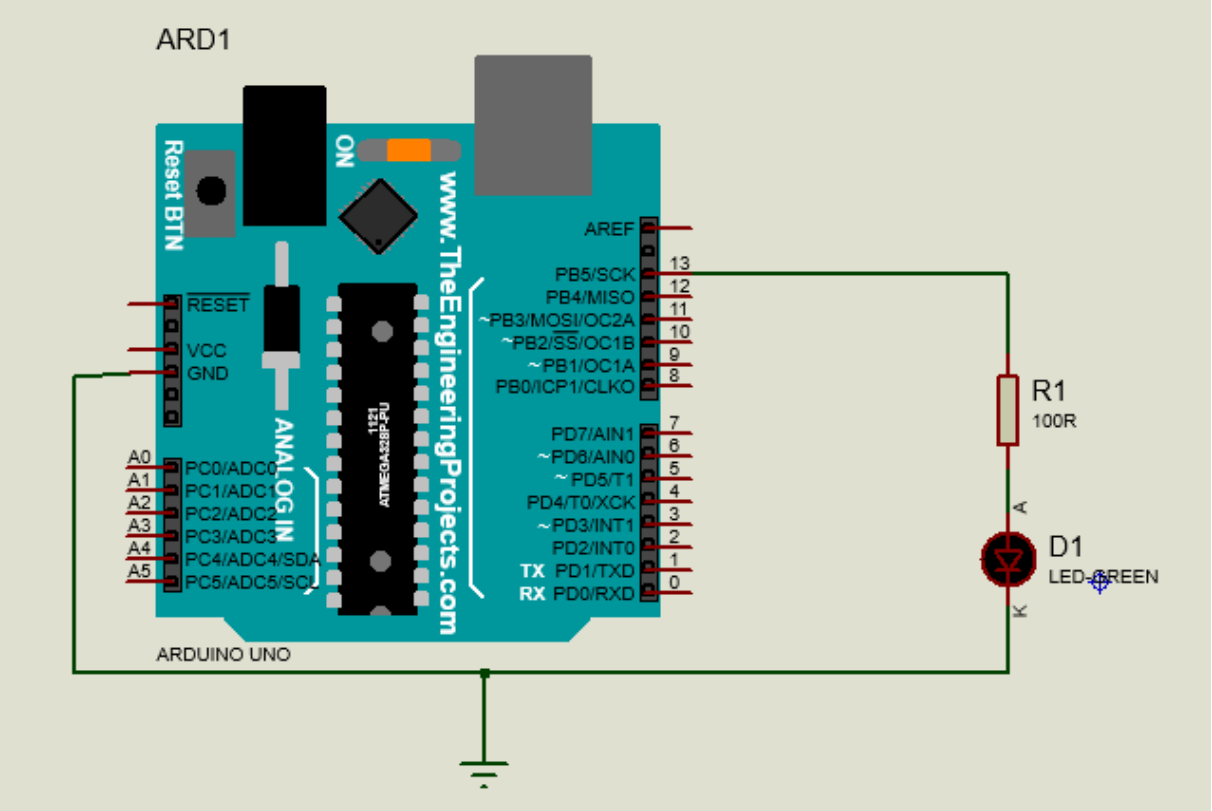
exercise §

```
1  void setup() {
2    pinMode (13, OUTPUT );
3  // ici la borne 13 est configurée en sortie
4  }
5
6  void loop() {
7
8  digitalWrite (13, HIGH);
9  /*ici la borne 13 passe à un etat
10  * haut soit 5V
11  */
12 }
```

Enregistrement terminé.

12 Arduino/Genuino Uno sur COM5

Ce programme met la borne 13 à un état logique haut. Si on connecte une led sur la borne 13 de la carte en série avec une résistance allant (330 Ohm à 2.2k Ohm) on verra la led s'allumer. Soit le schéma suivant :

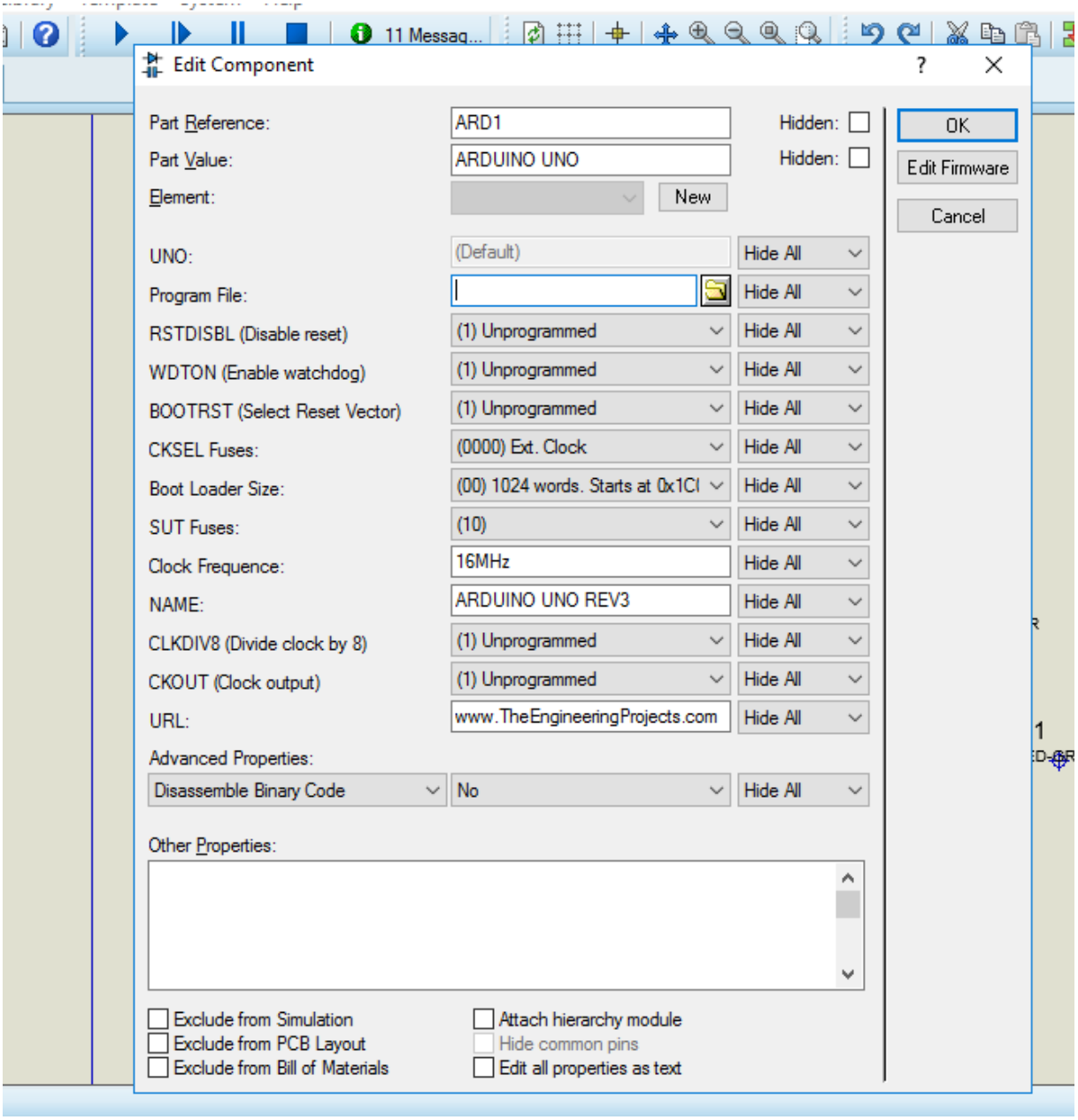


Pour faire la simulation du montage avant de faire le téléversement :
Il faut cliquer sur :

- ✓ Croquis
- ✓ Puis sur exporter les binaires compilées

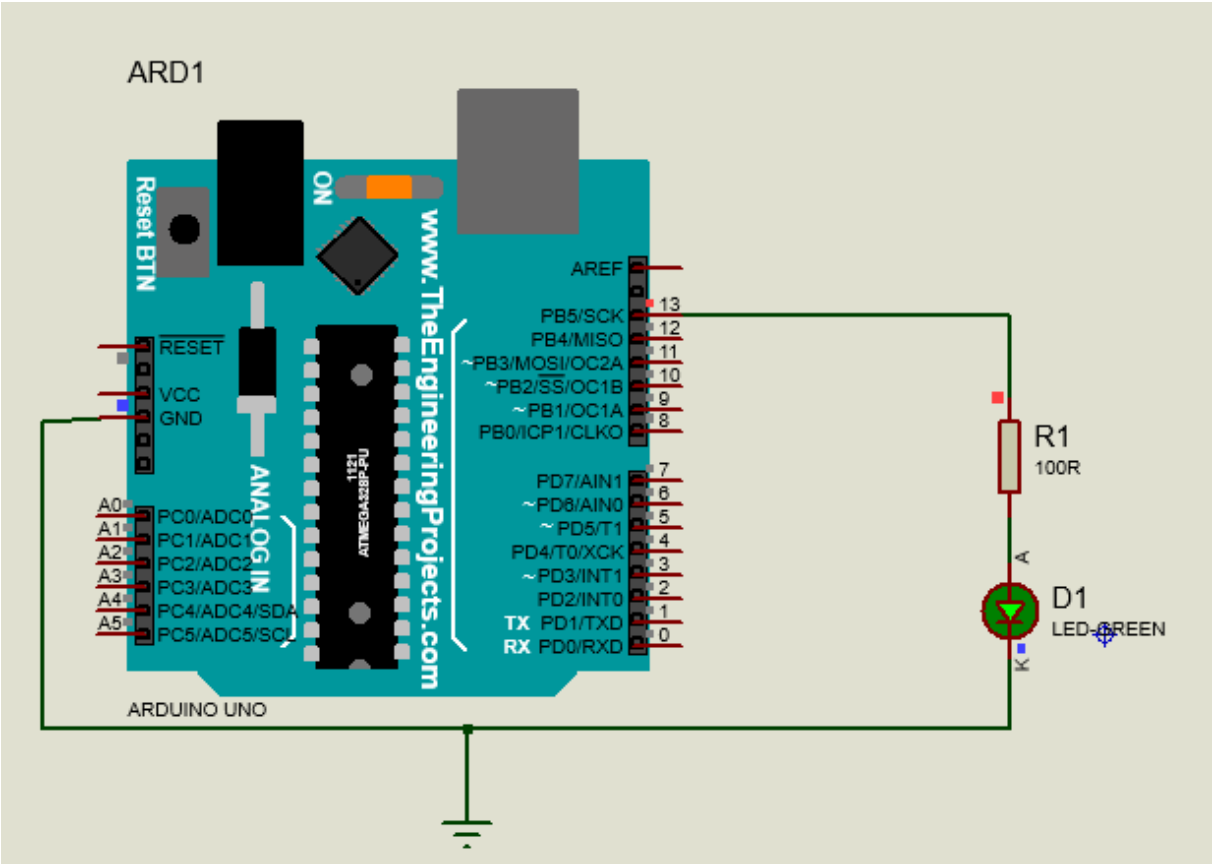
Après cette action deux fichiers d'extension .hex vont se créer dans le dossier où vous avez enregistré votre programme. Alors pour faire la simulation

- ➢ vous allez dessiner le schéma en proteus.
- ➢ Une fois vous avez terminé à dessiner le schéma en proteus faite double clic gauche sur l'image de l'arduino en proteus.
- ➢ Une fois vous faites cette double clic une nouvelle fenêtre apparait



- Puis mettez le chemin d'accès d'un de fichiers .hex obtenus lors l'export des binaires compilés dans le champ Program File
- Puis cliquez sur OK
- Et enfin sur Play dans proteus pour lancer la simulation

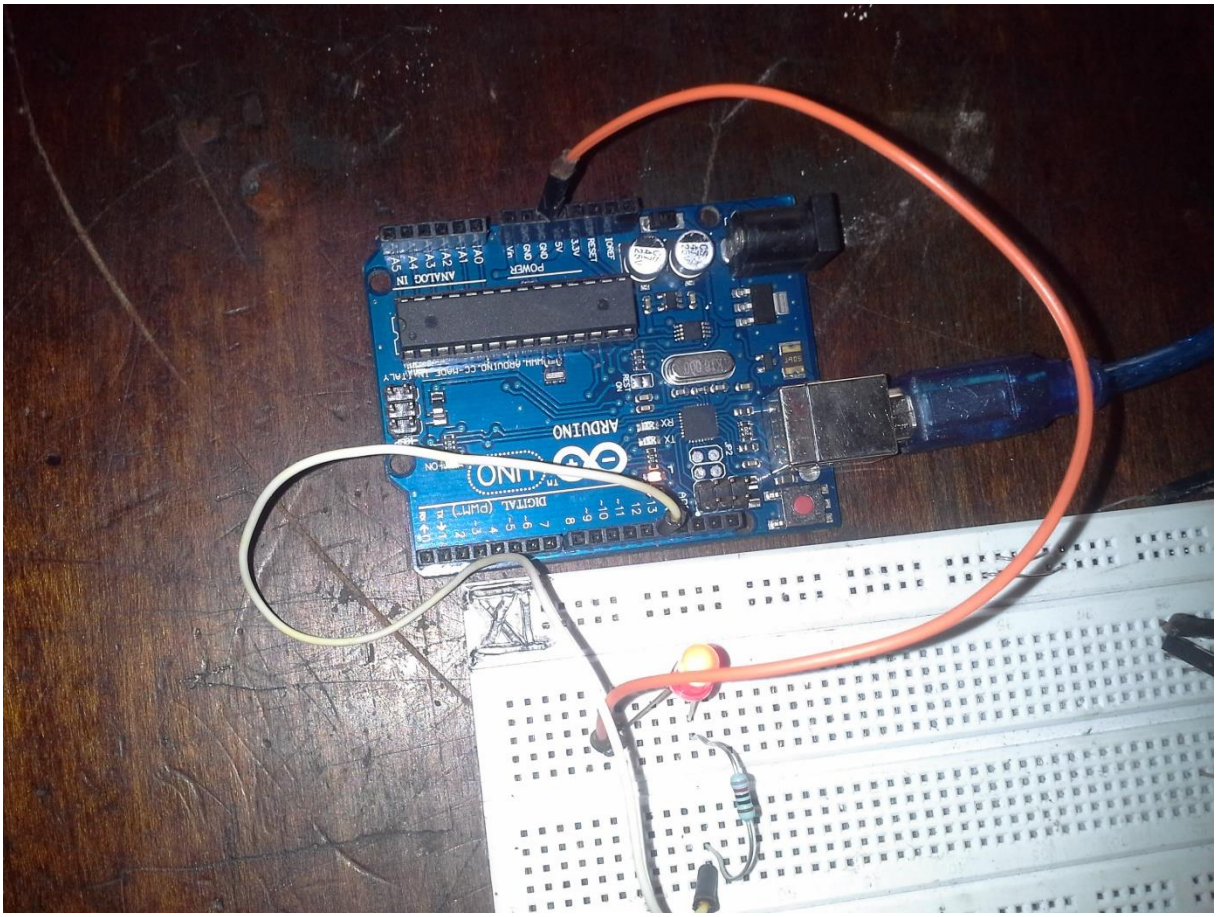
Image de simulation :



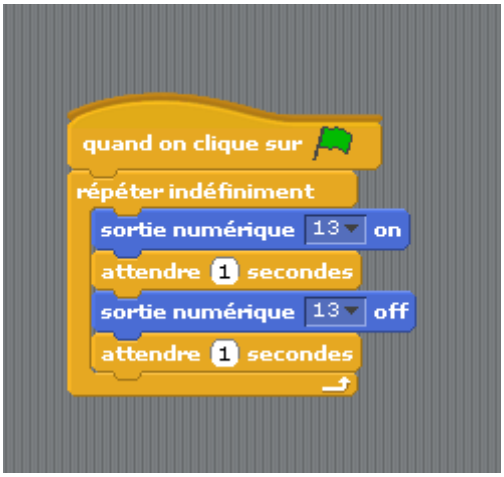
Alors pour envoyer le programme dans la carte arduino :

- Il faut clic sur outils
- Puis type de carte pour indiquer au logiciel le nom de votre carte dans notre cas nous choisirons : Arduino/Genuino UNO
- Puis cliquer sur port et là vous allez indiquer au logiciel là où se trouve votre carte arduino
- Et enfin vous allez cliquer sur la flèche pour envoyer le programme dans la carte

Photo de réalisation :



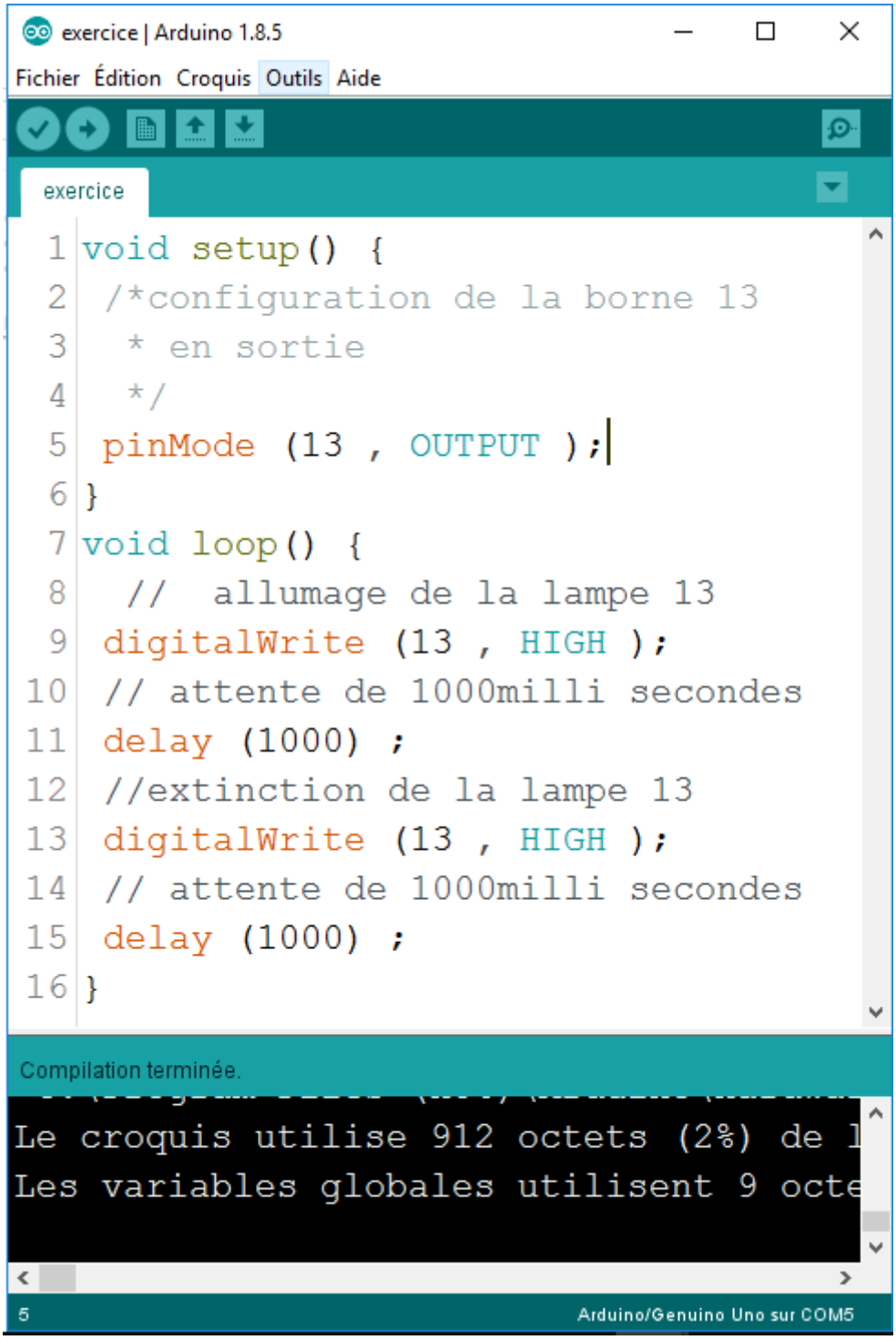
Pour faire clignoter une led se trouvant à la borne 13 on doit donc procéder de la manière :



La fonction qui permet d'attendre pendant un temps c'est :

On a : **delay (nombre de temps en milli secondes) ;**

Code faire clignoter une led se trouvant à la borne 13 :



Les instructions se trouvant dans le void loop () s'exécute de manière infini se pourquoi la led clignotera aussi longtemps que la carte est alimenter. Pour envoyer le programme dans la carte :

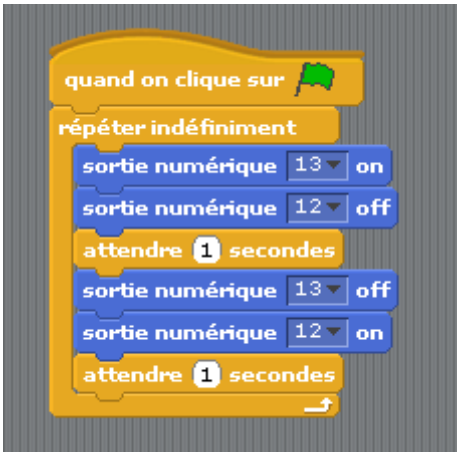
Il faut suivre les procédures énoncé précédemment donc :

- ✓ Il faut cliquer sur outils
- ✓ Puis type de carte dans notre cas Arduino/Genuino Uno
- ✓ Puis préciser le port USB de la machine où se trouve branché la carte ARDUINO
- ✓ Enfin cliquer sur la flèche pour faire le téléversement

AUTRES PROGRAMMES PERMETTANT LA GESTION NUMERIQUE DE SORTIE

Programme permettant d'allumer une led se trouvant à la borne 13 tout en éteignant une autre led se trouvant à la borne 12 pendant une seconde puis fait le contraire donc éteint la 13 en allumant 12 pendant un seconde et cela tant que la carte est alimenté.

Algorithme du programme :



exercice | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

exercice §

```
1  void setup() {
2   /*configuration de la borne 13 et
3    * 12
4    */
5   pinMode (13 , OUTPUT );
6   pinMode (12 , OUTPUT );
7  }
8    // programme principal
9  void loop() {
10  digitalWrite (13 , HIGH );
11  digitalWrite (12 , LOW );
12  delay (1000) ;
13  digitalWrite (13 , LOW );
14  digitalWrite (12 , HIGH );
15  delay (1000) ;
16 }
```

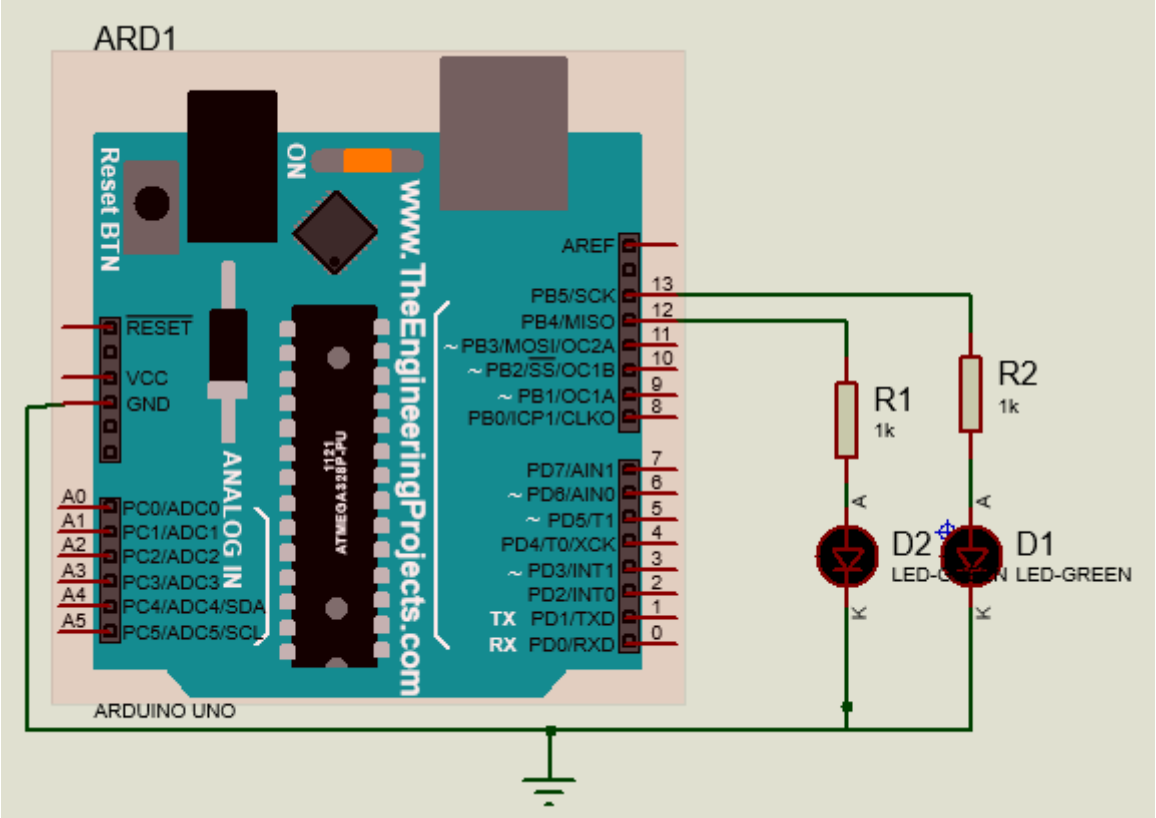
Compilation terminée.

```
Le croquis utilise 912 octets (2%) de l
Les variables globales utilisent 9 octe
```

8 Arduino/Genuino Uno sur COM5

Ce programme permet de faire clignoter se trouvant à la borne :

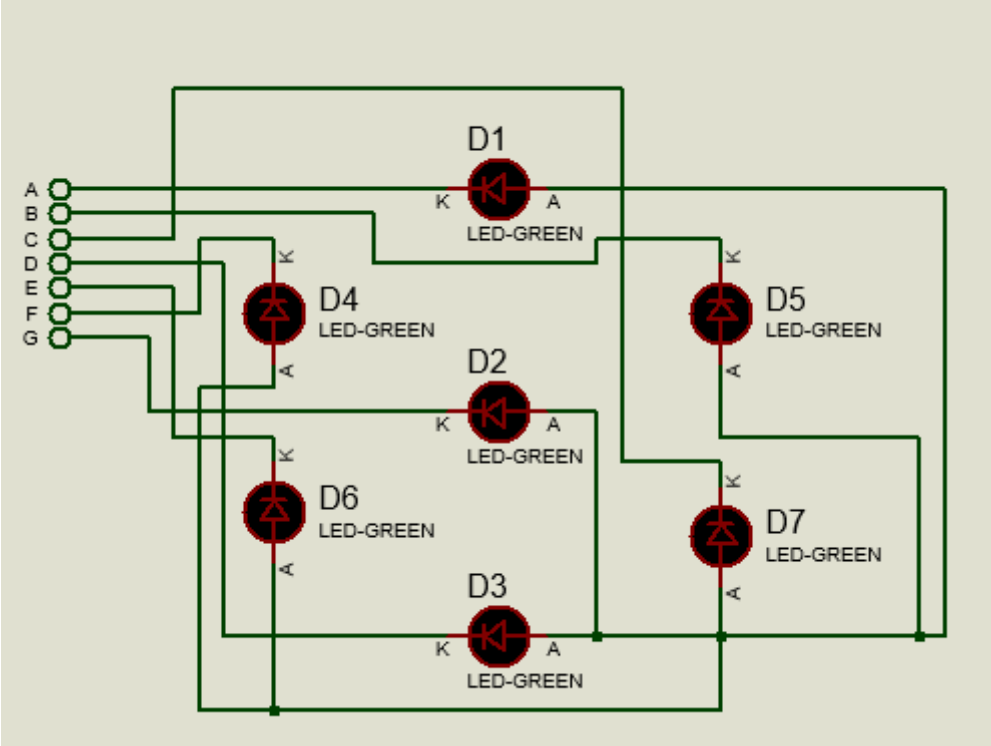
Schémas du système :



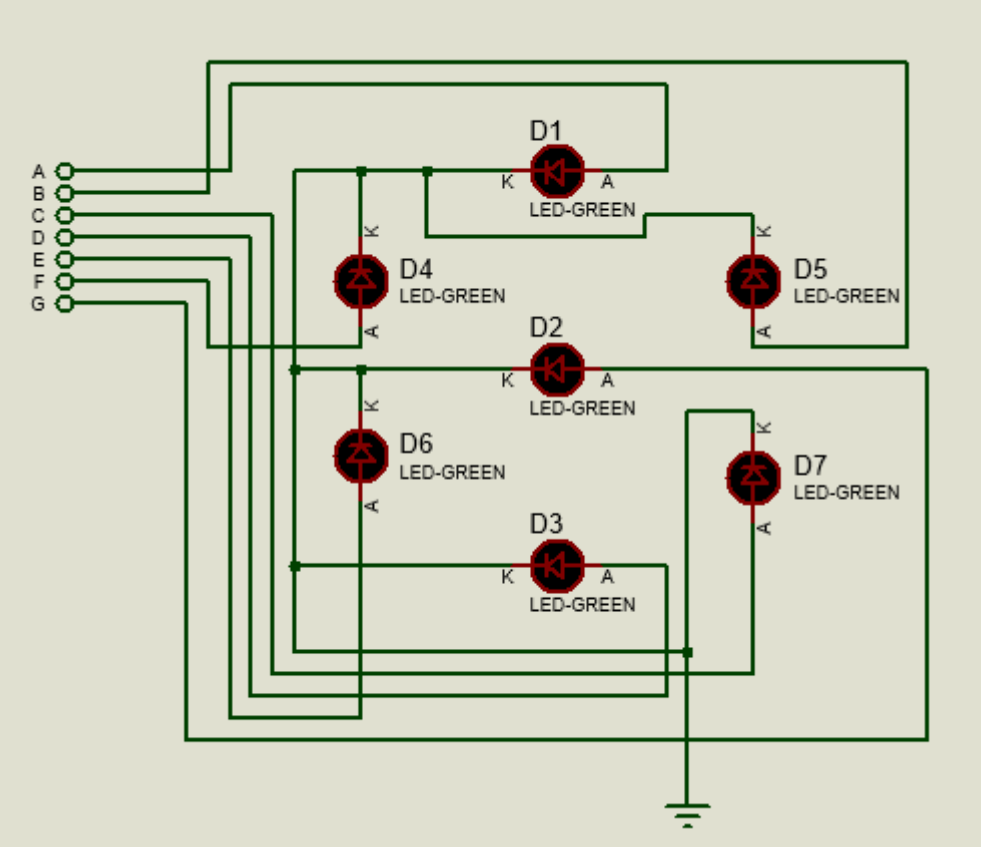
AFFICHEUR 7 SEGMENTS

L'afficheur 7 segments est un des afficheurs le plus utilisé en électronique. Le mot 7 segments vient du fait qu'il est constitué par 7 leds généralement notés par : (A, B, C, D, E, F, G). Cet afficheur peut être à anode ou à cathode commun selon que la borne commune de toutes les leds est l'anode ou la cathode :

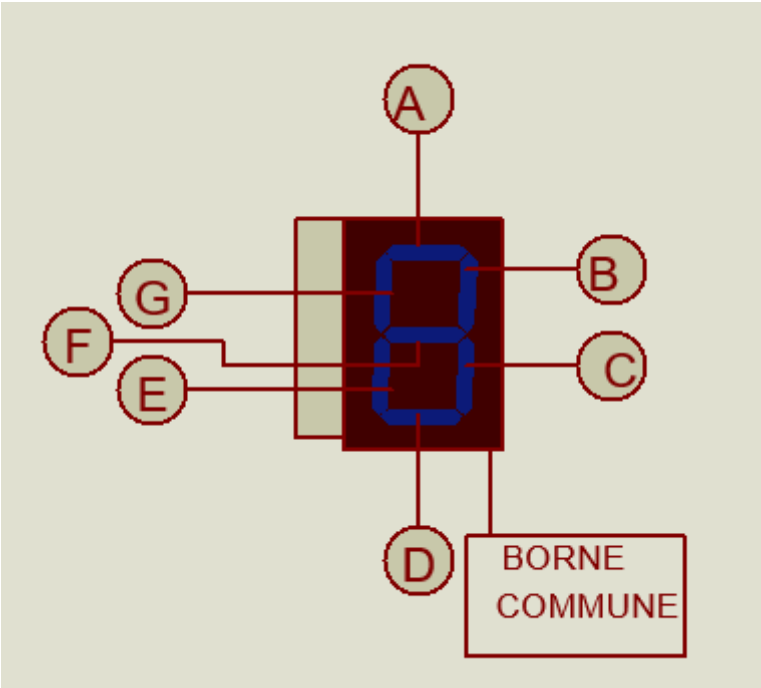
a. STRUCTURE D'UN AFFICHEUR A ANODE COMMUN



b. STRUCTURE D'UN AFFICHEUR A CATHODE COMMUN



Dans les schémas pour éviter beaucoup de fils on le représente par :



Ici la borne commune sera la masse ou le +Vcc (5V) selon que l'afficheur est anode ou cathode commun.

Dans la suite nous utiliserons l'afficheur cathode commun donc la borne commune sera à la masse

Nous allons faire un compteur qui compte de 0 à 3 secondes

Algorithme pour écrire sur un afficheur 7 segments :

- Pour écrire 1 : il faut allumer les leds : A, B
- Pour écrire 2 : il faut allumer les leds : A, B, C, E, G
- Pour écrire 3 : il faut allumer les leds : A, B, C, D, G
- Pour écrire 4 : il faut allumer les leds : F, B, C
- Pour écrire 5 : il faut allumer les leds : A, F, G, C, D
- Pour écrire 6 : il faut allumer les leds : A, F, E, D, C, G
- Pour écrire 7 : il faut allumer les leds : A, B, C
- Pour écrire 8 : il faut allumer les leds : A, B, C, D, E, F, G
- Pour écrire 9 : il faut allumer les leds : A, B, C, D, F, G

Programme permettant de compter de 0 à 3 secondes :

_seg | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

_seg §

```
/
 *  programme permettant
 *  de compte avec un afficheur 7 segments
 *  de 0 à 3 secondes.
 */

void setup() {
 pinMode (13, OUTPUT); // vers la led A
 pinMode (12, OUTPUT); //vers la led B
 pinMode (11, OUTPUT); // vers la led C
 pinMode (10, OUTPUT); // vers la led D
 pinMode (9, OUTPUT); // vers la led E
 pinMode (8, OUTPUT); // vers la led F
 pinMode (7, OUTPUT);  // vers la led G
}

void loop() {

  // allumage de 0

  digitalWrite (13, HIGH); //allumage de la led  A
  digitalWrite (12, HIGH); //allumage de la led B
  digitalWrite (11, HIGH); //allumage de la led C
  digitalWrite (10, HIGH); //allumage de la led D
  digitalWrite (9, HIGH); //allumage de la led  E
  digitalWrite (8, HIGH); //allumage de la led F
  digitalWrite (7, LOW); //extinction  de la led  G
```

```
28
29    delay (1000); // attente d'une seconde
30
31     // allumage de 1
32
33  digitalWrite (13, LOW);   // extinction de la led A
34   digitalWrite (12, HIGH); // allummage de la led B
35   digitalWrite (11, HIGH); // allummage de la led C
36   digitalWrite (10, LOW);  // extinction de la led D
37   digitalWrite (9, LOW);   // extinction de la led E
38   digitalWrite (8, LOW);   // extinction de la led F
39   digitalWrite (7, LOW);   // extinction de la led G
40
41   delay (1000); // attente d'une seconde
42
43   digitalWrite (11, LOW); // extinction de la led C
44
45   digitalWrite (12, HIGH); // allummage de la led B
46   digitalWrite (13, HIGH); //allumage de la led  A
47   digitalWrite (10, HIGH); //allumage de la led D
48   digitalWrite (9, HIGH);  //allumage de la led  E
49   digitalWrite (7, HIGH); //allumage de la led  G
50
51   delay (1000); // attente d'une seconde
52
53    digitalWrite (9 , LOW); // extinction de la led E
```

```
54    digitalWrite (11, HIGH); // allumage de la led C
55
56     delay (1000); // attente d'une seconde
57
58 }
```

Schéma du système :

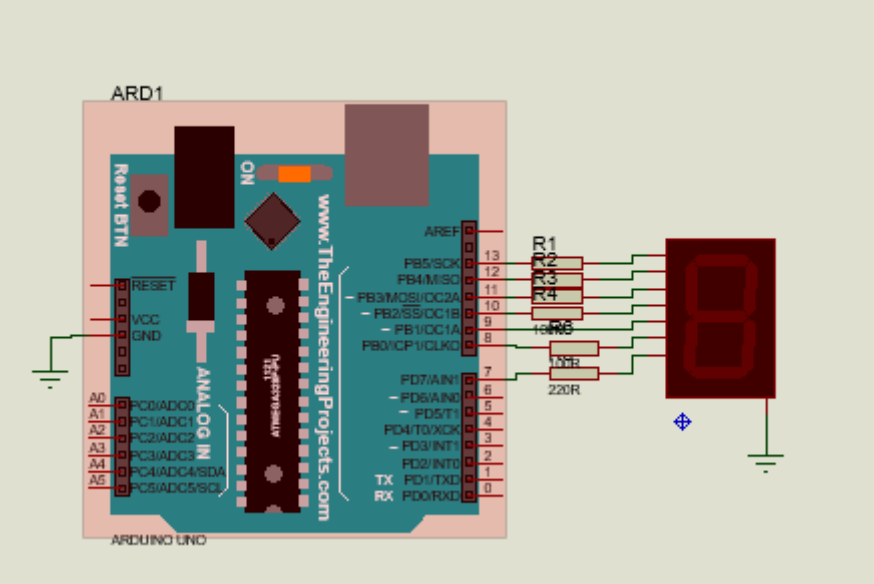


Image de simulation lors de l’affichage de zéro :

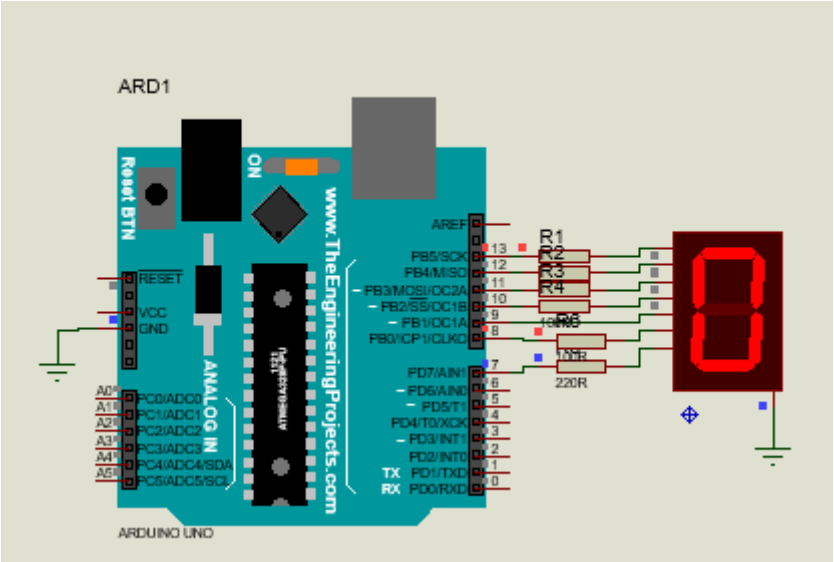


Image de simulation lors de l’affichage de Un :

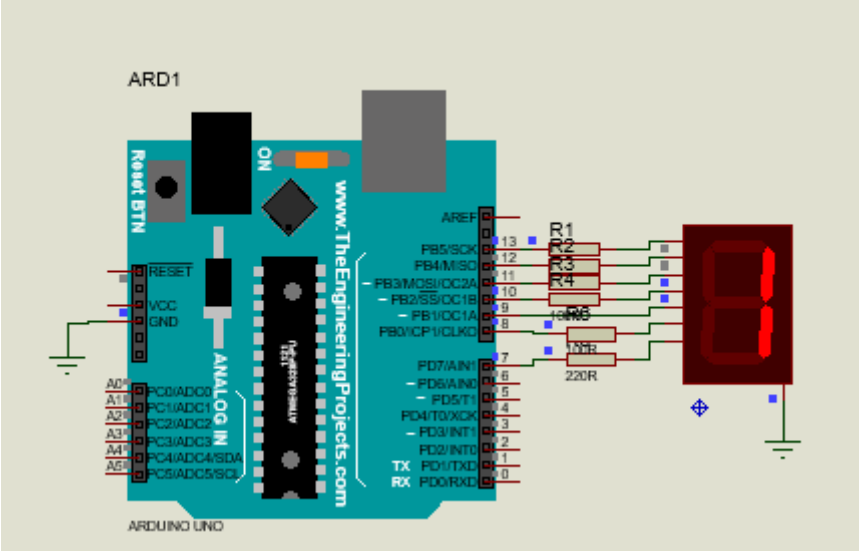


Schéma de simulation lors de l’affichage de deux :

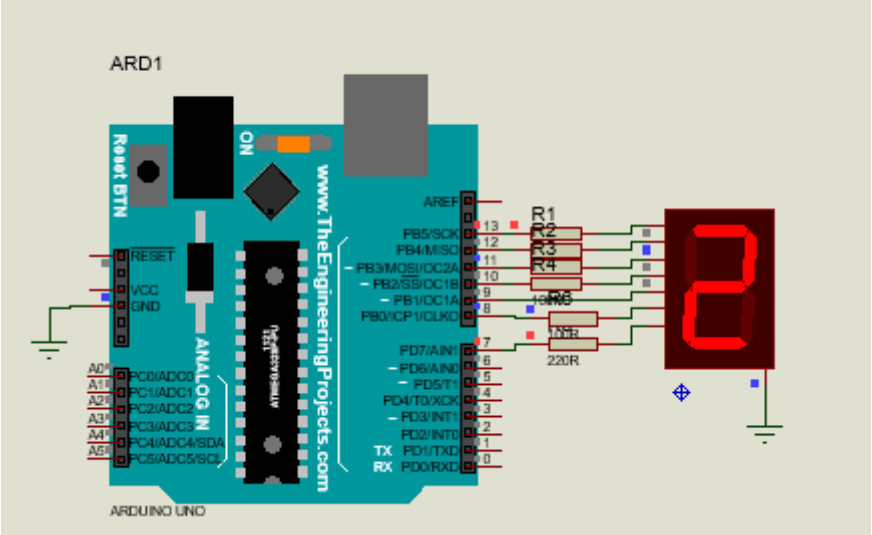
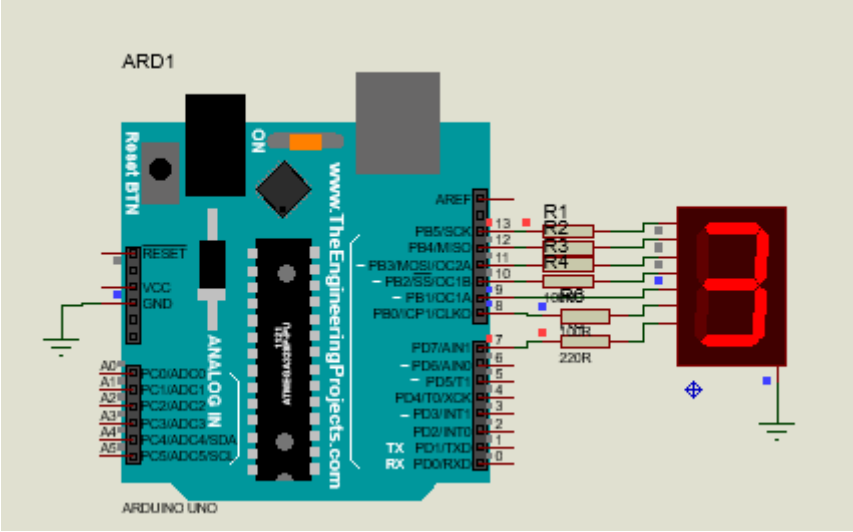


Image de simulation lors de l'affichage de 3 :



LES MOTS CLES DEFINE

Quand on écrit **digitalWrite (13, HIGH)** et qu' à la borne 13 il est connecté une led cette dernière s'allume mais remarque : si une personne lue notre code ne saura pas si la borne 13 commande une led si nous ne la disons pas en plus si on fait un programme où il y a beaucoup de leds verte, rouge et orange en programmant avec de nombre on peut facilement oublié quelle est la broche qui commande la led verte ou rouge , la led jaune ou orange pour ce faire on utilise les defines

Exemple d'un programme faisant clignoter une lampe avec le define

define | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

define §

```
1
2  #define ledJaune 13
3   /*
4 *  partout dans le programme
5 *  ou on ecrira ledJaune cela sera la meme
6 *  que si on ecrivez 13 car on definie dans
7  *  le programme que ledJaune = 13
8  */
9 void setup() {
10   pinMode (ledJaune ,OUTPUT);
11
12 }
13
14 void loop() {
15   digitalWrite (ledJaune , HIGH);
16   delay (1000);
17   digitalWrite (ledJaune , HIGH);
18   delay (1000);
19 }
```

Enregistrement terminé.

8 Arduino/Genuino Uno sur COM5

Ce programme permet de faire clignoter une ledJaune se trouvant à la borne 13. Remarqué que ce programme est plus parlant que le précédent grâce à l'utilisation de define.

GESTION NUMERIQUES DE BROCHES DE L'ARDUINO EN SORTIE PAR ADRESSAGE AUX REGISTRES

Cette partie intéresse les électroniciens si vous ne l'été pas ne vous sentez pas obliger de lire cette partie.

Le microcontrôleur se trouvant sur la carte ARDUINO UNO est un atmega 328p c'est un microcontrôleur il est constitué par des registres dans cette partie nous allons nous allons voir comment configurer l'arduino en utilisant ses registres.

Les registres que nous utiliserons ce sont les registres : DDRB, DDRC, DDRD, PORTA, PORTB et PORTC

Les registres DDRB, DDRC, DDRD permettent de faire la configuration. Les registres PORTA, PORTB et PORTC permettent de faire le manipulation. L'utilisation de ces registres se fait sur 8 bit

CONFIGURATION DE BROCHES EN SORTIE

Pour faire la configuration des broches en sortie nous utiliserons

sketch_sep24b | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

Ouvrir

sketch_sep24b §

```
1  #include <avr/io.h>
2  #include <util/delay.h>
3
4  int main (void)
5  {
6     /* ces trois instructions suivant
7      *  font la configuration des broches
8      *  de l'arduino en sortie
9      */
10      DDRB = 255 ;
11      DDRC = 255 ;
12      DDRD = 255 ;
13 }
14
```

UTILISATION DE BROCHES

Comme exercice faisons un programme qui fait clignoter une led se trouvant à la borne 13 :

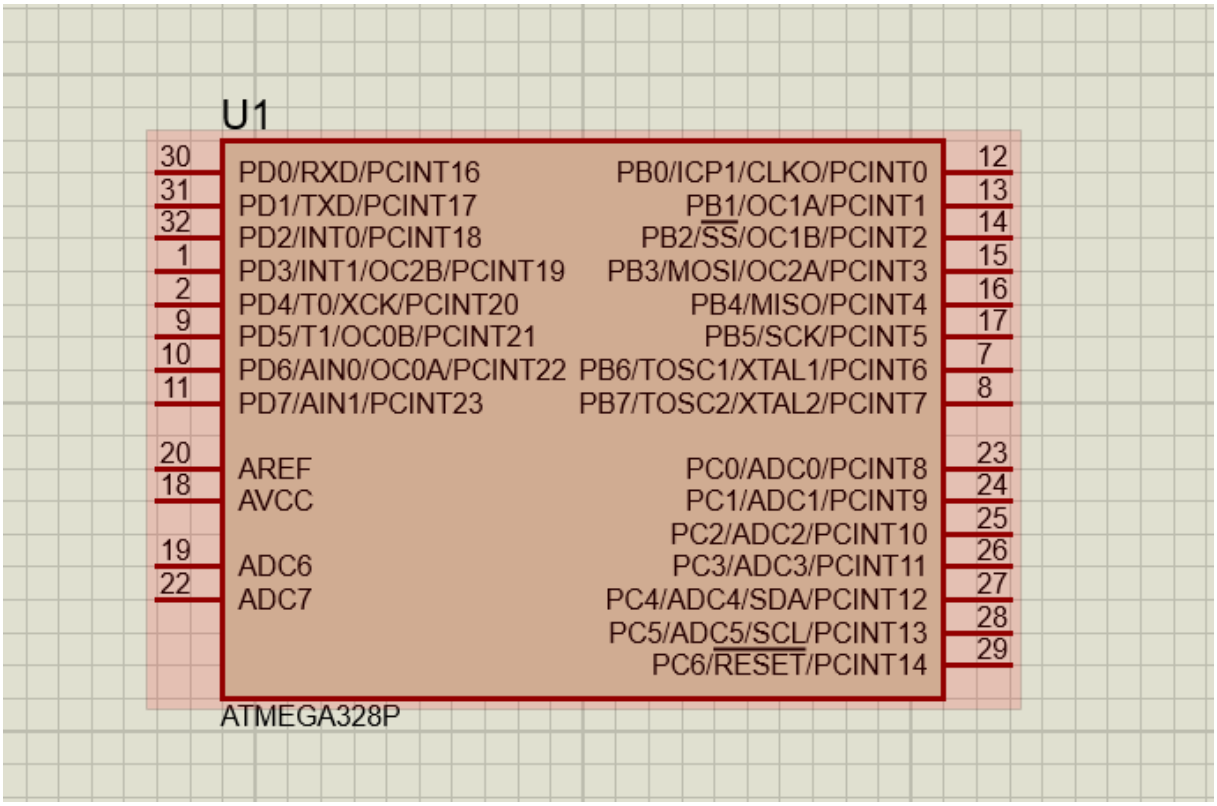
sketch_sep24b | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

sketch_sep24b

```
#include <avr/io.h>
#include <util/delay.h>

int main (void)
{
   /* ces trois instructions suivant
    *  font la configuration des broches
    *  de l'arduino en sortie
    */
     DDRB = 255 ;
     DDRC = 255 ;
     DDRD = 255 ;

      while (1) {
            PORTB = 0b00100000;
            _delay_ms (1000);
             PORTB = 0b00000000;
              _delay_ms (1000);


      }
```

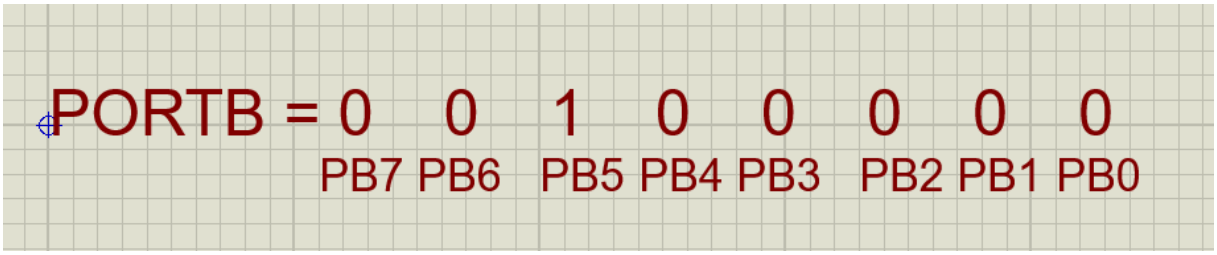
Téléversement terminé



En voyant sur cette figure vous voyez que le PORTB :

Va de PB0 à PB7 dans le cas de l'arduino on va de PB0 à PB5 car PB6 et PB7 sont réservés au quartz

Quand on écrit PORTB = 0b00100000 ;



- **1 : Pour dire de mettre à haut (lui donner 5V) cette broche**
- **0 : Pour dire de mettre à bas (lui donner 0) cette broche**

Cette technique est beaucoup plus utiliser dans de programme où on veut minimiser la capacité (la taille en octet) du programme.

GESTION NUMERIQUES DE BROCHES DE L'ARDUINO EN ENTREE

Ici c'est l'arduino qui reçoit les informations venant du monde extérieur (appui sur un bouton, signal venant d'un détecteur 0-5V,…).

- **Procédure pour faire la gestion numérique des broches de l'arduino en entrée :**

- ✓ Configurer la broche que l'on veut utiliser en entrée dans le void setup () l'instruction qui permet de faire cela est : **pinMode (numéro de la broche, INPUT).** Par cette instruction la broche configurée peut être utilisé comme entrée numérique. On comprend bien que dans le fonctionnement normal la broche peut être à 1 ou à 0
- ✓ Tester l'état de la broche et agir selon ce dernier la fonction qui permet de faire cela est : **digitalRead (numéro de la broche)** cette fonction analyse l'état de la broche et on peut agir selon que la broche est à 1 ou à 0 donc dans le programme on doit tester en utilisant les conditions de programmation (if { } else { })

EXEMPLE D'UN PROGRAMME PERMETTANT DE FAIRE CLIGNOTER UNE LAMPE SE TROUVANT A LA BORNE 13 QUAND ON APPUI SUR UN BOUTON SE TROUVANT A LA BORNE 8

testBOUTON | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

testBOUTON

```
1
2 void setup() {
3  pinMode (13 ,OUTPUT);
4   pinMode (8 , INPUT);
5 }
6
7 void loop() {
8    if (digitalRead (8) == LOW)
9    {
10     while (1)
11      {
12         digitalWrite (13 , HIGH);
13         delay (1000);
14         digitalWrite (13 , LOW );
15         delay (1000);
16      }
17
18   }
19
20 }
```

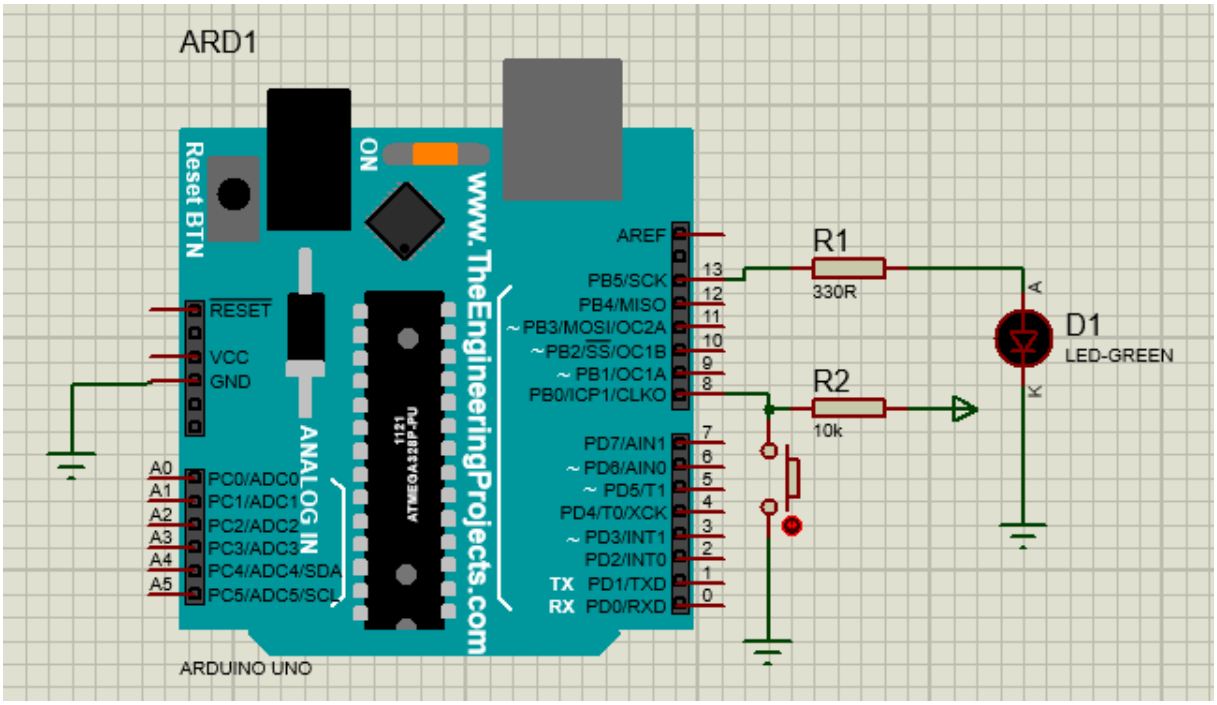
Enregistrement terminé.

14 Arduino/Genuino Uno sur COM5

Comme on veut allumer une lampe à la borne 13 on la configure en sortie et un bouton à la borne 8 qu'on configure en entrée. Grace à ce programme si on branche une led sur la borne 13 on la verra clignoter tant que la carte est alimentée si la borne 8 a été mise à la masse.

```
if (digitalRead (8) == LOW)
{

  /*
   *  ces instructions s'executent
   *  si la borne 8 est à 0
   */
   }


if (digitalRead (8) == HIGH)
{

  /*
   *  ces instructions s'executent
   *  si la borne 8 est à 1
   */
   }
```

L'instruction == c'est l'instruction de comparaison.



Ce schéma permet montre la réalisation du système

Remarque : sur le schéma la résistance R2 est une résistance de PULLUP permettant de fixer sur la broche 8 un état logique haut (1) ce pourquoi dans le programme on teste si la borne 8 passe à (0). Cette résistance R2 permet d'attribuer un certain état à la broche de l'arduino

si vous ne le faites pas la borne 8 va prendre des valeurs aléatoires soit 0 soit 1 qui peut nuire au fonctionnement du programme.

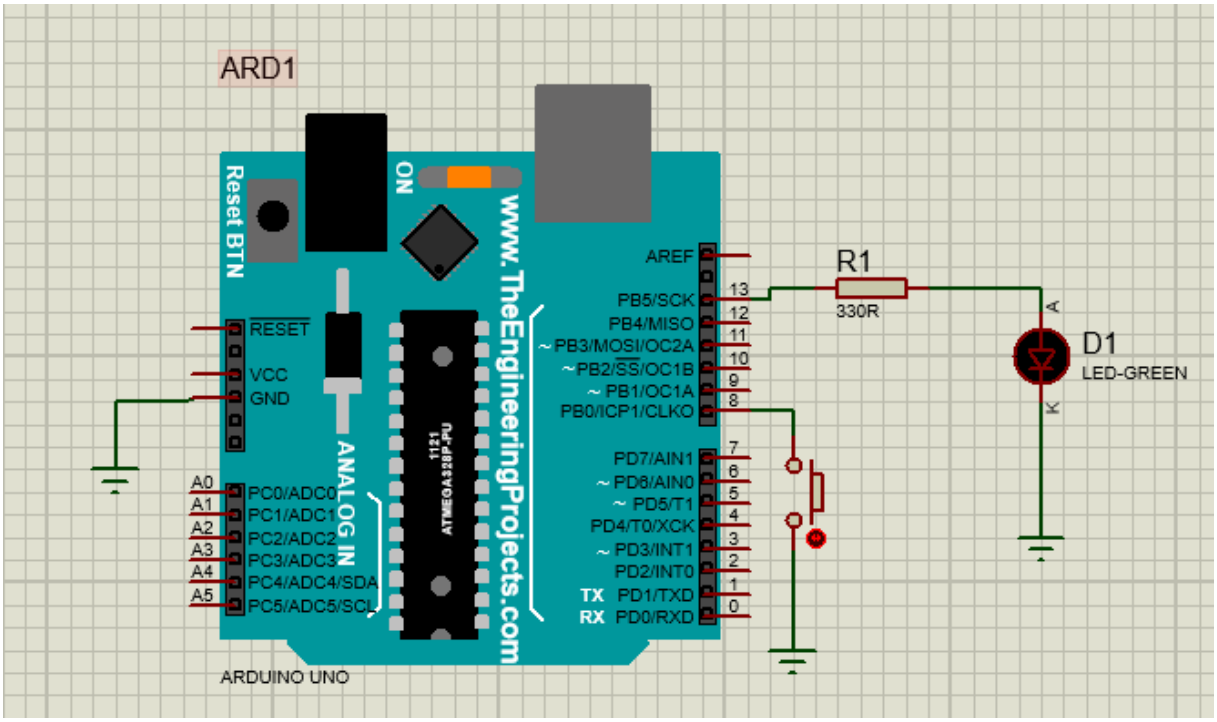
COMMENT S'EN PASSER DE LA REISTANCE PHYSIQUE DE PULLUP

Pour se passer de cette résistance on utilise une fonction pour faire la configuration :

pinMode (numéro de la broche, INPUT_PULLUP)

Exemple d'un code utilisant la fonction si dessus :

L'avantage ici et que dans le schéma la résistance de PULLUP est enlevé du système et cela réduit le nombre de composant à rassembler et moins de câblage pour faire la réalisation :



Code source

testBOUTON | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

testBOUTON §

```
1
2  void setup() {
3   pinMode (13 ,OUTPUT);
4    pinMode (8 , INPUT_PULLUP);
5  }
6
7  void loop() {
8
9     if (digitalRead (8) == LOW)
10    {
11
12      /*
13       *  ces instructions s'executent
14       *  si la borne 8 est à 1
15       */
16       }
17
18    }
19
20 }
```

Enregistrement terminé.

9 Arduino/Genuino Uno sur COM5

LES OPERATEURS LOGIQUES

Symbole	Opérateur
&&	Opérateur ET logique
||	Opérateur OU logique
!	Opérateur NON logique

Ces opérateurs sont importants lorsqu'on veut utiliser plusieurs boutons poussoirs

Exemple programme permettant de faire clignoter une led connecté à la borne 13 lorsqu'on appui sur deux boutons l'un se trouvant sur la borne 8 par rapport à la masse et l'autre se trouvant à la borne 9 par rapport à la masse à la fois :

testBOUTON | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

testBOUTON §

```
1
2  void setup() {
3    pinMode (13 ,OUTPUT);
4    pinMode (8 , INPUT_PULLUP);
5    pinMode (9 , INPUT_PULLUP);
6
7
8  }
9
10 void loop() {
11  if (digitalRead (8) == LOW && digitalRead (9) == LOW )
12    {
13      while (1)
14       {
15          digitalWrite (13 , HIGH);
16          delay (1000);
17          digitalWrite (13 , LOW );
18          delay (1000);
19       }
20
21    }
```

11 Arduino/Genuino Uno sur COM5

Exemple programme permettant de faire l'action si on appuie sur un bouton ou sur l'autre :

testBOUTON | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

testBOUTON §

```
1
2  void setup() {
3    pinMode (13 ,OUTPUT);
4    pinMode (8 , INPUT_PULLUP);
5    pinMode (9 , INPUT_PULLUP);
6
7
8  }
9
10 void loop() {
11  if (digitalRead (8) == LOW || digitalRead (9) == LOW )
12    {
13      while (1)
14       {
15          digitalWrite (13 , HIGH);
16          delay (1000);
17          digitalWrite (13 , LOW );
18          delay (1000);
19       }
20
21    }
```

11 Arduino/Genuino Uno sur COM5

Ce programme fait clignoter une led si on appuis pas sur l'un ou l'autre bouton poussoir :

testBOUTON | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

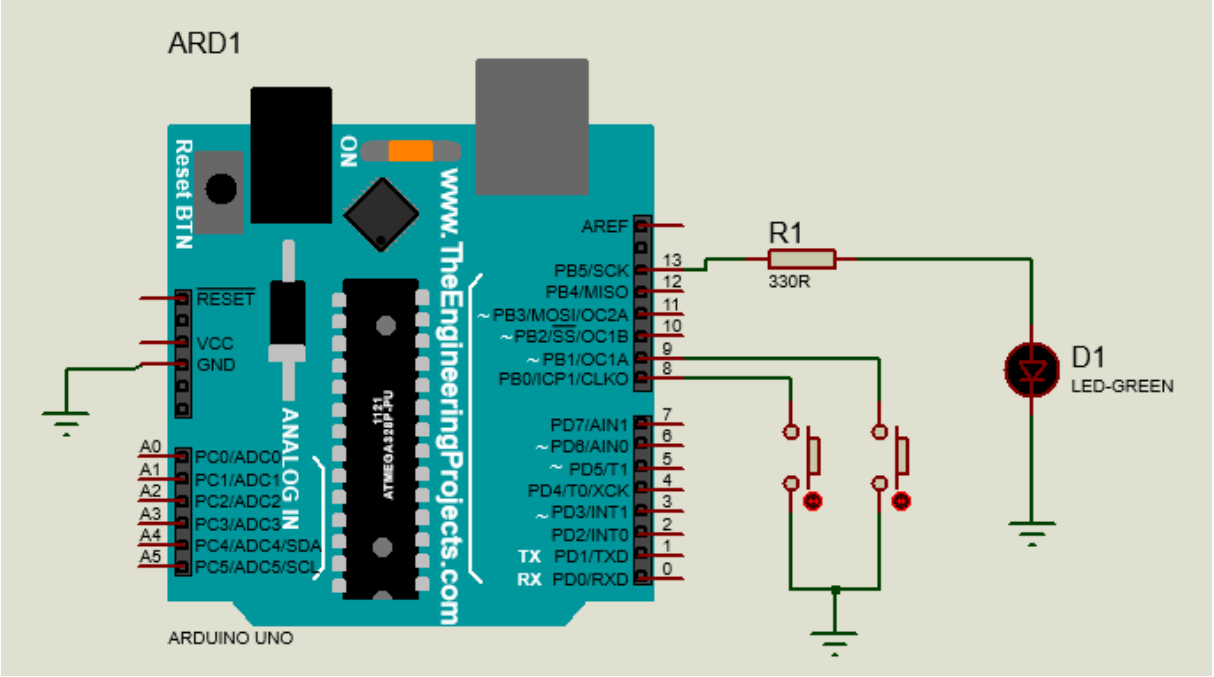
Nouveau

testBOUTON §

```
1
2  void setup() {
3    pinMode (13 ,OUTPUT);
4    pinMode (8 , INPUT_PULLUP);
5    pinMode (9 , INPUT_PULLUP);
6
7
8  }
9
10 void loop() {
11  if (!digitalRead (8) == LOW  )
12    {
13      while (1)
14       {
15          digitalWrite (13 , HIGH);
16          delay (1000);
17          digitalWrite (13 , LOW );
18          delay (1000);
19       }
20
21    }
```

11 Arduino/Genuino Uno sur COM5

SCHEMA AVEC UTILISATION DES DEUX BOUTONS



GESTION ENTREE DES BROCHES DE L'ARDUINO PAR ADRESSAGE AUX REGISTRES :

exercice | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

exercice

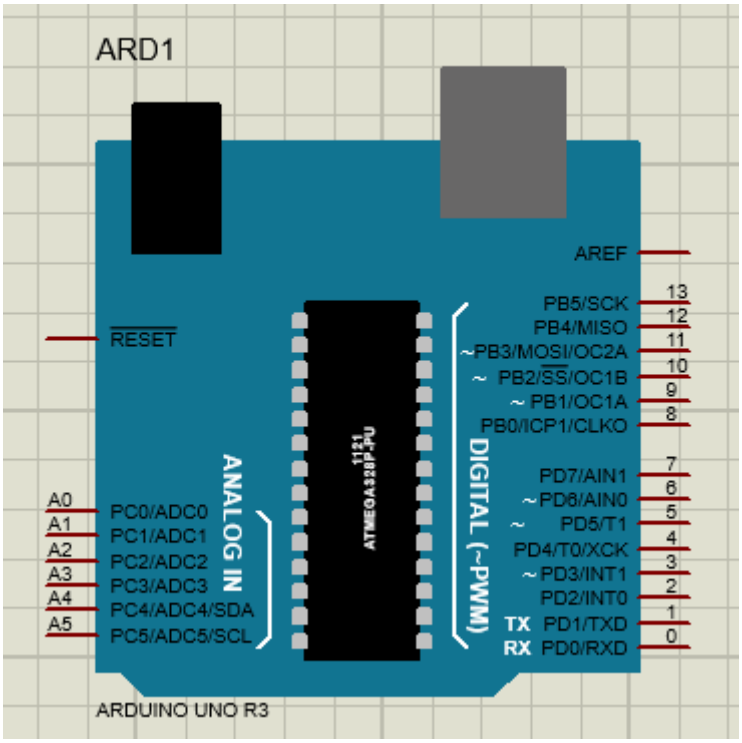
```
1
2  #include <avr/io.h>
3  #include <util/delay.h>
4 int main (void)
5 {
6    DDRB = 0xFF ; // configuration du port B en sortie
7    DDRD &= ~0x08; // configuration de PD3  en entree
8
9
10  if ((PIND&0x08)!= 0)  // teste si le bouton passe à 1
11    {
12    while (1) {
13        PORTB = 1 ;  // activation de la broche PB0 à 1
14        delay (1000); // delay de 1 seconde
15        PORTB = 0 ;  // mise à 0 du port B
16        delay (1000); // delay de 1 seconde
17              }
18           }
19       }
20
21
```

CHAPITRE2. GESTION NUMERIQUE DES PORTS DE LA CARTE ARDUINO EN UTILISANT LE REGISTRE

1. INTRODUCTION

Dans cette partie nous allons présenter comment faire le traitement des broches de la carte ARDUINO en utilisant l'adressage de registre. Cette programmation est important car il facile certaines choses il est optimal et permet d'être proche de l'architecture interne du microcontrôleur.

2. GESTION DE BROCHE EN SORTIE



Sur la figure ci-dessus on voit très que chaque broche porte un nom bien déterminé. Pour faire la gestion des broches de la carte ARDUINO en sortie il faut avoir la connaissance principalement de deux registres dont :

- ✓ **Le registre DDRx** : ce registre permet de faire la configuration des broches en entrée ou en sortie.
- ✓ **Et le registre PORTx :** ce registre permet d'affecter l'état à une broche donné du microcontrôleur dans notre cas c'est juste 5V (1) ou OV (0).

Pour faire la gestion de broches de la carte ARDUINO en sortie on a juste deux étapes :

1. **LA CONFIGURATION**

   On a trois registre de configuration pour la gestion en numérique des broches de la carte ARDUINO.

   DDRB

DDRC
DDRD
Ce registre vos permettre la configuration

Pour configurer toutes les broches du port D en sortie on va écrire :

DDRD = 0b11111111 ;
Ici on configure toutes les broches allant de 0 à 7 de la carte ARDUINO en sortie

DDRD = 0b$X_7X_6X_5X_4X_3X_2X_1X_0$

Mettre :
X_7 : Si on met ce bit à 1 on configure la broche 7 de la carte ARDUINO en sortie et si on met ce bit à 0 on configure la broche en entré.

X_6 : Si on met ce bit à 1 on configure la broche 6 de la carte ARDUINO en sortie et si on met ce bit à 0 on configure la broche en entré.

X_5 : Si on met ce bit à 1 on configure la broche 5 de la carte ARDUINO en sortie et si on met ce bit à 0 on configure la broche en entré.

X_4 : Si on met ce bit à 1 on configure la broche 4 de la carte ARDUINO en sortie et si on met ce bit à 0 on configure la broche en entré.

X_3 : Si on met ce bit à 1 on configure la broche 3 de la carte ARDUINO en sortie et si on met ce bit à 0 on configure la broche en entré.

X_2 : Si on met ce bit à 1 on configure la broche 2 de la carte ARDUINO en sortie et si on met ce bit à 0 on configure la broche en entré.

X_1 : Si on met ce bit à 1 on configure la broche 1 de la carte ARDUINO en sortie et si on met ce bit à 0 on configure la broche en entré.

X_0 : Si on met ce bit à 1 on configure la broche 0 de la carte ARDUINO en sortie et si on met ce bit à 0 on configure la broche en entré.

PORTD = 0b$X_7X_6X_5X_4X_3X_2X_1X_0$

X_7 : Si on met ce bit à 1 l'état la broche 7 de la carte ARDUINO passe à 1 et si on met ce bit à 0 l'état la broche passe à 0.

X_6 : Si on met ce bit à 1 l'état la broche 6 de la carte ARDUINO passe à 1 et si on met ce bit à 0 l'état la broche passe à 0.

X_5 : Si on met ce bit à 1 l'état la broche 5 de la carte ARDUINO passe à 1 et si on met ce bit à 0 l'état la broche passe à 0.

X_4 : Si on met ce bit à 1 l'état la broche 4 de la carte ARDUINO passe à 1 et si on met ce bit à 0 l'état la broche passe à 0.

X_3 : Si on met ce bit à 1 l'état la broche 3 de la carte ARDUINO passe à 1 et si on met ce bit à 0 l'état la broche passe à 0.

X_2 : Si on met ce bit à 1 l'état la broche 2 de la carte ARDUINO passe à 1 et si on met ce bit à 0 l'état la broche passe à 0.

X_1 : Si on met ce bit à 1 l'état la broche 1 de la carte ARDUINO passe à 1 et si on met ce bit à 0 l'état la broche passe à 0.

X_0 : Si on met ce bit à 1 l'état la broche 0 de la carte ARDUINO passe à 1 et si on met ce bit à 0 l'état la broche passe à 0.

DDRB = 0b$\boldsymbol{X_7X_6X_5X_4X_3X_2X_1X_0}$

Mettre :

$\boldsymbol{X_7X_6}$ **:** sont non utilisés c'est ici ou on connecte le quartz on pourra voir comment les utiliser en utilisant l'oscillateur interne.

X_5 : Si on met ce bit à 1 on configure la broche 13 de la carte ARDUINO en sortie et si on met ce bit à 0 on configure la broche en entré.

X_4 : Si on met ce bit à 1 on configure la broche 12 de la carte ARDUINO en sortie et si on met ce bit à 0 on configure la broche en entré.

X_3 : Si on met ce bit à 1 on configure la broche 11 de la carte ARDUINO en sortie et si on met ce bit à 0 on configure la broche en entré.

X_2 : Si on met ce bit à 1 on configure la broche 10 de la carte ARDUINO en sortie et si on met ce bit à 0 on configure la broche en entré.

X_1 : Si on met ce bit à 1 on configure la broche 9 de la carte ARDUINO en sortie et si on met ce bit à 0 on configure la broche en entré.

X_0 : Si on met ce bit à 1 on configure la broche 8 de la carte ARDUINO en sortie et si on met ce bit à 0 on configure la broche en entré.

PORTB = 0b$\boldsymbol{X_7X_6X_5X_4X_3X_2X_1X_0}$

$\boldsymbol{X_7X_6}$: Ces broches sont non utilisé on montrera comment utilisé ces broches en utilisant l'oscillateur interne

X_5 : Si on met ce bit à 1 l'état la broche 13 de la carte ARDUINO passe à 1 et si on met ce bit à 0 l'état la broche passe à 0.

X_4 : Si on met ce bit à 1 l'état la broche 12 de la carte ARDUINO passe à 1 et si on met ce bit à 0 l'état la broche passe à 0.

X_3 : Si on met ce bit à 1 l'état la broche 11 de la carte ARDUINO passe à 1 et si on met ce bit à 0 l'état la broche passe à 0.

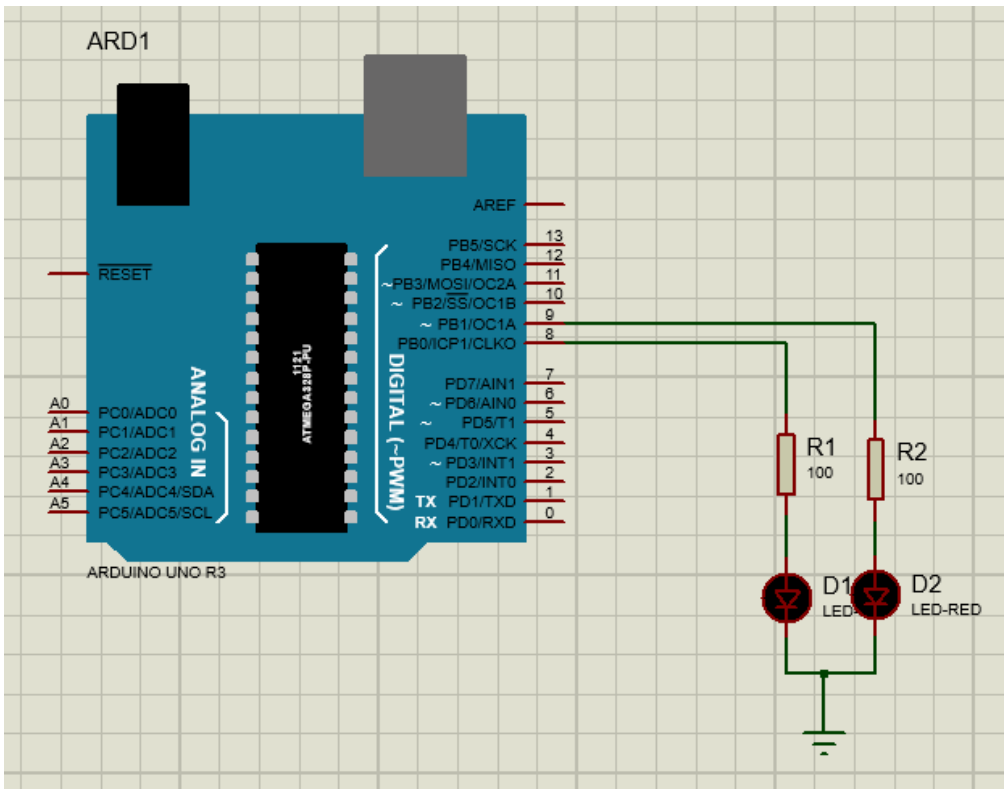
X_2 : Si on met ce bit à 1 l'état la broche 10 de la carte ARDUINO passe à 1 et si on met ce bit à 0 l'état la broche passe à 0.

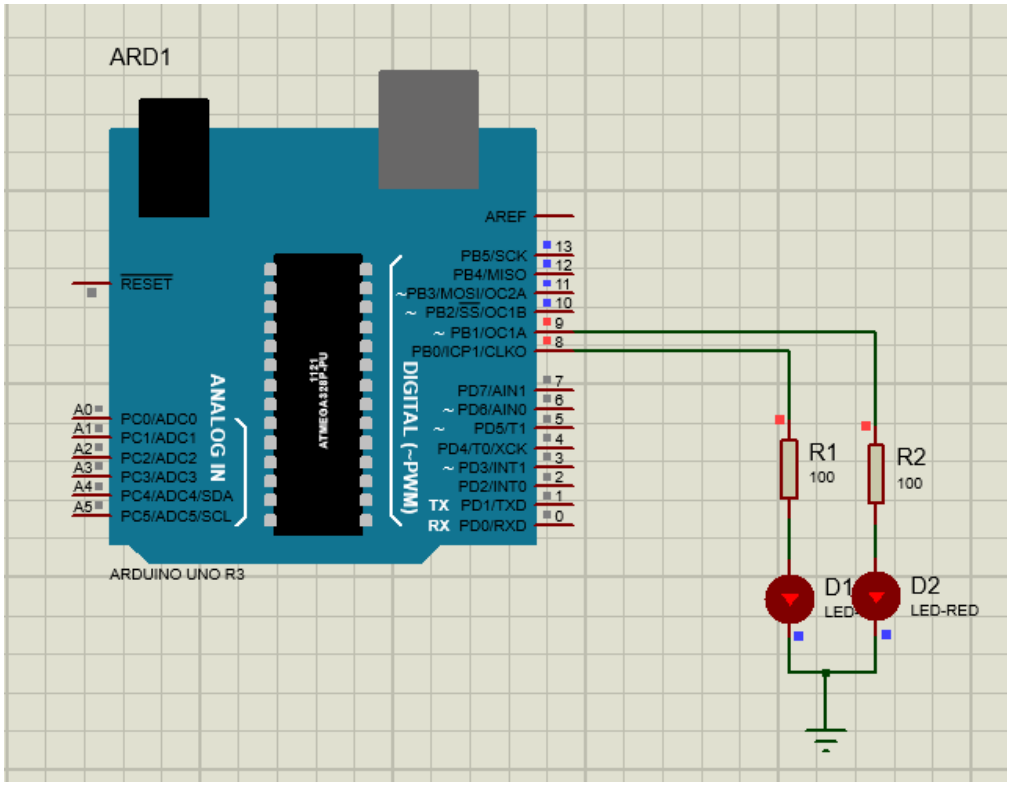
X_1 : Si on met ce bit à 1 l'état la broche 9 de la carte ARDUINO passe à 1 et si on met ce bit à 0 l'état la broche passe à 0.

X_0 : Si on met ce bit à 1 l'état la broche 8 de la carte ARDUINO passe à 1 et si on met ce bit à 0 l'état la broche passe à 0.

<u>EXEMPLE DE CODE AVEC REGISTRE CLIGNOTEMENT DE DEUX LEDS</u>

```
void setup() {

   DDRB = 0b11111111 ;

}

void loop() {

  PORTB = 0b00000011 ;
  delay (1000) ;
  PORTB = 0b00000000;
  delay (1000);
}
```

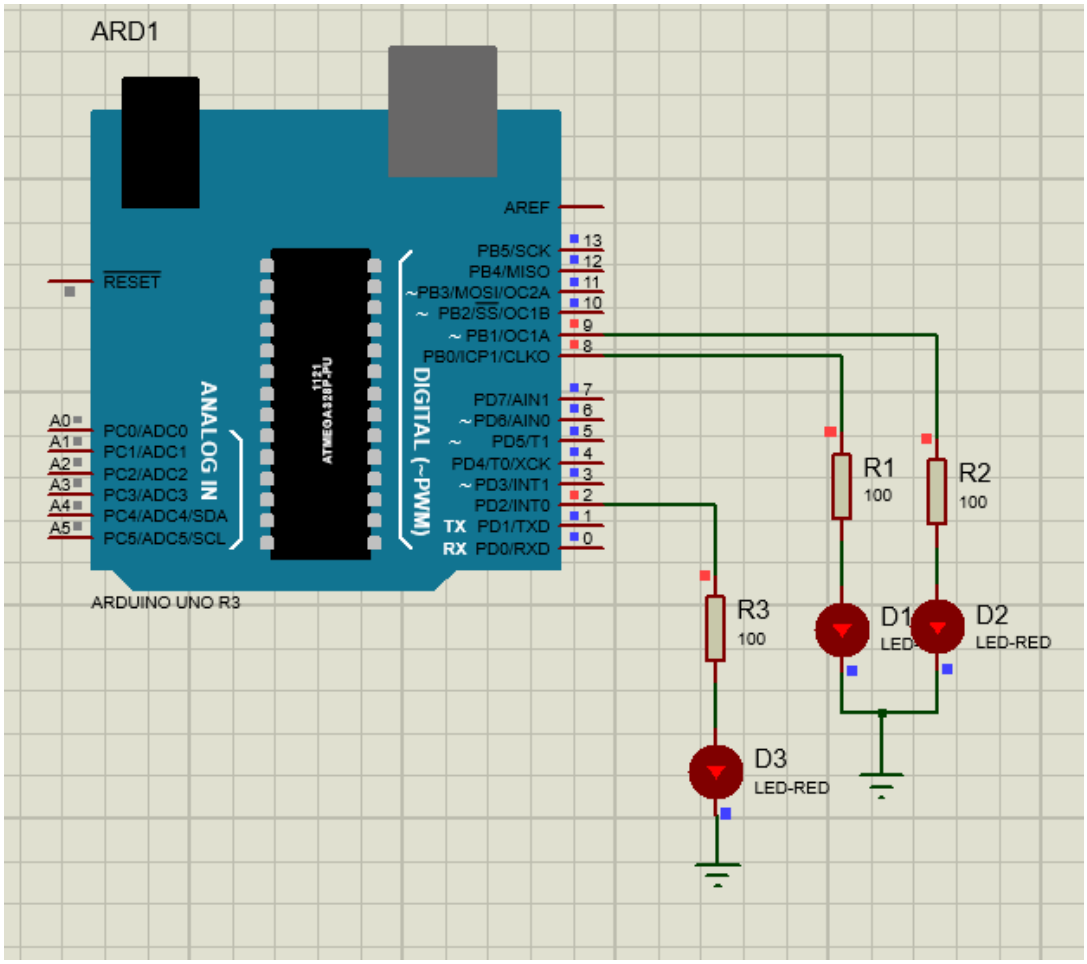




Ici on voit bien que les leds sur les broches 9 et 8 clignotent toute marche très bien.

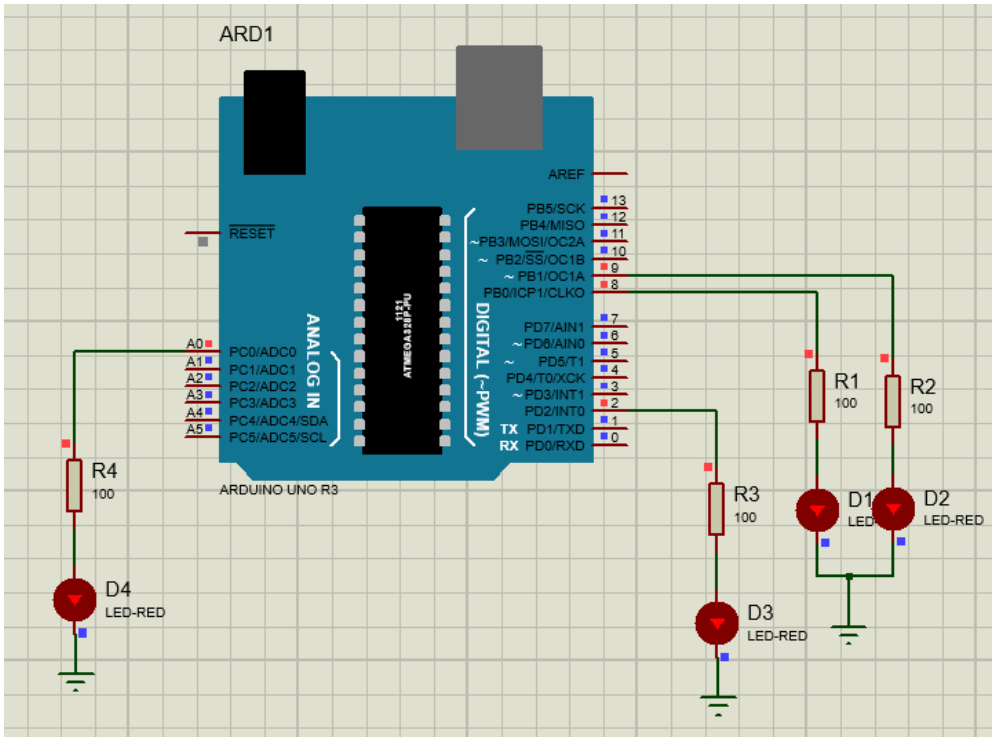
Par la suite utilisons aussi le PORTD

```
void setup() {

    DDRB = 0b11111111 ;
    DDRD = 0b11111111 ;

}

void loop() {

   PORTB = 0b00000011 ;
   PORTD = 0b00000100 ;
   delay (1000) ;
   PORTB = 0b00000000;
   PORTD = 0b00000000;
   delay (1000);
}
```



Ici on utilise aussi Le PORTC

```
void setup() {

   DDRB = 0b11111111 ;
   DDRD = 0b11111111 ;
   DDRC = 0b11111111 ;

}

void loop() {

  PORTB = 0b00000011 ;
  PORTD = 0b00000100 ;
  PORTC = 0b00000001 ;
  delay (1000) ;
  PORTB = 0b00000000;
  PORTD = 0b00000000;
  PORTC = 0b00000000 ;
  delay (1000);
}
```

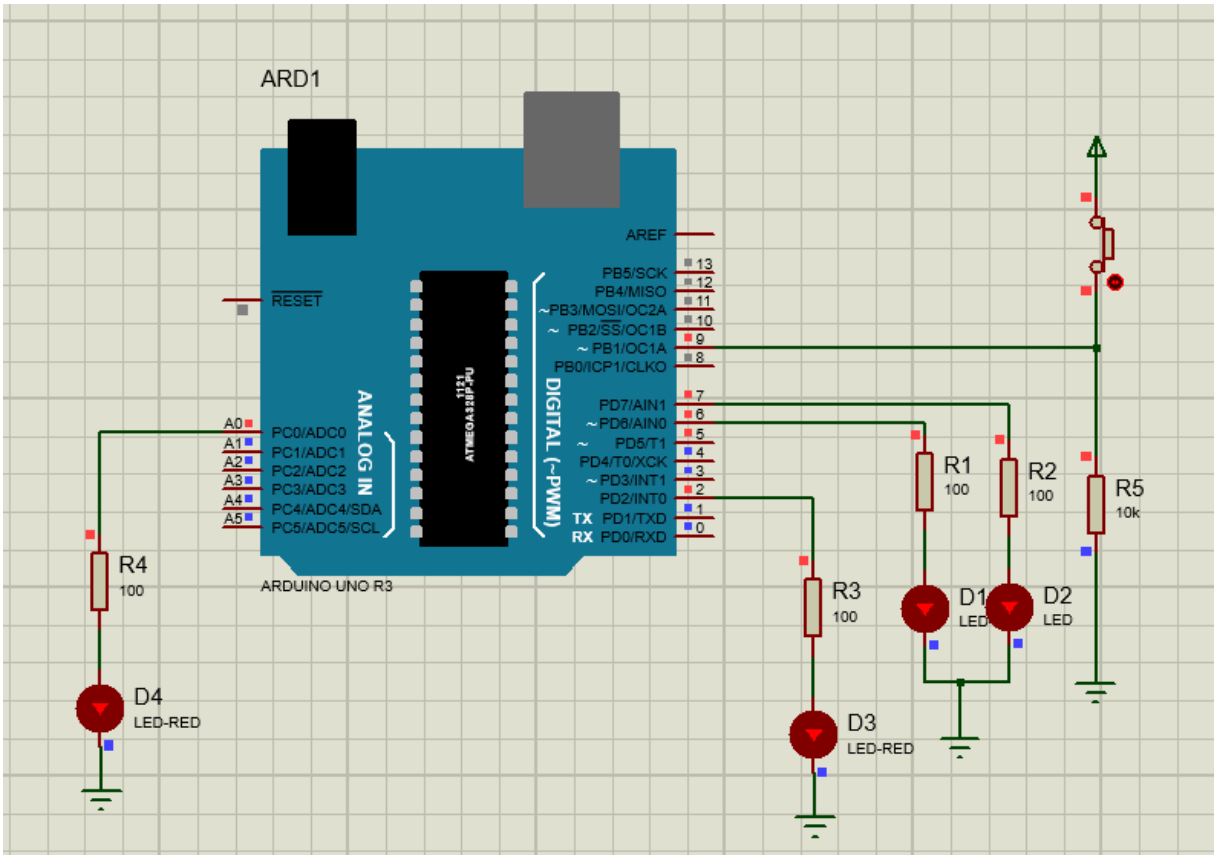


LA GESTION DE BROCHE EN ENTREE

Dans cette partie nous allons montrer comment configurer les broches en entrée en utilisant l'adressage aux registres. Ici aussi il y a deux étapes.

```
void setup() {

   DDRB = 0b00000000 ;
   DDRD = 0b11111111 ;
   DDRC = 0b11111111 ;

}

void loop() {

 if (PINB & 0X001) {


  PORTD = 0b11100100 ;
  PORTC = 0b00000001 ;

  delay (1000) ;

  PORTD = 0b00000000;
  PORTC = 0b00000000 ;
  delay (1000);

 }
```

Ici nous venons de faire la configuration de toutes les broches du port B donc en allant de la broche 8 à la broche 13 de la carte ARDUINO en entrée.



Donc pour le registre DDRx si le bit est à 1 cela veut dire que la broche est configurer en sortie et si ce 0 donc la broche est configurer en entrée

En écrivant DDRB = 0b00000000 ; cela veut dire que nous configurons toutes les broches de du port B en entrée.

Alors pour la condition on utilise le registre PINx

```
if (PINB & 0X001) {

 // ICI ON LES INSTRUCTIONS SI LA CONDITION EST VRAIE

}
```

Ici dans notre cas on teste l'état de la broche PB0.

PINB & 0X001

Ici on a : 0b11111111 & 0x01 = 0b$X_7X_6X_5X_4X_3X_2X_2X_1X_0$

Si :

X_7 : Passe à 1 dans ce cas on teste l'état de port bit 7 en entrée et non si 0

X_6 : Passe à 1 dans ce cas on teste l'état de port bit 6 en entrée et non si 0

X_5 : Passe à 1 dans ce cas on teste l'état de port bit 5 en entrée et non si 0

X_4 : Passe à 1 dans ce cas on teste l'état de port bit 4 en entrée et non si 0

X_3 : Passe à 1 dans ce cas on teste l'état de port bit 3 en entrée et non si 0

X_2 : Passe à 1 dans ce cas on teste l'état de port bit 2 en entrée et non si 0

X_1 : Passe à 1 dans ce cas on teste l'état de port bit 1 en entrée et non si 0

X_0 : Passe à 1 dans ce cas on teste l'état de port bit 0 en entrée et non si 0

Quand on met un 'b' il s'agit d'un codage binaire mais quand on met un 'x' il s'agit d'un codage hexadécimal

Chers amis la programmation ARDUINO est bonne mais elle est encore trop bonne quand on fait la programmation registre on se sent proche du processeur surtout nous qui sommes électroniciens nous devons plus nous rapprocher de cette programmation registre car nous rapproche plus de l'architecture du microprocesseur.

3.1 ETAT D'UN PORT NUMERIQUE

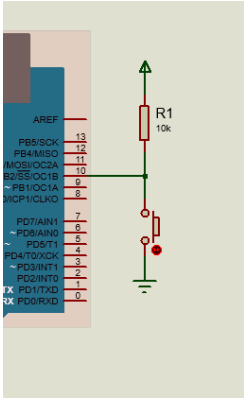
Lorsqu'une broche numérique est utilisée comme entrée il peut avoir trois états :

1. Etat logique 1
2. Etat logique 0
3. Ou Etat aléatoire

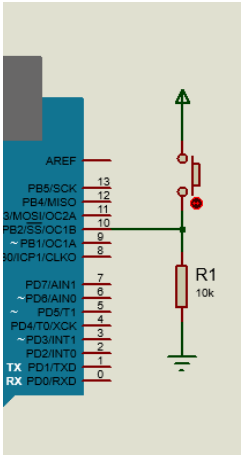
Alors selon le programme on teste un état ou un autre pour agir

Mais certes le dernier état (Etat aléatoire) pose un problème car la broche change d'état d'une manière aléatoire ce qui n'est pas bon pour l'exécution du programme qui peut agir d'une manière bizarre c'est pourquoi il est toujours important de fixer l'état d'une broche.

Voici comment faire pour éviter les états aléatoires :



Ici on vient de fixer l'état de la broche à 1 puis lorsqu'on appuie sur le bouton poussoir la broche passe à 0. Et cela permet de fixer l'état à 1 avant qu'on appuie sur le bouton poussoir.



Ici on vient de fixer l'état de la broche à 0 puis lorsqu'on appuie sur le bouton poussoir la broche passe à 1. Et cela permet de fixer l'état à 0 avant qu'on appuie sur le bouton poussoir.

Il y a aussi un moyen de configurer une résistance de pull up (tirant au plus) interne se trouvant l'intérieur de l'atmega328p de la carte ARDUINO cette méthode semble souvent très intéressant lorsqu'on on pas trop de résistance physique.

CHAPITRE 3 LES VARIABLES

LES VARIABLES

Par définition une variable est un espace qu'on loue dans une case mémoire pour y stocker des informations.

En arduino une variable est caractérisée par : son nom, sa portée et son type et par là sa taille. Le nom d'une variable doit être explicatif.

PORTEE D'UNE VARIABLE

Une variable peut-être publique ou locale une variable dit locale lorsqu'elle est déclarée dans une fonction et ici il n'est accessible que dans cette fonction.

Une variable est dit publique lorsqu'il est en dehors des fonctions on l'appelle encore variable globale car il est accessible partout.

TYPE DES VARIABLES

Les types des variables peuvent être énumérés dans le tableau suivant :

TYPE	STOCKAGE	LIMITATION	TAILLE EN BITS	TAILLE EN OCTETS
int	entiers	-32 768 à +32 767	16	2
long	entiers	-2 147 483 648 à +2 147 483 647	32	4
char	entiers	-128 à +127	8	1
float	décimale	-3.4 * 10^38 à +3.4 * 10 ^38	32	4
double	décimale	-3.4 * 10^38 à +3.4 * 10 ^38	32	4
byte	Entier non négatif	0 à 255	8	1
word	Entier non négatif	0 à 65535	16	2
boolean	Entier non négatif	0 à 1	32	4

COMMENT CHOISIR LE TYPE D'UNE VARIABLE

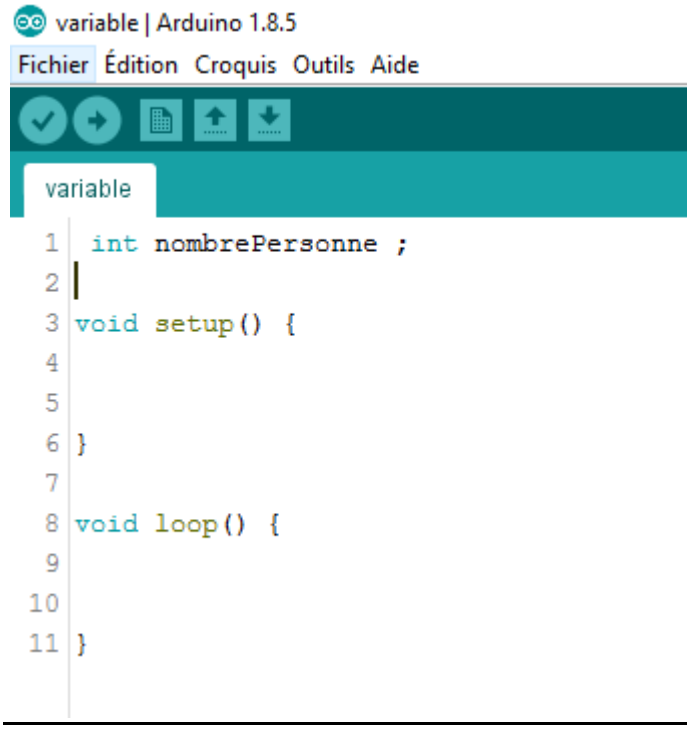
On choisit le type d'une variable en fonction de l'information qui sera stocké dans la variable. Si l'on veut stocker une information entière on utilise les variables entières

Exemple1 : Si on veut faire un programme permettant de compter les nombres de personnes qui passe dans une porte on utilisera une variable entière car on ne peut pas avoir un nombre de personne qui est décimal.

Exemple2 : Si on veut faire un programme permettant de mesurer la température la variable de la température sera une variable décimal car la température est pour la plupart de fois décimale.

DECLARATION DE VARIABLES DANS ARDUINO

DECLARATION DE LA VARIABLE GLOBAL



Déclaration des plusieurs variables globales

variable | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

variable §

```
int nombrePersonne ;

float temperature ;

void setup() {


}

void loop() {


}
```

Il en a de même pour les autres variables on utilise on déclare en appelant le type et le nom de la variable.

DECLARATION DE LA VARIABLE GLOBAL

Une variable locale est déclarée dans une fonction

Exemple d’une variable locale

variable | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

variable §

```
1
2
3 void setup() {
4
5  float temperature ;
6 }
7
8 void loop() {
9
10
11 }
```

Cette variable temperature est locale ne peut être visible que dans la fonction setup

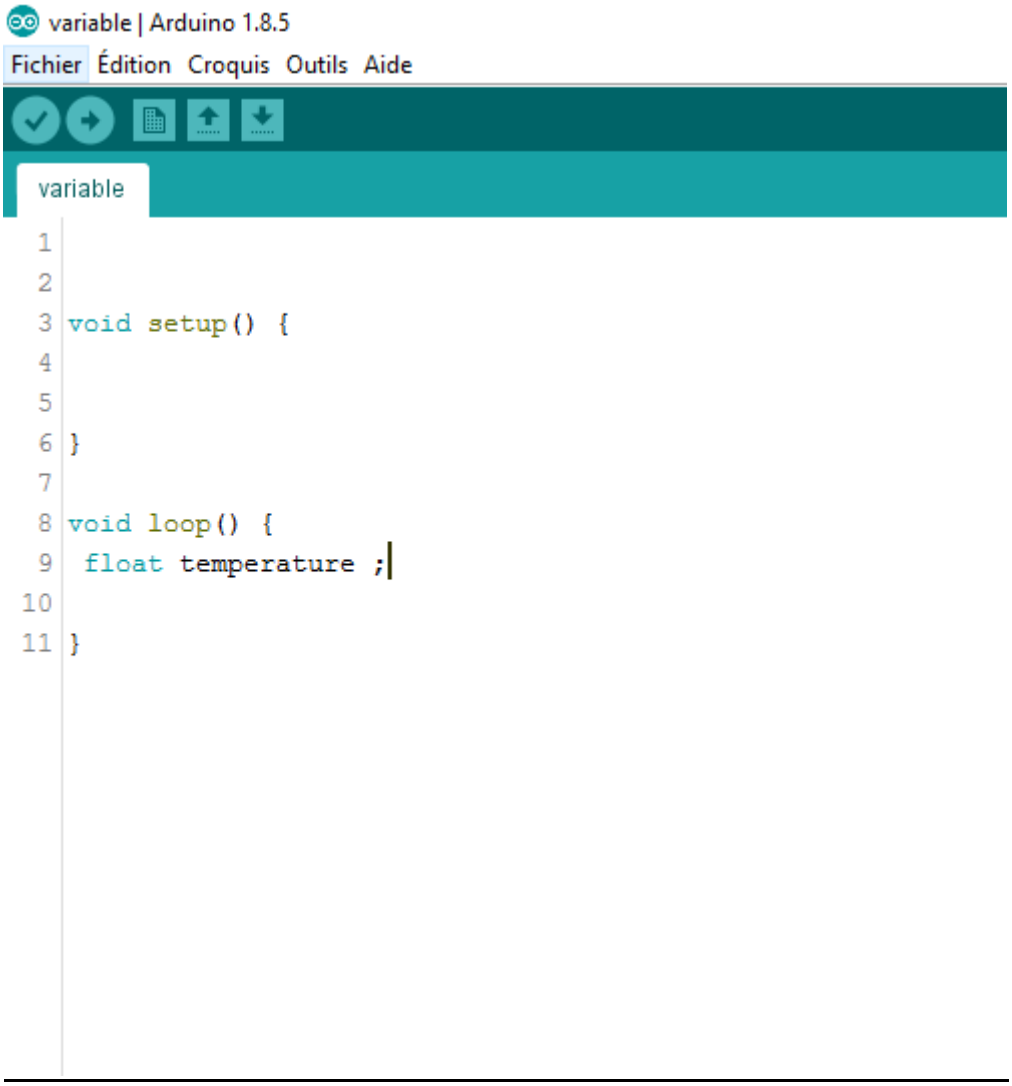
CONTROL DE L'ALLUMAGE D'UNE LAMPE ET EXTINCTION PAR PLUSIEURS APPUIS SUR BOUTON POUSSOIR

Dans ce présent article je m'en veux présenter un code permettant d'allumer une lampe quand on appuie six fois sur le bouton poussoir et qui l'éteint quand on appuie deux fois.

```
 int nombreAppuie = 0 ;

void setup() {

  pinMode (13 , OUTPUT) ;
  pinMode (8, INPUT_PULLUP) ;

}

void loop() {
 if (digitalRead (8) == LOW) {
     nombreAppuie++ ;
 }

  if ( nombreAppuie== 4) {
     digitalWrite (13 , HIGH);
  }

   if ( nombreAppuie  == 6) {
     digitalWrite (13 , LOW) ;
      nombreAppuie  = 0 ;
   }



}
```

EXPLICATION

Dans ce code on commence par la déclaration et l'initialisation de la variable : nombreAppuie
Alors quand la broche 4 passe à 0 (quand on appuie sur le bouton poussoir) la variable nombreAppuie se voit implémenter (donc se voit ajouter 1). Lorsque la variable nombreAppuie = 4 après 4 appuis sur bouton poussoir la borne 13 passe à 1.
Lorsque nombreAppuie = 6 la borne 13 passe à 0 et on fait revenir nombreAppuie à 0 pour permettre au système de s'exécuter en boucle.



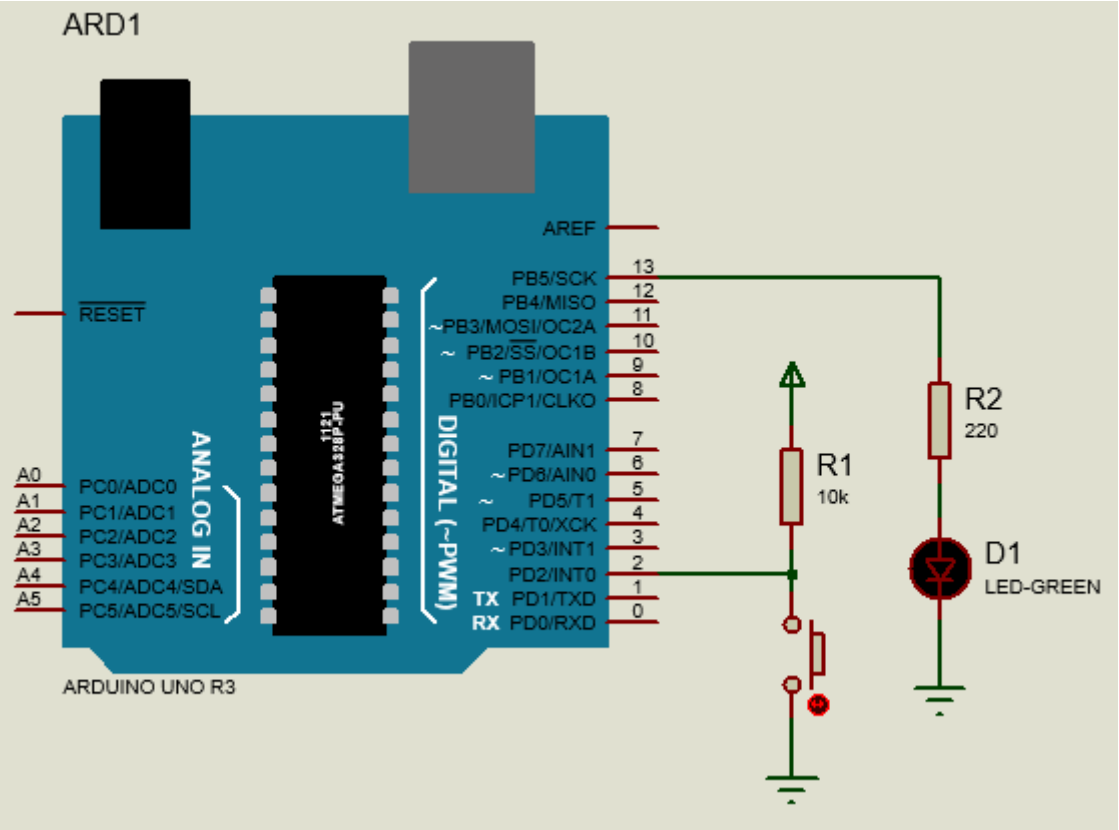
CHAPITRE4. PROBLEME DE REBONDS

INTRODUCTION

Dans ce présent article nous allons traiter sur le problème de rebond un problème couramment rencontré dans le domaine de la programmation embarquée.

DEVELOPPEMENT

Le problème de rebond intervient lorsque vous utiliser une broche du microcontrôleur en entrée soit le montage suivant :



Ce montage ci-dessous fonctionne de cette façon :

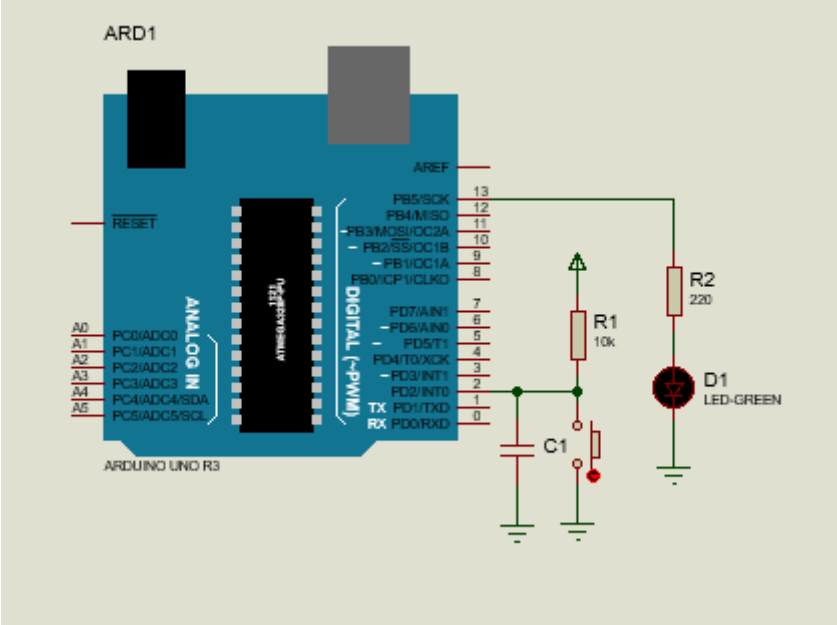
Quand on appuie sur le bouton poussoir une fois la lampe s'allume quand on appuie pour la deuxième fois elle s'éteint et ainsi de suite.

Le problème est que quand on appuie sur le bouton poussoir on peut penser qu'on appuyer une seule fois mais sachant que le microcontrôleur fonctionne il considère que vous avez appuyé plusieurs fois. En plus le bouton poussoir est constitué par de lames vous pouvez penser que vous avez appuyé une seule fois mais les lames du bouton poussoir se touchent plusieurs fois pour palier à cela on utilise les anti-rebonds voir la suite.

Pour trouver une solution on utilise l'option du hardware ou du software.

La solution en hardware consiste à mettre le bouton poussoir en parallèle avec un condensateur. Sachant que le condensateur s'oppose aux variations brusques de tension il joue le rôle d'absorber les variations de tension lorsqu'on appuie sur le bouton poussoir.

On peut faire le schéma suivant :



Ici on évite les rebonds en utilisant un condensateur cette technique est bonne et rapide mais si on a plusieurs cela peut être un problème en ce qui concerne le coût et la taille du montage. Ce pour cela qu'on peut utiliser la solution software que nous allons présenter ci-dessous :

La solution software cette solution consiste à faire un programme qui va rendre aveugle le programme face aux rebonds pour ce faire on utilise juste la fonction **delay (constante)** pour voir pratiquement ce que nous venons d'expliquer faisons un programme permettant d'allumer une lampe quand on appuie sur le bouton pour la première fois et o l'éteint quand on appuie pour la deuxième fois :

```
 int compte ;
void setup() {
 pinMode(8,INPUT_PULLUP);
 pinMode (9, OUTPUT);
}

void loop() {
   if (digitalRead (8) == LOW) {
     delay (300) ;  // on met le microcontroleur en pause pour eviter les rebonds
     compte++ ;
   }
    if (compte == 1) {
       digitalWrite (9,HIGH) ;
    }
    if (compte == 2) {
       digitalWrite (9,LOW) ;
    }
}
```

Pour éviter ce problème de rebonds avec le software juste après appuie sur le bouton on active le delay pour rendre le microcontrôleur insensible aux rebonds.

Je pense qu’en lisant cet article vous avez trouvez une manière de résoudre le problème de rebonds

CHAPITRE5. LES CONDITIONS EN ARDUINO

- Si on veut qu'une lampe puisse s'allumer lorsqu'on appuie sur un ou sur deux boutons
- Si on veut faire tourner un moteur quand la sortie d'un détecteur passe à 1
- Etc.

Pour faire une condition en langage de programmation on utilise l'instruction :

If (condition)

Dans la condition on utilise des opérateurs logiques donnée dans le tableau suivant :

Symbole	Nom de l'opérateur
&&	ET
||	OU
!	NON

<u>Table logique de l'opérateur ET (&&) :</u>

Condition1	Condition2	Résultat (&&)
fausse	fausse	On n'a rien
fausse	vraie	On n'a rien
vraie	fausse	On n'a rien
vraie	vraie	On a un résultat

Condition1	Concdition2	Résultat (&&)
0	0	0
0	1	0
1	0	0
1	1	1

Lorsqu'on on utilise l'opérateur ET on ne peut avoir un résultat que si toutes les conditions sont respecter exemple :

If (condition1 && condition2) {

Instructions

}

Les instructions se trouvant dans les accolades ne peuvent s'exécuter que si les conditions (condition1 et condition2) sont respectées. Si l'un de condition ne pas respecter rien ne va se passer

<u>Table logique de l'opérateur OU (||)</u>

Condition1	Condition2	Résultat (||)
Fausse	fausse	On n'a rien
fausse	vraie	On a un résultat
vraie	fausse	On a un résultat
vraie	vraie	On un résultat

Condition1	Condition2	Résultat (||)
0	0	0
0	1	1
1	0	1
1	1	1

If (condition1 || condition2) {

Instructions

}

Les instructions se trouvant les accolades s'exécute si au moins l'une de condition est vérifiée.

Exemple d'un programme ARDUINO permettant d'allumer une lampe se trouvant à la borne 13 quand on appuie sur un bouton se trouvant à la broche 8.

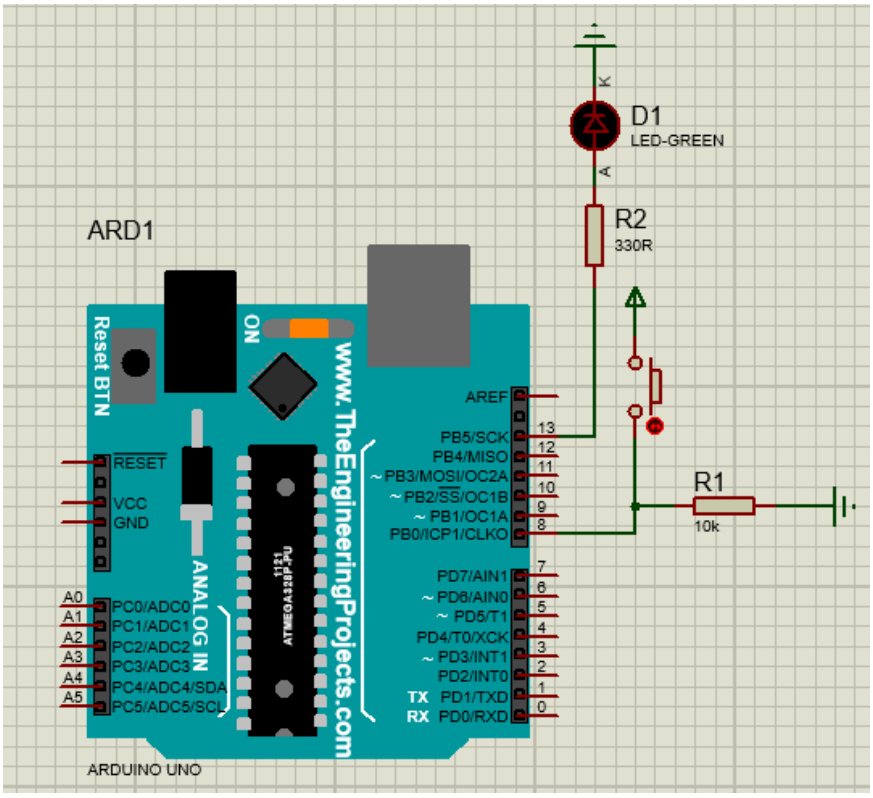
condition | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

condition §

```
1 void setup() {
2
3  pinMode (8,INPUT)// configuration de la broche en entrée
4  pinMode (13,OUTPUT);// configureation de la broche en sortie
5
6 }
7
8 void loop() {
9
10   if (digitalRead(8) == HIGH ) // si on appuie sur le bouton
11    {
12   pinMode (13,HIGH); // allumage de la led se trouvant sur le 13
13    }
14 }
```

SCHEMAS DE REALISATION



Exemple d'un programme ARDUINO permettant d'allumer une lampe se trouvant à la borne 13 quand on appuie à la fois sur les boutons se trouvant aux broches 8,9 et 10

condition | Arduino 1.8.5

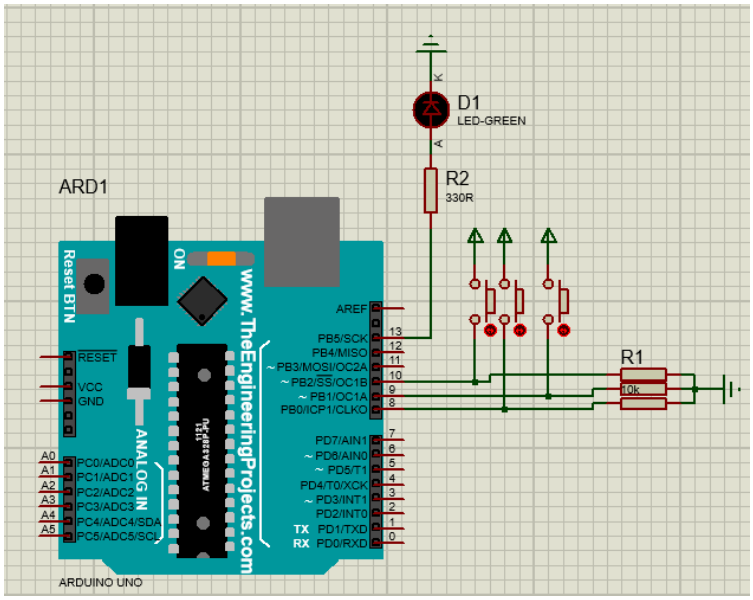
Fichier Édition Croquis Outils Aide

condition §

```
void setup() {

 pinMode (8,INPUT)// configuration de la broche en entrée
 pinMode (9,INPUT)// configuration de la broche en entrée
 pinMode (10,INPUT)// configuration de la broche en entrée
 pinMode (13,OUTPUT);// configureation de la broche en sortie

}

void loop() {

  if (digitalRead(8) == HIGH && digitalRead(9) == HIGH && digitalRead(10) == HIGH ) //condition
   {
  pinMode (13,HIGH); // allumage de la led se trouvant sur le 13
   }
}
```

Ici la lampe ne s'allumer que si on appui simultanément sur les boutons 8,9 et 10

SCHEMAS DE REALISATION



Vous constatez la présence des résistances en effet ces dernières permettent d'éviter les états aléatoires.

Pour la logique OU on a

Exemple d'un programme ARDUINO permettant d'allumer une lampe se trouvant à la borne 13 quand on appuie sur au moins un des boutons se trouvant aux broches 8,9 et 10

condition | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

Vérifier

condition §

```
1 void setup() {
2
3  pinMode (8,INPUT)// configuration de la broche en entrée
4  pinMode (9,INPUT)// configuration de la broche en entrée
5  pinMode (10,INPUT)// configuration de la broche en entrée
6  pinMode (13,OUTPUT);// configureation de la broche en sortie
7
8 }
9
10 void loop() {
11
12   if (digitalRead(8) == HIGH || digitalRead(9) == HIGH || digitalRead(10) == HIGH ) //condition
13    {
14   pinMode (13,HIGH); // allumage de la led se trouvant sur le 13
15    }
16 }
```

CONDITION SWITCH

INTRODUCTION

En programmation lorsqu'on a trop des conditions « ***if*** » qui sont répétitifs on conseille d'utilise la condition « ***switch*** » imaginer par exemple un programme où on agit en fonction de la valeur d'une variable donnée pour allumer les lampes on a le code suivant :

```
void loop() {

  if (i == 0) {
    digitalWrite (13,HIGH);
    }

  if (i == 1) {
    digitalWrite (12,HIGH);
    }

  if (i == 2) {
    digitalWrite (11,HIGH);
    }

  if (i == 3) {
    digitalWrite (10,HIGH);
    }

  if (i == 4) {
    digitalWrite (9,HIGH);
    }
}
```

Pour ce code nous pouvons facilement voir que nous allumons une ou autre led selon la valeur de la variable i. Il est possible de faire la même chose en utilisant la condition « ***switch*** »

```
void loop() {

   switch (i) {
     case 1 :
        digitalWrite (13,HIGH);
        break;
     case 2 :
        digitalWrite (12,HIGH) ;
     case 3 :
        digitalWrite (11,HIGH);
     case 4 :
        digitalWrite (10,HIGH);
   }
}
```

En utilisant la condition switch on minimise l'écriture du code. Ici dans le code si haut on commence d'abord par vérifier la valeur de i :

1. Si i = 1 on active la broche 13 puis on fait break
2. Si i = 2 on active la broche 12 puis on fait break
3. Si i = 3 on active la broche 11 puis on fait break
4. Si i = 4 on active la broche 10 puis on fait break

Alors si toute les contions ne sont pas respectées on peut aussi faire une action et pour cela on utilise l'instruction « ***default*** »

```
void loop() {

   switch (i) {
     case 1 :
        digitalWrite (13,HIGH);
        break;
     case 2 :
        digitalWrite (12,HIGH) ;
     case 3 :
        digitalWrite (11,HIGH);
     case 4 :
        digitalWrite (10,HIGH);

        default :
        digitalWrite (9,HIGH);
         break ;
   }
}
```

Dans ce code ci haut si la valeur de i de 1 de 2 de 3 ou de 4 c'est le default qui va s'exécuter

Le default s'exécute lorsque toutes les conditions sont fausses.

Chers amis voici comment fonction la condition switch.

CHAPITRE6. LES TABLEAUX

LES TABLEAUX

En programmation on utilise les tableaux pour la déclaration des variables de même nature cela permet de réduire les nombres de ligne de codes. Dans un tableau on déclare à la fois plusieurs variables de même type.

CARACTERISTIQUES D'UN TABLEAU

Un tableau est caractérisé par :

- Son type : un tableau peut stocker des informations de type entières (int), caractère (char) ou autres types
- Son nom : comme une variable, un tableau est caractérisé par son nom c'est ce dernier qui permet de l'identifier dans les codes.
- Et enfin les nombres d'éléments qu'il peut contenir : c'est le nombre des variables se trouvant en son sein.

DECLARATION DU TABLEAU

```
1
2 int tabLed [4] = {10,11,12,13};
3
```

Ici nous venons de déclarer et d'initialisation un tableau de type entier, ayant 4 éléments. Soit quatre variables indépendantes créer dans un même tout.

UTILISATION DES DONNEES DU TABLEAU

Un tableau est un ensemble des variables indexées. Le premier élément du tableau est indexé 0 et le deuxième élément du tableau est indexé 1 le troisième élément est indexé 2 le quatrième est indexé 3 et ainsi de suite.

Si on écrit par exemple :

```
tabLed [0];
```

Ecrivant ceci on obtient la valeur du premier élément du tableau : dans notre cas on aura une valeur de 10.

```
tabLed [1];
```

En écrivant ceci on obtient la valeur deuxième élément du tableau : dans notre cas on aura une valeur de 11.

```
tabLed [2];
```

En écrivant ceci on obtient la valeur du troisième élément du tableau : dans notre cas on aura une valeur de 12.

```
tabLed [3];
```

En écrivant ceci on obtient la valeur du quatrième élément du tableau : dans notre cas on aura une valeur de 13.

On peut appliquer sur les tableaux toutes les opérations possibles qu'on applique sur les variables car en effet un tableau de par sa définition est un ensemble des variables.

En programmation arduino le tableau va faciliter certaines tâches. Posons le problème imaginer un projet ou on doit configurer les broches 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,12 et 13 en sorties comme on fera :

```
void setup() {

  pinMode (2,OUTPUT);
  pinMode (3,OUTPUT);
  pinMode (4,OUTPUT);
  pinMode (5,OUTPUT);
  pinMode (6,OUTPUT);
  pinMode (7,OUTPUT);
  pinMode (8,OUTPUT);
  pinMode (9,OUTPUT);
  pinMode (10,OUTPUT);
  pinMode (11,OUTPUT);
  pinMode (12,OUTPUT);
  pinMode (13,OUTPUT);
}

void loop() {


}
```

Avec cette méthode on fait la configuration mais en utilisant beaucoup des lignes de codes ; et ce qui qu'on peur en temps plus on a des sorties plus écrit beaucoup mais avec la technique des tableaux on résout ce problèmes résolvons donc le même problème en utilisant les tableaux on a :

```
int tableauLED [12] {2,3,4,5,6,7,8,9,10,11,12,13} ;

void setup() {
   for (int i = 0 , i<13 , i++) {

     pinMode (tableauLED [i] , OUTPUT);

   }


}

void loop() {


}
```

En utilisant les boucles le code devient très petit peu importe le nombre de sorties que l'on va configurer. Explication du code utilisant le tableau.

Premièrement pour ce code on utilise le tableau et dans ce dernier on énumère les valeurs de toutes les broches que l'on va configurer en sortie l'instruction qui permet cela est le suivant :

```
int tableauLED [12] {2,3,4,5,6,7,8,9,10,11,12,13} ;
```

Pour faire la configuration des broches en sortie on utilise la boucle for : cette boucle permet de parcourir le tableau et par là on configure chaque élément du tableau en sortie :

```
void setup() {
   for (int i = 0 , i<13 , i++) {

     pinMode (tableauLED [i] , OUTPUT);

   }


}
```

Ici on fait la configuration en parcourant le tableau avec la boucle for :

Au début de la boucle : i =0 ce qui respecte la condition d'entré dans la boucle et par là la broche 2 de la carte arduino est configuré en sortie puis i passe 1 avec i=1 la condition d'enté dans la boucle est vérifié puis la broche 3 est aussi configuré en sortie puis i passe à 2. La boucle va continuer jusqu'à ce que la condition ne sera plus vraie : donc lorsque i sera égale à 13 par là on voit que 13

n’est pas inférieur après on quitte ; le programme quitte la boucle for après avoir configuré les broches se trouvant dans le tableau en sortie.

Dans la suite nous allons faire un petit programme permettant de faire clignoter des leds :

```
int tableauLed [3] = {10,12,13} ;
void setup() {

   for (int i=0 ; i>4; i++) {
     pinMode( tableauLed [3],OUTPUT) ;
   }
}

void loop() {
  digitalWrite ( tableauLed [0] ,HIGH) ;
  digitalWrite ( tableauLed [1] ,HIGH) ;
  digitalWrite ( tableauLed [2] ,HIGH) ;
  delay (1000);
  digitalWrite ( tableauLed [0] ,LOW) ;
  digitalWrite ( tableauLed [1] ,LOW) ;
  digitalWrite ( tableauLed [2] ,LOW) ;
delay (1000);



}
```

```
  int a ;
  int b ;

 int tableauLed [3] = {10,12,13} ;
void setup() {

   for (int i=0 ; i>4; i++) {
     pinMode( tableauLed [3],OUTPUT) ;
   }
}

void loop() {

  while (a<4) {
      digitalWrite ( tableauLed [i] ,HIGH) ;
      a++;
  }

}
```

Ce dernier programme lui : permet d’allumer toutes les LED se trouvant sur les broches 10, 12,13.

CHAPITRE7 LES BOUCLES

Les boucles sont utilisées pour ne pas répéter les codes.

Lorsqu'on veut qu'une portion de code s'exécute en boucle sous une certaine condition on utilise les boucles

Il existe principalement trois sortes de boucles :

- La boucle for
- La boucle while
- Et la boucle do while

1. BOUCLE for

Avant d'expliquer la boucle for nous allons d'abord commencer par exposé un problème nécessitant son utilisation. Pour ce faire faisons un programme permettant de faire clignoter dix fois une led se trouvant à la borne 13.

sketch_feb13a | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

sketch_feb13a §

```
1  void setup() {
2    pinMode (13,OUTPUT);
3
4     digitalWrite (13,HIGH);
5     delay (1000);
6      digitalWrite (13,LOW);
7      delay (1000);
8       digitalWrite (13,HIGH);
9     delay (1000);
10     digitalWrite (13,LOW);
11     delay (1000);
12      digitalWrite (13,HIGH);
13    delay (1000);
14     digitalWrite (13,LOW);
15     delay (1000);
16      digitalWrite (13,HIGH);
17    delay (1000);
18     digitalWrite (13,LOW);
19     delay (1000);
20      digitalWrite (13,HIGH);
21    delay (1000);
22     digitalWrite (13,LOW);
23     delay (1000);
24     digitalWrite (13,HIGH);
```

```
25    delay (1000);
26     digitalWrite (13,LOW);
27     delay (1000);
28      digitalWrite (13,HIGH);
29    delay (1000);
30     digitalWrite (13,LOW);
31     delay (1000);
32      digitalWrite (13,HIGH);
33    delay (1000);
34     digitalWrite (13,LOW);
35     delay (1000);
36      digitalWrite (13,HIGH);
37    delay (1000);
38     digitalWrite (13,LOW);
39     delay (1000);
40      digitalWrite (13,HIGH);
41    delay (1000);
42     digitalWrite (13,LOW);
43     delay (1000);
44 }
45
46 void loop() {
47
48 }
```

Ici on voit que pour faire clignoter une la lampe 10 fois on doit l'allumer et l'éteindre dix fois on doit répéter cela dans le code ce qui rend le code très long pour pallier à cela on utilise les boucles.

Comment utiliser :

for(initialisation ; condition ; opération)

{

Instructions

}

Les instructions de cette boucle s'exécutent que si la condition est vraie

Faisons alors l'exercice de faire clignoter une led 10 fois avec ***la boucle for***

sketch_feb13a | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

sketch_feb13a

```
1  void setup() {
2    pinMode (13,OUTPUT);
3
4     for (int i =0 ;i<10;i++)
5     {
6      digitalWrite (13,HIGH);
7      delay (1000);
8      digitalWrite (13,LOW);
9      delay (1000);
10    }
11
12 }
13
14 void loop() {
15
16 }
```

Ce code permet de faire la même chose que le premier mais il est plus court par rapport au premier car il utilise la notion de boucle.

<u>COMMENT FONCTIONNE LA BOUCLE FOR</u>

```
for (int i =0 ;i<10;i++)
{
 digitalWrite (13,HIGH);
 delay (1000);
 digitalWrite (13,LOW);
 delay (1000);
}
```

On commence par une initialisation en affectant à une certaine variable une valeur initiale :

```
(int i =0
```

Dans notre cas on a une variable entière i qu'on affecte une valeur initiale de 0

En initialisant la variable i à 0 on entre dans la boucle que si **<u>la condition est vraie</u>**

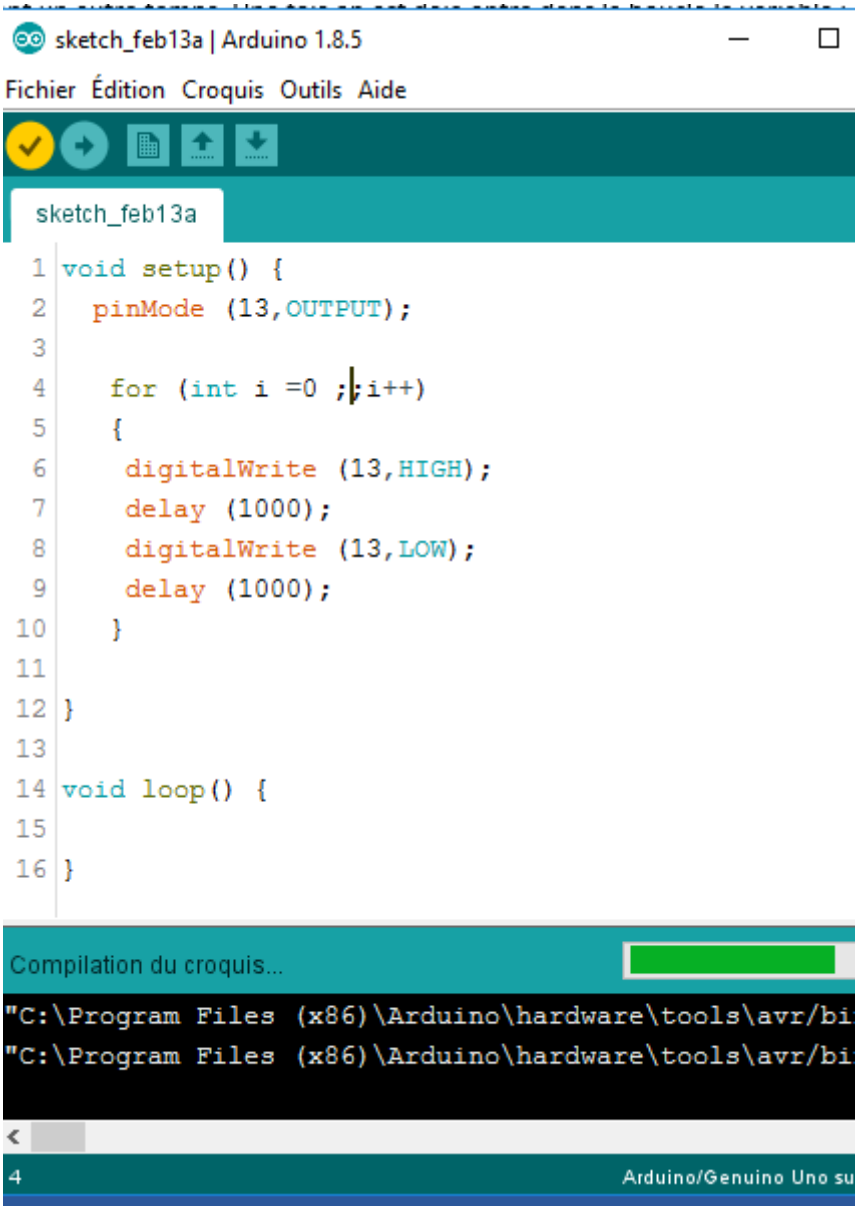
Premièrement on voit que i égale à 0 cela implique que i est inférieur à 10 comme la condition est vraie on entre dans la boucle et on effectue l'opération i++ on augmente à i 1 façons de signaler le premier entré dans la boucle. Bien sûr en entrant dans la boucle on exécute les instructions ce

trouvant dans la boucle dans notre cas on allume la led se trouvant à la borne 13 pendant un temps puis on l'éteint pendant un autre temps. Une fois on est déjà entré dans la boucle la variable i passe à 1 après exécution des instructions de la boucle on vérifie si la condition est encore vraie sachant que i est déjà à 1 et 1 est inférieur à 10 on entre encore dans la boucle car la condition est vraie puis on ajoute encore à i 1 il devient 2 et ainsi de suite jusqu'à ce que i sera égale à 10 la condition ne plus respecte car 10 n'est pas supérieur à 10. Puis on quitte la boucle

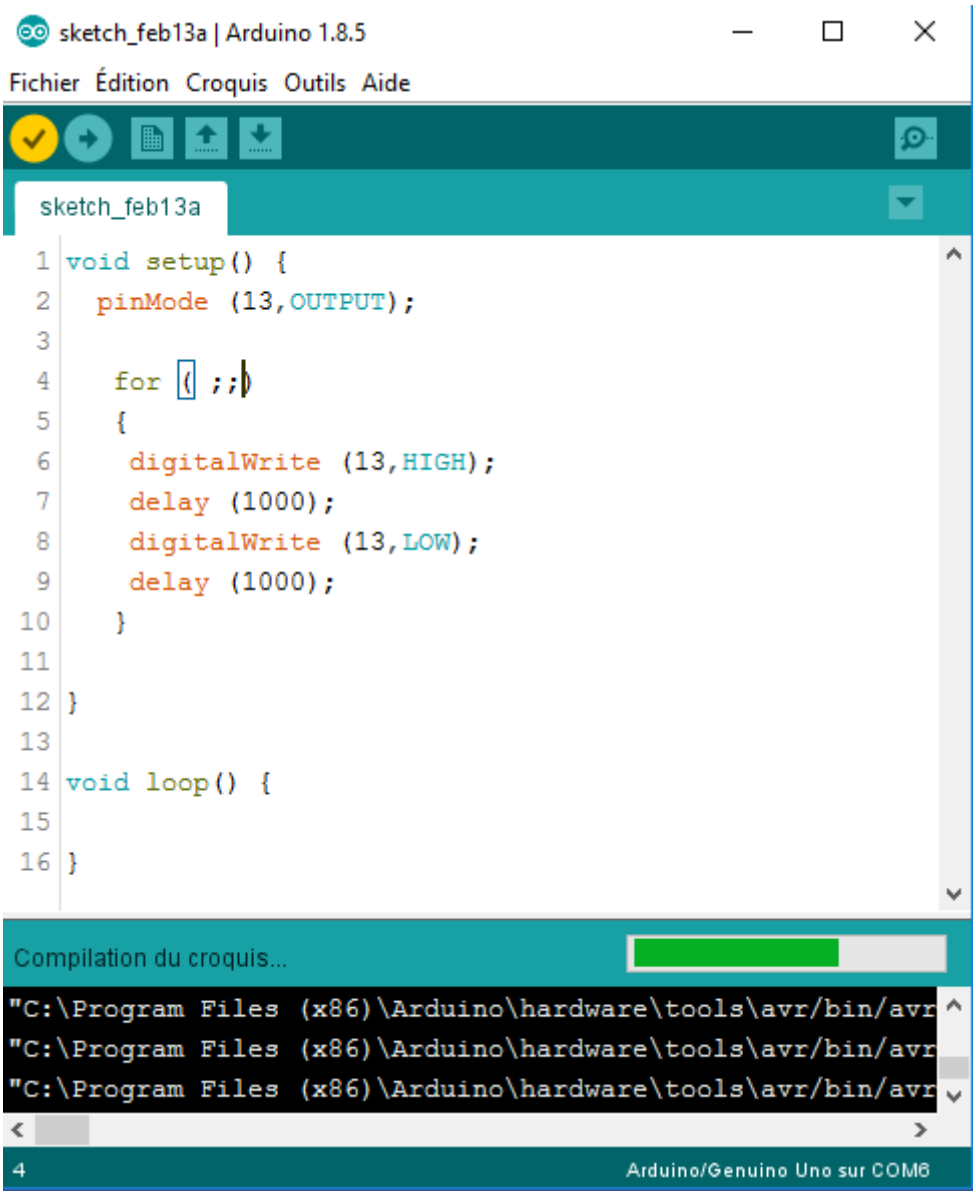
Donc cette boucle s'exécute tant que la condition est vérifiée.

<u>COMMENT FAIRE UNE BOUCLE INFINIE AVEC LA BOUCLE</u>

Avec la boucle for pour faire une boucle infinie il suffit seulement que la condition soit vraie à chaque instant. Où carrément n'est pas mettre la condition.



Dans ce code on a juste enlevé la condition et cela effectue une boucle infinie. On peut encore écrire de la sorte :



2. <u>BOUCLE while</u>
   Programme permettant de faire un programme permettant de faire clignoter dix fois une led se trouvant à la borne 13.

while(condition)

{

Instructions

}

Les instructions de la boucle while s'exécute si la condition est vraie.

L'opération s'effectue dans le corps de la boucle. Donc ici dans ce programme on commence d'abord par créer la variable à qui on affecte une valeur de 0. Sachant qu'au début i =0 la condition pour entrer dans la boucle est respectée ce qui fait qu'on exécute les instructions de la boucle puis i passe s'incrémente. On exécutera les instructions de la boucle tant que la condition est vérifiée donc tant que i est inférieure à 10 ce qui permettra 10 entrées dans la boucle et par là dix clignotement de la tension se trouvant à la borne 13.

3. <u>Boucle do while</u>

La grande différence entre la boucle do while et les autres sortes de boucles est que cette boucle s'exécute au moins une seule fois. Pour cette boucle on exécute d'abord les instructions se trouvant en son sein puis on fait la vérification de la condition après ; si la condition est vraie on ré exécute la boucle si la condition est fausse on quitte la boucle. Il convient de signaler en programmation que cette boucle est rarement utilisée.

La boucle do while (condition) s'exécute au moins une seule fois.

Il signifie fait tant que la condition est vraie.

<u>N.B : En programmation ARDUINO il est conseillé d'utiliser la boucle while car elle marche toujours dans toutes les conditions. Les autres boucles surtout la boucle for crée très souvent des complications car elle est implémenté dans le void loop ().</u>

CHAPITRE8 LES FONCTIONS

1 LES FONCTIONS

1.1. PRESENTATION

De fois il peut arriver que vous faites un programme qui a beaucoup de lignes de codes et cela rend difficile sa lecture en cas de problème. De fois il arrive que vous répète des instructions et cela rend la programmation longue en effet par répétition des codes vous perdez le temps. Pour pallier a cela on utilise les fonctions.

Pratiquement une fonction un ensemble des codes qu'on groupe dans un seul paquet et qu'on pourra utiliser partout dans notre programme comme on veut.

Il existe deux types des fonctions :

- Les fonctions vides : ensemble d'instruction ne retournant pas de résultat
- Les fonctions typées : ils ont type donné et retournent un résultat

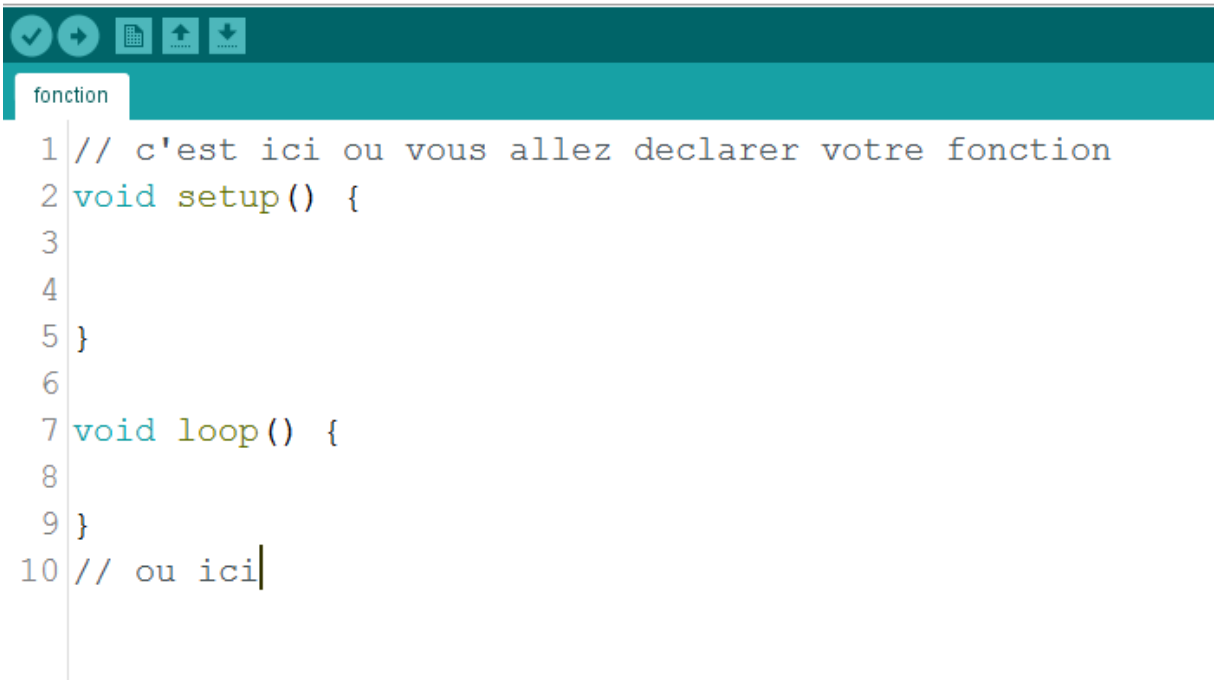
1.2. CARACTERISTIQUES D'UNE FONCTION

Une fonction est caractérisée par :

- **Son nom** : le nom de la fonction doit spécifier ce que fait la fonction : par exemple une fonction qui clignote une led vous pouvez lui donner le nom clignotement
- **Par son type** : la fonction peut être vide ou typée**.** Les fonctions vides sont déclarées avec le mot clé « **void** » et les fonctions typées sont déclarées avec le mot clé qui est le nom de leurs types.
- **Et ses paramètres :** les paramètres d'une fonction sont facultatifs une fonction peut ou ne pas les avoir.

1.3 COMMENT CREER UNE FONCTION

En arduino pour créer une fonction on doit aller à la fin c'est-à-dire après l'accolade fermente de void loop ou au début c'est-à-dire avant le void setup.



1.3 CREATION D'UNE FONCTION

1.3.1 CREATION D'UNE FONCTION VIDE SANS PARAMETRES

Par exemple créons une fonction permettant de faire clignoter une lampe se trouvant à la borne 13.

fonction | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

fonction §

```
1  #define led 13
2
3  void clignotement () {          /* ici nous créons une fonction vide */
4
5    digitalWrite (13,HIGH);
6    delay(1000);
7    digitalWrite (13,LOW);
8    delay(1000);
9  }
10 void setup() {
11   pinMode (led,OUTPUT);   /*ici nous confugurons notre broche qui va gérer la led en sortie*/
12
13 }
14
15 void loop() {
16   clignotement ();            /*ici nous appelons notre fonction*/
17 }
18
```

Pour utiliser la fonction la fonction on se sert de son nom. Par exemple ici haut nous avons appelé clignotement dans le void loop et les instructions se trouvant dans la fonction void clignotement créé dans le programme vont s'exécuter.

1.3.2 CREATION D'UNE FONCTION AVEC PARAMETRES

fonction §

```
1  int pinLed ;
2
3  void clignotement (int pinLed) {          /* ici nous créons une fonction vide avec parametre */
4
5    pinMode (pinLed,OUTPUT);
6    digitalWrite (pinLed,HIGH);
7    delay(1000);
8    digitalWrite (pinLed,LOW);
9    delay(1000);
10 }
11 void setup() {
12      /*ici nous confugurons notre broche qui va gérer la led en sortie*/
13
14 }
15
16 void loop() {
17   clignotement (2);            /*ici nous appelons notre fonction*/
18 }
19
```

Ici c’est la led se trouvant à la borne 2 qui va clignoter car c’est elle qui a été placé dans le paramètre de la fonction.

fonction §

```
int pinLed1 ;
int pinLed2 ;

void clignotement (int pinLed1,int pinLed2 ) {        /* ici nous créons une fonction vide avec deux parametre */
   pinMode (pinLed1,OUTPUT);
   pinMode (pinLed2,OUTPUT);
   digitalWrite (pinLed1,HIGH);
   digitalWrite (pinLed2,HIGH);
   delay(1000);
   digitalWrite (pinLed1,LOW);
   digitalWrite (pinLed2,LOW);
   delay(1000);
}
void setup() {
     /*ici nous confugurons notre broche qui va gérer la led en sortie*/

}

void loop() {
  clignotement (2,4);           /*ici nous appelons notre fonction*/
}
```

Cette fonction permet de faire clignoter deux lampes l'une se trouvant à la borne 2 et l'autre se trouvant à la borne 4.

Selon le choix on peut créer une fonction qui a plusieurs paramètres.

1.3.3 CREATION D'UNE FONCTION TYPEE

Les fonctions typés ce sont de fonction qui retourne un résultant

1.3.4 FONCTION TYPEE SANS PARAMETRE

Vérifier

fonction §

```
1
2
3
4 int ledAclignoter  () {         /* ici nous créons une fonction typee sans paramètres */
5   return 13;
6 }
7 void setup() {
8    pinMode (ledAclignoter() , OUTPUT);  /*ici nous confugurons notre broche qui va gérer la led en sortie*/
9
10 }
11
12 void loop() {
13   digitalWrite (ledAclignoter(),HIGH);           /*ici nous appelons notre fonction*/
14 }
15
```

Les fonctions typées retournent une certaine valeur par exemple ici haut nous avons créé une fonction typée retournant 13.

1.3.5 FONCTION TYPEE AVEC PARAMETRES

fonction §

```
1
2
3
4 int ledAclignoter  (int led) {         /* ici nous créons une fonction typee avec paramètre */
5   return led;
6 }
7 void setup() {
8    pinMode (ledAclignoter(2) , OUTPUT);  /*ici nous confugurons notre broche qui va gérer la led en sortie*/
9
10 }
11
12 void loop() {
13   digitalWrite (ledAclignoter(2),HIGH);           /*ici nous appelons notre fonction*/
14 }
15
```

Cette fonction permet de retourner la valeur se trouvant dans son paramètre.

Dans notre cas il retourne 2

CHAPITRE9 LE MACRO ET LE DEFINE

INTRODUCTION

En programmation C et ARDUINO on utilise le define « ***#define*** » pour remplace une valeur donnée par un nom ou une instruction ; on utilisera les macros lorsqu'on veut remplacer par un nom tout un ensemble d'instructions. Dans cette présente partie nous allons commencer par présenter le « ***#define*** » puis nous présenterons les macros.

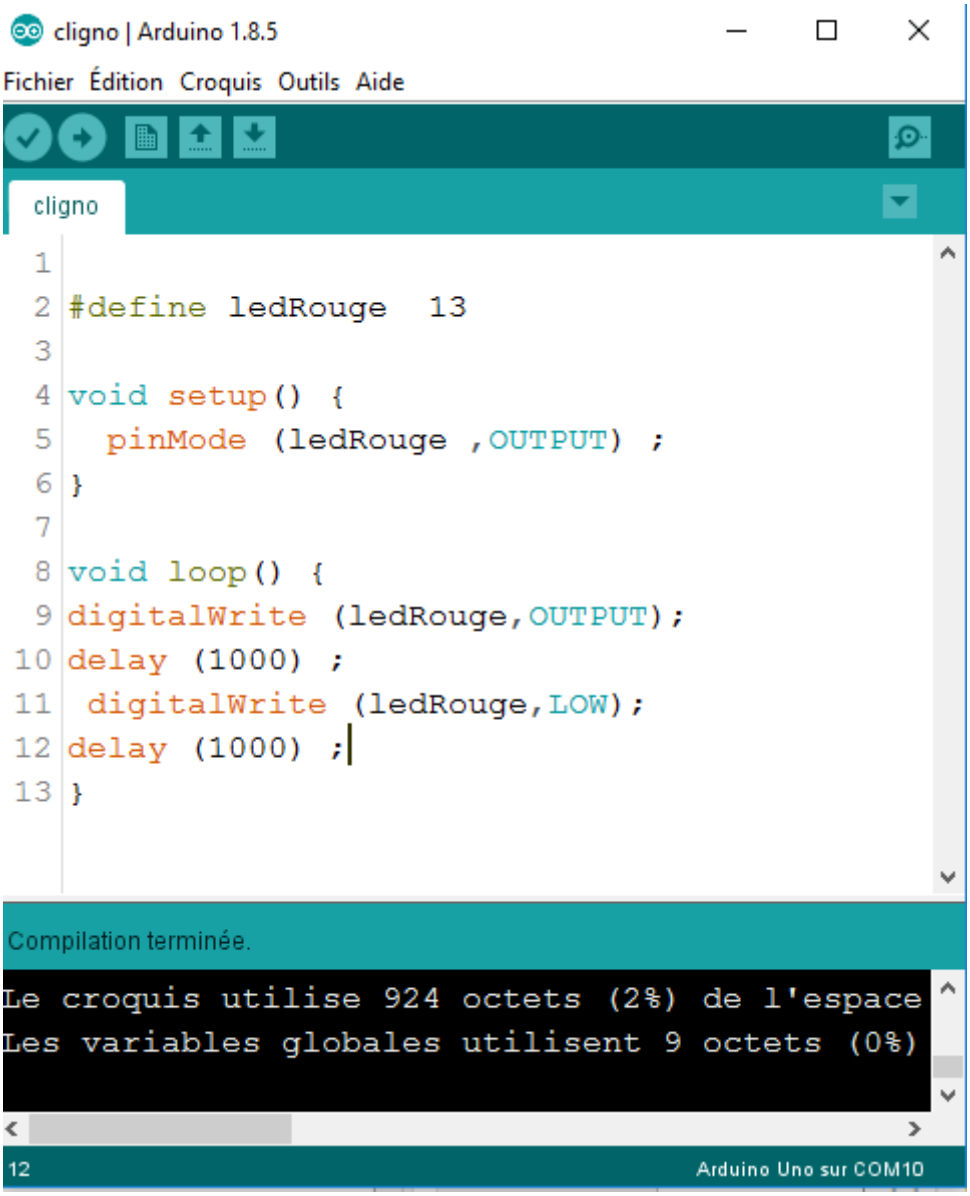
LE DEFINE

Montrons comment fonctionner par un exercice pour faire clignoter une led se trouvant à une borne 13 de la carte ARDUINO voici le code que l'on écrit :

cligno

```
1  void setup() {
2    pinMode (13 ,OUTPUT) ;
3  }
4
5  void loop() {
6  digitalWrite (pin,OUTPUT);
7  delay (1000) ;
8   digitalWrite (pin,LOW);
9  delay (1000) ;
10 }
```

Mais remarquons une chose que notre programme n'est pas explication : on ne sait sur la broche 13 se trouve quel équipement pour ce faire on peut utiliser le define comme ceci :



Dans le code ci-haut on utilise la notion define l'instruction :

```
#define ledRouge 13
```

Grace à cette instruction le programme remplacera par 13 partout où on écrit ledRouge cet ainsi qu'on voit facilement que le deuxième code est trop explicatif que le premier par le fait qu'on sait que c'est la ledRouge qui va clignoter. Par là on voit que la notion de define essais de donner une certaine abstraction dans notre code.

Le même code ci-haut peut se faire en utilisant une variable voir :

cligno

```
1
2 int ledRouge = 13;
3
4 void setup() {
5   pinMode (ledRouge ,OUTPUT) ;
6 }
7
8 void loop() {
9 digitalWrite (ledRouge,OUTPUT);
10 delay (1000) ;
11  digitalWrite (ledRouge,LOW);
12 delay (1000) ;
13 }
```

Compilation terminée.

```
Le croquis utilise 936 octets (2%) de l'espace
Les variables globales utilisent 9 octets (0%)
```

2 Arduino Uno sur COM10

En utilisant le variable vous pouvez voir nous avons un programme de 936 Octets mais en utilisant le define on a un programme de 924 Octets par cette différence vous voyez que le define ne consomme pas de mémoire c'est juste une simple définition dans le programme. Donc en utilisant le define vous remplacez un nom par une valeur.

LE MACRO

Pour le define on remplace une valeur ou une instruction par un nom ; mais pour le macros on remplace tout un ensemble d'instruction par un mot :

LE MACRO SANS PARAMETRE

clignotement §

```
#define clignotement() digitalWrite (13,HIGH);\
                        delay (1000) ; \
                        digitalWrite (13,LOW);\
                        delay (1000) ;
```

Avec ce programme on remplacera le mot clignotement() par l'ensemble de code on peut voir dans ce code qu'une led se trouvant à la borne 13 va clignoter :

clignotement §

```
#define clignotement() digitalWrite (13,HIGH);\
                        delay (1000) ; \
                        digitalWrite (13,LOW);\
                        delay (1000) ;



void setup() {

 pinMode (13, OUTPUT);
}

void loop() {

  clignotement() ;

}
```

Compilation terminée.

```
Le croquis utilise 924 octets (2%) de l'espace de stockage de p
Les variables globales utilisent 9 octets (0%) de mémoire dynam
```

Dans ce programme on appelle juste le mot clignotement() qui appelle à son tour l'ensemble de code énumérés lors de la création du macros à l'entête du programme.

LE MACRO AVEC PARAMETRES

Pour le macros avec paramètres voici la déclaration :

```
#define clignotement(pin) digitalWrite (pin,HIGH);\
                          delay (1000) ; \
                          digitalWrite (pin,LOW);\
                          delay (1000) ;
```

Dans ce cas ci-dessus on a un macros avec paramètres.

Par la suite on peut faire clignoter une led juste en utilisant la notion de macros : ici on fait clignoter une led se trouvant à la borne 13.

clignotement

```
#define clignotement(pin) digitalWrite (pin,HIGH);\
                          delay (1000) ; \
                          digitalWrite (pin,LOW);\
                          delay (1000) ;



void setup() {

 pinMode (13, OUTPUT);
}

void loop() {

  clignotement(13) ;

}
```

Compilation terminée.

```
Le croquis utilise 924 octets (2%) de l'espace de stockage de p
Les variables globales utilisent 9 octets (0%) de mémoire dynam
```

Pourquoi on utilise le macros : il consomme peu d'espace dans la mémoire. Pour vous faire voir voici un programme qui fait la même mais qui consomme trop des mémoires :

clignotement §

```
1
2   void clignotement (int pin) {
3     digitalWrite (pin,HIGH);
4     delay (1000) ;
5     digitalWrite (pin,LOW);
6     delay (1000) ;
7
8   }
9
10 void setup() {
11
12  pinMode (13, OUTPUT);
13 }
14
15 void loop() {
16
17   clignotement(13) ;
18
19 }
20
21
```

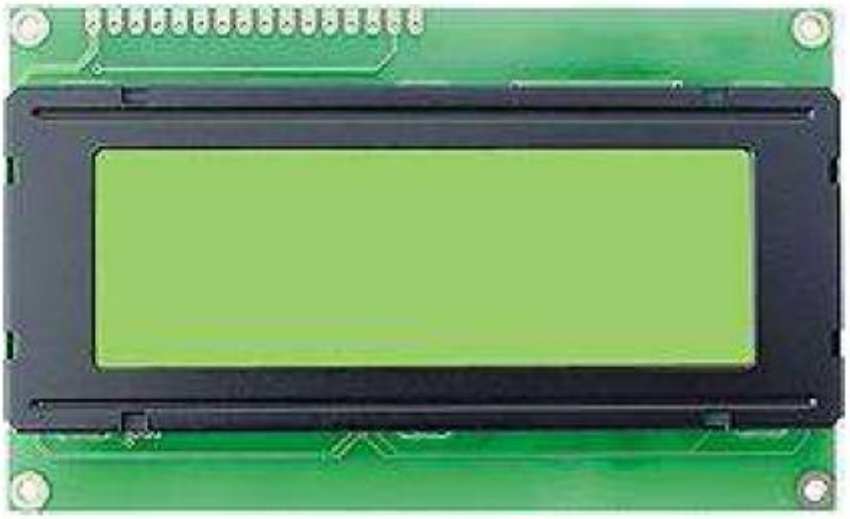
Compilation terminée.

```
Le croquis utilise 936 octets (2%) de l'espace de stockage de progr
Les variables globales utilisent 9 octets (0%) de mémoire dynamique
```

Ici vous voyez très qu'en utilisant la fonction le programme est de 936 Octets mais en utilisant le macros on a un programme de 924 Octets. Chers amis je pense vous avez bien compris le macros et le define.

CHAPITRE10 GESTION DE L'ECRAN LCD

Un écran LCD est un écran utilisant le principe des cristaux liquides pour afficher.



Il est utilisé en électronique pour donner des informations aux utilisateurs (ici il s'agit de texte)

L'arduino permet de gérer de cet écran avec une facilité. Cet écran est constitué par un certain nombre de broches qui sont présentées dans le tableau suivant :

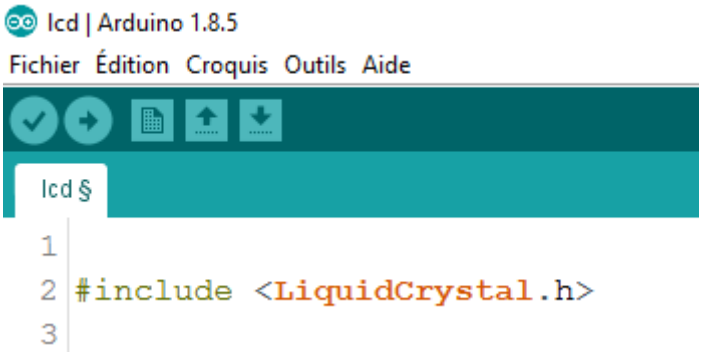
BROCHES	FONCTIONS DES BROCHES
Alimentation positive (Vcc)	Ces broches assurent l'alimentation électrique de l'écran LCD
Masse (GND)	
Contraste (VEE)	Permettant de régler le contraste de l'écran : • Pour VEE = 0V le contraste est maximal • Pour VEE = 5V le contraste est minimal
Enable (E)	Entrée de validation : elle permet de valider les données sur front descendant. Lorsque E est à 0 (E = 0) le bus de données est à l'état haute impédance
Register Select (RS)	Il permet d'indiquer à l'afficheur si l'on souhaite réaliser une commande (RS = 0) par des instructions spécifiques ou écrire une donnée (envoie du code caractère à afficher) sur le bus (RS = 1)
Read/Write (R/W)	Il s'agit d'une entrée de lecture et d'écriture • Si R/W = 1 : lecture • Si R/W = 0 : écriture

D0…D7	Bus des données bidirectionnel. Il permet de transférer les instructions ou les données à l'afficheur LCD.

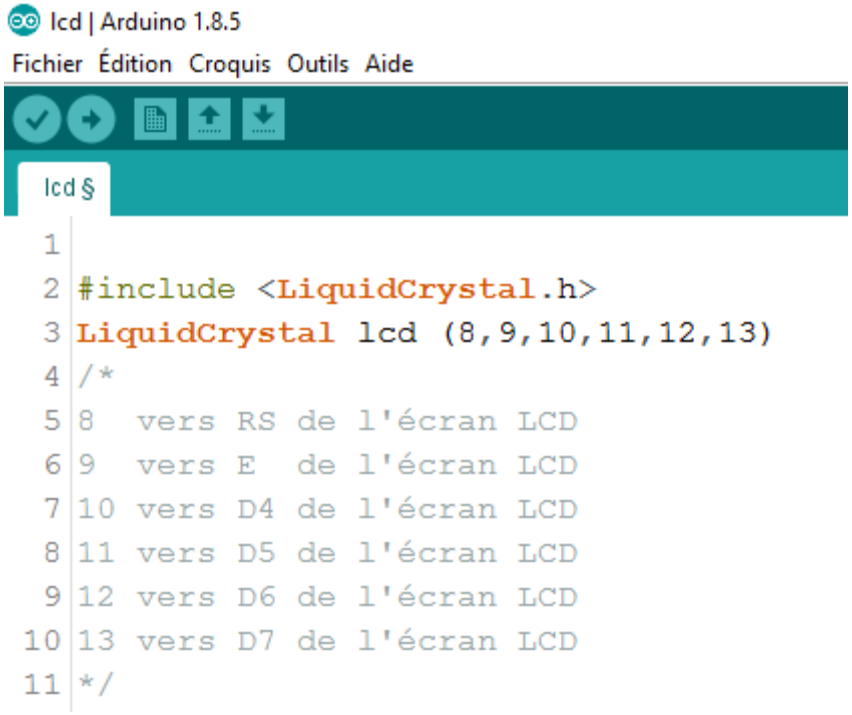
PROCEDURE DE GESTION D'UN ECRAN AVEC ARDUINO

Pour gérer un écran LCD avec arduino on doit :

1. Appeler la libraire de gestion de l'écran à l'entête du programme



2. Appeler le constructeur de l'écran



3. Initialiser l'écran :

lcd | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

Ouvrir

lcd §

```
1
2  #include <LiquidCrystal.h>
3  LiquidCrystal lcd (8,9,10,11,12,13)
4  /*
5  8  vers RS de l'écran LCD
6  9  vers E  de l'écran LCD
7  10 vers D4 de l'écran LCD
8  11 vers D5 de l'écran LCD
9  12 vers D6 de l'écran LCD
10 13 vers D7 de l'écran LCD
11 */
12 void setup() {
13   lcd.begin (16,2);
14   /*
15    * il s'agit d'un écran ayant :
16    *  16 colonnes
17    *  2 lignes
18    */
19
```

4. Manipulation

lcd | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

lcd §

```
1
2 #include <LiquidCrystal.h>
3 LiquidCrystal lcd (8,9,10,11,12,13)
4
5 void setup() {
6   lcd.begin (16,2);
7
8
9 }
10
11 void loop() {
12 lcd.setCursor (0,0);
13 lcd.print ("ECRAN LCD");
14
15 }
```

lcd.setCursor (0,0) ; permet de positionner le curseur à la position initiale

Pour afficher les variables rien de plus difficile :

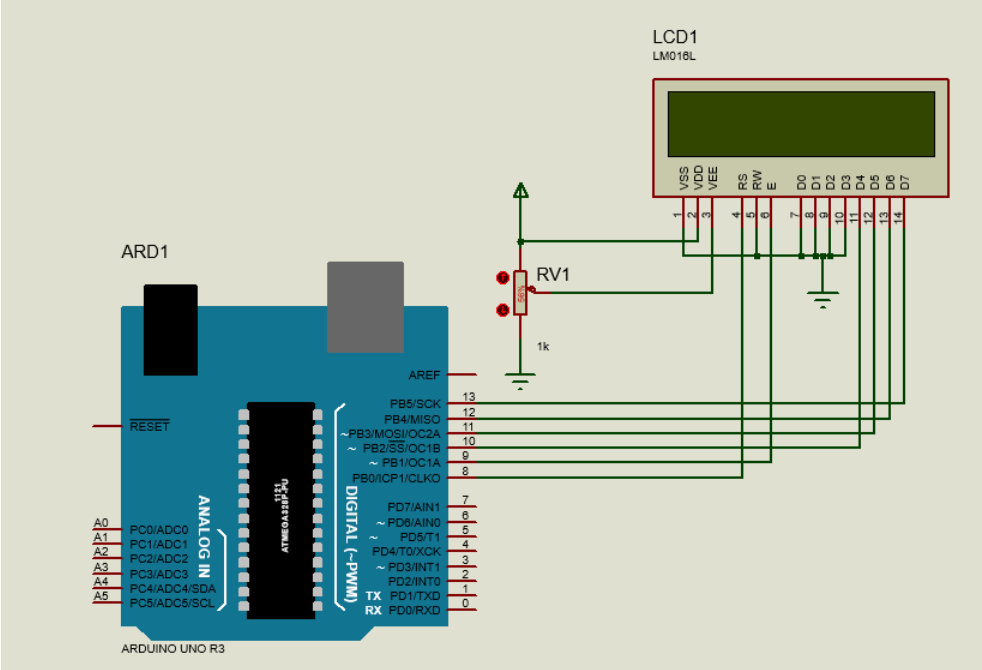
lcd | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

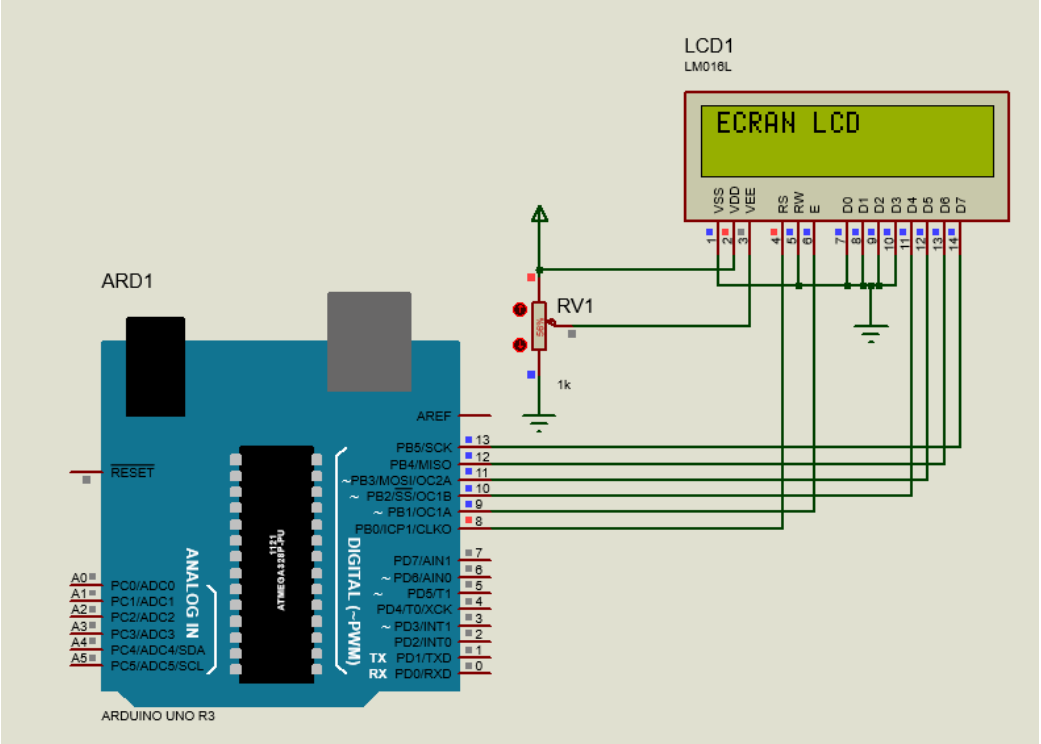
lcd §

```
1
2  #include <LiquidCrystal.h>
3  LiquidCrystal lcd (8,9,10,11,12,13)
4  int variable = 24 ;
5
6  void setup() {
7    lcd.begin (16,2);
8
9
10 }
11
12 void loop() {
13 lcd.setCursor (0,0);
14 lcd.print ("ECRAN LCD");
15
16 lcd.setCursor (0,1) // ici on se positionne a la deuxime ligne premiere colonne
17 lcd.print (variable);// ici on affiche la variable
18
19 }
```

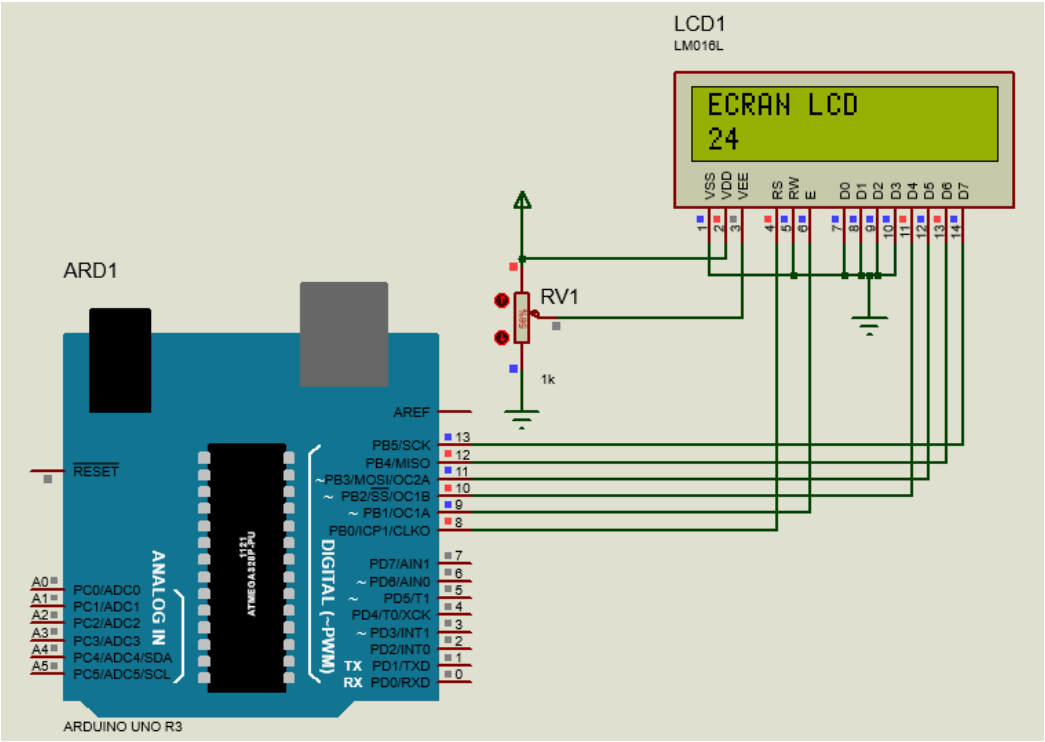
SCHEMAS DE CABLAGE DE L'ECRAN LCD



QUELQUES RESULTATS DE SIMULATION



La simulation ci-dessus c'est la simulation du premier code.



La simulation ci-dessus c'est la simulation du deuxième code

LES AUTRES FONCTIONS PERMETTANT DE GERER UN ECRAN LCD

Il existe plusieurs fonctions permettant à un écran d'effectuer une tache donnée parmi eux on peut citer :

1. noDisplay() : cette fonction fait disparaitre les textes au niveau de l'écran LCD
2. display() : cette fonction fait apparaitre les textes sur l'écran LCD
   exemple d'utiisation de la fonction display et la fonction noDisplay :

```
#include <LiquidCrystal.h>
LiquidCrystal lcd (8,9,10,11,12,13) ;
int variable = 24 ;

void setup() {
 lcd.begin (16,2) ;


}

void loop() {

  lcd.display() ;
  lcd.setCursor (0,0) ;
  lcd.print ("ECRAN LCD");

  lcd.setCursor (0,1) ;
  lcd.print (variable);
  delay (2000) ;
  lcd.noDisplay () ;
  delay(2000);

}
```

   Le code permet d'afficher les informations sur l'écran pendant 2 secondes puis fait disparaitre ces informations donc n'affiche rien sur l'écran pendant 2 secondes et cela dans une boucle infinie.

3. clear() : cette fonction permet d'efface l'écran LCD

codelcd §

```
#include <LiquidCrystal.h>
LiquidCrystal lcd (8,9,10,11,12,13) ;
int variable = 24 ;

void setup() {
 lcd.begin (16,2) ;

 lcd.print ("ECRAN LCD");
 delay (2000) ;

  lcd.clear ();
}

void loop() {



}
```

La fonction clear () efface les informations ce très différent de la fonction noDisplay() elle qui fait disparaitre les informations de l'écran même si parfois les résultats peuvent êtres les mêmes. Par ces deux codes voyons la différence des fonctions clear() et noDisplay()

codelcd §

```
#include <LiquidCrystal.h>
LiquidCrystal lcd (8,9,10,11,12,13) ;
int variable = 24 ;

void setup() {
 lcd.begin (16,2) ;

 lcd.print ("ECRAN LCD");
 delay (2000) ;

  lcd.clear ();
}

void loop() {

lcd.clear ();
delay (1000) ;
lcd.clear ();
delay (1000) ;


}
```

Pour ce code utilisant le clear() l'écran va juste afficher pendant deux secondes puis sera effacé pour ne plus jamais s'alumer

```
#include <LiquidCrystal.h>
LiquidCrystal lcd (8,9,10,11,12,13) ;
int variable = 24 ;

void setup() {
 lcd.begin (16,2) ;

 lcd.print ("ECRAN LCD");

}

void loop() {

lcd.display ();
delay (1000) ;
lcd.noDisplay ();
delay (1000) ;


}
```

Dans le code ci-dessus l'écran va plutôt clignoter.
Par ces exemples la différence entre es fonctions clear() et noDisplay() est claire.

4. home () : cette fonction met le curseur de l'écran au coordonnées (x,y) = (0,0)

   Cette fonction permet de placer le curseur au point (0,0) elle est équivalente à la fonction : setCursor (0 ,0).
   Dans le code on l'écrit :

```
lcd.home() ;
```

5. scrollDisplayLeft() : cette fonction fait aller le texte écrit par l'écran LCD vers la gauche

codelcd §

```
#include <LiquidCrystal.h>
LiquidCrystal lcd (8,9,10,11,12,13) ;
int variable = 24 ;

void setup() {
 lcd.begin (16,2) ;

 lcd.print ("ECRAN LCD");

}

void loop() {
// lcd.home();
 lcd.print ("ECRAN LCD");
 lcd.scrollDisplayLeft() ;
  delay (2000);
}
```

6. scrollDisplayRight() : cette fonction fait aller le texte écrit par l’écran LCD vers la droite

codelcd §

```
#include <LiquidCrystal.h>
LiquidCrystal lcd (8,9,10,11,12,13) ;
int variable = 24 ;

void setup() {
 lcd.begin (16,2) ;

 lcd.print ("ECRAN LCD");

}

void loop() {
// lcd.home();
 lcd.print ("ECRAN LCD");
 lcd.scrollDisplayRight() ;
  delay (2000);
}
```

CREATION DE SES PROPRES CARACTERES

Pour arriver à créer un caractère on suit les procédures suivants :

1. On crée un tableau qui se sera en fonction du caractère que l'on veut créer :

0	0	0	0	0
X	0	0	0	X
0	0	0	0	0
0	0	0	0	0
X	0	0	0	X
0	X	X	X	0
0	0	0	0	0
0	0	0	0	0

Cette photo a été tré livre du site du zero traitant sur arduino

```
byte creationPropre[8] = {
B11111,
B10001,
B00000,
B00000,
B10001,
B01110,
B00000,
B00000
};
```

Ici on crée un tableau ce qui constitue un seule caractère. Ces sont les éléments du tableau 1 ou 0 qui détermine comment sera le caractère.

2. On apprend à l'écran le caractère en question avec la fonction :

```
lcd.createChar(0, creationPropre);
```

On apprend ce caractère à l'écran en lui affectant une adresse de 0.

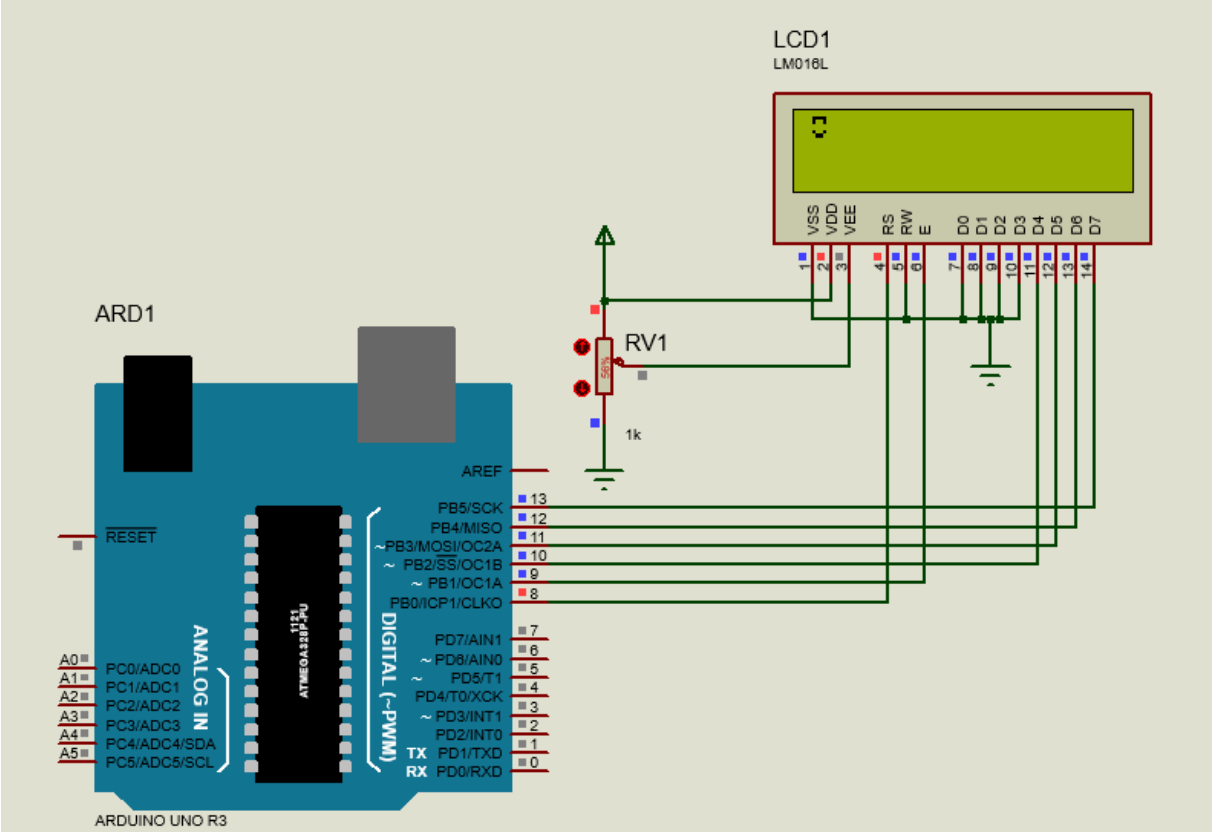
3. On affiche enfin le caractère avec la fonction :

```
lcd.write((uint8_t) 0);
```

Enfin on a le code suivant :

codelcd §

```
#include <LiquidCrystal.h>
LiquidCrystal lcd (8,9,10,11,12,13) ;
byte creationPropre[8] = {
B11111,
B10001,
B00000,
B00000,
B10001,
B01110,
B00000,
B00000
};
void setup() {
 lcd.createChar(0, creationPropre);
 lcd.begin (16,2) ;
 lcd.home () ;
 lcd.write((uint8_t) 0);

}

void loop() {

}
```

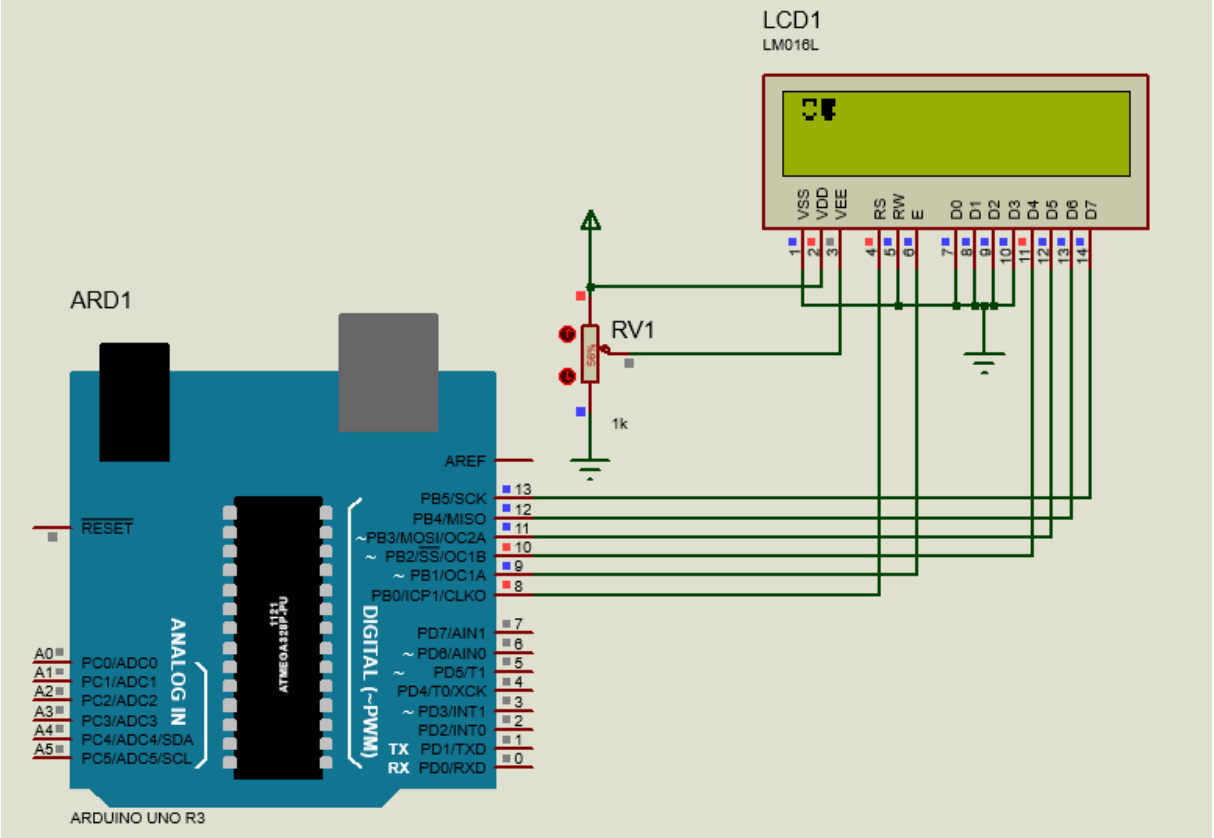


Départ ces résultats de simulation on voit que l’écran affiche le caractère que l’on a créé.

Si on veut afficher deux caractères rien de plus compliqué il suffit juste de donner à chaque caractère une adresse qui permettra d’accéder à ces informations.

codelcd §

```
#include <LiquidCrystal.h>
LiquidCrystal lcd (8,9,10,11,12,13) ;
byte creationPropre[8] = {
B11111,
B10001,
B00000,
B00000,
B10001,
B01110,
B00000,
B00000
};

byte creationAutre[8] = {
B11111,
B11111,
B11110,
B11110,
B11111,
B01110,
B00000,
B00000
};
void setup() {
 lcd.createChar(0, creationPropre);
 lcd.createChar(1, creationAutre);
 lcd.begin (16,2) ;
 lcd.home () ;
 lcd.write((uint8_t) 0);
 lcd.write((uint8_t) 1);

}

void loop() {

}
```



Départ la simulation on voit très bien que les caractères s’affiche très bien. C’est par cette simulation que nous mettons fin à notre traité merci et bonne lecture.

EXERCICE : FAITE UN HORLOGE SIMPLE SANS REGLAGE

montre §

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(8,9,10,11,12,13);

int seconde;
int mint;
int heure;

  void affiche () {
  lcd.setCursor(0,0);
  lcd.print(heure);
  lcd.print(" ");

  lcd.setCursor(3,0);
  lcd.print(":");
  lcd.setCursor(4,0);
  lcd.print(mint);
  lcd.print(" ");
  lcd.setCursor(7,0);
  lcd.print(":");
  lcd.setCursor(8,0);
  lcd.print(sec);
  lcd.print(" ");

  }
```

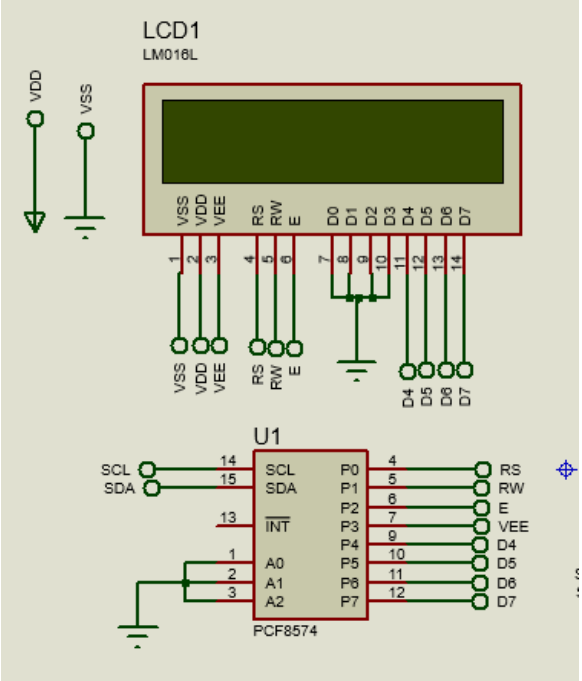
```
void loop() {

      if (seconde == 60) {
         mint++ ;
         seconde =0 ;
      }

       if (mint == 60) {
         mint =0  ;
         heure++ ;
      }

       if (heure == 24) {
         heure =0 ;

      }

       affiche () ;
       delay (400) ;
       sec++ ;
  }
```

Avec ce programme on fait une horloge simplement.

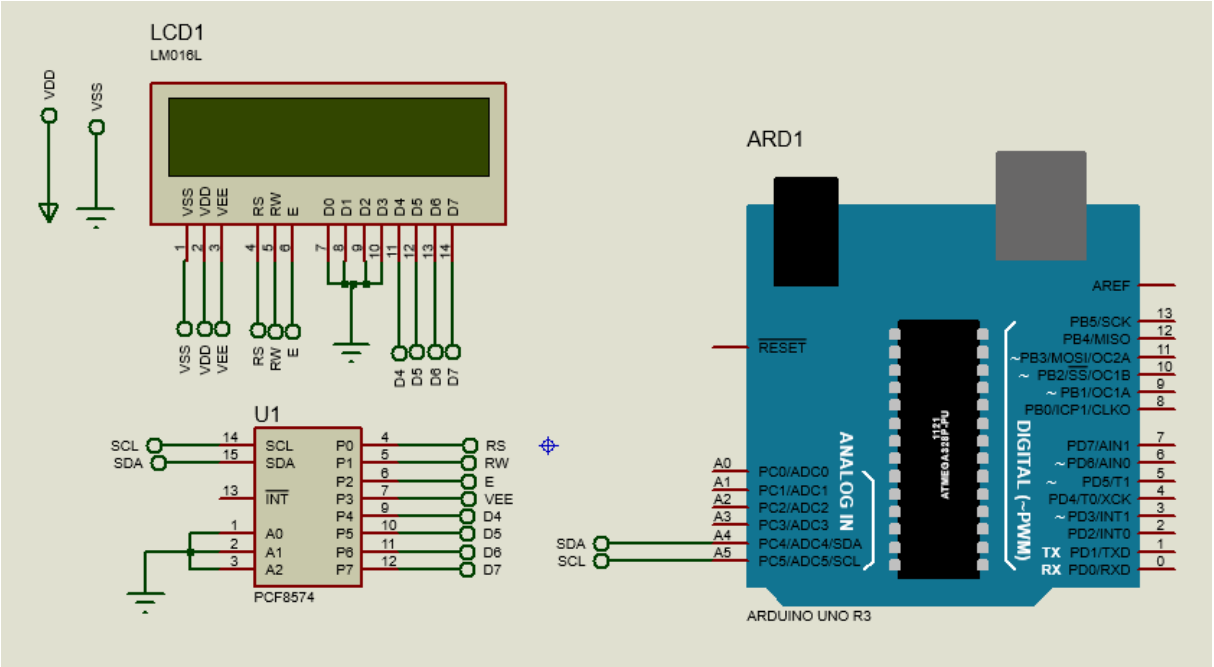
ECRAN LCD AVEC I2C

Comme vous avez bien constaté notre écran LCD prend jusqu'à six broches de la carte ARDUINO. Cela peut devenir un problème lorsque nous travaillons sur un projet qui a beaucoup de sorties et beaucoup d'entrées car la LCD prend en elle seule 6 broches ; en plus dans la pratique câblé cet écran de fois cet encombrant câblé 6 fils en plus d'autres fils d'alimentation et le potentiomètre de réglage. Il est possible de palier à cela en utilisant un écran avec une puce utilisant la communication I2C. Selon les variantes de l'écran on peut utiliser : PCF 8574, PCF 8574A ou PCF 8574T

Schéma de cablage d'un écran LCD avec I2C :



Le schéma ci-dessus représente un écran LCD avec I2C utilisant le PCF 8574.



CODE

Pour le code il faudra une librairie permettant de gérer cet écran LCD I2C en arduino.

Arduino-LiquidCrystal-I2C-library-master	14/12/2016 13:11	WinRAR archive	8 Ko

Il faut installer cette librairie sur ARDUINO c'est cette librairie qui permet de gérer un écran LCD avec I2C.

PRESENTATION DU CODE

avecI2C §

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
```

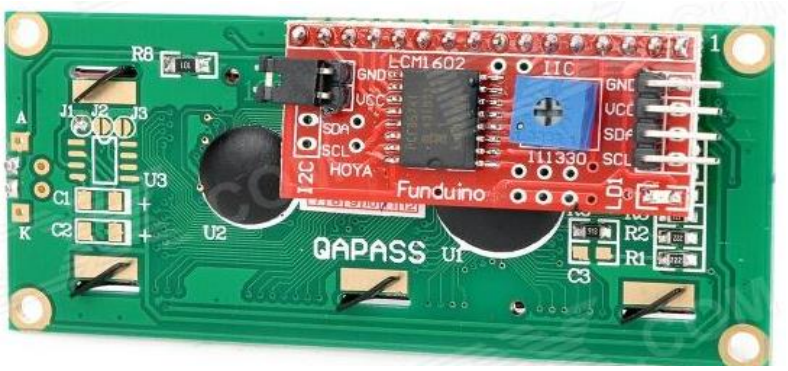
En codant on commence par l'implémentation des bibliothèques ***<Wire.h>*** pour gérer la communication I2C et ***<LiquidCrystal_I2C.h>***

Puis par après on doit préciser l'adresse de la puce I2C de l'écran. L'adresse varie selon que l'on a utilisé un écran avec PCF 8574, PCF 8574A ou PCF 8574T.

Lorsque vous acheter l'écran par défaut l'adresse est de :

PUCE DU LCD	ADRESSE
PCF 8574	0x20
PCF 8574A	0x3F
PCF 8574T	0x27

Vous pouvez savoir le nom de la puce en regardant sur le dos de l'écran :



Selon le circuit PCF qui pilote votre écran vous devez alors dans le code préciser l'adresse de votre écran LCD

```
#define I2C_ADDR 0x20  //decommenter ceci en cas d'utilisation du PCF8574

 //#define I2C_ADDR    0x27 // decommenter ceci en cas d'utilisation du PCF8574T
// #define I2C_ADDR    0x3F ////decommenter ceci en cas d'utilisation du PCF8574A
```

Ici nous avons laissé l'adresse du PCF8574 car c'est le circuit que nous avons puis utiliser.

```
#define BACKLIGHT_PIN     3
#define En_pin  2
#define Rw_pin  1
#define Rs_pin  0
#define D4_pin  4
#define D5_pin  5
#define D6_pin  6
#define D7_pin  7

LiquidCrystal_I2C  lcd (I2C_ADDR,En_pin,Rw_pin,Rs_pin,D4_pin,D5_pin,D6_pin,D7_pin);
```

Ici on spécifie les broches de l'écran tout en créant aussi l'objet de la classe **LiquidCrystal_I2C.**

Le nom de l'objet c'est **lcd**.

Par après vienne les configurations de l'écran :

```
lcd.begin (16,2); // specification du nombre de colonnes et celui des lignes
```

Ici on précise qu'il s'agit d'un écran LCD avec deux lignes et 16 colonnes.

```
lcd.setBacklight(HIGH);
```

Cette instruction permet d'activer le retro éclairage de la LCD. Si vous écrivez

```
lcd.setBacklight(LOW);
```

Vous désactivez le retro éclairage.

Pour le reste c'est-à-dire l'affichage le défilement, … les instructions de l'écran sans I2C restent les mêmes pour l'écran avec I2C.

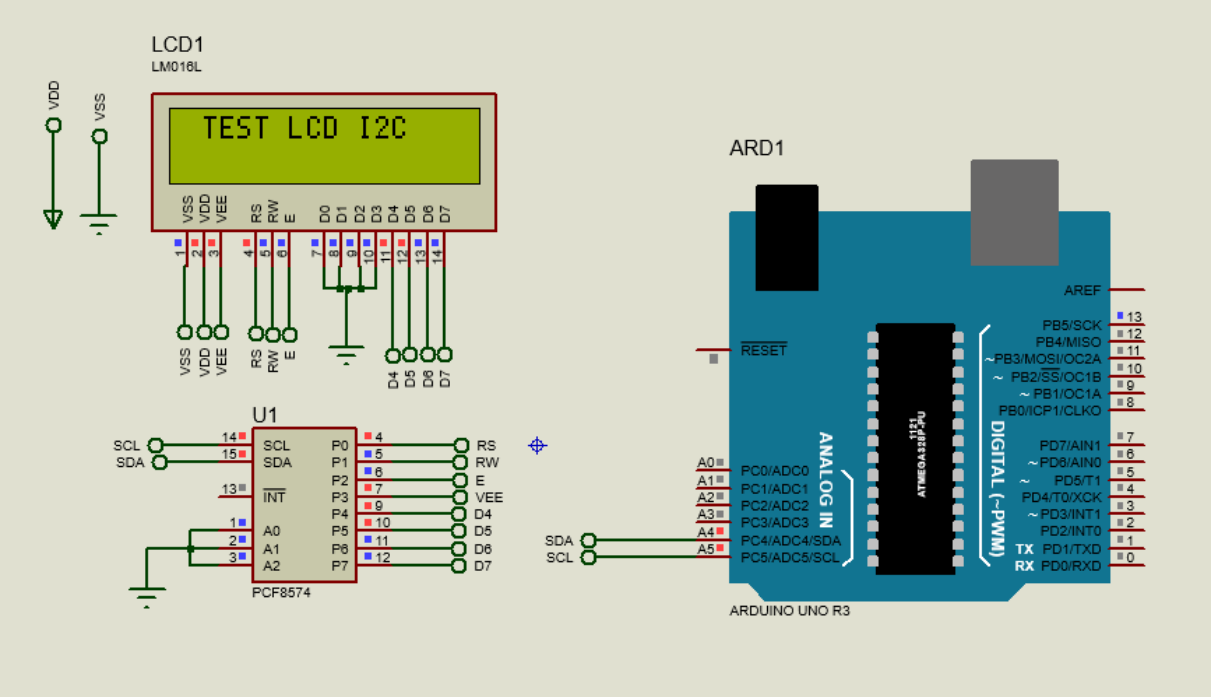
Voici le code :

avecI2C §

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
/*
 *  Lire sur le dos du LCD si son circuit integré est un shift register
 *  de type PCF8574 (adresse 0x20) ou PCF8574A (adresse 0x38) ou PCF8574T (adresse 0x27)
 */
#define I2C_ADDR 0x20  //decommenter ceci en cas d'utilisation du PCF8574

 //#define I2C_ADDR    0x27 // decommenter ceci en cas d'utilisation du PCF8574T
// #define I2C_ADDR    0x3F ////decommenter ceci en cas d'utilisation du PCF8574A
// <<----- Ajouter votre adresse.
/*
 * Pour connaitre cette adresse, utiiser le code I2C_scanner
 */
 // Broches du LCD ------> Broches du I2C (register)
#define BACKLIGHT_PIN     3
#define En_pin  2
#define Rw_pin  1
#define Rs_pin  0
#define D4_pin  4
#define D5_pin  5
```

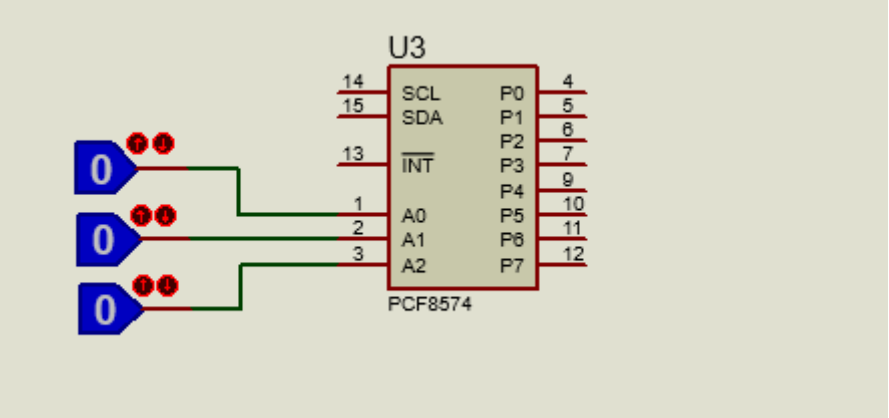
```
#define D6_pin  6
#define D7_pin  7

LiquidCrystal_I2C  lcd (I2C_ADDR,En_pin,Rw_pin,Rs_pin,D4_pin,D5_pin,D6_pin,D7_pin);

void setup()
{
 lcd.begin (16,2);

lcd.setBacklightPin(BACKLIGHT_PIN,POSITIVE);

lcd.setBacklight(LOW);
lcd.home (); // aller à home

 lcd.print(0,0);
 lcd.print("TEST LCD I2C");
 pinMode (13,OUTPUT);
 pinMode (7,INPUT);

}

  long distance ;

void loop()
{




}

```

RESULTAT DE SIMULATION



GESTION DE PLUSIEURS ECRAN AVEC I2C

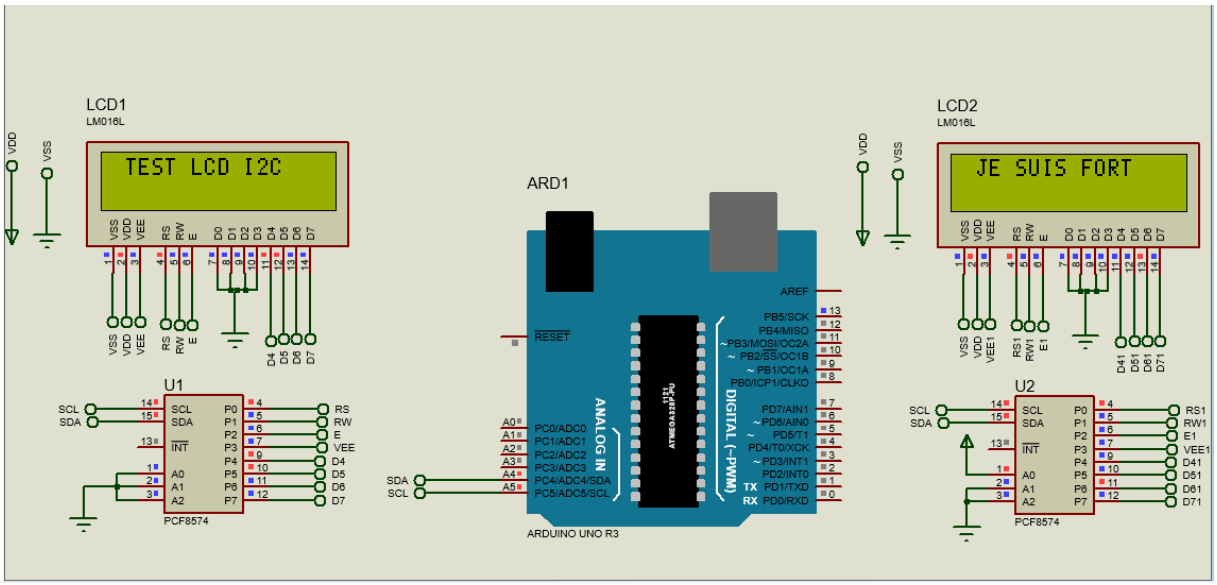
Pour gérer plusieurs écrans il suffit juste de spécifier l'adresse.



Les broches A0, A1 et A2 permettent de fixer l'adresse physique.

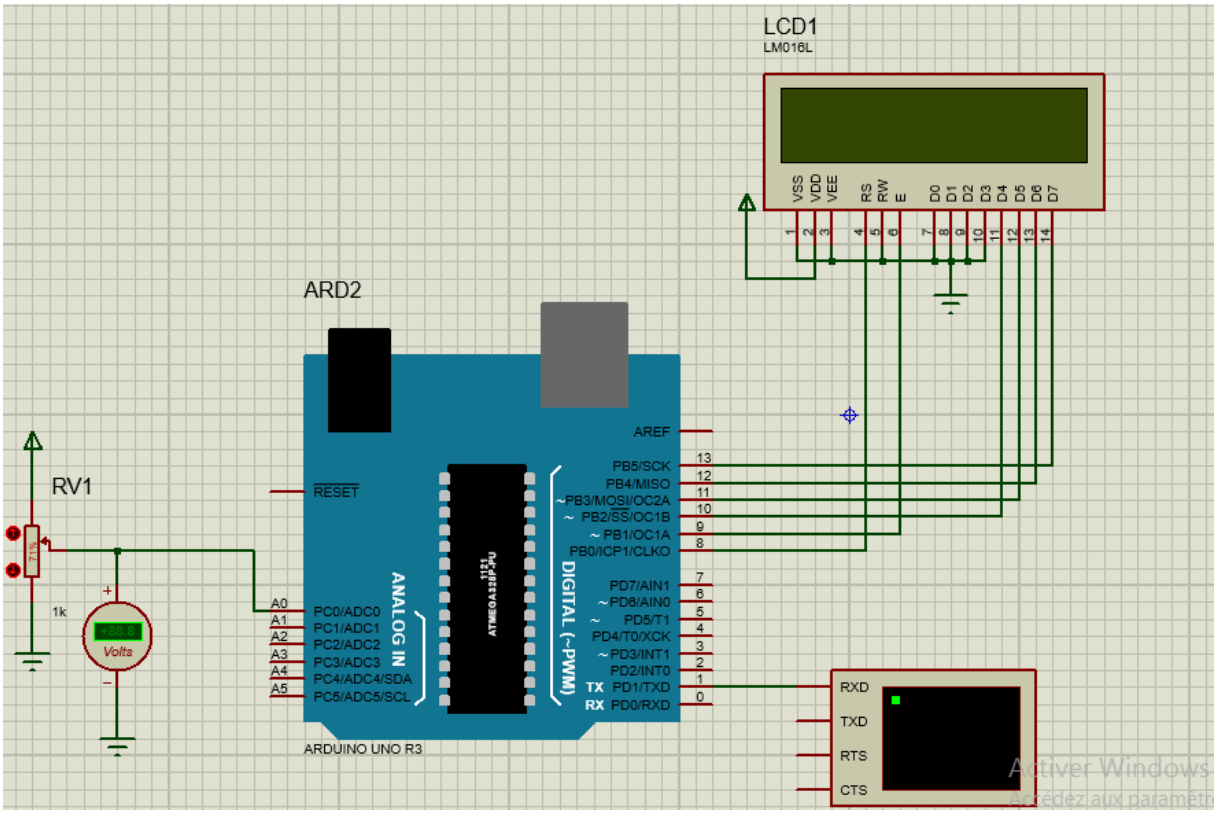
```
1 #include <Wire.h>
2 #include <LiquidCrystal_I2C.h>
3 /*
4  *  Lire sur le dos du LCD si son circuit integré est un shift register
5  *  de type PCF8574 (adresse 0x20) ou PCF8574A (adresse 0x38) ou PCF8574T (adresse 0x27)
6  */
7 #define I2C_ADDR 0x20  //decommenter ceci en cas d'utilisation du PCF8574
8 #define I2C_ADDR2 0x21
9
10  //#define I2C_ADDR    0x27 // decommenter ceci en cas d'utilisation du PCF8574T
11 // #define I2C_ADDR    0x3F ////decommenter ceci en cas d'utilisation du PCF8574A
12 // <<----- Ajouter votre adresse.
13 /*
14  * Pour connaitre cette adresse, utiiser le code I2C_scanner
15  */
16  // Broches du LCD ------> Broches du I2C (register)
17 #define BACKLIGHT_PIN     3
18 #define En_pin  2
19 #define Rw_pin  1
20 #define Rs_pin  0
21 #define D4_pin  4
22 #define D5_pin  5
23 #define D6_pin  6
24 #define D7_pin  7
25
26 LiquidCrystal_I2C  lcd (I2C_ADDR,En_pin,Rw_pin,Rs_pin,D4_pin,D5_pin,D6_pin,D7_pin);
27 LiquidCrystal_I2C  lcd1 (I2C_ADDR2,En_pin,Rw_pin,Rs_pin,D4_pin,D5_pin,D6_pin,D7_pin);
28
29 void setup()
30 {
31  lcd.begin (16,2);
32  lcd1.begin (16,2);
33
34 lcd.setBacklightPin(BACKLIGHT_PIN,POSITIVE);
35 lcd1.setBacklightPin(BACKLIGHT_PIN,POSITIVE);
36
37 lcd.setBacklight(LOW);
38 lcd1.setBacklight(LOW);
39
40 lcd.home (); // aller à home
41 lcd1.home (); // aller à home
42
43
44  lcd.print(0,0);
45  lcd1.print(0,0);
46
47  lcd.print("TEST LCD I2C");
48  lcd1.print("JE SUIS FORT");
49
50  pinMode (13,OUTPUT);
51  pinMode (7,INPUT);
52
53 }
54
55   long distance ;
56
57 void loop()
58 {
59
60 }
61
```

Avec ce code ci-dessus on gère deux écrans avec I2C.



Avec l'écran I2C on peut arriver à commander jusqu'à 7écrans différents en utilisant juste le SDA et le SCL et bien compris chaque écran doit avoir une adresse spécifique.

CHAPITRE11. CONVERSION ANALOGIQUE NUMERIQUE



CODE SOURCE

adc_arduino | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

adc_arduino §

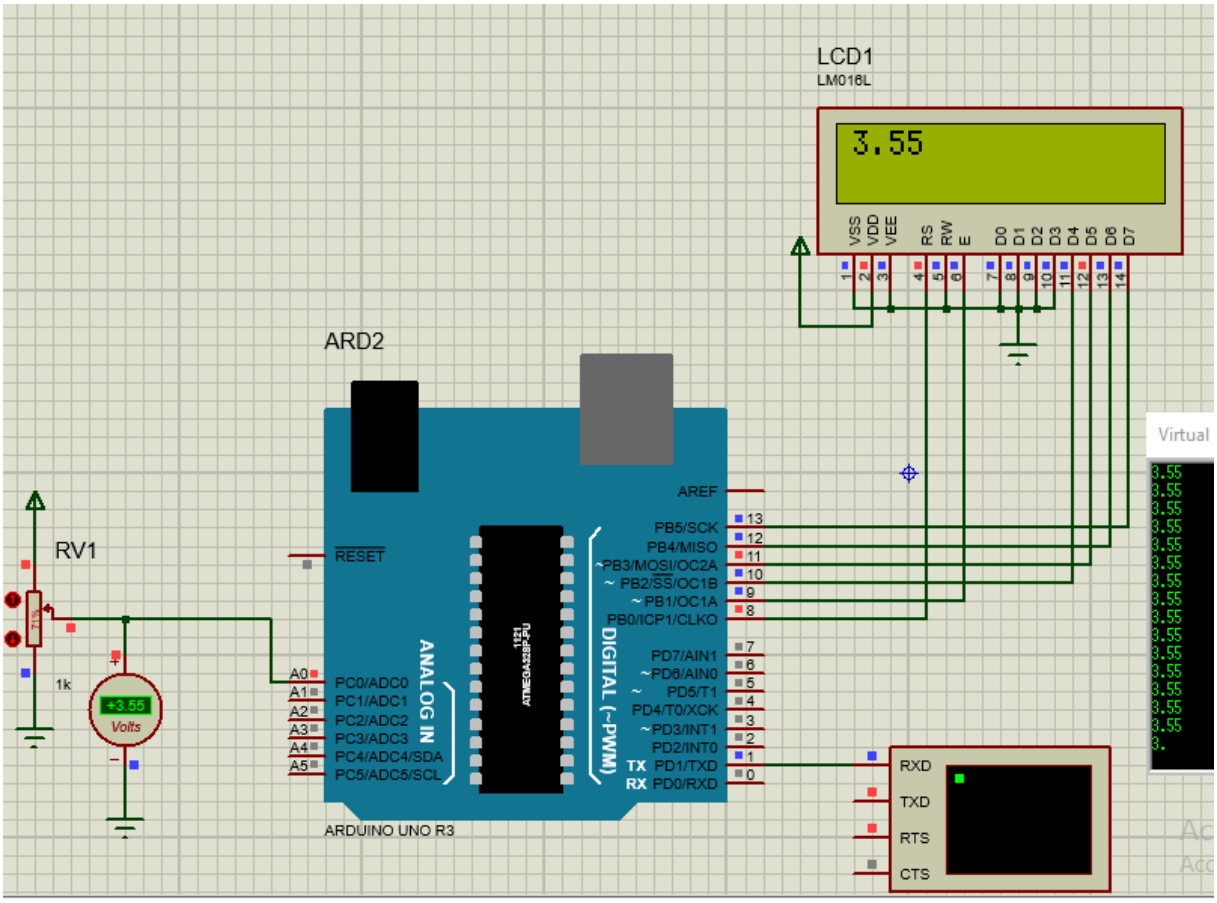
```
#include <LiquidCrystal.h>

float tensionBinaireMesuree;
float tensionDecimaleMesuree;

LiquidCrystal lcd (8,9,10,11,12,13);

void setup() {

Serial.begin (9600);

}

void loop() {

 tensionBinaireMesuree = analogRead(A0);
 tensionDecimaleMesuree = tensionBinaireMesuree*0.0048828;
  Serial.println (tensionDecimaleMesuree);
   lcd.setCursor (0,0);

   lcd.print (tensionDecimaleMesuree);

}
```

La fonction : analogRead(nom de la broche) permet de lire la valeur de la tension

Dans les parenthèses on met le nom de la broche d'entrée de la tension cette tension est en binaire

Allant de (0 à 1023) soit de (0 à 5V).

ESSAIS


LCD1
LM016L
3.55
VSS VDD VEE RS RW E D0 D1 D2 D3 D4 D5 D6 D7
ARD2
RV1
1k
+3.55
Volts
RESET
AREF
PB5/SCK
PB4/MISO
~PB3/MOSI/OC2A
~ PB2/SS/OC1B
~ PB1/OC1A
PB0/ICP1/CLKO
PD7/AIN1
~PD6/AIN0
~ PD5/T1
PD4/T0/XCK
~PD3/INT1
PD2/INT0
TX PD1/TXD
RX PD0/RXD
A0 PC0/ADC0
A1 PC1/ADC1
A2 PC2/ADC2
A3 PC3/ADC3
A4 PC4/ADC4/SDA
A5 PC5/ADC5/SCL
ANALOG IN
DIGITAL (~PWM)
ARDUINO UNO R3
RXD
TXD
RTS
CTS
Virtual
3.55

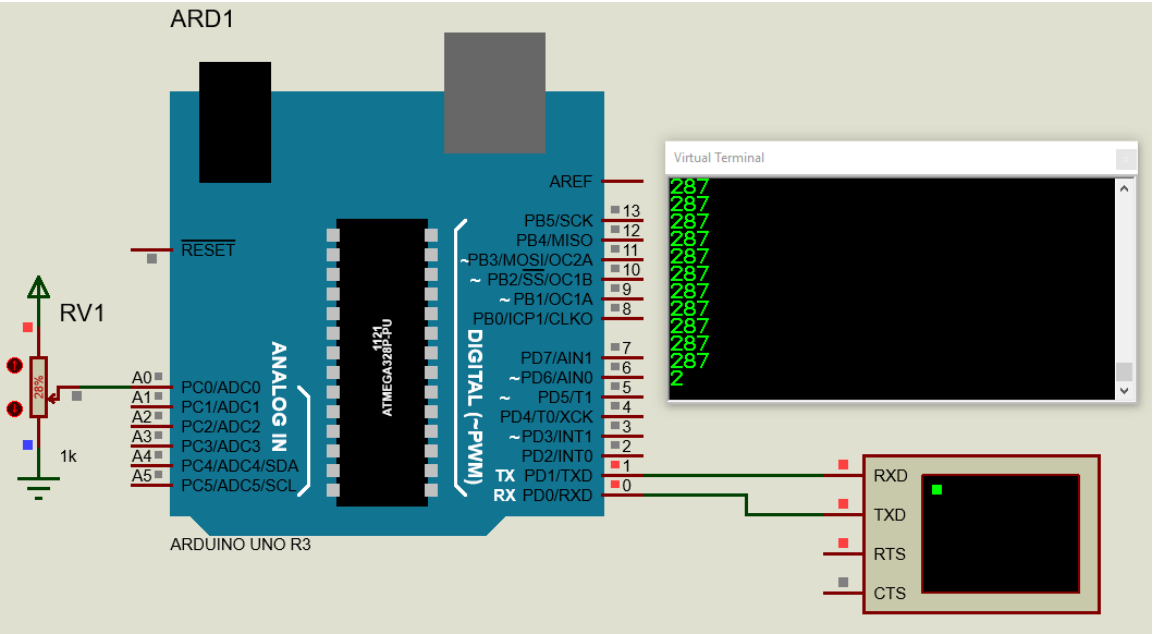
CHAPITRE11. CONVERSION ANALOGIQUE NUMERIQUE PAR LA METHODE DE REGISTRE

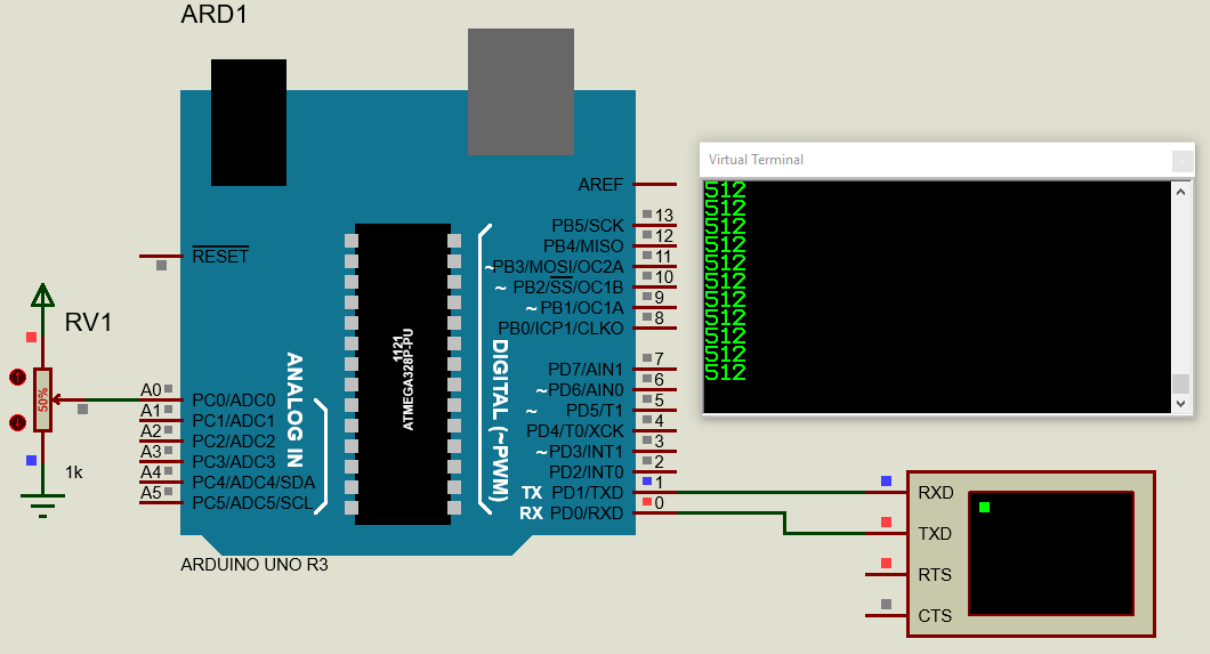
1. CODE DANS ARDUINO

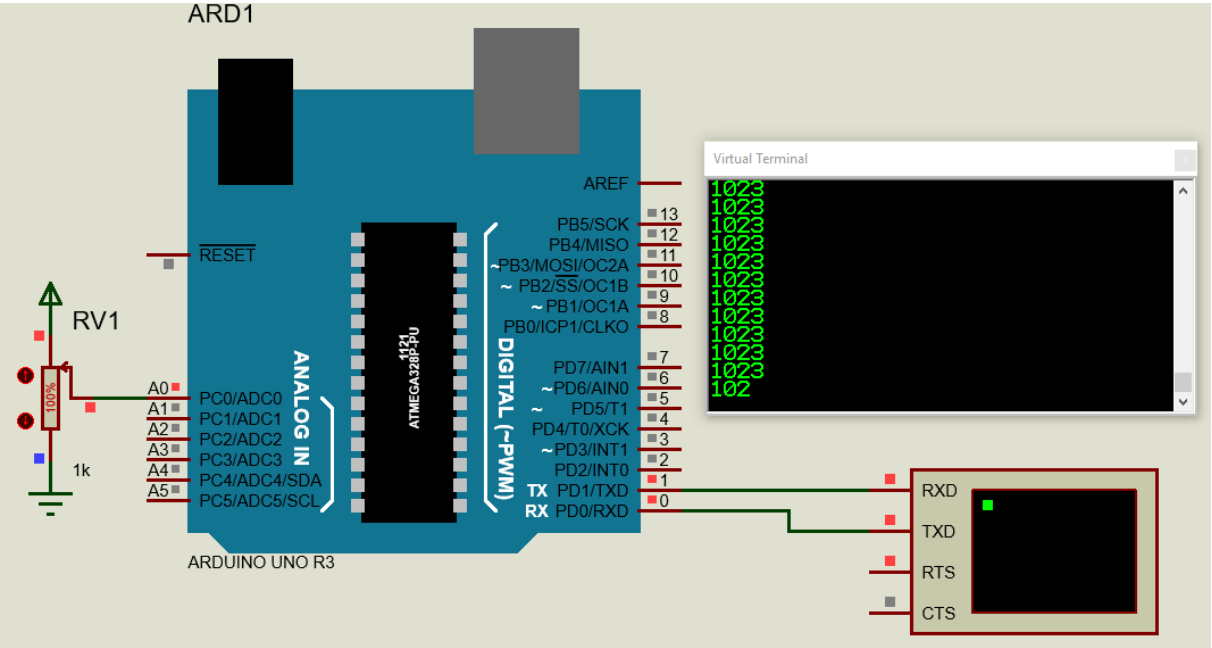
code_asm §

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <stdlib.h>

 uint16_t adcValeur ;

 void adc_Init() {
  ADCSRA  = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
  ADMUX   = (1<<REFS0);
 }
  uint16_t adc_Read(uint8_t chanal) {
      ADCSRA |=(1<<ADSC);

      while (ADCSRA & (1<<ADSC)) ;
      return ADCW ;
  }
  int main () {

     Serial.begin (9600) ;
     adc_Init();

       while (1) {
        adcValeur  = adc_Read (0);
        Serial.println (adcValeur) ;
       }
  }
```

2. SCHEMA ET SIMULATION


ARD1
AREF
RESET
PB5/SCK
PB4/MISO
~PB3/MOSI/OC2A
~ PB2/SS/OC1B
~PB1/OC1A
PB0/ICP1/CLKO
PD7/AIN1
~PD6/AIN0
~ PD5/T1
PD4/T0/XCK
~PD3/INT1
PD2/INT0
TX PD1/TXD
RX PD0/RXD
ANALOG IN
A0 PC0/ADC0
A1 PC1/ADC1
A2 PC2/ADC2
A3 PC3/ADC3
A4 PC4/ADC4/SDA
A5 PC5/ADC5/SCL
DIGITAL (~PWM)
ATMEGA328P-PU
RV1
28%
1k
ARDUINO UNO R3
Virtual Terminal
287
287
287
287
287
287
287
287
287
287
2
RXD
TXD
RTS
CTS


ARD1
AREF
RESET
PB5/SCK
PB4/MISO
~PB3/MOSI/OC2A
~ PB2/SS/OC1B
~PB1/OC1A
PB0/ICP1/CLKO
PD7/AIN1
~PD6/AIN0
~ PD5/T1
PD4/T0/XCK
~PD3/INT1
PD2/INT0
TX PD1/TXD
RX PD0/RXD
ANALOG IN
A0 PC0/ADC0
A1 PC1/ADC1
A2 PC2/ADC2
A3 PC3/ADC3
A4 PC4/ADC4/SDA
A5 PC5/ADC5/SCL
DIGITAL (~PWM)
ATMEGA328P-PU
RV1
50%
1k
ARDUINO UNO R3
Virtual Terminal
512
512
512
512
512
512
512
512
512
512
512
RXD
TXD
RTS
CTS



3. CONCLUSION

En voyant les images de simulation nous pouvons conclure que le code fonctionne très bien chers amis je vous conseille de programmer toujours de cette façon car cela vous appelle à la connaissance de l'architecture du microcontrôleur.

CHAPITRE12. COMMUNICATION UART (SERIE)

COMMUNICATION SERIE

Lorsque l'on veut que la carte ARDUINO puisse communiquer avec un ordinateur ou avec un autre système à microprocesseur on peut utiliser la communication série.

Dans cette communication la carte ARDUINO peut lire ou écrire des informations

ECRITURE DES INFORMATIONS

communication | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

communication §

```
1
2
3  void setup() {
4
5  Serial.begin (9600); /* ici on inialise la communication
6  en la confugurant en une vitesse de 9600 bps
7  */
8
9  }
10
11 void loop() {
12
13   Serial.print ("bonjour le monde");  /* ici ARDUINO envoie
14   une information sur son port SERIE qui sera recue
15   par le module ou la machine avec qui elle communique */
16 }
```

LECTURE DES INFORMATIONS

communication | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

communication §

```
#define led 13
int information ;
void setup() {

Serial.begin (9600); /* ici on inialise la communication
en la confugurant en une vitesse de 9600 bps
*/
pinMode (led , HIGH);  /* confuguration de la 13 en sortie*/
}
void loop() {
   if (Serial.available () > 0) /*ici on verifie la disponibilite des donnees*/
   {
     information = Serial.read (); /*ici on lue les donnees disponible et on le stocke dans la variable*/
   }
       if (information == '1') /*ici on test si l'information est  1 pour allumer la led à 13*/
       {
           digitalWrite (led, HIGH);
       }

         if (information == '0') /*ici on test si l'information est  0 pour éteindre la led à 13*/
         {
            digitalWrite (led, LOW);
         }
}
```

CHAPITRE13 AJOUT DES LIBRAIRIES EN ARDUINO

INTRODUCTION

En programmation une librairie est un ensemble des codes bien finis qu'on met dans un seul et qu'on peut réutiliser quand on a besoin. Les librairies facilitent la programmation. C'est-à-dire on fait les choses une seule fois pour toutes et ce qui reste ce juste l'initialisation.

INSTALLATION D'UNE LIBRAIRIE EN ARDUINO

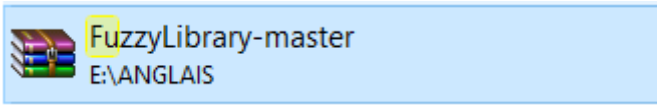
En ARDUINO il a deux manières d'ajouter une librairie :

1. En passant directement par le fichier .zip
2. En décompressant la librairie dans un dossier portant le même puis on copie le dossier dans les librairies d'arduino

On va commencer par la première méthode nous allons voir comment nous pouvons augmenter la librairie :

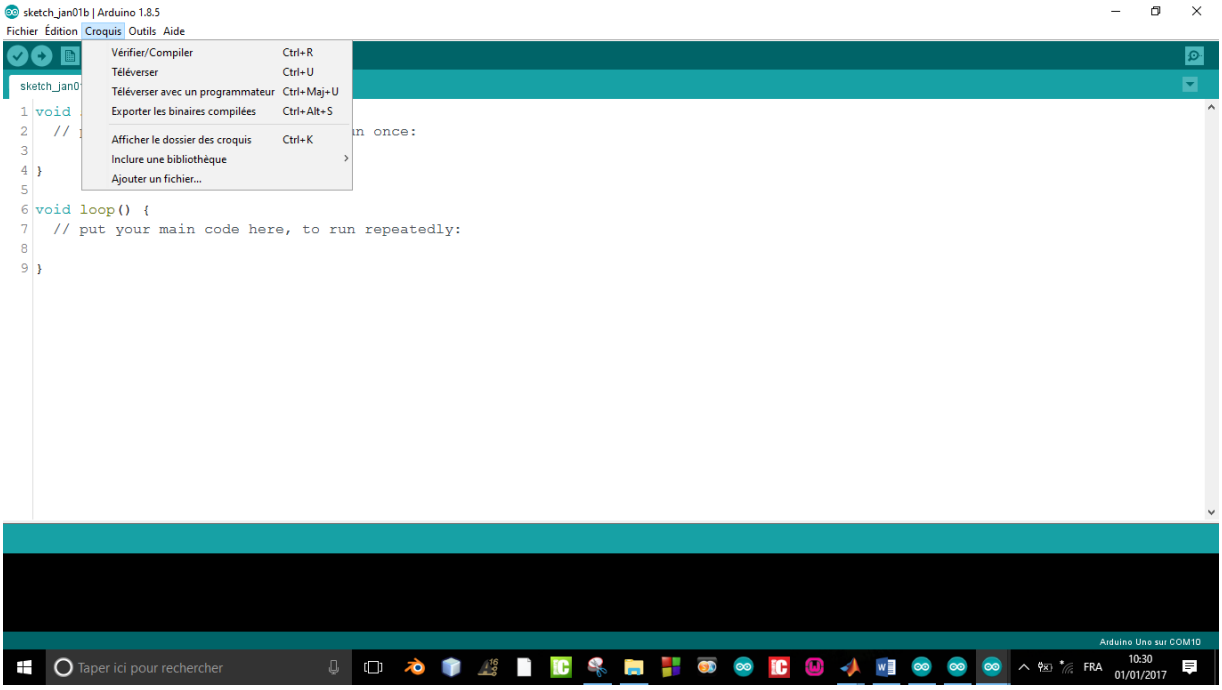
Etape premier :

1. On doit d'abord avoir le fichier de la librairie à installer sous format .ZIP qui se trouve sur votre ordinateur

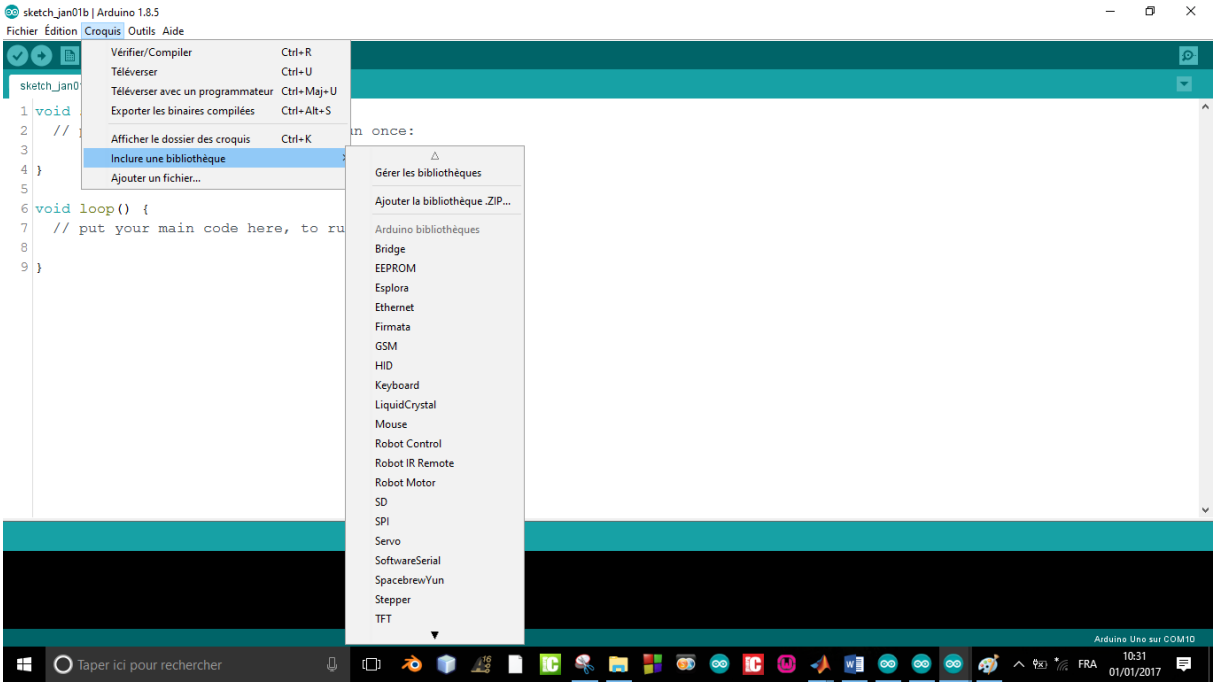


Ici il s'agit d'une librairie de la logique floue.

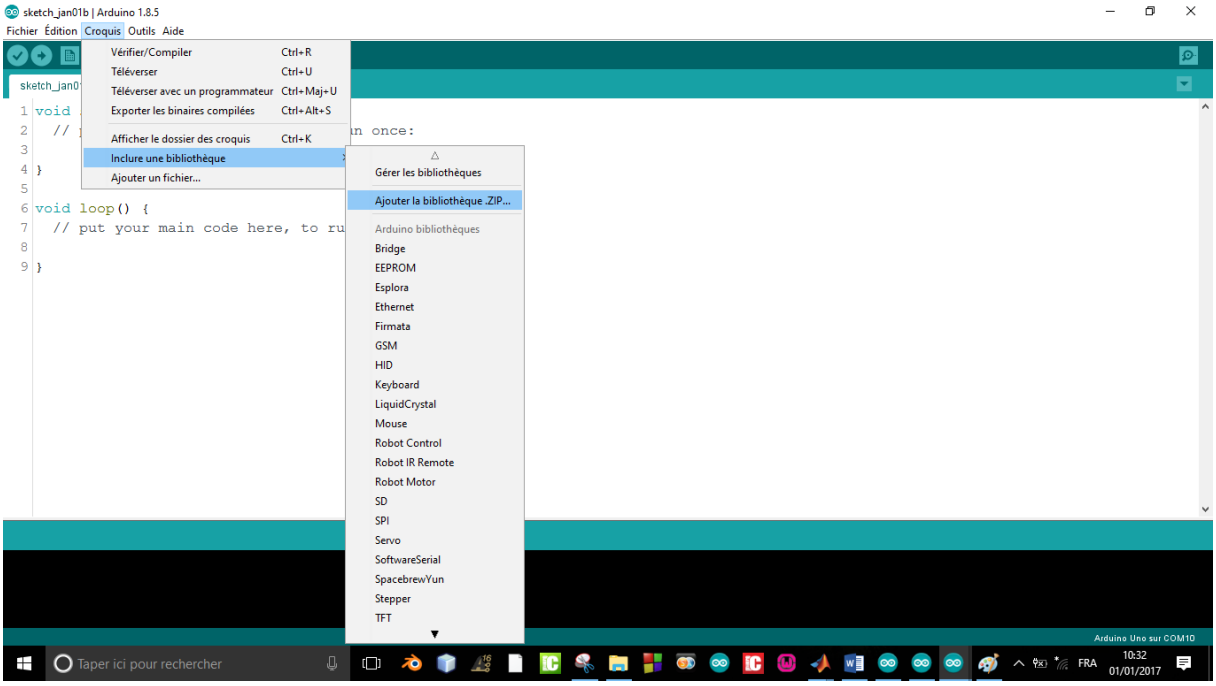
2. Cliquez sur croquis (menu du logiciel ARDUINO)



3. Puis inclure une bibliothèque



4. Puis ajouter la bibliothèque .ZIP



5. L’opération 4 vous guide là où dans les répertoires de l’ordinateur puis vous cherchez là où se trouve votre bibliothèque puis vous double cliqué sur le fichier .ZIP de la librairie si tout s’est bien passé on vous signalera que la bibliothèque ou la librairie a été bien ajouté.

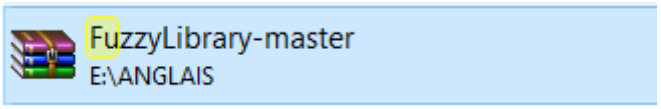
   Si tout s’est bien passé vous allez voir :

```
La bibliothèque a été ajoutée à votre dossier de bibliothèques. Veuillez regarder le menu « Importer bibliothèque »
Bibliothèque non valide trouvée dans C:\Program Files (x86)\Arduino\libraries\con
Bibliothèque non valide trouvée dans C:\Program Files (x86)\Arduino\libraries\con
Bibliothèque non valide trouvée dans C:\Program Files (x86)\Arduino\libraries\con
```

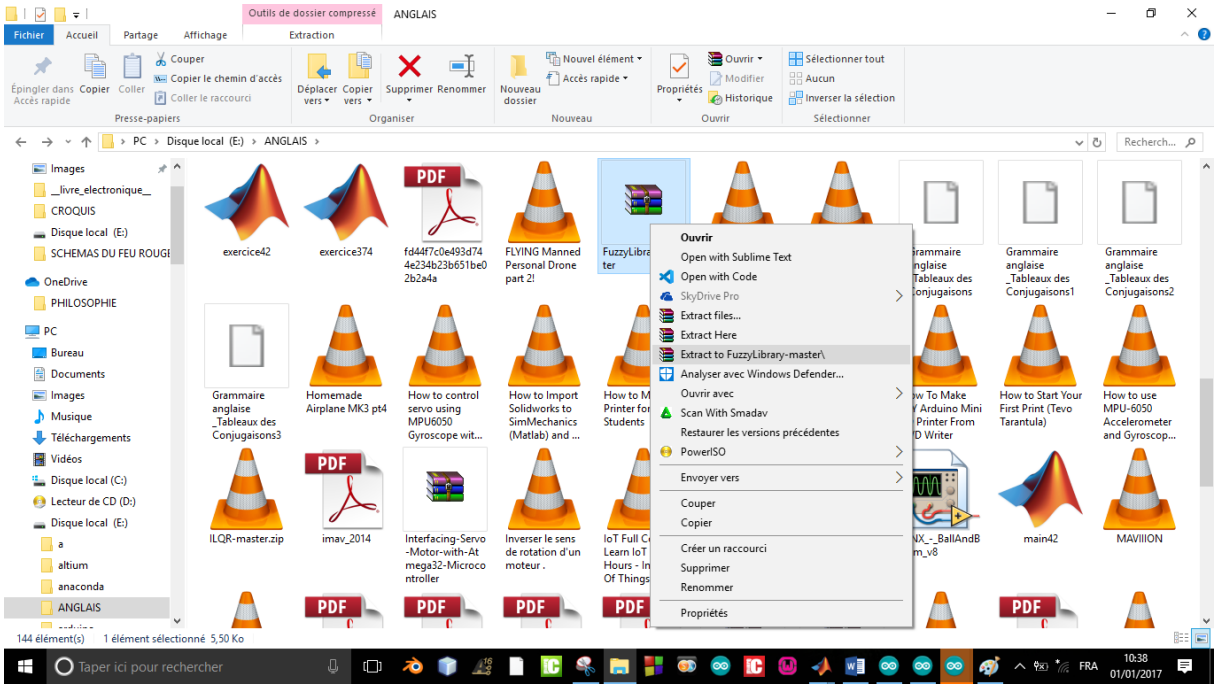
Etape deuxième :

Voici comment on installe la librairie par une autre méthode :

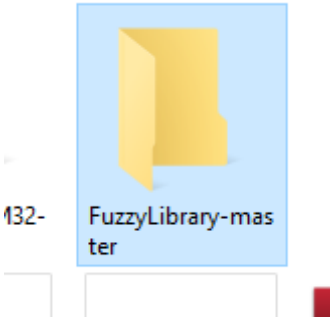
1. On télécharge la librairie sous format .ZIP ou autre format :



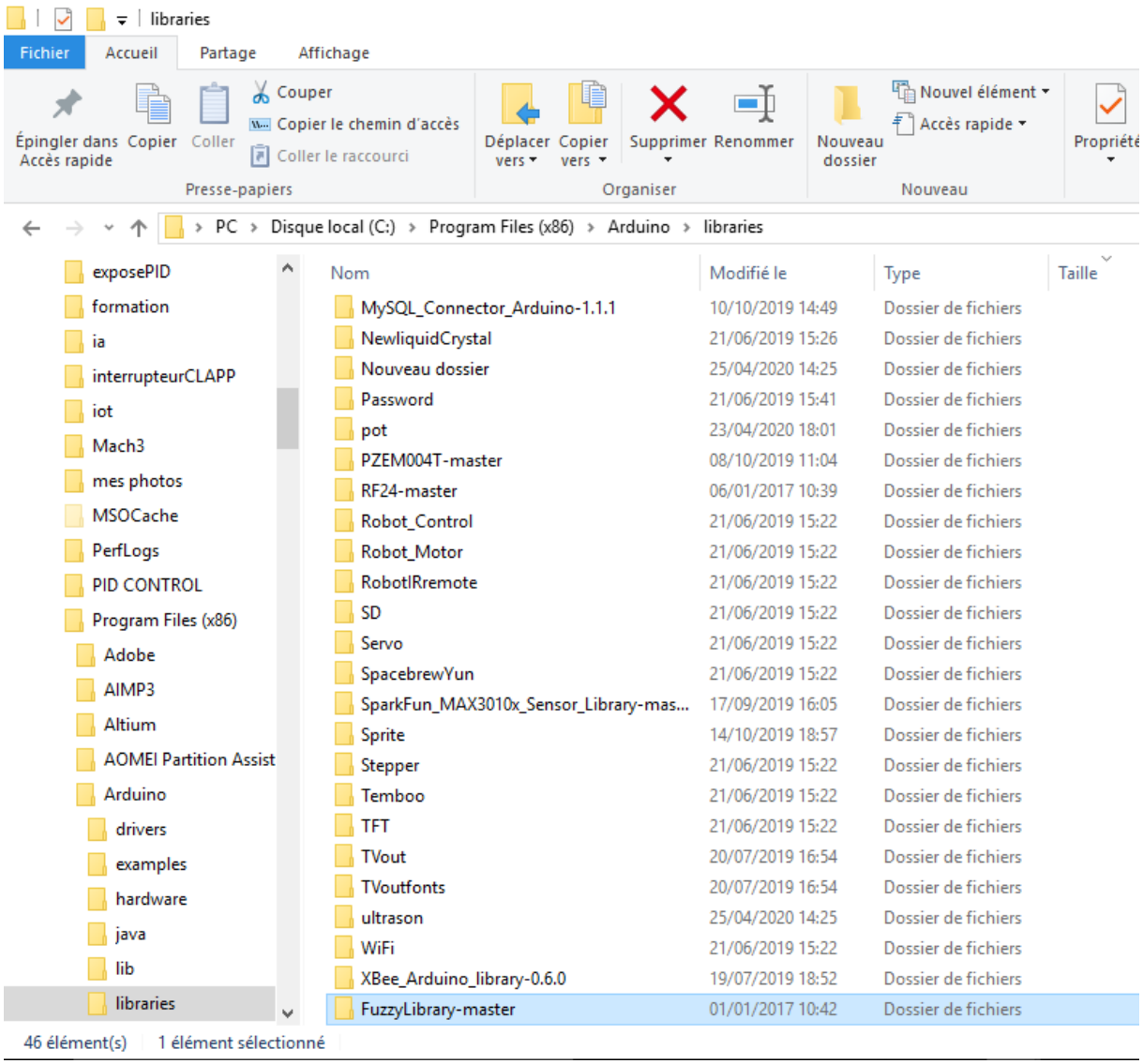
2. On décompresse le fichier .ZIP en un dossier portant le même nom :



3. On obtient un dossier portant le même nom que le fichier .ZIP



4. Puis la dernière opération on part copié ce dossier dans les librairies d'arduino au chemin : C:\Program Files (x86)\Arduino\libraries



Voici les deux principales manières d'ajout d'une librairie dans ARDUINO.

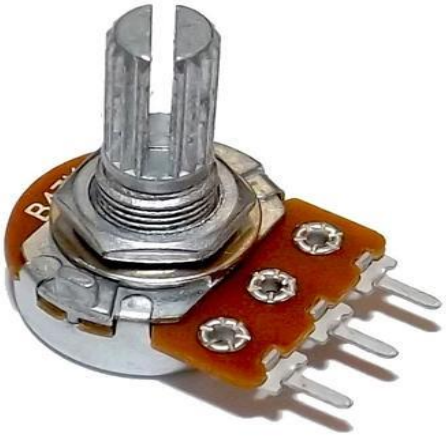
CHAPITRE14 LES MODULES EN ARDUINO

Les modules sont utilisés en ARDUINO pour permettre à la carte de faire une tache bien déterminé par exemple : mesurer la température , mesurer la distance , envoyer des SMS , faire des appels et bien d'autres choses les modules permettent à la carte ARDUINO de faire des tâches.

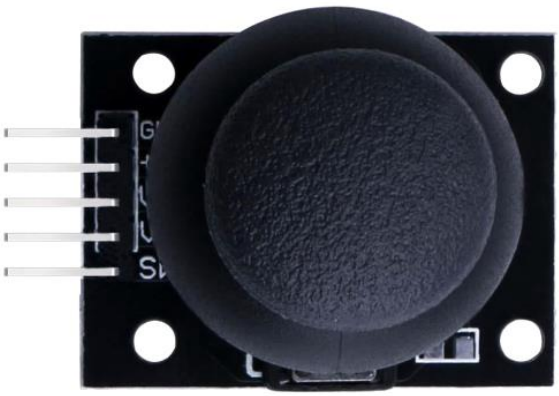
Les modules que nous allons étudier dans cette partie sont :

1. Le module MPU 9265

2. Le potentiomètre



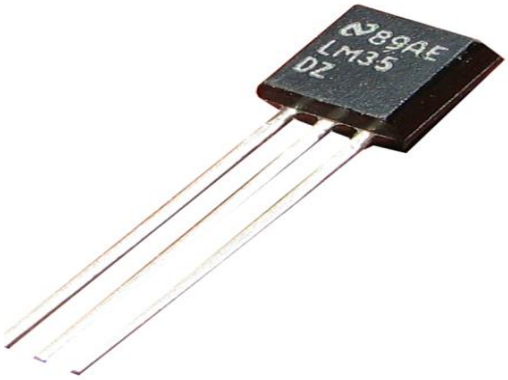
3. Le joystick



4. Le relais

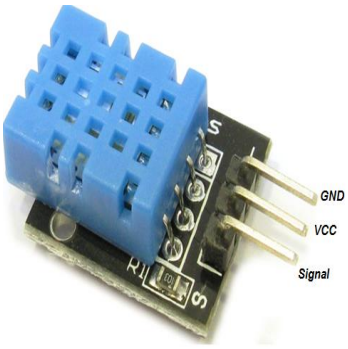
5. Le buzzer

6. Le capteur de température (LM 35)



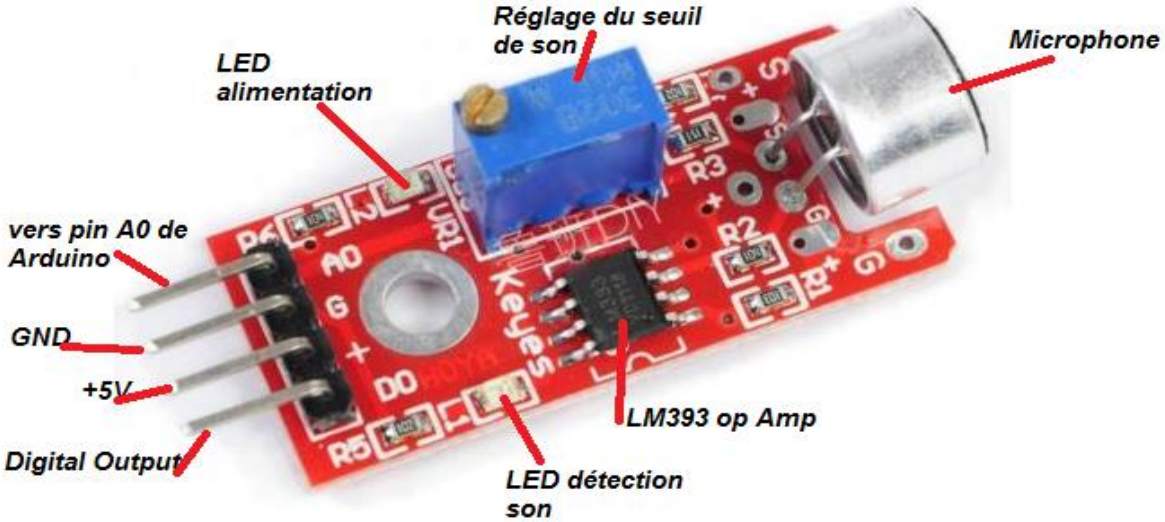
7. L’horloge temps réel

8. Détecteur de température et d’humidité



9. La photorésistance

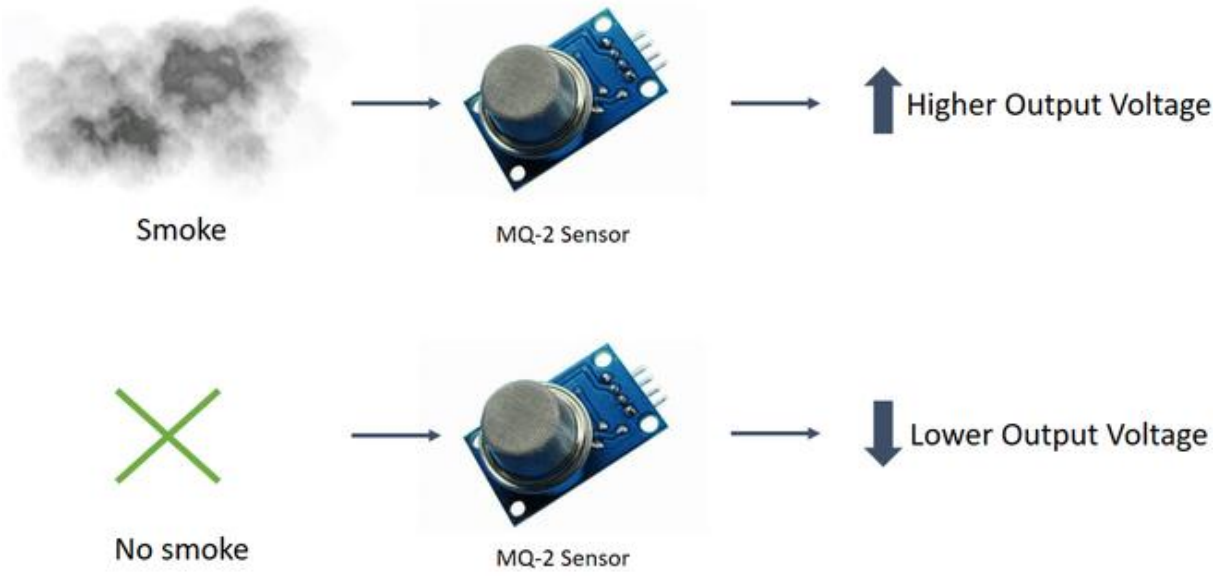
10. Détecteur de son



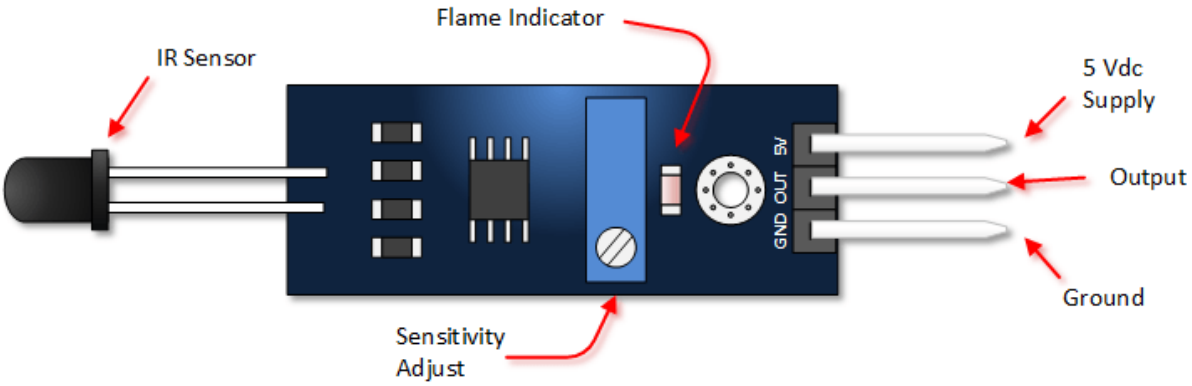
11. Détecteur de niveau d'eau

12. Capteur de pluie

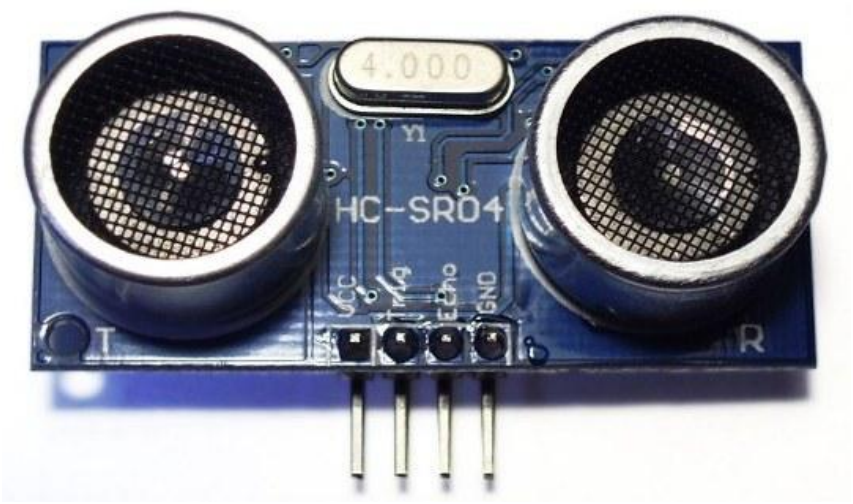
13. Détecteur de fumée



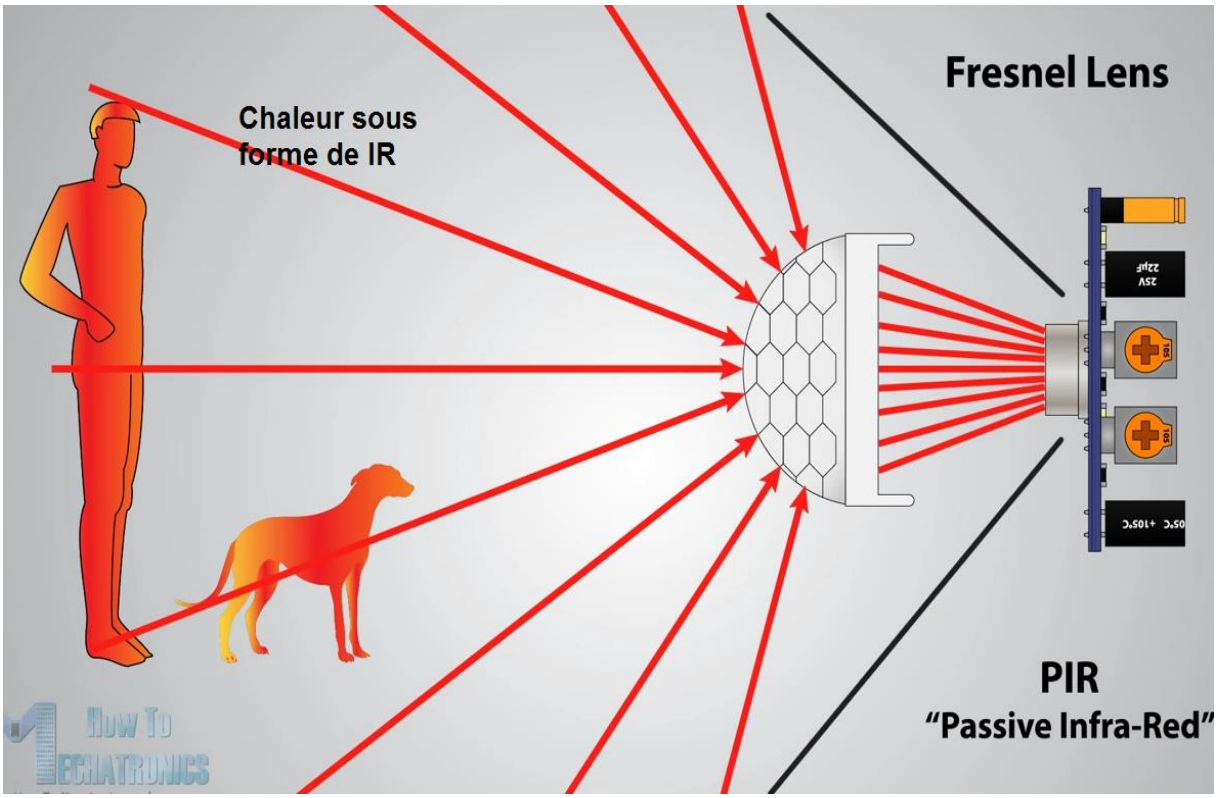
14. Détecteur de flamme



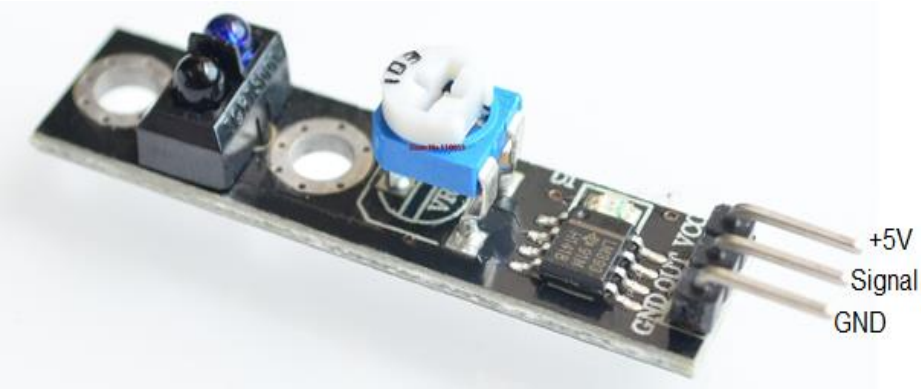
15. Module ULTRASON



16. Détecteur de mouvement



17. Module de traçage de parcours



18. Le servomoteur

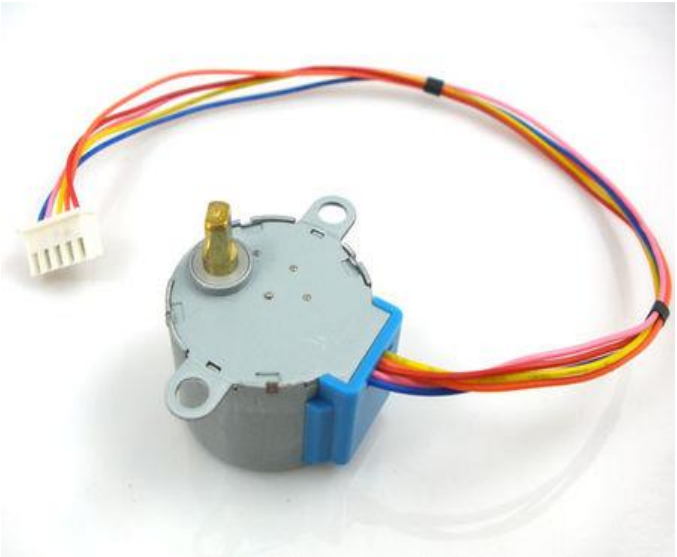
19. Le moteur brushless

20. Le module GPS

21. Le module GSM

22. Le module Bluetooth

23. Le module RFID
24. Le moteur PAS à PAS

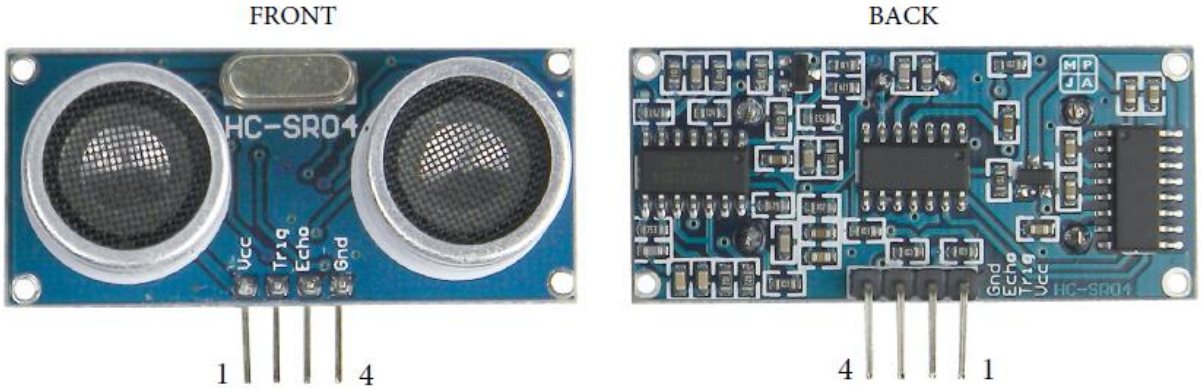


UTILISATION DES MODULES

1. MODULE ULTRASON

1.1. PRESENTATION

Un module ULTRASON est un module utilisant le principe de la chauve-souris il est équipé d'un émetteur (haut-parleur émettant les ultrasons) et d'un récepteur (microphone recevant les ultrasons) il est utilisé pour diverses applications détection d'un objet obstacle se trouvant dans son champs ou pour mesurer la distance. Le croquis ci-dessus montre l'image d'un module à ultrason :



Notre étude se fera sur un module à ultrason spécifique il s'agit du : **HC-SR04**

1.2 PRINCIPE DE FONCTIONNEMENT DU MODULE

Ce module permet de mesurer la distance entre lui et un objet se trouvant à sa face il envoie un signal sonore ultrasonic moyenne un petit haut-parleur s'il reçoit un écho en retour moyennant un petit microphone ultrasonic il déduit qu'il y a un obstacle puis il mesure la distance. Ce module donne le temps que le signal ultrasonic a fait pour faire un aller-retour

et connaissant la vitesse du son on déduit la distance aller-retour puis cette distance est divisée par deux pour trouver la distance entre l'objet et l'ultrason.

MANUPILATION DU MODULE HC-SR04

Pour la détection de l'obstacle :

On doit apporter la broche Trig à 1 (5V) pendant 10 microsecondes c'est cette impulsion qui enclenche l'émetteur envoyant un signal ultrasonique grâce à un haut-parleur. S'il y a un obstacle il y a un signal de retour qui est capté par le microphone ultrasonique et de là, la broche Echo passe à 1. Le temps de passage de cette broche ultrasonique à 1 est directement proportionnel avec la distance qu'il y a entre le module et l'obstacle.

Ce principe de fonctionnement peut être résumé par les graphiques suivantes :

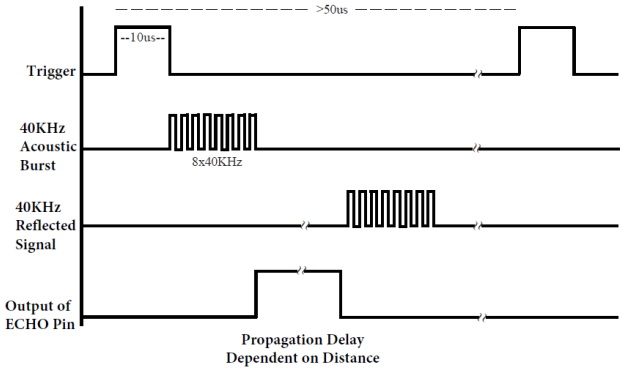


Fig4.19 diagramme de fonctionnement d'un module ultrasonique [4]

CARACTERISTIQUES ELECTRIQUES DU HC-SR04

Ce module est constitué par 4 broches ayant chacun un rôle bien déterminer voir tableau suivant [4] :

numéro de la broche		nom de la broche	fonction de la broche
1		VCC	5V (c'est la tension d'alimentation)
2		Trig	Trigger c'est la broche d'entrée (INPUT)
3		Echo	Echo c'est la broche de sortie (OUTPUT)
4		Gnd	Tension d'alimentation négative de l'alimentation

Paramètres électriques	
Tension de fonctionnement normal	+5V
Courant de fonctionnement normal	15mA
Fréquence d'opération	40Khz
Distance de détection	4m
Angle de mesure	15°

Entré du signal Trig	10microseconds minimum impulsion TTL
Sortie signal Echo	Proportionnel à la distance avec l'obstacle
Autres caractéristiques du module	
Dimensions de la carte du module	1-13/16" X 13/16" X 5/8"
Connections de la carte du module	4 X 0.1"

PROGRAMME PERMETTANT DE MESURE LA DISTANCE EN CENTIMETRE AVEC UN ULTRASON

ULTRASON | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

ULTRASON §

```
1
2  const int trigPin = 9 ;
3  const int echoPin = 10 ;
4  const int ledPin = 13 ;
5
6  long duration;
7  int distance ;
8  int safetyDistance;
9
10 void setup()
11 {
12
13 pinMode (trigPin , OUTPUT);
14 pinMode (echoPin , INPUT);
15 pinMode (ledPin , OUTPUT);
16
17
18 }
19
20 void loop()
21 {
22    digitalWrite ( trigPin , LOW );
23    delayMicroseconds (2);
24
```

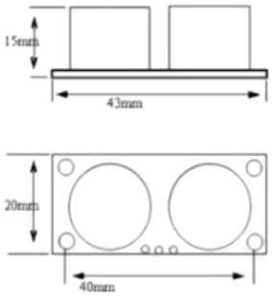
```
  digitalWrite ( trigPin ,HIGH );
  delayMicroseconds (10);
  digitalWrite ( trigPin , LOW );

  duration = pulseIn (echoPin , HIGH );
  distance = duration * 0.034/2;
 safetyDistance = distance;

  if (distance>120)
  {
    digitalWrite ( ledPin , HIGH );
  }
  else {
     digitalWrite ( ledPin , LOW );
  }
}
```

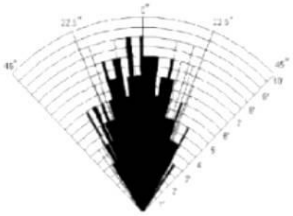
AUTRES INFORMATIONS IMPORTANTES POUR LES MODULES

Features:

- Power Supply :+5V DC
- Quiescent Current : <2mA
- Working Currnt: 15mA
- Effectual Angle: <15°
- Ranging Distance : 2cm – 400 cm/1" - 13ft
- Resolution : 0.3 cm
- Measuring Angle: 30 degree
- Trigger Input Pulse width: 10uS
- Dimension: 45mm x 20mm x 15mm

VCC = +5VDC
Trig = Trigger input of Sensor
Echo = Echo output of Sensor
GND = GND





Practical test of performance,
Best in 30 degree angle

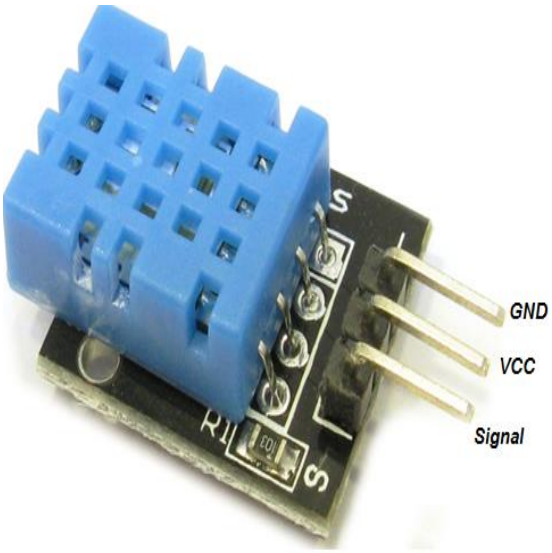
2. UTILISATION DU DHT11 ET DU DHT22

INTRODUCTION

Dans la pratique certain projet pousse à mesurer la température et l'humidité. Par exemple pour une couveuse ces informations (température et humidité) sont d'une importance très capitale dans cette partie on va voir comment on peut mesurer ces deux grandeurs en utilisant le DHT 11 d'une part et le DHT 22 de l'autre part.

DEVELOPPEMENT

Le DHT 11 et le DHT 22 sont présentés par la figure ci-dessous :



Image

Cette image montre l'apparence d'un DHT 11 et un DHT 22.

Il constitué par trois broches l'une pour la masse l'autre pour le VCC puis celle du signal. C'est celui du signal qui donne les informations sur la température et l'humidité.

UTILISATION DU DHT 11 et DHT 22 EN ARDUINO

Pour utiliser ces modules avec ARDUINO :

1. il faut d'abord télécharger la librairie facilitant leurs usages déjà disponible sur INTERNET. Ce pourquoi dans le code on commence par :

```
#include <DHT.h>
```

2. Une fois vous avez permis à votre code d'utiliser la librairie du DHT 11 ou du DHT 22 vous devez créer un objet qui vous permettra d'utiliser les fonctions de votre librairie :
   a. Si vous utilisez un DHT 11
      Voici comment sera la création de l'objet :

```
DHT dht(9, DHT11);
```

      Ici en premier paramètre on spécifie la broche ARDUINO sur lequel sera connectée la borne signal du DHT 11 ; et en deuxième paramètre on spécifie le DHT utilisé dans ce cas il s'agit du DHT 11.
      S'il s'agissait du DHT 22 on écrirait :

```
DHT dht(9, DHT22);
```

      Ici il s'agit d'un DHT 22 connecté dont la broche de signal est connectée au pin 9 de la carte ARDUINO.
3. Ici on fait une configuration de l'utilisation du module dans le void setup () { } avec l'instruction :

```
dht.begin();
```

Ici on doit prendre le nom de l'objet que l'on a créé point la fonction begin ()

4. Ce la dernière partie ici on obtient alors les informations relatives à la température et à l'humidité pour obtenir la température on a la fonction :

```
float t = dht.readTemperature();
```

   Pour obtenir l'humidité on a :

```
float h = dht.readHumidity();
```

Ces sont là les procédures d'utilisation du DHT 11 et du DHT 22.

On a le code complet pour le DHT 11 :

DHT §

```
#include <DHT.h>

DHT dht(9, DHT11);

void setup() {
 dht.begin();
 Serial.begin (9600);
}

void loop() {

  float h = dht.readHumidity();
  float t = dht.readTemperature();
  Serial.println (h);
   Serial.println (t);
   delay(300);

}
```

Ici les informations relatives à la température et à l'humidité sont affichées sur le moniteur série.

On a le code avec le DHT 22 :

DHT §

```
#include <DHT.h>

DHT dht(9, DHT22);

void setup() {
 dht.begin();
 Serial.begin (9600);
}

void loop() {

  float h = dht.readHumidity();
  float t = dht.readTemperature();
  Serial.println (h);
   Serial.println (t);
   delay(300);

}
```

SCHEMAS DU MONTAGE

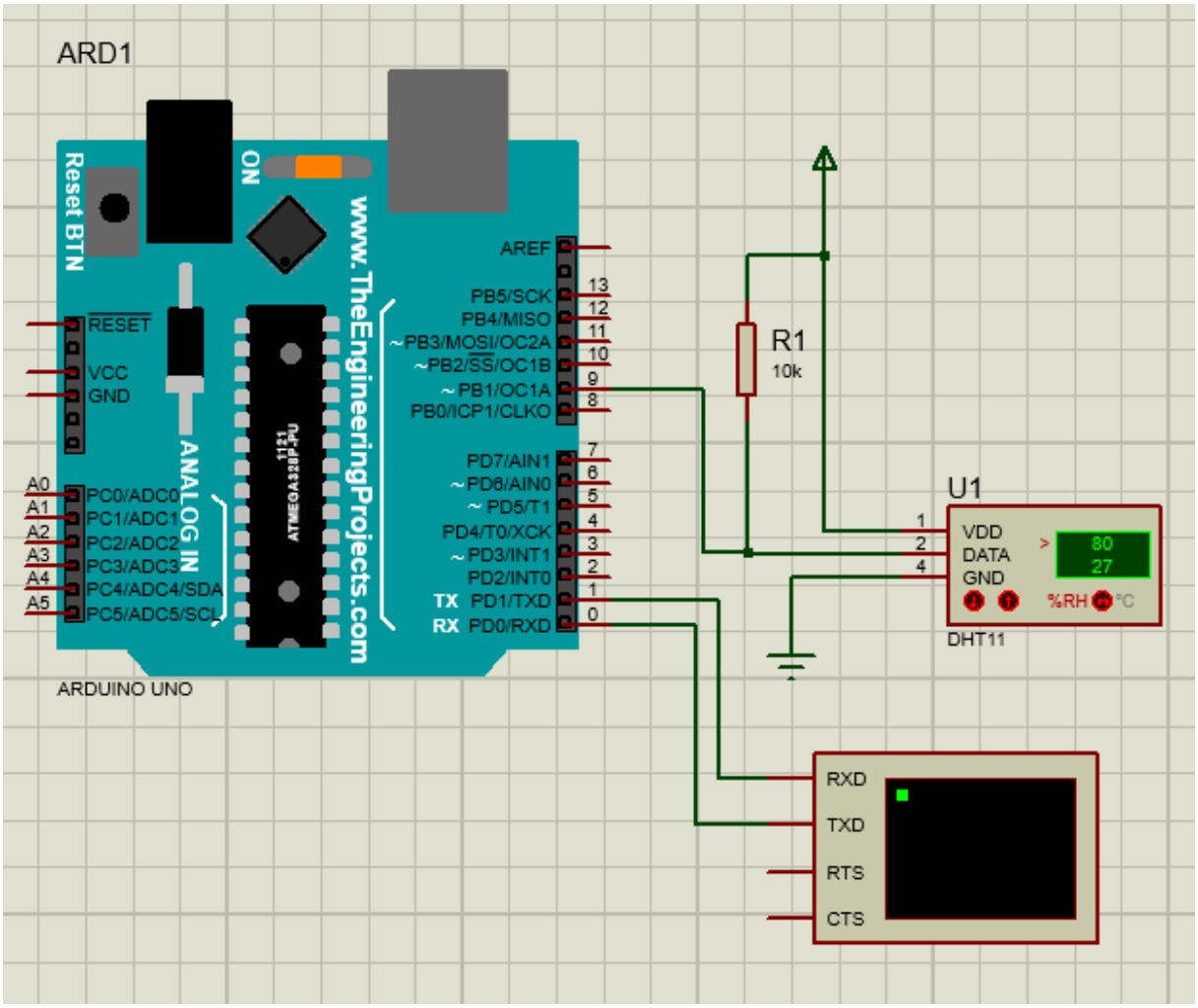
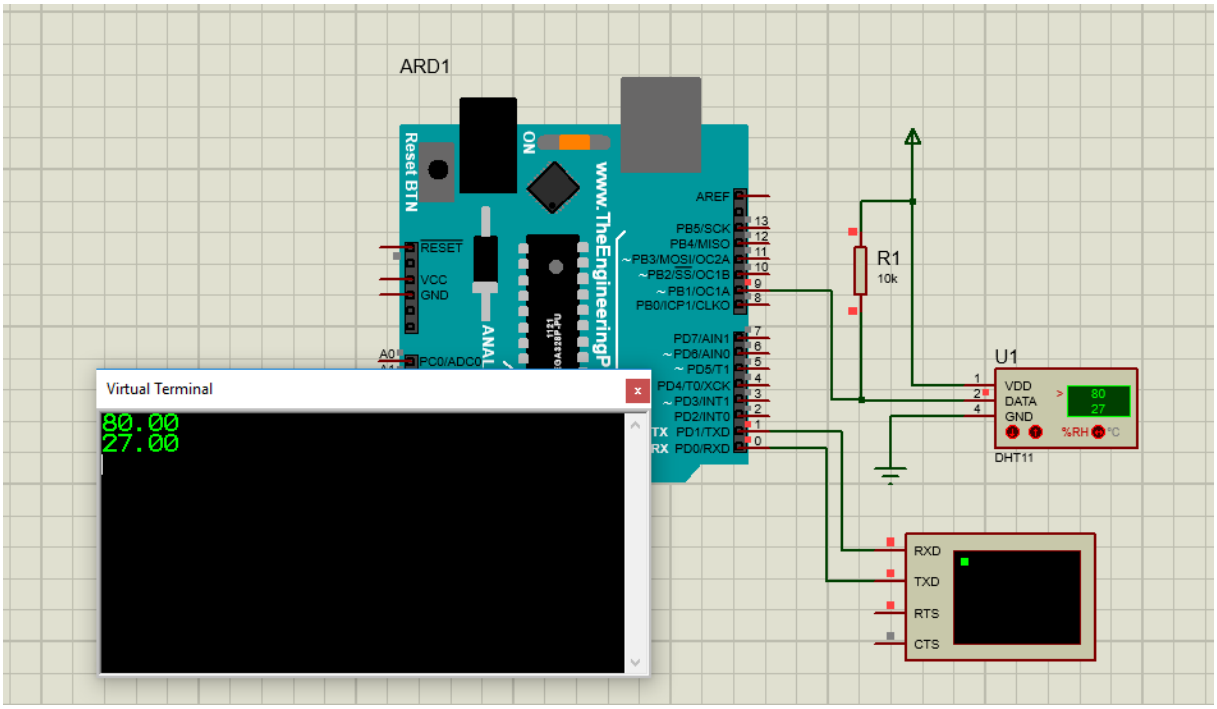


IMAGE DE SIMULATION



On voit clairement que ça marche très bien utiliser donc le DHT 11 ou le DHT 22 pour vos réalisations.

Il est utile de signal avant de terminer cette partie que le DHT 22 est une amélioration d DHT 11 donc on comprend que le DHT 22 et précis que le DHT 11.

3. LE SERVO MOTEUR

Introduction

Un servomoteur est un système mécanique pouvant prendre un angle que vous lui ordonné de prendre. Dans notre cas un servomoteur est constitué par un moteur DC couplé à des engrenages et d'une carte électronique permettant la commande du moteur selon l'angle que l'on veut avoir.

Fig1 image du servomoteur

Pour commander le servomoteur on suit la procédure suivante :

1. On inclut dans le code la librairie du servomoteur

```
#include <Servo.h>
```

2. On crée l'objet de la classe **Servo** de la librairie

```
Servo servoMoteur ;
```

3. On précise la broche sur laquelle on va connecter le servomoteur

```
servoMoteur.attach(5) ;
```

Ici par exemple le signal de commande du servomoteur sera disponible au niveau de la broche 5 de la carte ARDUINO

4. Par la suite il y a deux cas :
   - Soit on ordonne le servomoteur de faire un angle que l'on veut

```
servoMoteur.write(90) ;
```

La valeur se trouvant dans le paramètre de la fonction **write** correspond à l'angle que fera le servomoteur par exemple dans l'exemple ci-dessus le servomoteur fera un angle de 90°.

On a le programme suivant de gestion du servomoteur :

```
#include <Servo.h>

Servo servoMoteur ;
void setup() {

servoMoteur.attach (5) ;


}

void loop() {
  servoMoteur.write (90) ;
  delay (1000) ;
  servoMoteur.write (0) ;
  delay (1000) ;

}
```

Ce programme permet d'action le servomoteur dont le signal de commande venant de la broche 5 à 90° pendant une seconde puis à 0° pendant une seconde et cela dans une boucle infinie.

NB : pour la commande du servomoteur pour plus de précision on peut encore utiliser la fonction **writeMicroseconds**(2400)

```
#include <Servo.h>

Servo servoMoteur ;
void setup() {

servoMoteur.attach (5) ;


}

void loop() {
  servoMoteur.writeMicroseconds (1000) ;
  delay (1000) ;
  servoMoteur.writeMicroseconds (600) ;
  delay (1000) ;

}
```

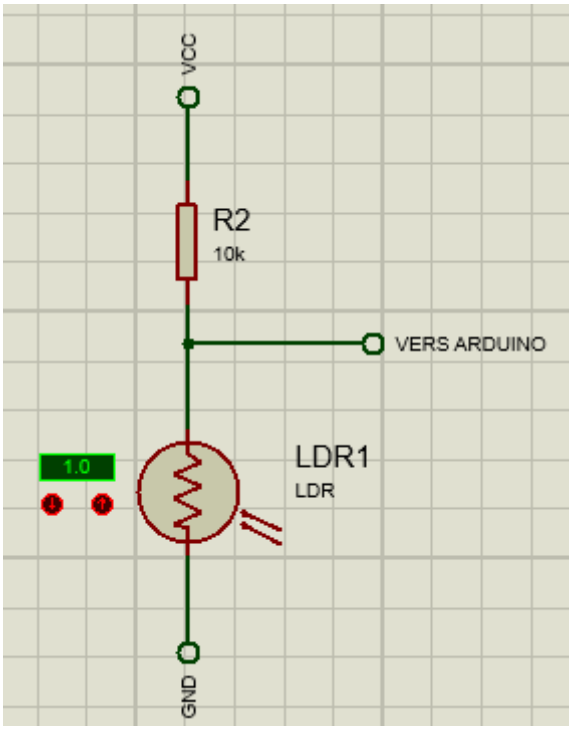
- Soit on lit l'angle que le servomoteur fait déjà

```
servoMoteur.read();
```

Cette fonction permet de lire l'angle sur lequel se trouve le servomoteur.

4. LA PHOTO RESISTANCE

La photorésistance est une résistance dont la valeur change en fonction de la lumière. Si la lumière est intense sa valeur diminue considérablement et si la lumière est faible sa valeur augmente considérablement. Il est généralement utiliser dans les interrupteurs crépusculaires les compteurs de tours et d'autres équipements. Pour bien l'utiliser on le met dans un pont diviseur avec une autre résistance puis on pour juger sur l'intensité de la lumière en fonction de la tension aux bornes de la photorésistance. Pour notre cas dans la programmation ARDUINO nous aurons juste la tache de mettre la photorésistance dans un pont diviseur et puis nous agirons dans le code en fonction de la tension aux bornes de cette photorésistance.



La tension aux bornes de cette photorésistance est trop grande lorsqu'il y a obscurité dans le code ARDUINO on pourra juste lire cette tension et en fonction de la valeur obtenue pourra décider de faire une action ou une autre. Par exemple s'il s'agit d'un interrupteur crépusculaire on décidera soit d'allumer les lampes ou pas. Voici un petit code que l'on peut utiliser par exemple pour un interrupteur crépusculaire.

```
 #define lampe 13
 #define infoObscurite 800
void setup() {
   pinMode (13,OUTPUT);
}

void loop() {

 if (analogRead(A0) == infoObscurite) {
   digitalWrite (lampe , HIGH);
 } else {
   digitalWrite (lampe , LOW);
 }

}
```

Ici la borne venant du milieu du pont diviseur de la première figure et la masse sont connectés à la carte ARDUINO. Cette borne du milieu est connectée à A0 et le masse et connecté à la masse d'ARDUINO.

5. MOTEUR BRUSHLESS

INTRODUCTION

Un moteur brushless est un moteur qui n'a pas de balais par opposition de moteurs à courant qui eux ont des balais ; c'est son des moteurs tournants à très grande vitesse ils sont couramment utilisés dans l'aéromodélisme (les drones : entre les avions RC et les quadrucoptères). Dans ce présent travail nous n'allons pas détailler comme ces moteurs fonctionnent mais nous allons principalement voir comment nous pouvons commander ces dernies avec un ARDUINO.

PROGRAMME ARDUINO

Pour commander un moteur brushless on utilise la même du servomoteur donc pour la programmation on commence :

1. Par l'ajout de la librairie **<Servo.h>**

```
#include <Servo.h>

```

2. On crée un objet de la classe Servo qui facilitera les manœuvres par l'instruction suivante :

```

Servo moteurBRUSHLESS ;

```

3. On affecte la broche de commande qui pourra contrôler la vitesse du moteur brushless dans cet exemple nous utilisons la broche 5.

```
moteurBRUSHLESS.attach(5);

```

4. On fait une activation du moteur par l'instruction :

```
moteurBRUSHLESS.writeMicroseconds (1000) ;

```

5. Puis on peut alors affecte au moteur une certaine vitesse par exemple ici nous active le moteur brushless à 1500 avec l'instruction :

```
moteurBRUSHLESS.writeMicroseconds (1500) ;
```

Pour la commande du moteur brushless on conseil que le paramètre de la fonction writeMicroseconds puis varier entre 1000 et 2000.

Voici alors le programme complet ou la vitesse est à 1500 :

```
#include <Servo.h>

 Servo moteurBRUSHLESS ;

void setup() {

 moteurBRUSHLESS.attach(5);

 moteurBRUSHLESS.writeMicroseconds (1000) ;
 delay (2000) ;

}

void loop() {

   moteurBRUSHLESS.writeMicroseconds (1500) ;


}
```

Chers ami voici en sommaire comment vous pouvez commander les moteurs BRUSHLESS.

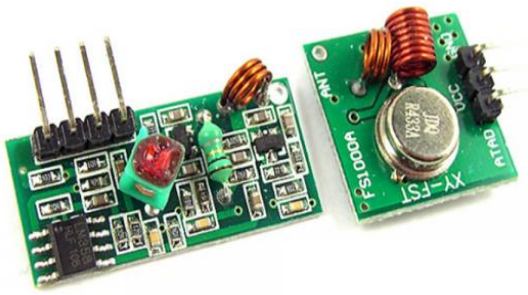
CHANGEMENT DU SENS DE ROTATION

Pour changer le sens de rotation du moteur Brushless il suffit juste de changer le cablage de phase allant vers le moteur.

6. MODULES RF

INTRODUCTION

Les modules RF sont utilisés lorsqu'on a besoin de faire une transmission des données à distance commandes des lampes ou autres systèmes à distance. Ces modules fonctionnent en paire (un récepteur et un émetteur) l'émetteur envoie les informations et le récepteur reçoit les informations. On peut voir ces modules sur la photo ci-dessous :



Ces modules fonctionnent sur une portée allant jusqu'à 200 mètres.

Pour bien comprendre faisons un exemple pour l'utilisation de ce module :

Exemple : on va faire un programme permettant d'allumer une led quand on appuie sur un bouton poussoir et cela à distance.

Commençons par le code de l'émetteur :

emetteurRF

```
#include <VirtualWire.h>
char *controller;
void setup() {
  pinMode (13 , OUTPUT) ;
  pinMode (8 , INPUT_PULLUP);
  vw_set_ptt_inverted(true);
  vw_set_tx_pin(12);
  vw_setup(4000);
}
void loop() {

   if (digitalRead (8) == LOW) {
    controller="1"  ;
    vw_send((uint8_t *)controller, strlen(controller));
    vw_wait_tx();
   }  else {
    controller="0"  ;
    vw_send((uint8_t *)controller, strlen(controller));
    vw_wait_tx();
   }
  }
```

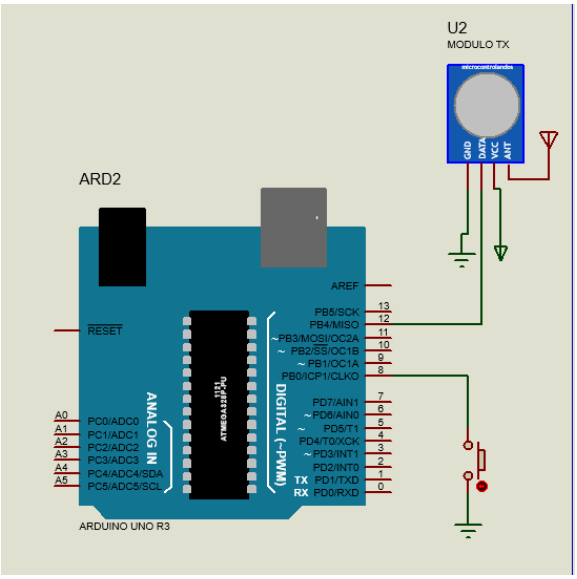
Par la suite nous avons le code du récepteur :

recepteurRF §

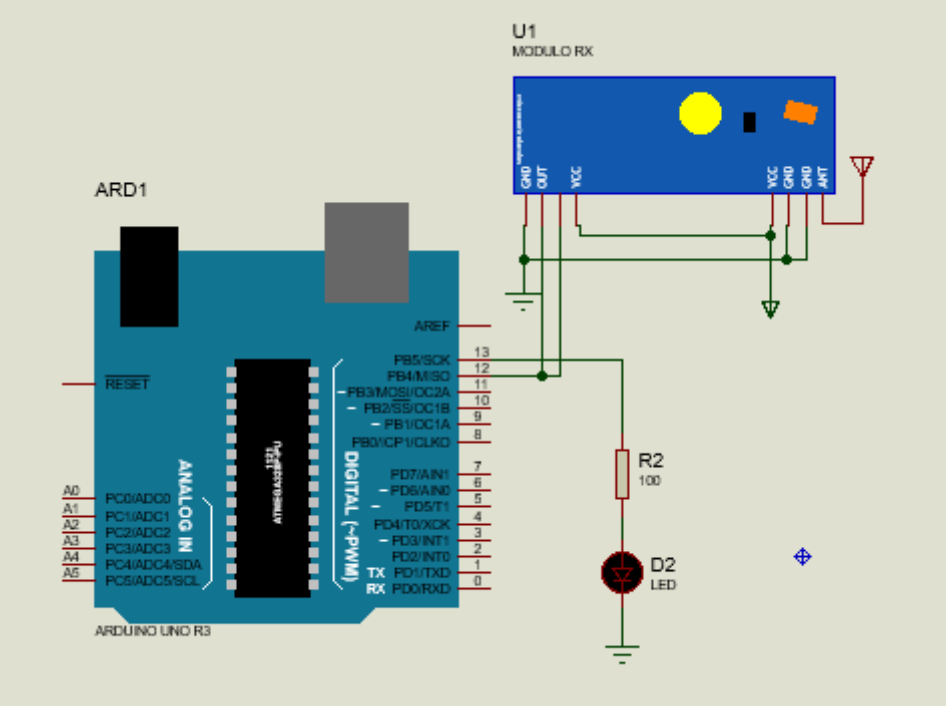
```
#include <VirtualWire.h>
void setup() {
 vw_set_ptt_inverted(true);
 vw_set_rx_pin(12);
 vw_setup(4000);
 vw_rx_start();
 pinMode(13, OUTPUT);
}
void loop() {
  uint8_t buf[VW_MAX_MESSAGE_LEN] ;
    uint8_t buflen = VW_MAX_MESSAGE_LEN ;
    if (vw_get_message(buf, &buflen))
    {
      if(buf[0]=='1'){
   digitalWrite(13,HIGH);
      }
   if(buf[0]=='0'){
  digitalWrite(13, LOW);
    }
  }
}
```

SCHEMAS DE CABLAGE

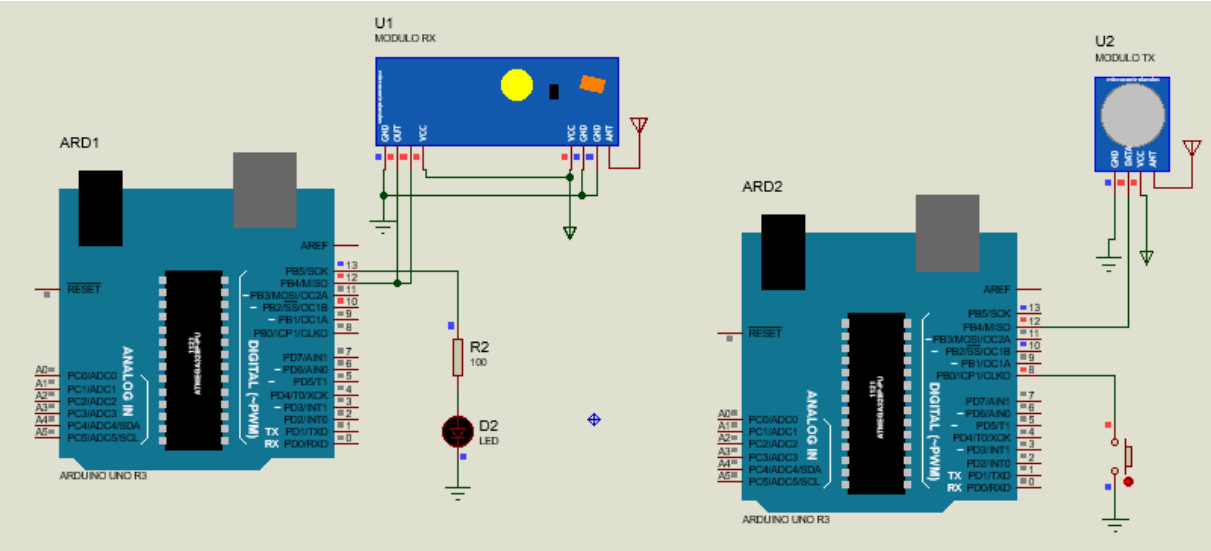
SCHEMA EMETTEUR



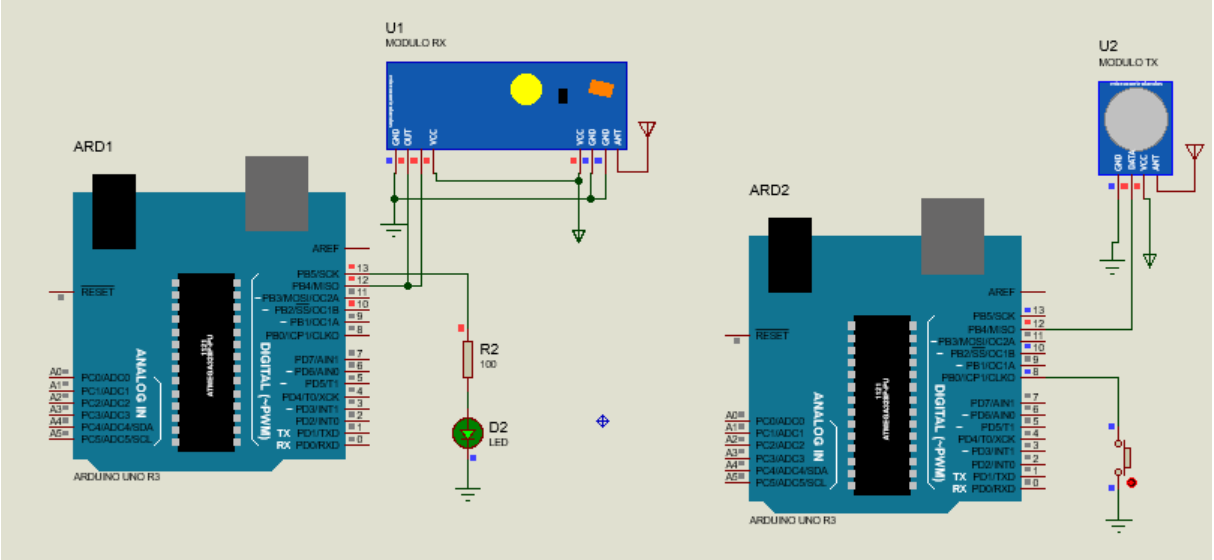
L’émetteur est constitué par le module RF et le bouton poussoir permettant de faire la commande la diode LED à distance.



RESULTAT DE SIMULATION



La figure ci-dessus représente la simulation avant appuis sur bouton poussoir rien ne se passe

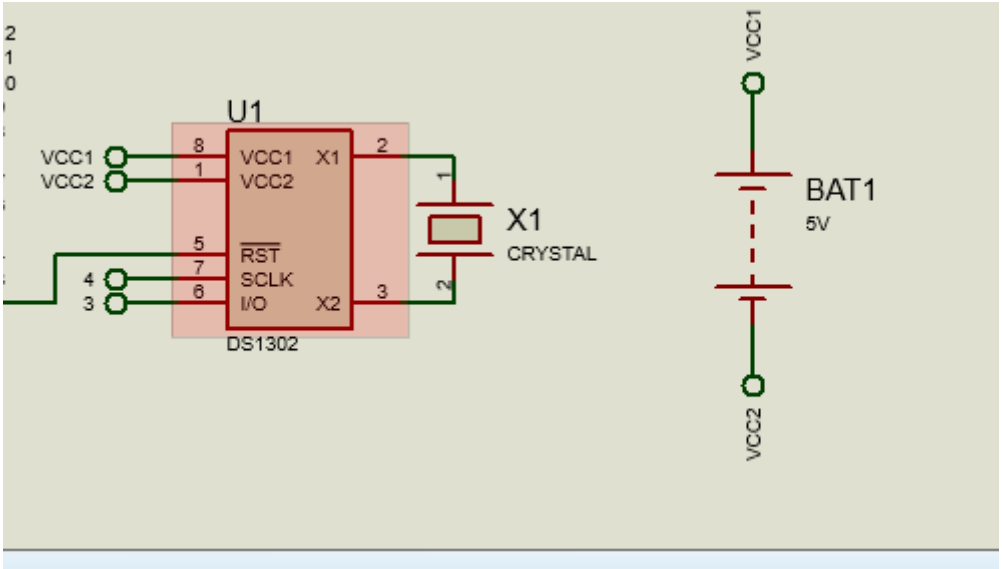


Quand on appuie sur le bouton poussoir se trouvant au niveau de l'émetteur la diode LED se trouvant au niveau du récepteur s'allume aussi longtemps que le bouton est appuyée. Par cette exemple je vous voyez comment faire la commande sans fil avec arduino c'est trop facile.

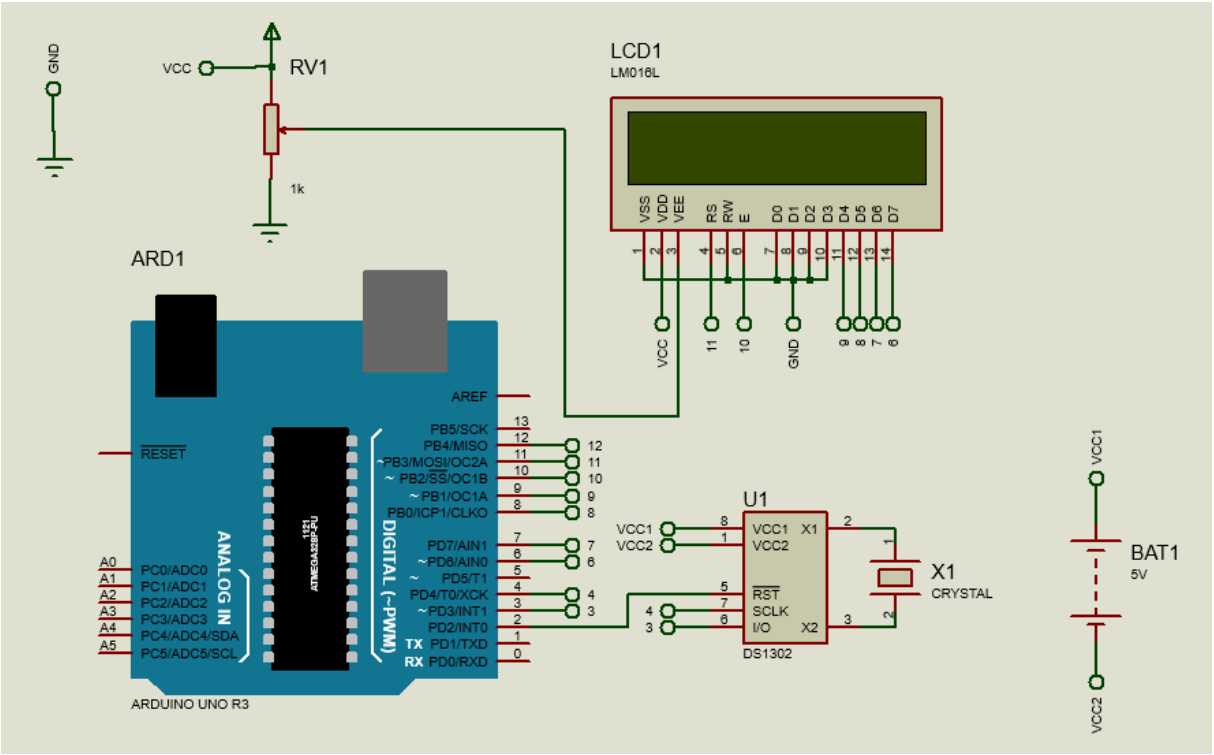
7. HORLOGE TEMPS REEL

INTRODUCTION

Lorsqu'on gérer le temps avec ARDUINO plusieurs possibilités existe et parmi on peut citer l'horloge temps réel dans notre cas nous allons étudier l'horloge DS 1302. Il existe d'autres modules permettant de gérer le temps mais le principe d'utilisation reste le même. Schéma du DS 1302 :



Ici on va utiliser une librairie ayant toujours le nom de **DS1302**
Donc si vous ne l'avez pas vous pouvez facilement le télécharger sur internet gratuitement
Voici le schéma de cablage :



Ce schéma présente le cablage du module DS 1302 sur la carte ARDUINO mais aussi la connexion de l'écran.
Par la suite voici le code :

```
#include <LiquidCrystal.h>
#include <DS1302.h>
```

Ici on appelle la librairie de l'écran et du module DS 1302

```
// Init the DS1302
DS1302 rtc(2, 3, 4);
```

Ici on déclare les broches ARDUINO qui seront connectées sur le module DS 1302

2 ira vers RST BAR du module DS 1302
3 ira vers SCLK du module DS 1302
4 ira vers I/O du module DS 1302

```
  rtc.setDOW(FRIDAY);
  rtc.setTime(12, 0, 0);
  rtc.setDate(6, 8, 2010);
```

Ces fonctions permettent le réglage de la montre
rtc.setDOW (FRIDAY) ici on règle le jour de la semaine à vendredi
rtc.setTime (12,0,0) ici on règle le temps à 12h 00min 00sec
rtc.setDate (6,8,2010) ici on règle la date au 6 aout 2010

```
(rtc.getTimeStr())
```

Cette fonction retourne temps en String.

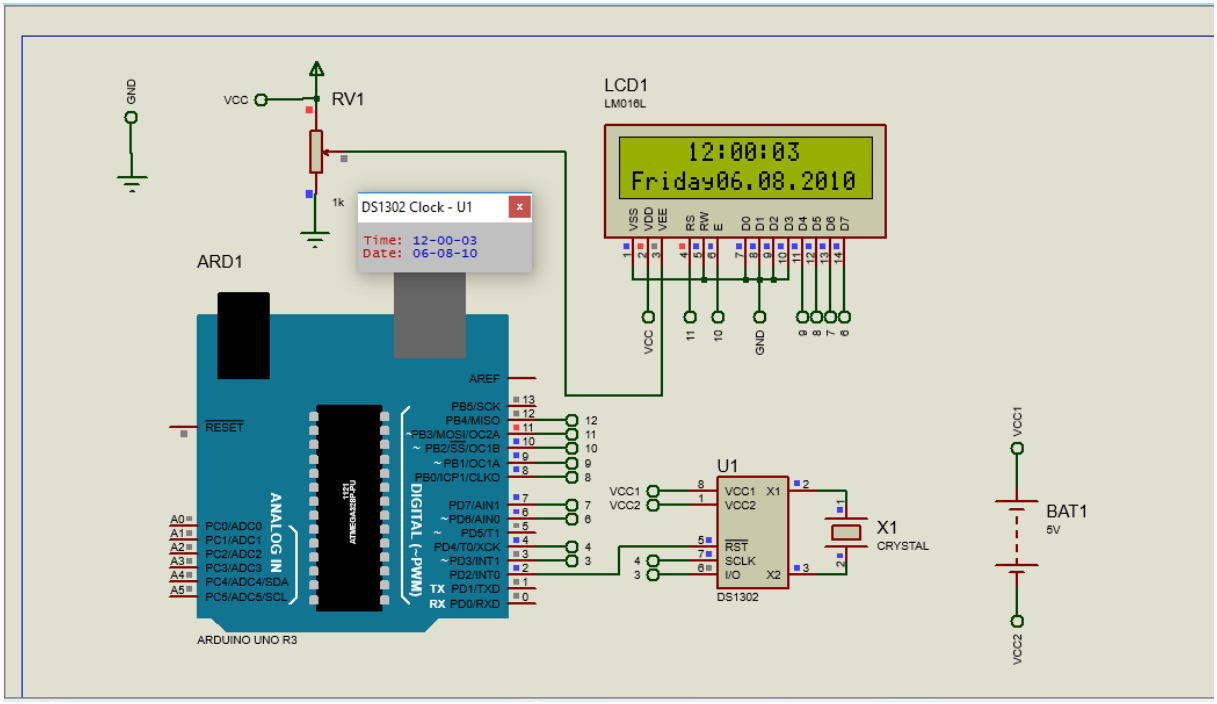
```
rtc.getDateStr());
```

Cette fonction retourne la date en String

Enfin on a le code :

```
#include <LiquidCrystal.h>
#include <DS1302.h>

// Init the DS1302
DS1302 rtc(2, 3, 4);

// Init the LCD
LiquidCrystal lcd(11, 10, 9, 8, 7, 6);

void setup()
{
  // Set the clock to run-mode, and disable the write protection
  rtc.halt(false);
  rtc.writeProtect(false);

  // Setup LCD to 16x2 characters
  lcd.begin(16, 2);

  // The following lines can be commented out to use the values already stored in the DS1302
  rtc.setDOW(FRIDAY);        // Set Day-of-Week to FRIDAY
  rtc.setTime(12, 0, 0);     // Set the time to 12:00:00 (24hr format)
  rtc.setDate(6, 8, 2010);
}

void loop()
{
  // Display time centered on the upper line
  lcd.setCursor(4, 0);
  lcd.print(rtc.getTimeStr());

  // Display abbreviated Day-of-Week in the lower left corner
  lcd.setCursor(0, 1);
  lcd.print(rtc.getDOWStr(FORMAT_SHORT));

  // Display date in the lower right corner
  lcd.setCursor(6, 1);
  lcd.print(rtc.getDateStr());

  // Wait one second before repeating :)
  delay (10);
}
```

SCHEMA EN SIMULATION :



Ici vous voyez en simulation que tout fonctionne.

NB : Mais constate une chose nous avons fait le réglage de notre montre dans le void setup cela veut dire si on réalimente notre montre une fois des alimenter il peut encore se régler ce qui n'est pas intéressant ce pourquoi on va envoyer ce code ci-dessous juste pour le réglage du RTC (horloge temps réel) puis après on envoie le code sans réglage :

horloge §

```
// Init the DS1302
DS1302 rtc(2, 3, 4);
// Init the LCD
LiquidCrystal lcd(11, 10, 9, 8, 7, 6);
void setup()
{
  // Set the clock to run-mode, and disable the write protection
  rtc.halt(false);
  rtc.writeProtect(false);

  // Setup LCD to 16x2 characters
  lcd.begin(16, 2);

  // The following lines can be commented out to use the values already stored in the DS1302
//  rtc.setDOW(FRIDAY);        // Set Day-of-Week to FRIDAY
//  rtc.setTime(12, 0, 0);     // Set the time to 12:00:00 (24hr format)
//  rtc.setDate(6, 8, 2010);
}

void loop()
{
```

```
  // Display time centered on the upper line
  lcd.setCursor(4, 0);
  lcd.print(rtc.getTimeStr());

  // Display abbreviated Day-of-Week in the lower left corner
  lcd.setCursor(0, 1);
  lcd.print(rtc.getDOWStr(FORMAT_SHORT));

  // Display date in the lower right corner
  lcd.setCursor(6, 1);
  lcd.print(rtc.getDateStr());

  // Wait one second before repeating :)
  delay (10);
}

```

Le deuxième code est sans réglage et c'est luis qui doit rester dans la carte ARDUINO et par là on peut même débrancher la carte ARDUINO et rebrancher et puis tout va bien.

8. LE CLAVIER

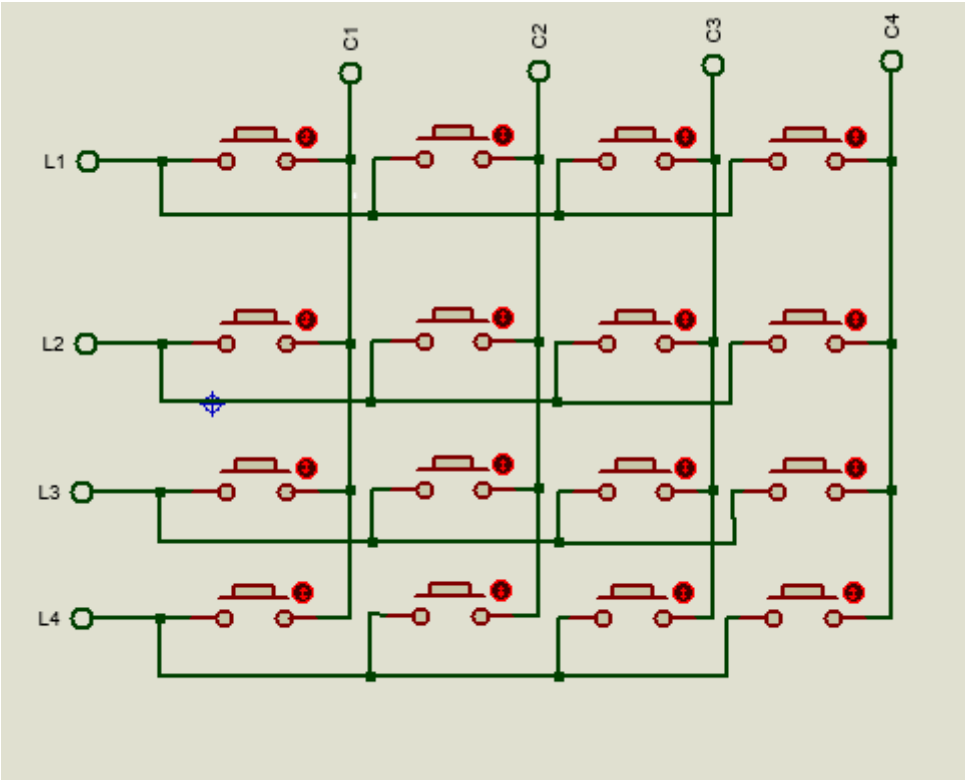
INTRODUCTION

Le clavier permet de saisir les données dans cette présente partie nous allons montrer comment utiliser le clavier avec ARDUINO.

UTILISATION DU CLAVIER

Ici nous allons utiliser un clavier matriciel

On le schéma matriciel suivant 4 fois 4



CODE DU SYSTEME

```
#include <Keypad.h>
const byte ligne = 4;
const byte colonne = 4 ;
char keys [ligne][colonne] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte lignePins[ligne]={2,3,4,5};
byte colonnePins[colonne] = {6,7,8,9};
Keypad keypad = Keypad(makeKeymap (keys),lignePins, colonnePins,ligne,colonne);
void setup() {
  Serial.begin(9600);
}
void loop() {
   char key = keypad.getKey();
  if(key!= NO_KEY){
    Serial.println(key);
  }
}
```

On commence par inclure la bibliothèque permettant la gestion du clavier :

```
#include <Keypad.h>
```

Puis on préciser le nombre de colonnes et des lignes du clavier

```
const byte ligne = 4;
const byte colonne = 4 ;
```

Puis on créer un tableau multidimensionnel dans lequel on définit les différentes valeurs du clavier :

```
char keys [ligne][colonne] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
```

Puis on spécifie les broches de la carte qui seront connectées au clavier : on spécifie ceux de lignes et de colonnes.

```
byte lignePins[ligne]={2,3,4,5};
byte colonnePins[colonne] = {6,7,8,9};
```

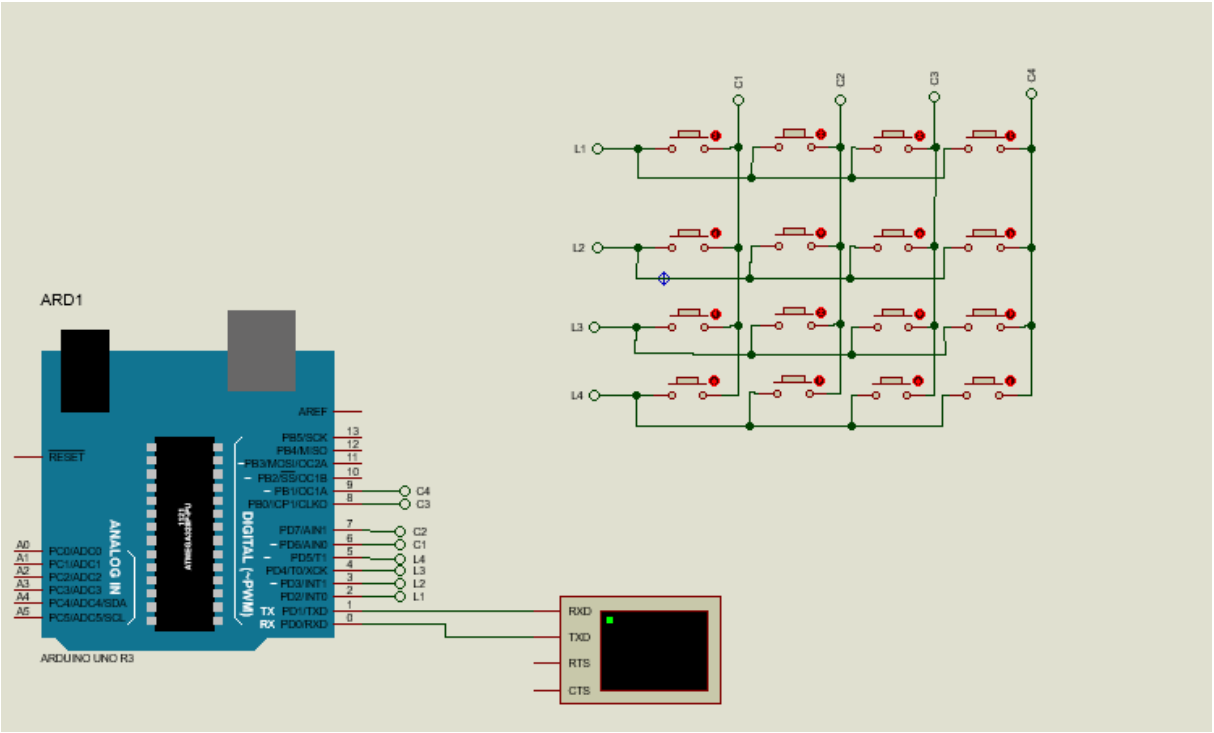
Puis on crée l'objet keypad pour l'utilisation des fonctions de la classe Keypad.

```
Keypad keypad = Keypad(makeKeymap (keys),lignePins, colonnePins,ligne,colonne);
```

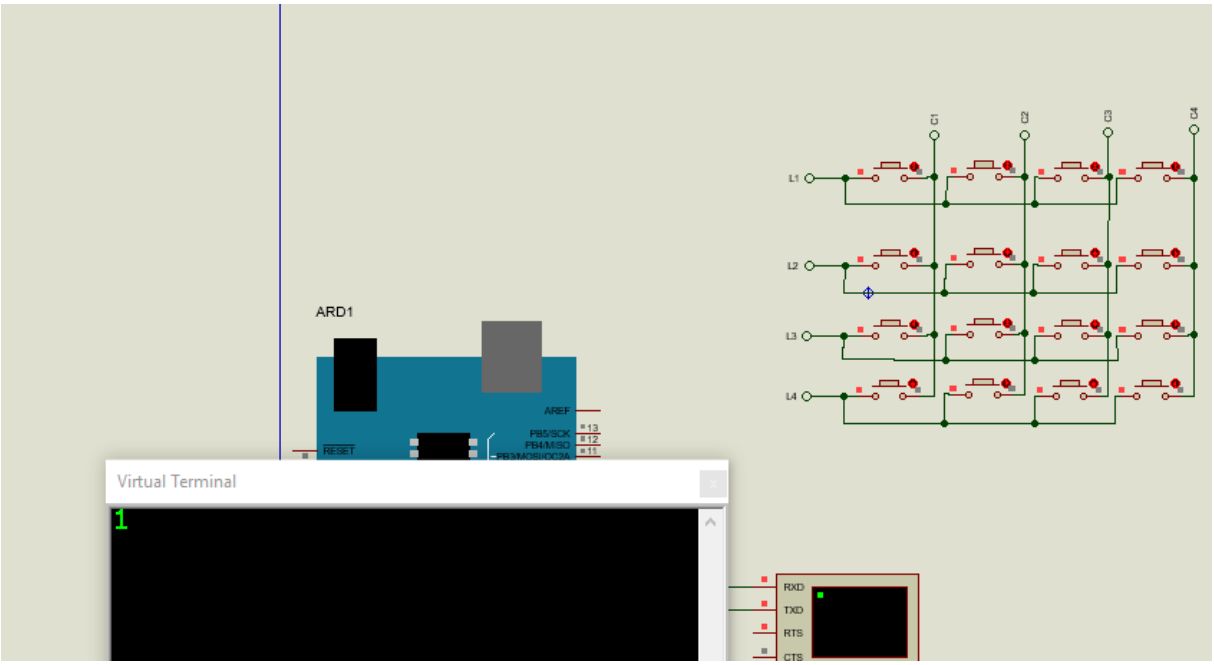
Pour lire les informations quand on a appuyé sur le clavier on utilise la fonction :

```
  char key = keypad.getKey();
```

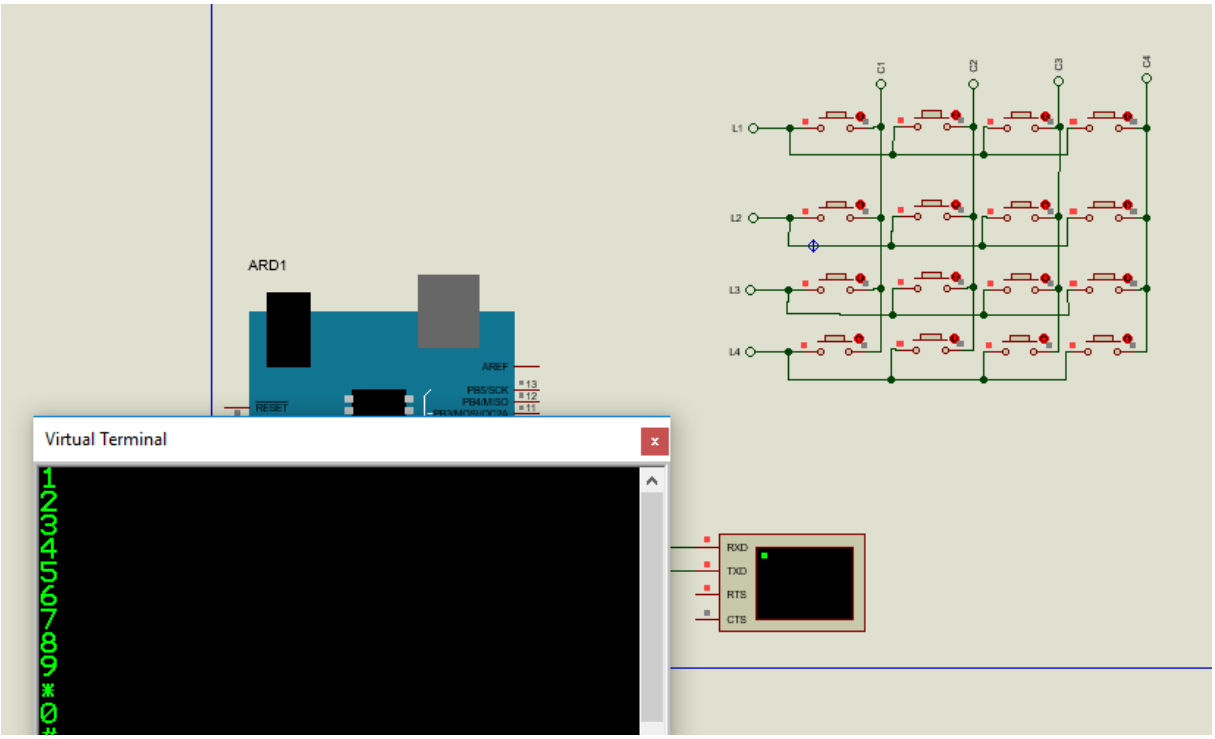
SCHEMA GENERALE DU SYSTEME :



RESUSULAT DE SIMULATION



La figure ci-dessus ce quand on appuie sur 1.



Cette figure montre l’affichage après plusieurs appuies.

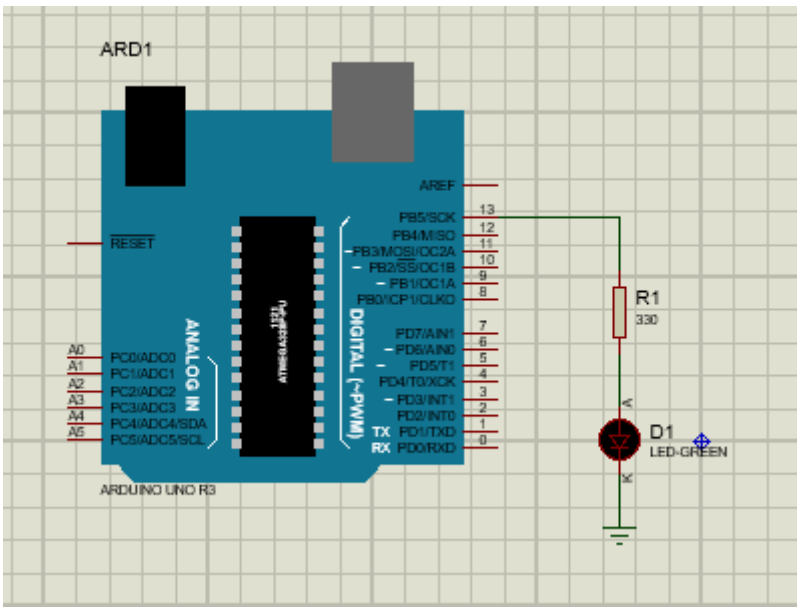
J’espère grâce à cet article vous saurez comment bien utiliser le clavier et cela avec facilité

CHAPITRE15 LA FONCTION MILLIS ET LA FONCTION MAP

1. FONCTION MILLIS

INTRODUCTION

Dans cette partie nous allons parler sur l'utilisation de la fonction millis(). Il s'agit d'une fonction qui permet de compter le temps en milliseconde lorsque la carte ARDUINO est alimentée. Une fois la carte ARDUINO est alimentée cette fonction compte le temps en milliseconde pendant tout le temps que la carte est alimentée



Pour pratiquer nous allons faire clignoter une LED en utilisant la fonction millis().

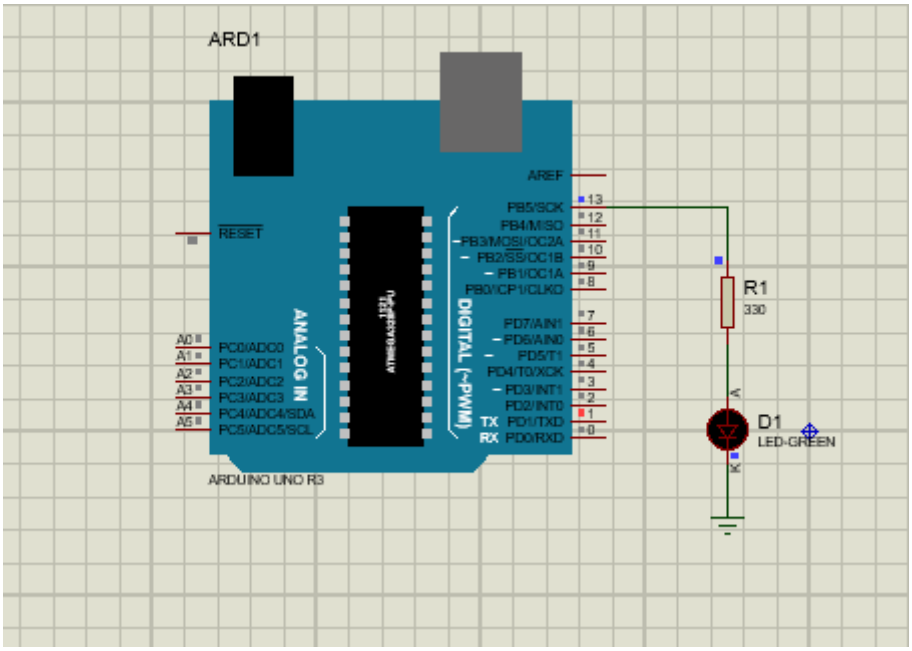
```
long temps ;
 boolean etat ;

void setup() {
 pinMode (13 ,OUTPUT);
}

void loop() {

        if (millis() - temps > 1000) {
           temps = millis();
           etat = !etat ;
        }

         digitalWrite (13 , etat) ;
}
```

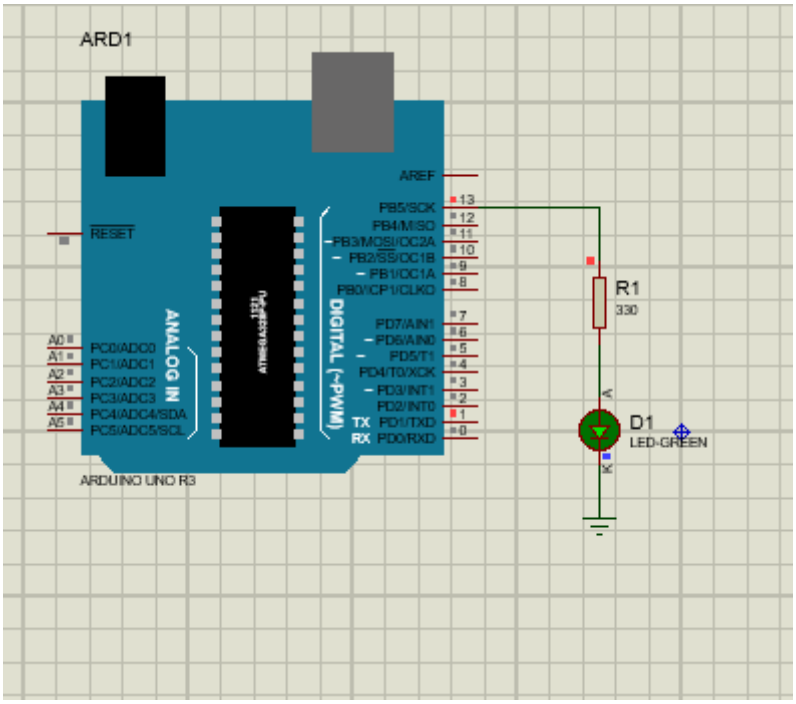
Cette fonction millis() utilise l’interruption TIMER0 cela fait que la carte ARDUINO

Peut faire autre chose tout en clignotant la LED.

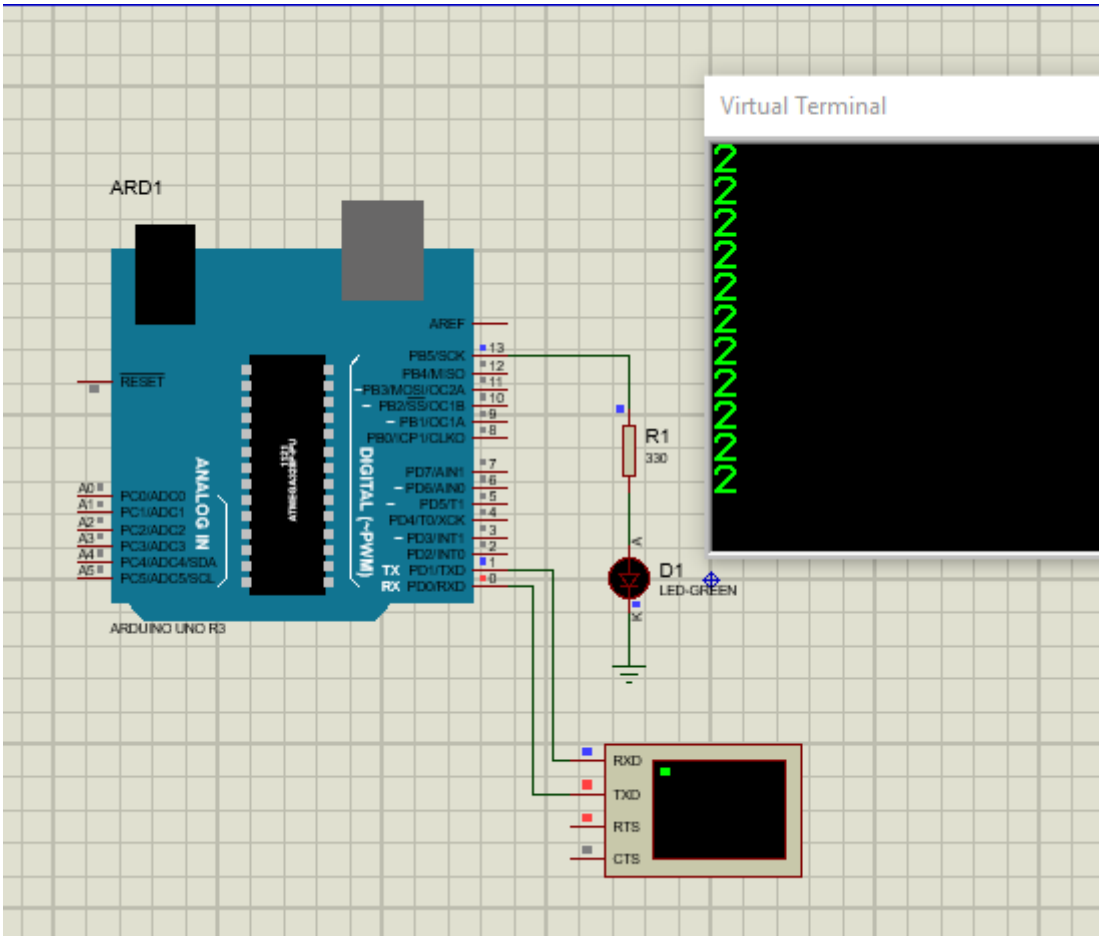
Il est aussi utile à signaler que la fonction delay (ms) met la carte ARDUINO en pause cela fait que si la carte ARDUINO est en delay (ms) même si vous appuyez sur un bouton poussoir la carte est insensible la fonction millis () va permettre d’éviter cela.

Le code ci-haut permet de clignoter une led se trouvant sur la borne 13 de la carte ARDUINO






```
long temps ;
 boolean etat ;

   int compte  ;

void setup() {
 pinMode (13 ,OUTPUT);
 Serial.begin (9600) ;
}

void loop() {

       if (millis() - temps > 1000) {
          temps = millis();
          etat = !etat  ;
           compte = compte + 1 ;
       }

        Serial.println (compte) ;
      digitalWrite (13  , etat) ;
}
```



Dans le deuxième code on clignote la lampe et on compte le temps en seconde puis on affiche ca sur le moniteur série.

2. FONCTION MAP

LA FONCTION MAP EN ARDUINO

La fonction map est une fonction en arduino qui permet de faire une correspondance des valeurs sur deux gammes bien précise.

<u>Utilisation de la fonction</u>

```
map(var , 0,255 , 300, 400);
```

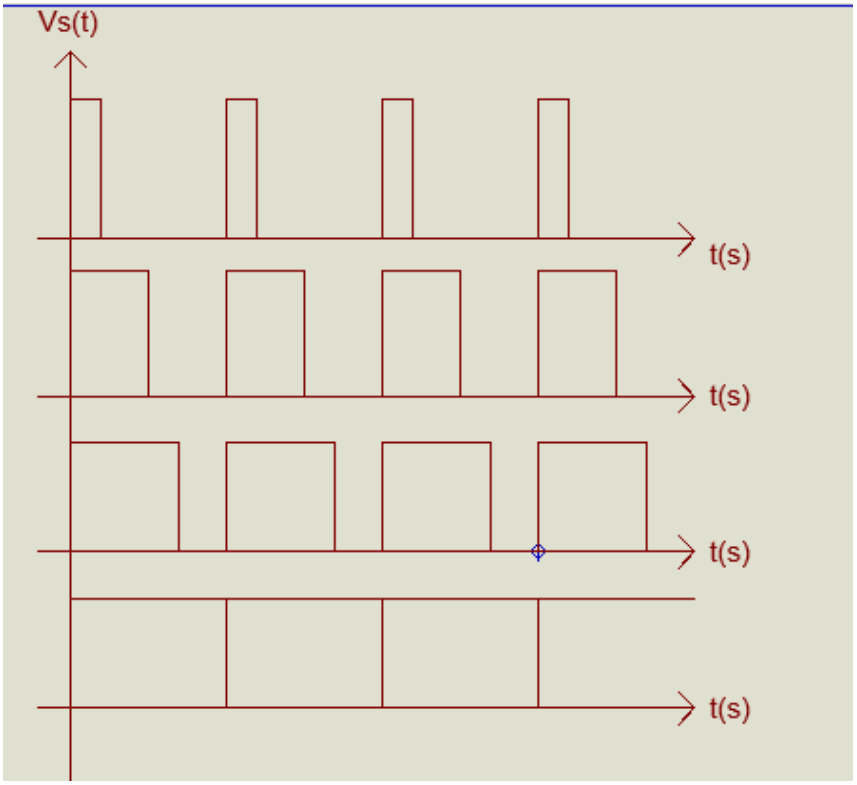
Cette fonction est constituée de cinq paramètres.

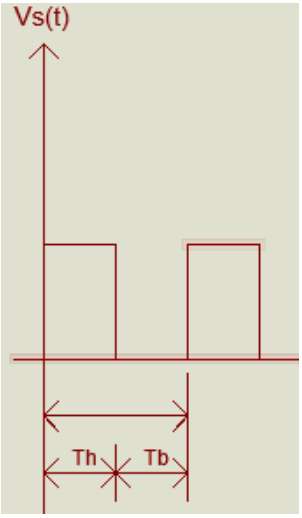
On a la variable var qui varie de 0 à 255 mais cette fonction retourne une valeur variant entre 300 et 400. Voici en quelques mots comment fonctionne cette fonction.

CHAPITRE16 LA COMMANDE PWM

COMMANDE D'UN MOTEUR A COURANT CONTINU PAR PWM

Lorsqu'on veut varier la vitesse d'un moteur électrique la première idée qui vient à l'esprit est de varier la tension aux bornes de ce dernier. Sachant que le moteur demande une tension minimale pour démarrer il devient difficile de contrôler le moteur de façon précise surtout pour de petites vitesses avec une tension qui varie de manière continue. Pour pallier à cela on utilise les signaux de modulation à largeur d'impulsion (PWM) par cette technique on envoie au moteur une tension maximale ou une tension nulle et en agissant sur le temps où la tension est maximale et le temps où la tension est minimale on varie la tension moyenne et par conséquent on varie la vitesse du moteur.





Signaux PWM

En variant le temps haut (Th) et le temps bas (Tb) ; on varie la tension moyenne et par là on varie la vitesse du moteur à courant continu voir équations suivantes :

On sait que : $V_s moy = \frac{1}{T}\left[\int_0^{Th} Vs_{(t)} \,.\, dt + \int_{Th}^{Th+Tb} Vs_{(t)} . dt\right]$ (1)

Dans les intervalles :

$[0, Th]$: La tension $Vs_{(t)} = Vs_{max}$

$[0, Th]$: La tension $Vs_{(t)} = 0$

Alors l'intégrale (1) devient :

$V_s moy = \frac{1}{T}\left[\int_0^{Th} Vs_{max} \,.\, dt + \int_{Th}^{Th+Tb} O . dt\right]$

$\Rightarrow V_s moy = \frac{1}{T}\left[\int_0^{Th} Vs_{max} \,.\, dt\right]$

Avec Vs_{max} $une\ constante$

On a :

$$V_s moy = \frac{Vs_{max}}{T}\left[\int_0^{Th} 1 . dt\right]$$

$\Rightarrow V_s moy = \frac{Vs_{max}}{T} \,.\, Th$

Avec : $T = Th + Tb$

On a :

$\boldsymbol{V_s moy = \frac{Vs_{max}}{Th+Tb} \,.\, Th}$ **(Tension moyenne)**

En arduino lorsqu'on veut faire la génération des signaux PWM on commence d'abord par identifier la broche de la communication. Pratiquement sur cette broche on met un symbole alternatif.

En générant un signal PWM avec ARDUINO on peut aller de 0 à 5V soit de 0 à 100%

On a le code suivant :

sketch_feb11b | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

sketch_feb11b §

```
1
2  void setup() {
3
4  pinMode (11, OUTPUT); /* ici on
5  configure la broche qui va générer le
6  signal PWM en sortie
7  */
8  }
9
10 void loop() {
11
12 analogWrite (11, 50);
13  /*
14   *  ici on génére le signal PWM
15   *  à la borne 11
16   *  on de 0 à 100 %
17   *   soit de 0 à 255
18   *   dans la fonction analogWrite
19   */
20
21 }
```

CHANGEMENT DE FREQUENCE

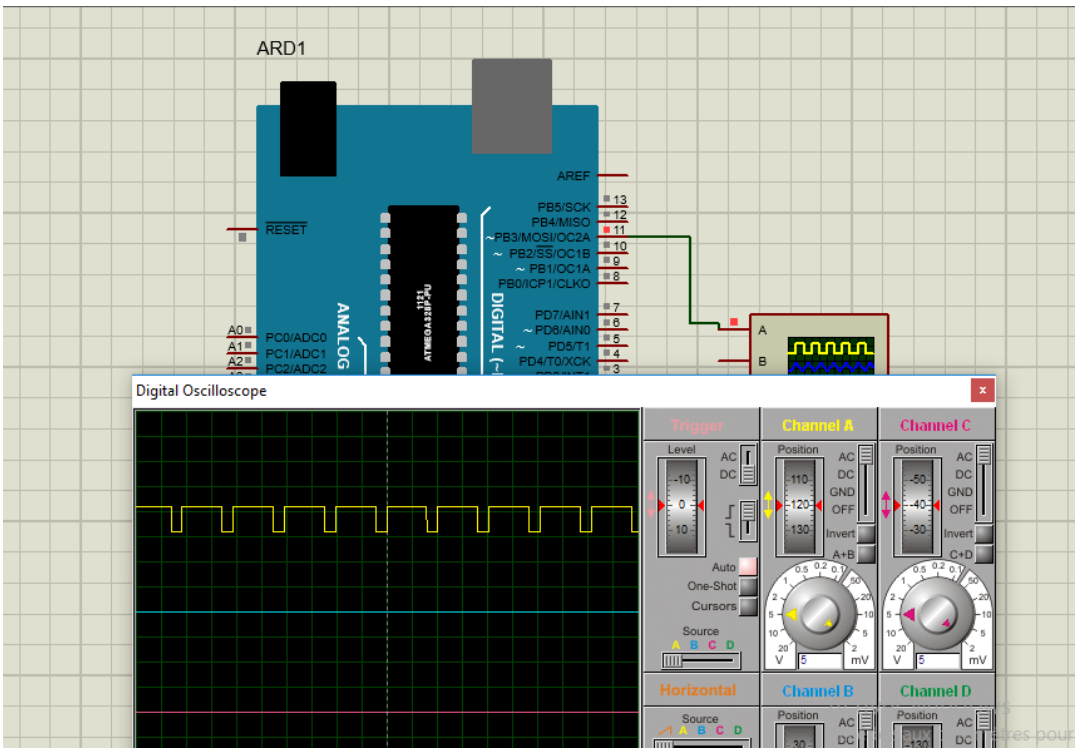
INTRODUCTION

Dans ce présent travail nous allons montrer comment changer la fréquence du signal PWM avec ARDUINO. Lorsqu'on fait un projet de fois on a besoin de changer la fréquence du PWM pour la commande du moteur

CODE POUR UNE FREQUENCE DE 62 500 Hz

pwm_freq §

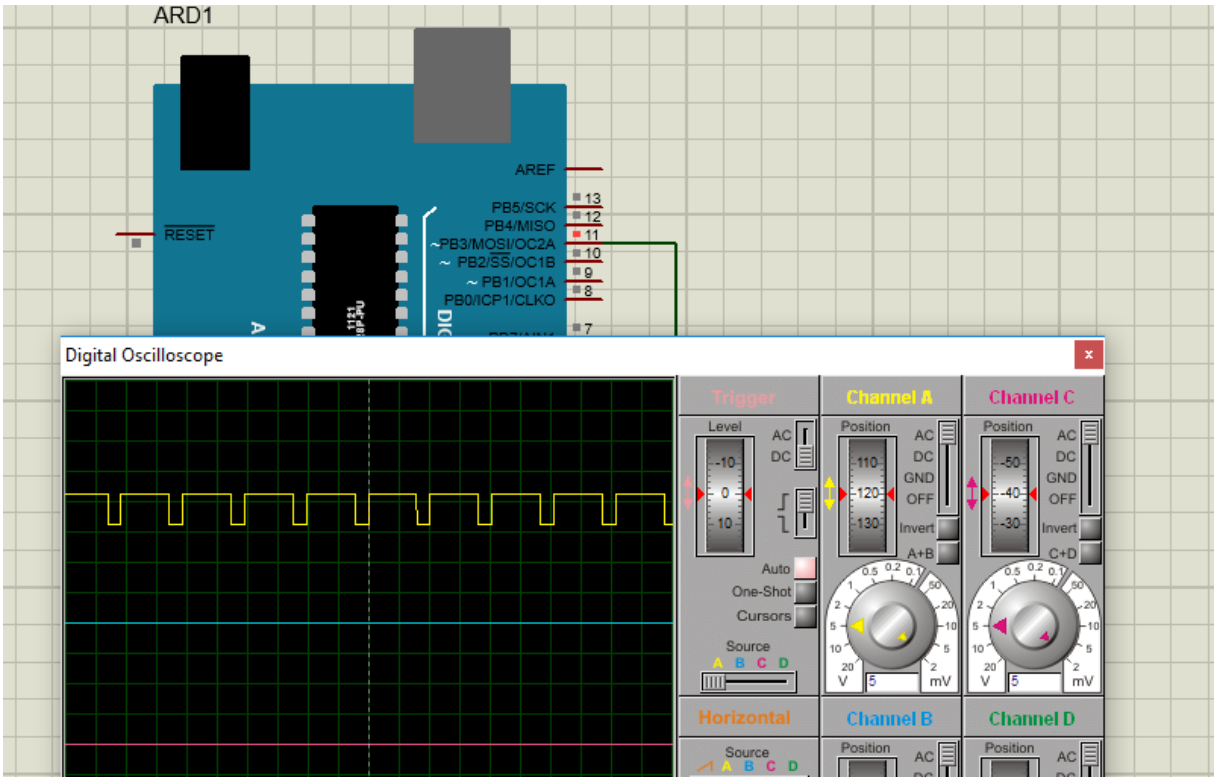
```
1
2   int obtentionFrequence = 0x01 ;
3
4 void setup() {
5
6   pinMode (11 ,OUTPUT);
7   TCCR0B = TCCR0B & 0b11111000 | obtentionFrequence ;
8
9 }
10
11 void loop() {
12
13  analogWrite (11 , 200);
14
15 }
```



CODE POUR UNE FREQUENCE DE 62 500 Hz/8

pwm_freq §

```
1
2   int obtentionFrequence = 0x02 ;
3
4 void setup() {
5
6   pinMode (11 ,OUTPUT);
7   TCCR0B = TCCR0B & 0b11111000 | obtentionFrequence ;
8
9 }
10
11 void loop() {
12
13  analogWrite (11 , 200);
14
15 }
```



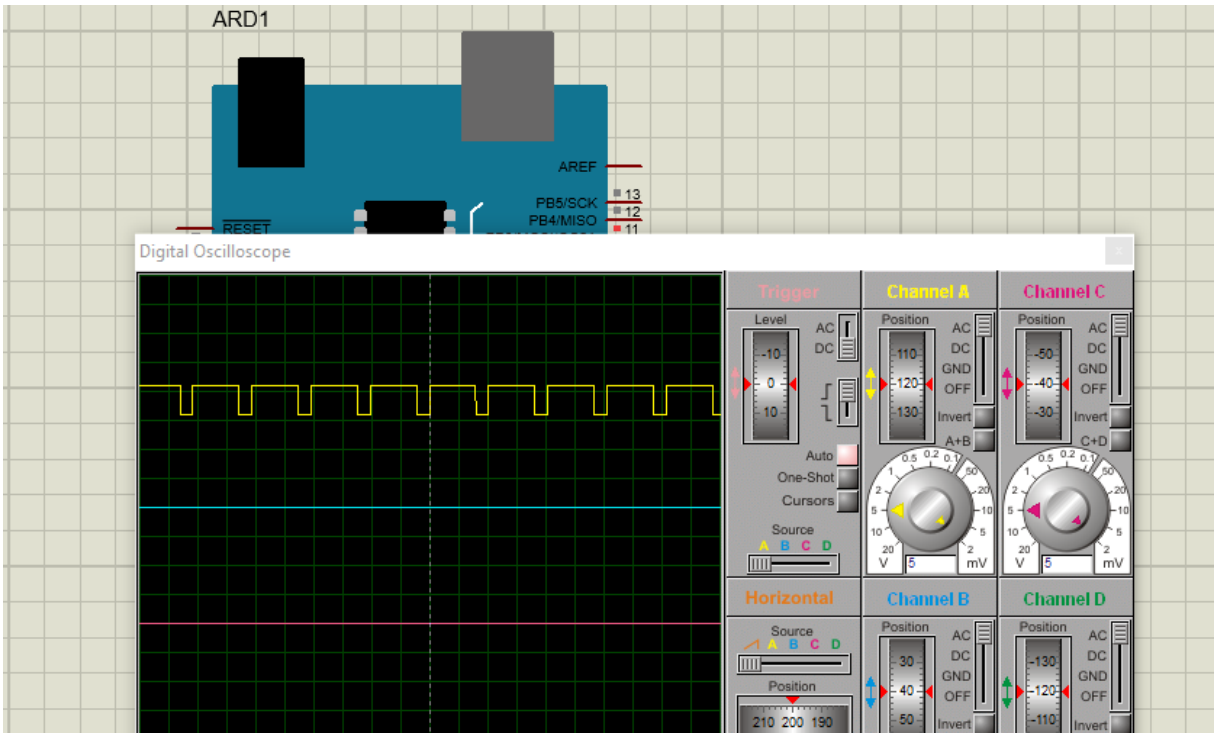
CODE POUR UNE FREQUENCE DE 62 500 Hz/64

pwm_freq | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

pwm_freq §

```
1
2   int obtentionFrequence = 0x03 ;
3
4 void setup() {
5
6   pinMode (11 ,OUTPUT);
7   TCCR0B = TCCR0B & 0b11111000 | obtentionFrequence ;
8
9 }
10
11 void loop() {
12
13  analogWrite (11 , 200);
14
15 }
```



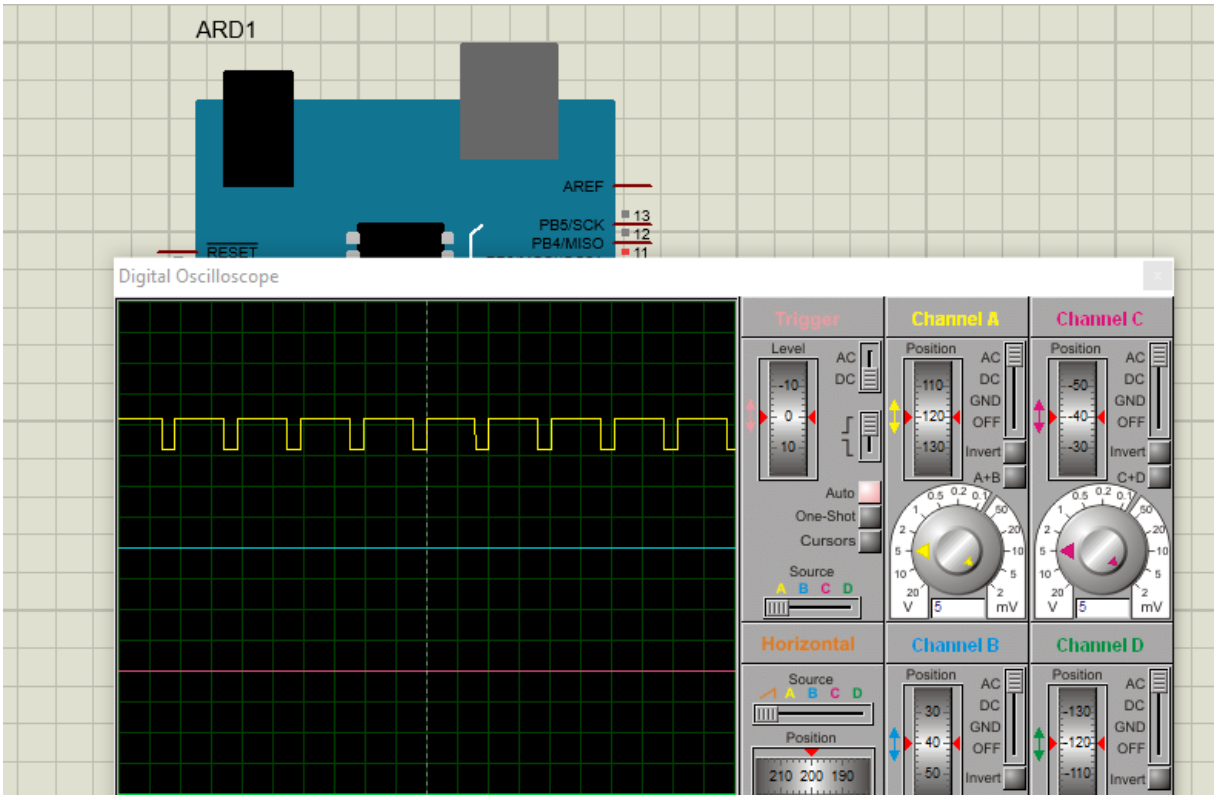
CODE POUR UNE FREQUENCE DE 62 500 Hz/256

pwm_freq | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

pwm_freq §

```
1
2   int obtentionFrequence = 0x04 ;
3
4 void setup() {
5
6   pinMode (11 ,OUTPUT);
7   TCCR0B = TCCR0B & 0b11111000 | obtentionFrequence ;
8
9 }
10
11 void loop() {
12
13  analogWrite (11 , 200);
14
15 }
```

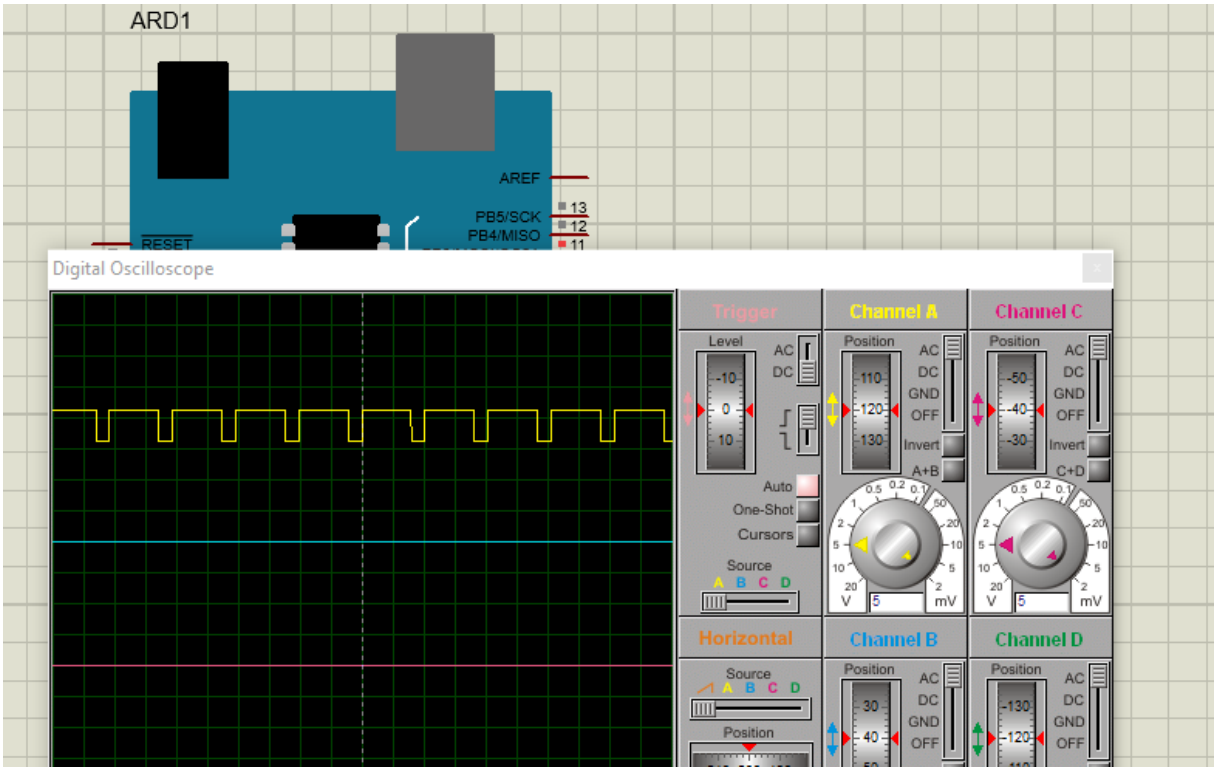


CODE POUR UNE FREQUENCE DE 62 500 Hz/1024

Fichier Édition Croquis Outils Aide

pwm_freq §

```
1
2   int obtentionFrequence = 0x05 ;
3
4 void setup() {
5
6   pinMode (11 ,OUTPUT);
7   TCCR0B = TCCR0B & 0b11111000 | obtentionFrequence ;
8
9 }
10
11 void loop() {
12
13  analogWrite (11 , 200);
14
15 }
```


ARD1
AREF
PB5/SCK
PB4/MISO
13
12
11
RESET
Digital Oscilloscope
Trigger
Level
AC
DC
Auto
One-Shot
Cursors
Source
A B C D
Channel A
Position
AC
DC
GND
OFF
Invert
A+B
V
mV
Channel C
C+D
Horizontal
Channel B
Channel D

CHAPITRE17 LE POINTEUR

INTRODUCTION

Un pointeur est une variable contenant l'adresse d'une autre variable. Il est caractérisé par son nom et son type et la variable qu'il pointe.

Le pointeur est très important en C tous programmeurs Arduino, C, C++ digne de ce nom doit maitriser la notion des pointeurs.

CREATION D'UN POINTEUR

```
1 int *nomPointeur ;
2
```

On déclare la variable en commençant par le symbole de multiplication

Un pointeur pointe sur une variable voir :

```
1 int var = 10;
2 int *nomPointeur ;
3 *nomPointeur =&var ;
4
```

Ici on voit que notre pointeur pointe la variable sur la variable var

En écrivant

```
*nomPointeur ;
```

Ici on accède à la valeur de la variable que l'on pointe

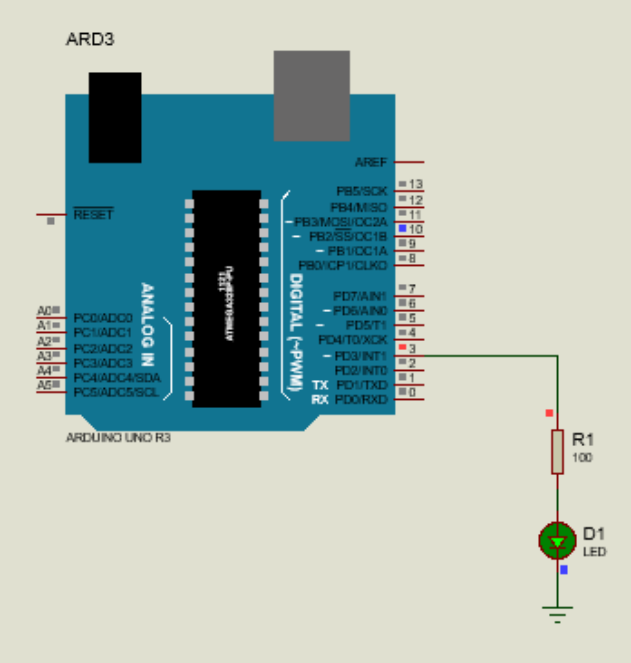
```
nomPointeur ;
```

Ici on accède à l'adresse de la variable que l'on pointe.

CODE USANT LE POINTEUR

pointeur

```
1  int pin = 10;
2
3
4  void setup() {
5   pinMode (pin,OUTPUT) ;
6   pinMode (3,OUTPUT);
7  }
8
9  void loop() {
10 changementBroche (&pin) ;
11 digitalWrite (pin,HIGH) ;
12 delay (1000) ;
13 digitalWrite (pin,LOW) ;
14 delay (1000) ;
15 }
16
17  void changementBroche (int *pointeBroche) {
18   *pointeBroche = 3 ;
19  }
20
```



Ici sur la simulation ci-dessus on voit que la lampe clignote à la borne 3.

CHAPITRE18 LA STRUCTURE

Introduction

Les structures sont utilisées pour qu'on y stocke des données ils peuvent stocker des informations de plusieurs types différents ce qui le différencient bien des tableaux stockant des informations de même types.

CREATION D'UNE STRUCTURE

On crée une structure de cette manière :

```
struct led {

  int ledPin ;
  float allumee = 1 ;
  float eteinte = 0 ;

};
```

On crée une structure avec le mot clé : *truct nom_de_la_structure { attributs de la structure }*

Les attributs sont des variables se trouvant au seins de structure ces variables peuvent être de tous types entiers, flottants, caractères ou chaines de caractères.

Une fois on a déjà créer la structure on doit créer une variable de cette structure et par là rien de plus compliqué il suffit juste d'utiliser le nom de la structure puis le nom de la variable de la structure on l'instruction suivante :

```
led leRouge ;
```

Ici on crée une variable de la structure et cette variable a comme nom *ledRouge*

```
 struct led {

   int ledPin ;
   float allumee = 1 ;
   float eteinte = 0 ;

 };

  led leRouge ;

void setup() {
  leRouge.ledPin = 13  ;
  pinMode (leRouge.ledPin ,OUTPUT) ;
}

void loop() {

  digitalWrite (leRouge.ledPin , leRouge.allumee) ;
  delay (1000) ;
  digitalWrite (leRouge.ledPin , leRouge.eteinte) ;
  delay (1000) ;


}
```

Pour accéder aux informations se trouvant dans la structure on utilise le nom de la variable de la structure on fait point puis on met le nom de la variable dont on veut utiliser et se trouvant dans la structure.

Par exemple le programme ci haut consiste à faire clignoter une led se trouvant à la broche 13 avec un délai d'une seconde.

CHAPITRE19 SURCHAGE DE METHODES

INTRODUCTION

La surdéfinition des fonctions est un concept utiliser en programmation il s'agit d'une technique permettant de donner à deux ou plusieurs fonctions un même nom mais avec un nombre de paramètres différents. Cette technique est encore appelée la surcharge de fonctions ou surcharges de méthodes. Elle est utilisé lorsqu'on veut avoir une même donc portant un même nom mais s'exécutant différemment.

```
1  void allumgeLED (int ledA) {
2    digitalWrite (ledA, HIGH) ;
3  }
4
5   void allumageLED (int ledA , int ledB) {
6      digitalWrite (ledA, HIGH) ;
7       digitalWrite (ledB, HIGH) ;
8   }
9
10  void allumageLED (int ledA, int ledB , int ledC) {
11     digitalWrite (ledA, HIGH) ;
12      digitalWrite (ledB, HIGH) ;
13       digitalWrite (ledc, HIGH) ;
14  }
```

Ici il s'agit de trois fonctions portant le même mais avec des paramètres différents

On voit que la première fonction a un seul paramètre la deuxième fonction a deux paramètres et afin la troisième fonction qui a trois paramètres.

Ce fonction porte le même nom mais il y a une différence au niveau de paramètres ici on parle de la surcharge des fonctions.

Alors pour appeler une fonction ou l'autre on procède de cette manière :

```
27 allumageLED (3, 4 , 5)  ;
28 allumageLED (6,7);
29 allumageLED (8);
```

En appelant le nom de la fonction avec un seul paramètre ici on appelle la première fonction

En appelant le nom de la fonction avec deux paramètres ici on appelle la deuxième fonction

En appelant le nom de la fonction avec trois paramètres on appelle la troisième fonction

CODE COMPLET

```
1  void allumgeLED (int ledA) {
2    digitalWrite (ledA, HIGH) ;
3  }
4
5   void allumageLED (int ledA , int ledB) {
6      digitalWrite (ledA, HIGH) ;
7       digitalWrite (ledB, HIGH) ;
8   }
9
10   void allumageLED (int ledA, int ledB , int ledC) {
11      digitalWrite (ledA, HIGH) ;
12       digitalWrite (ledB, HIGH) ;
13        digitalWrite (ledc, HIGH) ;
14   }
15 void setup() {
16
17  pinMode (3 , OUTPUT) ;
18  pinMode (4 , OUTPUT) ;
19   pinMode (5 , OUTPUT) ;
20  pinMode (6, OUTPUT) ;
21   pinMode (7 , OUTPUT) ;
22  pinMode (8 , OUTPUT) ;
23 }
24
```

```
25 void loop() {
26
27 allumageLED (3, 4 , 5)  ;
28 allumageLED (6,7);
29 allumageLED (8);
30 }
```

Ici on peut bien que ce code consiste à faire passer à 1 les broches 3, 4, 5, 6,7 et 8

Voilà en quelques mots la notion de constructeur.

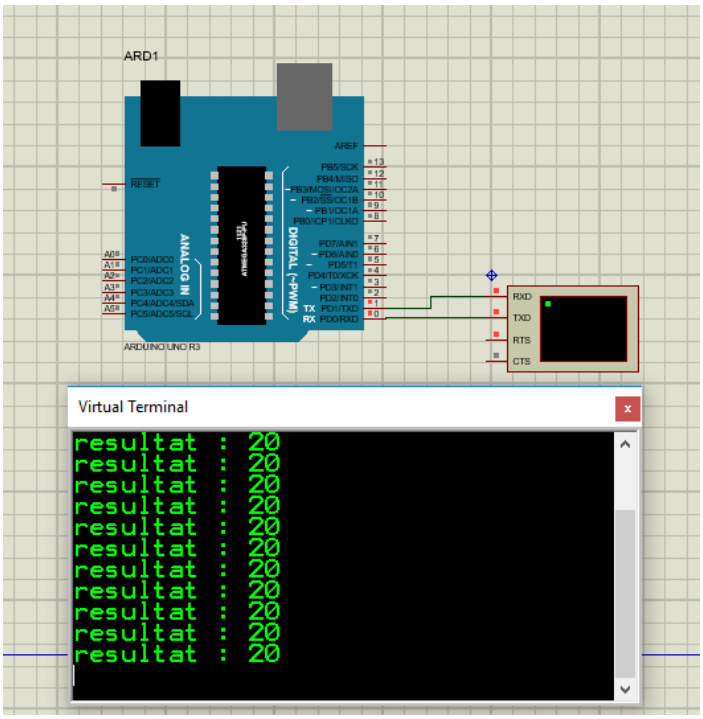
CHAPITRE20 LES FONCTIONS TEMPLATES

INTRODUCTION

Imagine par exemple que vous créez une fonction permettant de multiplier deux variables int placées en paramètres ; vous écrirez :

```
1
2  int multiplication( int a , int b) {
3    return a*b ;
4  }
5 void setup() {
6   Serial.begin (9600) ;
7
8 }
9
10 void loop() {
11  Serial.print ("resultat : ") ;
12  Serial.println (multiplication( 4,5)) ;
13
14 }
```

En analysant cette fonction nous voyons très bien qu'elle retourne une valeur de 20 car multipliant deux paramètres.

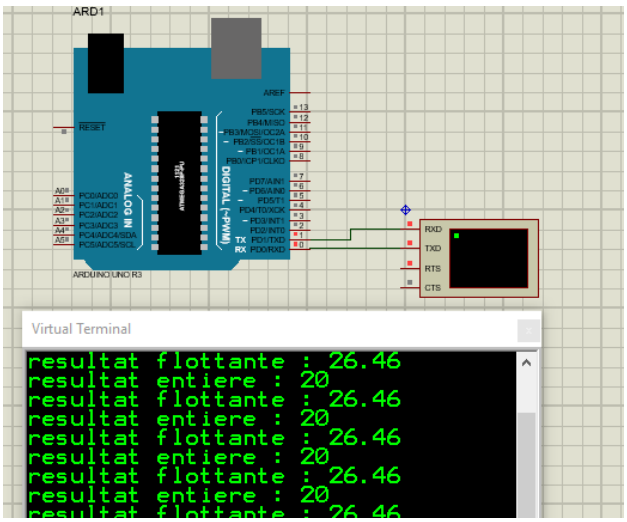


Ici on voit très bien qu'on a le résultat 20 mais on voit très bien que cette fonction ne retourne que les valeurs entières. Pour traiter les valeurs flottantes il faudra encore créer une autre fonction permettant de gérer les valeurs flottantes on a :

test_template §

```
1
2  int multiplication( int a , int b) {
3    return a*b ;
4  }
5
6  float multiplicatioFlottante (float a , float b) {
7     return a*b ;
8  }
9 void setup() {
10   Serial.begin (9600) ;
11
12 }
13
14 void loop() {
15  Serial.print ("resultat entiere : ") ;
16  Serial.println (multiplication( 4,5)) ;
17
18   Serial.print ("resultat flottante : ") ;
19  Serial.println (multiplicatioFlottante( 4.9,5.4)) ;
20
21
22 }
```

Résultat de simulation :

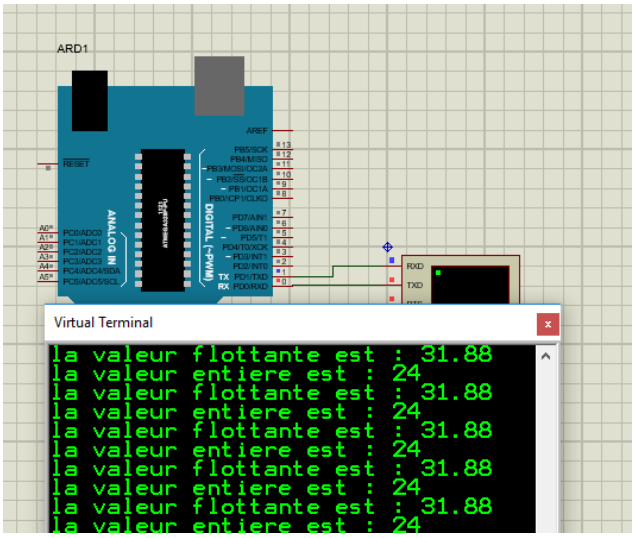


Ici nous voyons très bien qu'il nous fallait créer deux fonctions l'une pour traiter les valeurs entières et l'autre pour traiter les valeurs flottantes. Les fonctions patrons ou templates vont raccourcir cette méthode en effet il s'agit de fonction dont le type qui sera retourné varie en fonction des paramètres donc il s'agit d'une fonction qui s'adapte facilement par rapport au type que vous lui fournissaient en paramètres.

fonction_patron §

```
1
2 template <class T> T classTemplate  (T a, T b){
3  return a*b;
4 }
5
6
7 void setup() {
8   Serial.begin (9600);
9
10 }
11
12 void loop() {
13
14 Serial.print ("la valeur entiere est : ") ;
15 Serial.println (classTemplate (6,  4)) ;
16
17 Serial.print ("la valeur flottante est : ") ;
18 Serial.println (classTemplate (6.67,  4.78)) ;
19 }
```

Résultat de simulation :



Avec la fonction patron on a aussi des bons résultats mais on écrits moins de code

fonction_patron

```
1
2 template <class T> T classTemplate  (T a, T b){
3
4  return a*b;
5
6 }
7
```

Cette fonction ci-haut est une **fonction Template.**

CHAPITRE21 UN PEU DE PROGRAMMATION ORIENTEE OBJET

INTRODUCTION
La programmation orientée objet est une nouvelle manière de programmer. On programme comme si on manipule des objets du monde du réel et par là le codage devient simple. Cette programmation facilite le développement en groupe et permet de faire de programme robuste et évolutif dans le temps il est important que tout programmeurs voulant loin dans la programmation de connaitre cela.
CONCEPTS DE LA PROGRAMMATION ORIENTEE OBJET (POO)
Il y a 4 concepts en programmation orienté objet :
1. L'encapsulation
2. L'abstraction
3. L'héritage
4. Et le polymorphisme

1. ENCAPSULATION

Une carte ARDUINO contient un nombre limité des entrées et sorties numériques imaginé alors un programmeur qui veut faire clignoter une LED se trouvant la broche 100 de la carte ARDUINO en écrivant :

```
1  void setup() {
2     pinMode (100,OUTPUT) ;
3
4  }
5
6  void loop() {
7   digitalWrite (100,HIGH) ;
8   delay (1000) ;
9   digitalWrite (100,LOW) ;
10  delay (1000) ;
11
12 }
```

PROCEDURE POUR FAIRE LA POO EN ADUINO AVEC PLUSIEURS FICHIERS
On clique sur :

Puis il y une liste qui apparait puis on choisit nouvel onglet
Alors il faut donner au nouvel onglet un nom

Vous devez donner le nom de l'onglet .cpp :

Puis vous créez un autre fichier vous donnez le nom suivi de .h :

La fonction point h contient les entêtes des fonctions
La fonction point cpp contient la définition des fonctions
Faisons un programme permettant de faire clignoter une LED en utilisant la notion de classe :
Nous avons le code du fichier .h :

mail § | clignot.cpp | clignot.h

```
#include <Arduino.h>
class led{
  public :
  void allumee();

  void eteinte();

  void configuration();



  private :
  int pinLed=13;
};
```

Code du fichier .cpp
Dans ce fichier on fait l'implémentation des méthodes et ici dans ce fichier on implémente les trois méthodes : allumee () ; eteinte () ; configuration ().

mail § | clignot.cpp | clignot.h

```cpp
#include "clignot.h"
void led :: allumee(){
  digitalWrite(pinLed,HIGH);

}
void led :: eteinte(){
  digitalWrite(pinLed,LOW);

}
void led :: configuration(){
  pinMode(pinLed,OUTPUT);

}
```

Et enfin nous avons le code principal :

mail § | clignot.cpp | clignot.h

```
1  #include"clignot.h"
2  led ledjaune ;
3  void setup() {
4    ledjauneconfiguration();
5
6
7  }
8
9  void loop() {
10   ledjauneallumee();
11   delay(1000);
12   ledjauneeteinte();
13   delay(1000);
14
15
16 }
```

UN PEU D'ENCAPSULATION

INTRODUCTION

L'encapsulation est une technique de la programmation objet permettant aux programmes d'être robuste. Ce concept consiste à rendre la plupart d'attributs privés et accéder à ces derniers en passant par des fonctions publiques généralement appelés mutateurs et accesseurs.

Laissons la théorie et expliquons l'encapsulation par un exemple de la vie courante :

Imagine on vous donne un marché de faire une carte électronique pouvant contrôler la montée et la descente d'un ascenseur dans une maison de 100 étages. Bien sûr votre système aura un clavier auquel l'utilisateur pourra saisir l'étage auquel il se dirige. Alors imaginé que votre utilisateur saisi par exemple 500 donc il veut aller au niveau 500 et pourtant la maison n'a que 100 niveaux avant de faire la commande vous allez d'abord analyse la validité de la donnée c'est l'importance de l'encapsulation.

Imagine encore vous avez une carte ARDUINO UNO vous faites un programme ou une librairie permettant de gérer les entrés sorties puis du coup vous écrivez

pinMode (variable, OUPUT) avec variable = 100. Ce qui ne marchera jamais l'encapsulation va vous permettre d'analyser une donnée d'une variable.

Nous allons faire un petit projet permettant de faire clignoter une LED se trouvant à la borne que le programmeur choisi et s'il arrive que le programmeur a mal choisi soit il met un nombre trop supérieur ou un nombre négatif le système allume une lampe rouge.

CODE : ici l'utilisateur spécifie la broche sur laquelle la led pourra clignoter s'il choisit une valeur une valeur supérieur à 13 ou une valeur inférieure 0 une led se trouvant à la borne 3 signal un mauvais choix.

Code principale :

mainEncaps | cligno.cpp | cligno.h

```
#include "cligno.h"

clignot ledRouge ;

void setup() {

ledRouge.setPinLed(4);
pinMode (ledRouge.getPinLed (), OUTPUT) ;

}

void loop() {

  ledRouge.clignotement();

}
```

Fichier cligno.cpp :

mainEncaps | cligno.cpp § | cligno.h

```
#include "cligno.h"
void clignot :: clignotement (){
  digitalWrite (pinLed , HIGH);
  delay (1000) ;
  digitalWrite (pinLed , LOW);
  delay (1000) ;
}
void clignot :: setPinLed(int pinLed){
  if (pinLed < 0 || pinLed > 13 ) {
    pinMode (ledAlerte,OUTPUT);
    digitalWrite (ledAlerte,HIGH);
  } else {

    this->pinLed = pinLed  ;
    digitalWrite (ledAlerte,LOW);
  }
}
int clignot :: getPinLed () {
  return pinLed  ;
}

```

Fichier cligno.h :

mainEncaps | cligno.cpp | cligno.h

```
#include "Arduino.h"

 class clignot {

   public :
   void clignotement ();
   void setPinLed(int pinLed);
   int getPinLed () ;

   private :

    int pinLed  ;
    const int ledAlerte = 3;

   protected :

 };
```

Comme vous pouvez constater dans ce code nous faisons une analyse de la broche sur laquelle va clignoter la led en déclarant une variable ***int pinLed*** comme privée l'accès et nous accédons à cette variable par une fonction qu'on appelle **accesseur**. Cette fonction nous permet d'accéder à la valeur de la variable

```
int clignot :: getPinLed () {
  return pinLed  ;
}
```

Cette fonction ci-haut qu'on appelle **accesseur** prend le même type que celui de la variable car celle-ci retourne la valeur de la variable.

Ensuite nous avons aussi utilisée une fonction qu'on appelle mutateur : cette fonction permet d'affecter à la variable une valeur donnée bien sur sous certaines conditions de notre programme.

```
8  void clignot :: setPinLed(int pinLed){
9    if (pinLed < 0 || pinLed > 13 ) {
10     pinMode (ledAlerte,OUTPUT);
11     digitalWrite (ledAlerte,HIGH);
12   } else {
13
14     this->pinLed = pinLed  ;
15     digitalWrite (ledAlerte,LOW);
16   }
17 }
```

Ici on peut voir dans le code ci-dessus que l'on varie la valeur de la variable **pinLed** sous une certaine condition.

```
13
14     this->pinLed = pinLed  ;
15
```

Ce code montre que l'on change la valeur de la variable courante de la classe.

CHAPITRE22. CREATION DES LIBRAIRIES

INTRODUCTION

Les librairies ce sont des recueils de code qui permettent au programmeur d'utiliser des fonctionnalités prédéfinies. Imaginez par exemple vous faites un code très complexe et que ça vous prend trop du temps si vous ne voulais pas que les gens souffrent comme vous vous créer une librairie vous mettez à la portée de tous votre code. Chers amis il faut avoir un cœur ouvert lorsque vous êtes programmeur dans ce petit traité nous allons traiter sur la création des librairies.

COMMENT CREER UNE LIBRAIRIE

Pour utiliser créer une librairie on peut faire cela dans un éditeur de texte (Sublime Text3, Notepad, Bloc-notes,…) vous pouvez utiliser codeBlocks ou même le logiciel ARDUINO. Dans ce présent travail nous allons utiliser dans la première partie avec arduino puis nous allons montrer comment nous pouvons utiliser aussi Sublime Text3.

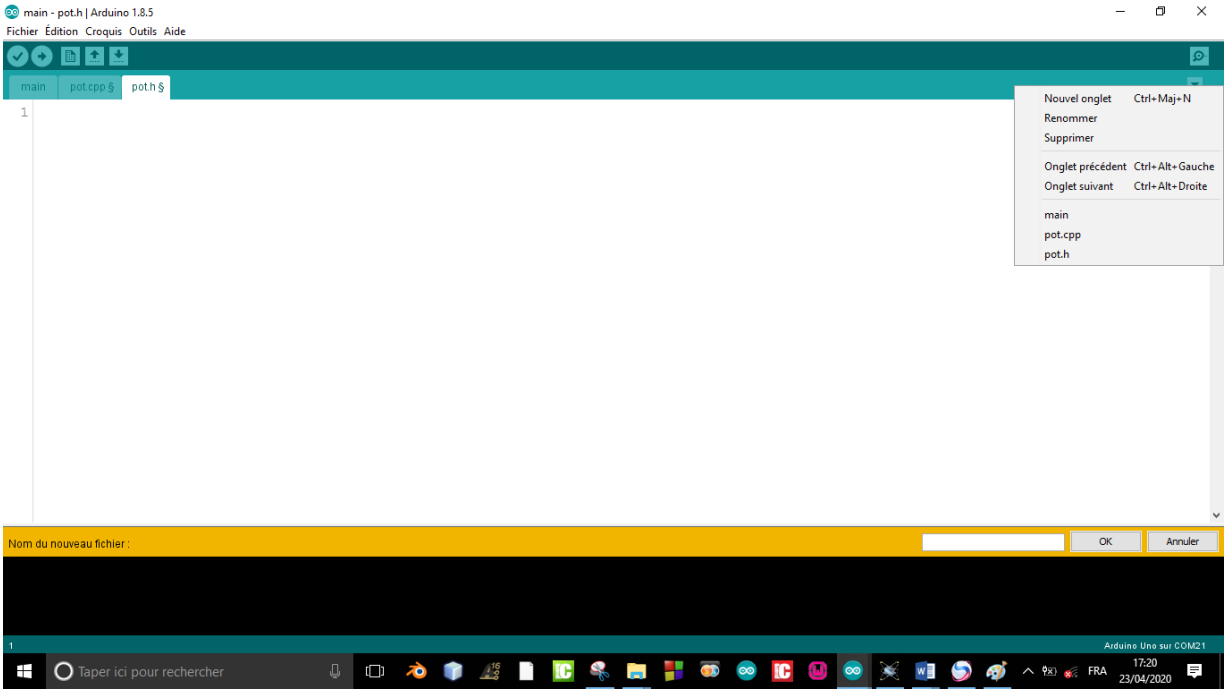
LE PROJET QUE NOUS ALLONS TRAITER

Comme projet nous allons créer une librairie qui permet de gérer le potentiomètre c'est-à-dire convertir la variation d'un potentiomètre en un angle.

Dans ce premier temps nous allons montrez comment nous allons faire ça avec ARDUINO :

Premièrement on crée un nouvel onglet pour y créer un fichier .h en cliquant sur le petit droit de l'éditeur ARDUINO

Puis vous cliquez sur nouvel onglet



Vous créez deux fichiers avec le nouvel onglet : un fichier .h et un fichier .cpp

Dans le fichier .h voici comment il faut faire le code :

On commence d'abord par écrire dans le fichier .h

```
1 #include "Arduino.h"
```

Ceci permet d'inclure dans ce fichier toutes les fonctions et les constantes de la programmation ARDUINO. Puis par la suite on crée la classe :

```
1 #include "Arduino.h"
2
3  class pot {
4
5   |
6  } ;
7
```

Ici dans la classe nous allons avoir des variables et des fonctions soient privés et publiques.

```
1  #include "Arduino.h"
2
3  class pot {
4
5    private :
6
7    public :
8
9  } ;
10
```

Les variables et les fonctions privés sont visibles uniquement dans la classe tandis que les variables et les fonctions privés sont visibles partout dans le programme. Pour notre cas nous allons créer un fonction tangle() qui mesure la tension venant d'un potentiomètre et la transforme en angle. Dans le fichier .h il est conseillé d'y mettre juste les entêtes.

Par la suite on le fichier .h suivant :

main - pot.h | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

main pot.cpp § pot.h §

```
#include "Arduino.h"

 class pot {

   private :

   public :

      float tangle (int pin) ;


 } ;

```

Ici nous venons de créer un fichier (pot.h).

Par la suite on a le fichier .cpp

main - pot.cpp | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

main | pot.cpp § | pot.h

```
#include "pot.h"

float pot :: tangle (int pin) {

    return map (analogRead(pin) ,0,1023,0,180) ;

}
```

Dans le fichier .cpp on implémente les fonctions que l'on a créées dans la classe.

Par-là la création de la librairie vient de prendre fin nous allons voir comment menant l'intégrer dans ARDUINO il suffit d'aller là où on a enregistré le projet puis on copie les deux fichier .cpp et le fichier .h

Nom	Modifié le	Type	Taille
main	23/04/2020 17:06	C++ Source	1 Ko
pot	23/04/2020 17:42	C++ source file	1 Ko
pot	23/04/2020 17:39	Header file	1 Ko

Ici nous allons copier le deux fichier pot.h et pot.cpp puis nous allons les mettre dans un dossier portant le même nom.

Nom	Modifié le	Type	Taille
pot	23/04/2020 17:48	Dossier de fichiers	
main	23/04/2020 17:06	C++ Source	1 Ko

Par-là notre librairie est déjà créer il nous suffit juste de copier le dossier pot dans le répertoire :

C:\Program Files (x86)\Arduino\libraries

Et voilà notre librairie est déjà prête et nous vous pouvons alors l'utiliser librement dans notre code voir :

sketch_apr23e | Arduino 1.8.5

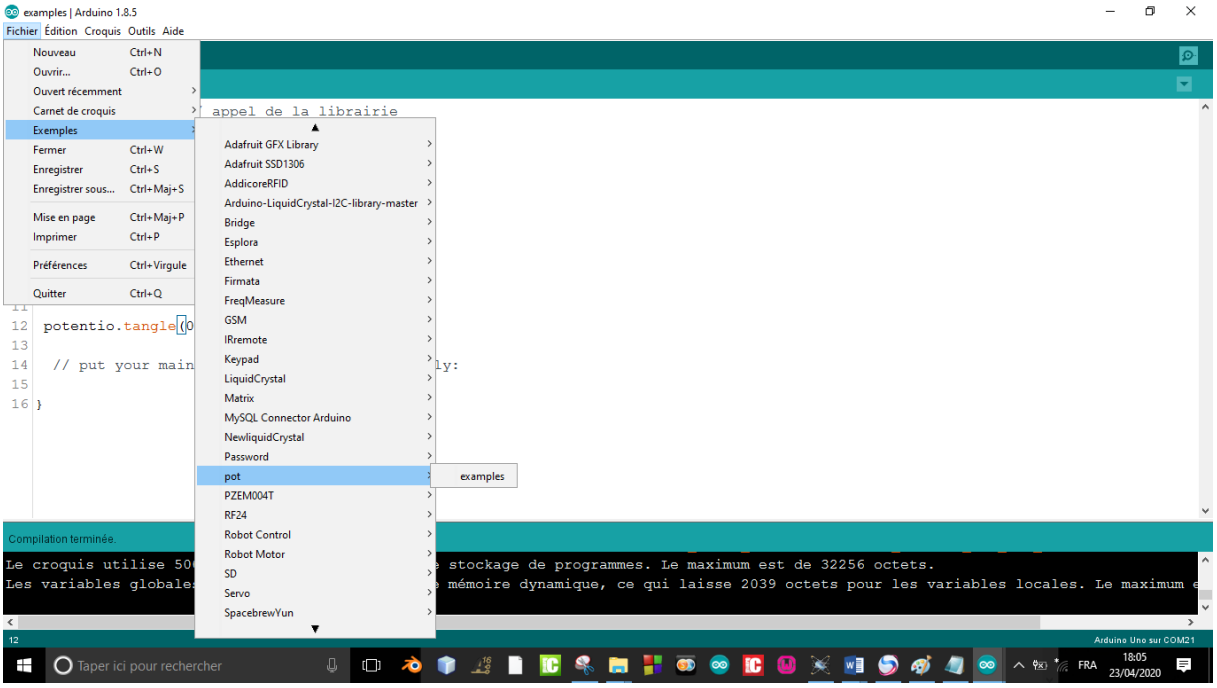
Fichier Édition Croquis Outils Aide

sketch_apr23e §

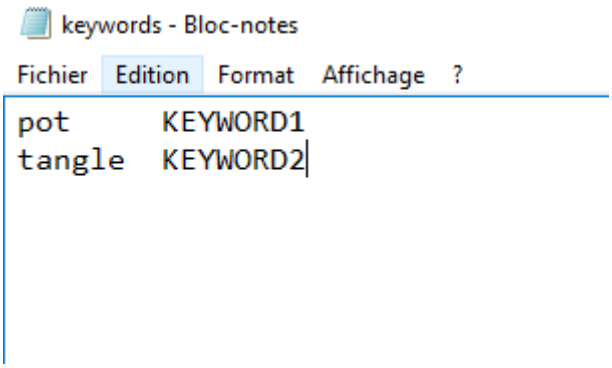
```
1  #include <pot.h> // appel de la librairie
2
3  pot potentio; //creation objet
4
5  void setup() {
6    // put your setup code here, to run once:
7
8  }
9
10 void loop() {
11
12  potentio.tangle();
13
14   // put your main code here, to run repeatedly:
15
16 }
```

Voilà notre librairie marche. Mais constaté une chose notre librairie n'est pas coloré en plus notre fonction n'est pas coloré pour faire une coloration il suffit juste de créer un fichier keywords.txt dans le dossier pot se trouvant dans : ***C:\Program Files (x86)\Arduino\libraries***

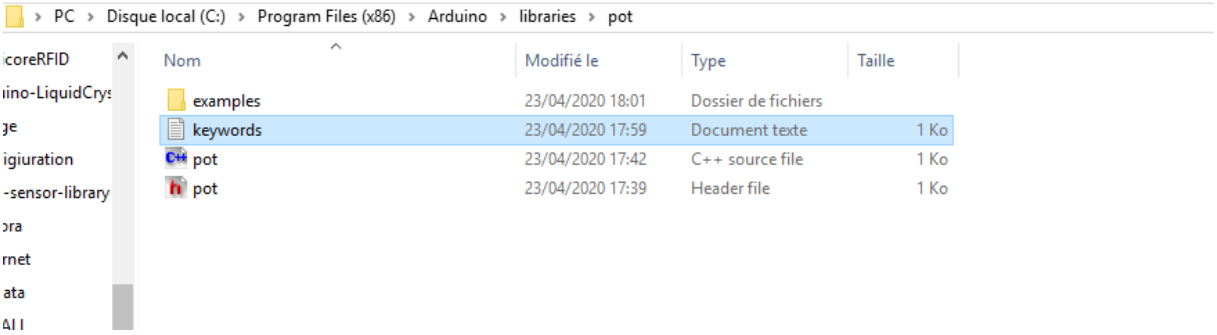
Et cela avec Bloc-notes ou autre éditeur de textes.



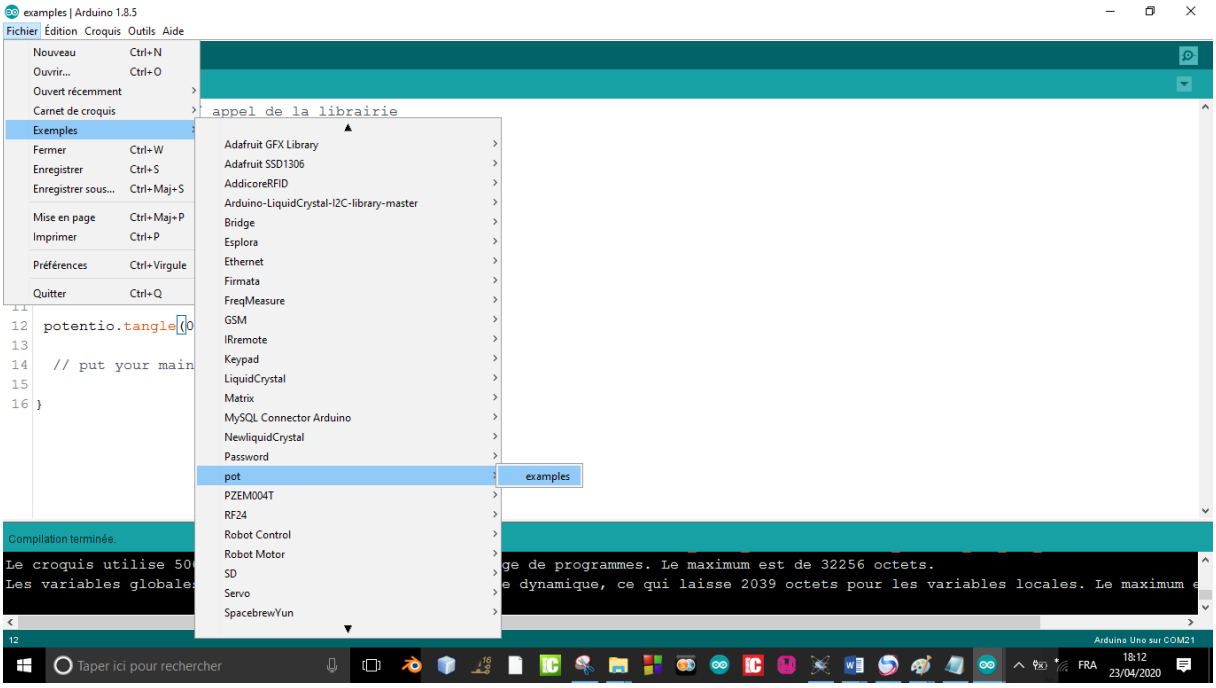
On voit très bien que la librairie **pot** est déjà intégrer dans arduino pour la coloration le fichier keywords :



Voici comment sera le dossier pot



Le fichier keywords va permet de colorer nos fonctions et notre librairie la fonction. Et le dossier examples contient un exemple de manipulation de notre librairie. Pour ouvrir l'exemple directement dans ARDUINO on fait :



Et quand on ouvre cet examples on a :

examples | Arduino 1.8.5

Fichier Édition Croquis Outils Aide

examples §

```
1  #include <pot.h> // appel de la librairie
2
3  pot potentio; //creation objet
4
5  void setup() {
6    // put your setup code here, to run once:
7
8  }
9
10 void loop() {
11
12  potentio.tangle(0);
13
14   // put your main code here, to run repeatedly:
15
16 }
```

On voit très bien que notre librairie a été très bien créer et que notre fonction est très bien colorée

Voilà créer alors vos librairies et contribué à l'évolution de la programmation ARDUINO

CHAPITRE23 STOCKER LES DONNEES SUR LA MEMOIRE EEPROM

INTRODUCTION

En arduino les variables perdent leurs valeurs lorsque le système est hors tension

Imagine alors que vous voulez faire un compteur d'énergie ou disons vous voulez faire un système permettant d'ouvrir une porte par mot de passe. Ce qui est sûr ce que en cas de mot de passe même si le système n'est pas alimenté il doit mémoriser le mot de passe pour ce faire on utilise une mémoire EEPROM gardant les informations même il y a absence de l'énergie électrique.

DEVELOPPEMENT

Il y a deux possibilités on peut stocker l'information ou s'en servir d'elle

ECRITURE DANS LA MEMOIRE EEPROM

On inclut la librairie EEPROM

Puis pour écrire dans la mémoire

On spécifie l'adresse puis la donnée

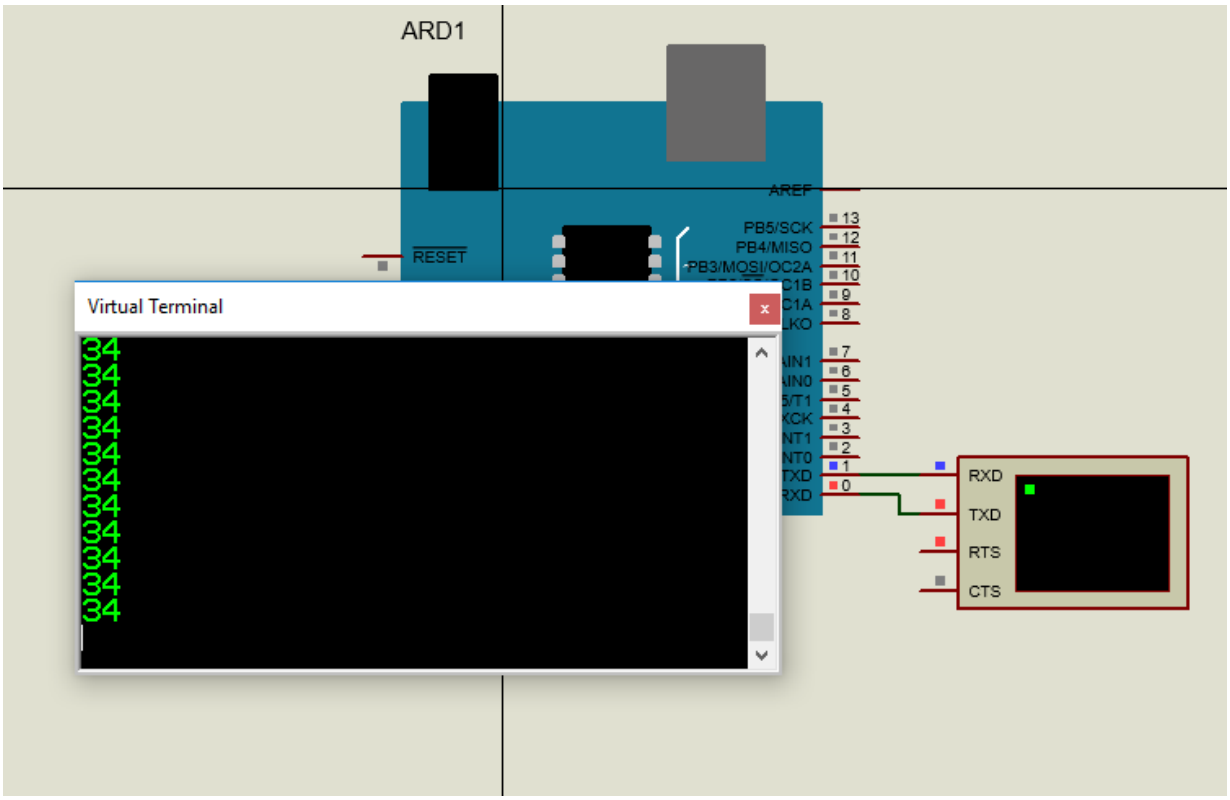
```
#include <EEPROM.h>

void setup() {

}

void loop() {
 EEPROM.write (0x09 , 34) ;

}
```

LECTURE DANS LA MEMOIRE EEPROM

```
#include <EEPROM.h>
 int infoEEPROM ;
void setup() {

}

void loop() {
 EEPROM.write (0x09 , 34) ;
 infoEEPROM = EEPROM.read(0x09);
}
```

Pour avoir les informations se trouvant dans la mémoire EEPROM on spécifie juste l'adresse sur laquelle se trouve l'information.

<u>RESULTAT DE SIMULATION</u>



Ici on voit très bien que l'information est 34 et ce seul qu'on enregistrer

CHAPITRE24. UN PEU DE CONTROL AVEC UN CONTROLEUR PID

INTRODUCTION

PID (Proportionnel intégrale Dérivé). Le contrôleur PID est l'un de contrôleur le plus utilisé dans le monde d'ingéniorat à cause de sa rapidité de mise en œuvre dans cette présente partie nous allons montrer comment nous pouvons concevoir un contrôleur en utilisant la carte ARDUINO.

DEVELOPEMENT

Avant de parler du contrôleur PID parlons d'abord de l'ingénierie automatique.

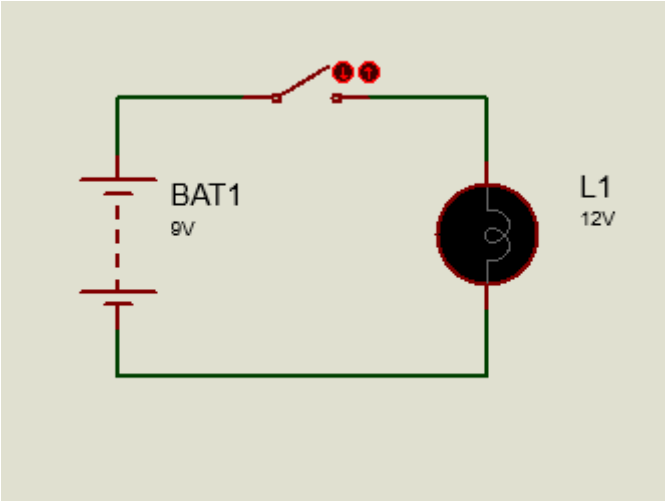
En automatique on distingue deux sortes des systèmes :

- Le système à boucle fermée
- Et le système à boucle ouverte

LE SYSTEME A BOUCLE OUVERT

Les systèmes à boucle ouvert sont des systèmes pour lesquels on fait la commande puis par la suite on ne s'intéresse pas à la sortie : il s'agit des systèmes à boucle ouvert parmi lesquels par exemple :

Dans la plupart des installations domestiques on fait juste l'allumage de lampes sans trop se préoccuper de l'intensité lumineuse de la lampe. Voir schémas ci-dessous :



La figure ci-dessus illustre un système à boucle ouvert : pour ce système lorsqu'on appuie sur l'interrupteur la lampe s'allume. Et on ne s'intéresse pas à savoir ce qui arrive au système après la commande.

LE SYSTEME A BOUCLE FERMEE

Un système à boucle fermée est un système pour lequel on ne s'intéresse pas uniquement de la commande mais plutôt de la sortie aussi. C'est l'exemple de système pour le drone ; système de position de l'antenne parabolique ; …

CHOIX DU SYSTEME

En ingéniorat on choisira qu'un système soit fermé selon que l'on a besoin d'avoir une précision sur la sortie

On choisira qu'un système soit ouvert selon qu'on n'a pas besoin d'une grande précision sur sa sortie

Dans cette partie nous allons nous intéresse aux systèmes à boucle fermés et plus particulièrement au contrôleur PID

PID

Il s'agit d'un contrôleur usant les paramètres P, I et D pour stabiliser un système donné par la suite nous allons présenter chaque paramètre et son rôle :

LE PARAMETRE P :

Il s'agit du paramètre proportionnel ce paramètre permet au contrôleur d'apporter une correction directement proportionnelle à l'erreur commise. Il est donné par la formule :

$$P = kp * e(t)$$

Le choix du paramètre kp :

Le paramètre kp est une constante sa valeur va beaucoup influencer le système à contrôler :

1. Si kp est trop grand cela va améliorer la rapidité du système ; le système sera trop rapide mais l'inertie fera qu'il aura de dépassement, il y aura des oscillations autour du point d'équilibre.
2. Si kp est petit cela fera que le système devient lent : on aura moins des oscillations moins de dépassement mais le système sera trop lent.
3. Si kp est à la valeur convenable le système sera rapide et moins oscillant

Mais remarquons une chose la formule de l'action proportionnelle est donnée par :

$$P = kp * e(t)$$

En partant de la formule ci-dessus on constate que plus l'erreur devient petit plus l'erreur devient petit plus le control devient petit et nous constatons très bien que si l'erreur est nulle le contrôle proportionnel n'est plus Cela fait à ce que le contrôleur P en lui seule ne peut pas suffire pour bien réguler le système. En effet l'erreur du système tend vers 0 le control disparait et quand le control disparait le système se déstabilise encore.

Bref : le paramètre P agit plus sur la rapidité du système et sa précision mais détériore la stabilité. Avec ce paramètre l'erreur final est toujours différente de 0.

LE PARAMETRE I

Le paramètre I agit sur la précision du système du système il introduit une lenteur dans le système ce qui permet il n'est jamais malgré la valeur de l'erreur ce qui permet de bien contrôlé la précision finale du système. Avec ce paramètre l'erreur finale du système tend vers 0. Il est donnée par l'équation :

$I = ki * \int_0^t e(t) * dt$

Le choix du paramètre ki :

Plus la valeur de ki est importante plus le système atteint vite sa valeur finale toute fois si la valeur de ki est trop grand cela peut affecter sur la stabilité du système

Le paramètre I va beaucoup plus agir sur la rapidité du système.

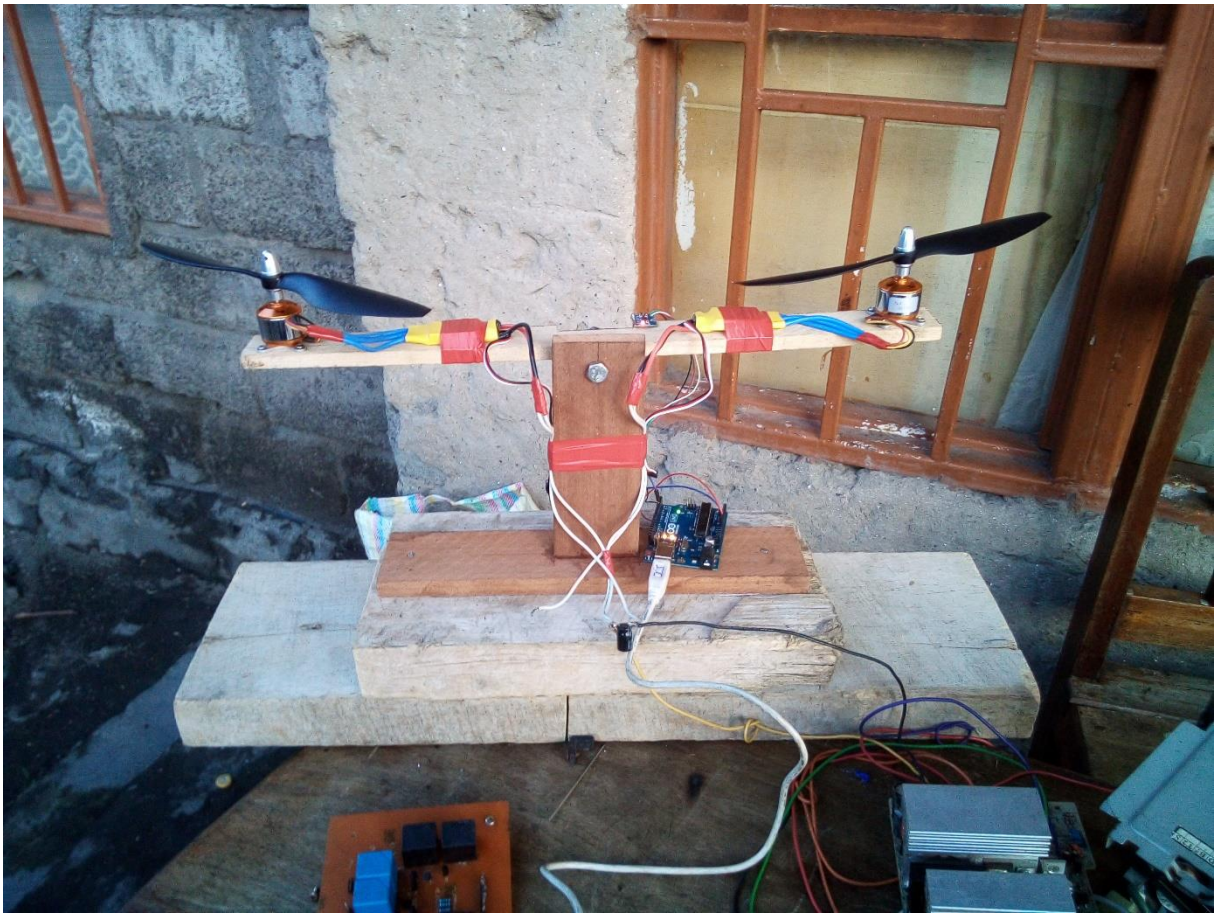
LE PARAMETRE D

Le paramètre D agit sur la stabilité du système plus la valeur de D est importante le dépassement se voit diminuer. Le paramètre D est choisi pour limiter le dépasseTapez une équation ici.ment. Il est donné par l'équation :

$$D = kd * \frac{e_p(t) - e_o(t)}{t_p - t_o}$$

IMPLEMENTATION DU CONTROLEUR PID SUR ARDUINO

Ici nous allons faire un projet permettant de contrôler deux moteur BRUSHLESS le système que nous allons contrôler est donné sur la photo suivant :



Comme on voit sur la photo nous devons imposer aux moteurs BRUSHLESS une certaine vitesse pour que l'axe de ce système puis faire un angle que nous lui avons imposé.

Nous avons utilisé :

1. Deux moteurs BRUSHLESS et ses ESC
2. Un MPU 6050
3. Un arduino UNO

Voici le code :

controlleurPID_CALIBRAGE

```
#include <MPU6050_tockn.h>
#include <Wire.h>
#include <Servo.h>
#include <Math.h>

MPU6050 mpu6050(Wire);

Servo moteurGauche ;
Servo moteurDroit ;

  float consigne = 0.00 ;
  float angleMesurer ;

   float erreur ;

   const float kp =3.00;
   const float kd =2.00;
   const float ki = 0.00588 ;

   float P ;
```

```
   float I ;
   float D ;

   float PID ;

  float temps ;
  float tempsEchantillonnage ;
  float tempsFuture ;

   float erreurPrecedente ;


    float vitesseMoteurGauche ;
   float vitesseMoteurDroit ;


       float vitesseInitial = 1300 ;

       float vitesseRegulerMoteurDroit ;
       float vitesseRegulerMoteurGauche ;
```

```
43
44          #define pinControllerMoteurGauche 7
45          #define pinControllerMoteurDroit 4
46
47           #define activationTempo 2000
48
49
50 void setup() {
51
52    Serial.begin(9600);
53    Wire.begin();
54    mpu6050.begin();
55    mpu6050.calcGyroOffsets(true);
56
57   moteurGauche.attach (pinControllerMoteurGauche);
58   moteurDroit.attach (pinControllerMoteurDroit) ;
59
60    temps = millis () ;
61
62   moteurGauche.writeMicroseconds (1000);
63   moteurDroit.writeMicroseconds (1000);
```

```
64
65    delay (activationTempo);
66 }
67
68 void loop() {
69
70    mpu6050.update();
71
72          tempsFuture = temps ;
73          temps = millis() ;
74          tempsEchantillonnage = (temps-tempsFuture)/1000 ;
75
76          angleMesurer = mpu6050.getAngleZ() ;
77
78         //Serial.print("angleZ : ") ;
79           //Serial.println (angleMesurer);
80
81             erreur = consigne - angleMesurer;
82
83         P = erreur * kp;
84
```

```
85        if (-3 < erreur < 3) {
86
87         I = I+(erreur*tempsEchantillonnage)*ki;
88
89        }
90
91         D = kd * ((erreur - erreurPrecedente) / tempsEchantillonnage);
92
93
94       PID = P + I + D ;
95
96        if (PID > 1000) {
97           PID = 1000 ;
98        }
99
100        if (PID < -1000) {
101           PID = -1000 ;
102        }
103
104
105      vitesseRegulerMoteurDroit = vitesseInitial + PID ;
```

```
105      vitesseRegulerMoteurDroit = vitesseInitial + PID ;
106      vitesseRegulerMoteurGauche = vitesseInitial - PID ;
107
108       if (vitesseRegulerMoteurDroit < 1000) {
109          vitesseRegulerMoteurDroit = 1000 ;
110       }
111
112       if (vitesseRegulerMoteurDroit > 2000) {
113          vitesseRegulerMoteurDroit = 2000 ;
114       }
115
116       if (vitesseRegulerMoteurGauche < 1000) {
117          vitesseRegulerMoteurGauche = 1000 ;
118       }
119
120       if (vitesseRegulerMoteurGauche > 2000) {
121          vitesseRegulerMoteurGauche = 2000 ;
122       }
123
124
125         moteurGauche.writeMicroseconds (vitesseRegulerMoteurDroit);
```

```
126         moteurDroit.writeMicroseconds (vitesseRegulerMoteurGauche);
127
128         erreurPrecedente  = erreur ;
129
130         //Serial.print ("TEMPS ECHANTILLONNAGE  :");
131
132         Serial.println (tempsEchantillonnage);
133
134 //        Serial.print ("PROPORTIONNEL P  :");
135 //        Serial.println (P);
136 //
137 //             Serial.print ("INTEGTRAL I  :");
138 //            Serial.println (I);
139 //
140 //           Serial.print ("DERIVEE D  :");
141 //            Serial.println (D);
142 //
143 //
144 //           Serial.print ("PID :");
145 //            Serial.println (PID);
146 //
147 //
148 //
149 //
150 //
151 //             Serial.print ("erreur :");
152 //             Serial.println (erreur);
153 //
154 //
155 //
156 //             Serial.print ("vitesse moteur droit");
157 //             Serial.println (vitesseRegulerMoteurDroit);
158 //
159 //             Serial.print ("vitesse moteur gauche");
160 //             Serial.println (vitesseRegulerMoteurGauche);
161
162 }
```

Nous imposons à notre système un angle que l'on note **consigne** et dans notre cas notre consigne est nulle les mesurons les angles puis nous agissons pour trouver

a. P on a le code :

```
P = erreur * kp;
```

b. I on a le code :

```
I = I+(erreur*tempsEchantillonnage)*ki;
```

N'oubliez pas que l'intégrale c'est une somme.

c. D on a le code :

```
D = kd * ((erreur - erreurPrecedente) / tempsEchantillonnage);
```

d. Et enfin pout le PID on a le code :

```
PID = P + I + D ;
```

CALIBRATION DU CONTROLEUR PID

INTRODUCTION

Pour le calibrage de contrôleur PID plusieurs contrôleurs ont vit le jour dans le monde de l'automatique ; quant à nous dans cette partie nous allons traiter sur le calibrage de ce contrôleur non en utilisant de toutes les méthodes existantes mais en utilisant la méthode de ZIEGLER-NICHOLS et la méthode de d'approximation.

CHOIX DE PARAMETRES PID PAR LA METHODE DE ZIEGLER-NICHOLS

Pour trouver les paramètres selon la méthode de ZIEGLER-NICHOLS :

1. On met les constantes kd et ki à zéro puis on augmente la constate kp jusqu'à ce qu'à amener le système à osciller de manière permanente on se trouve à cette valeur de kp à la limite de stabilité la constance kp auquel le système est à la limite de stabilité oscille est appelé kcr et la période d'oscillation Tcr. Pour trouver les paramètres en ayant Tcr et kcr on peut appliquer les formules du tableau suivant :

TYPE	kp	ki	kd
P	$0.5k_{cr}$		
PI	$0.4k_{cr}$	$0.8T_{cr}$	
PID	$0.6k_{cr}$	$0.5T_{cr}$	$0.125T_{cr}$

CHOIX DE PARAMETRE PID PAR LA METHODE APPROXIMATIVE

Personnellement j'aime la méthode approximative car avec cette méthode on a moins de dépassement mais certes cette méthode demande d'avoir une connaissance de rôle de chaque paramètre.

LE CHOIX DE LA CONSTATE kp

La constate kp agit sur la rapidité et la précision du système lorsque vous faites le choix de cette constate vous devez vous rassurez que votre système est rapide et précis. On commence par ce choisir un nombre au hasard de la constante kp puis on voit la réaction si cette valeur de kp est grand vous allez constater que votre système sera très rapide mais comme inconvénient vous pouvez constater le dépassement s'il a un grand dépassement donc il faut penser à diminuer la valeur de kp.

Bref : vous choisirez la constaté kp selon ce que vous désirez du système si vous voulez un système trop rapide choisissez donc une valeur de kp qui est grand mais avec risque de perdre en stabilité. Si vous choisissez une valeur petite vous aurez toujours un problème car votre système sera trop lent mais le de dépassement sera moindre voire inexistant. C'est pourquoi il est préférable de choisir une valeur intermédiaire pour que le système puisse agir pas trop rapidement ni trop lent mais souple. Mais dans toute la condition si la valeur de kp est trop élevée cela puis rendre le système complètement instable.

LE CHOIX DE LA CONSTATE kd

En visant la rapidité la constante kp introduit de fois et dans la plupart de cas un dépassement : c'est le rôle de la constante kd éliminer le dépassement. Plus vous augmenter la valeur de la constate kd plus le système a moins de dépassement. Mais le terme dérivé fait que le système devient trop sensible aux bruits.

Il faut toujours faire un choix en commence par 1 puis on voit le résultat si le résultat est bon on laisse la valeur telle quelle si le résultat n'est pas bon on augmente la valeur progressivement jusqu'à avoir un bon résultat. Si ce paramètre est bien choisi la stabilité du système se voit très améliorer dans le cas où on a mal choisit cette valeur le système peut se comporte trop bizarrement. Il faut bien noter que cette constante agit plus sur la stabilité du système.

LE CHOIX DE LA CONSTATE ki

La constante ki c'est la constante de l'intégrale elle agit sur la valeur finale du système sans cette constante on a du mal à voir l'erreur finale différente de zéro. Pour le choix de cette constante il faut faire trop attention car si vous choisissez aussi une très grande valeur vous pouvez voir votre système qui oscille trop donc , lors du choix de constate ki il est préférable de commencer avec de petite valeur ; voir la réaction et puis par après de grandes valeurs. Une constate ki bien choisi améliorera considérablement la précision du système. La valeur finale est théoriquement nulle lorsque cette constante a été très bien choisie.

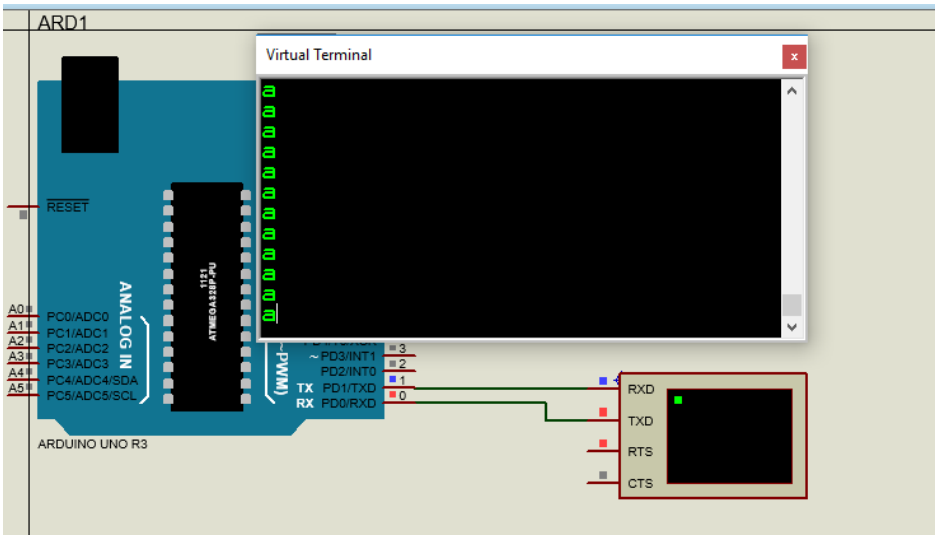
CONCLUSION

Calibrer un contrôleur PID veut dire trouver les paramètres kp, ki, kd chose qui n'est pas facile. Il existe plusieurs méthodes mais dans ce présent travail nous avons présenté la méthode de ZIEGLER-NICHOLS et la méthode d'essais approximatifs. Avec la méthode de ZIEGLER-NICHOLS on applique une procédure élaboré par les scientifiques ZIEGLER et NICHOLS voir ci-dessus pour trouver les paramètres du contrôleur tandis que par la méthode approximative on se choisi de paramètre on voit les résultats puis selon les résultats obtenus on peut savoir sur quel paramètre agir pour améliorer. Il n'est pas aussi facile de trouver les paramètres PID parfaits mais on peut trouver des paramètres acceptables pour système ou un autre.

CHAPITRE 1 LES CHAINES DE CARACTERES

La fonction charAt

```
1 String mot = "anticonstitutionnellement";
2 void setup() {
3 Serial.begin(9600);
4 }
5
6 void loop() {
7   Serial.println(mot.charAt(0));
8 }
```



Cette fonction permet d'obtenir un caractère dans une chaine caractère ici on obtient le premier caractère

En mettant 0 en paramètre de la fonction on obtient le premier caractère

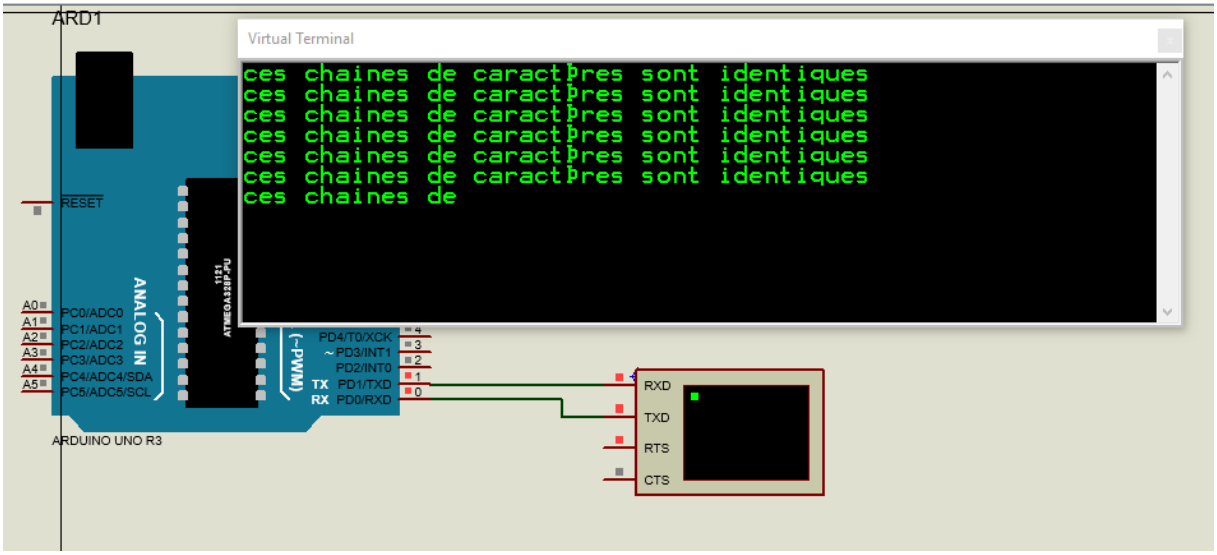
En mettant 1 en paramètre de la fonction on obtient le deuxième caractère

En mettant 1 en paramètre de la fonction on obtient le troisième caractère ainsi de suite.

La fonction equals

string_string §

```
1  String mot = "anticonstitutionnellement";
2  String mot1 = "anticonstitutionnellement";
3
4  void setup() {
5  Serial.begin(9600);
6  }
7
8  void loop() {
9   if(mot.equals(mot1) == true) {
10   Serial.println ("ces chaines de caractères sont identiques");
11  }
12  if(mot.equals(mot1) == false) {
13   Serial.println ("ces chaines de caractères ne sont pas identiques");
14  }
15 }
16
```

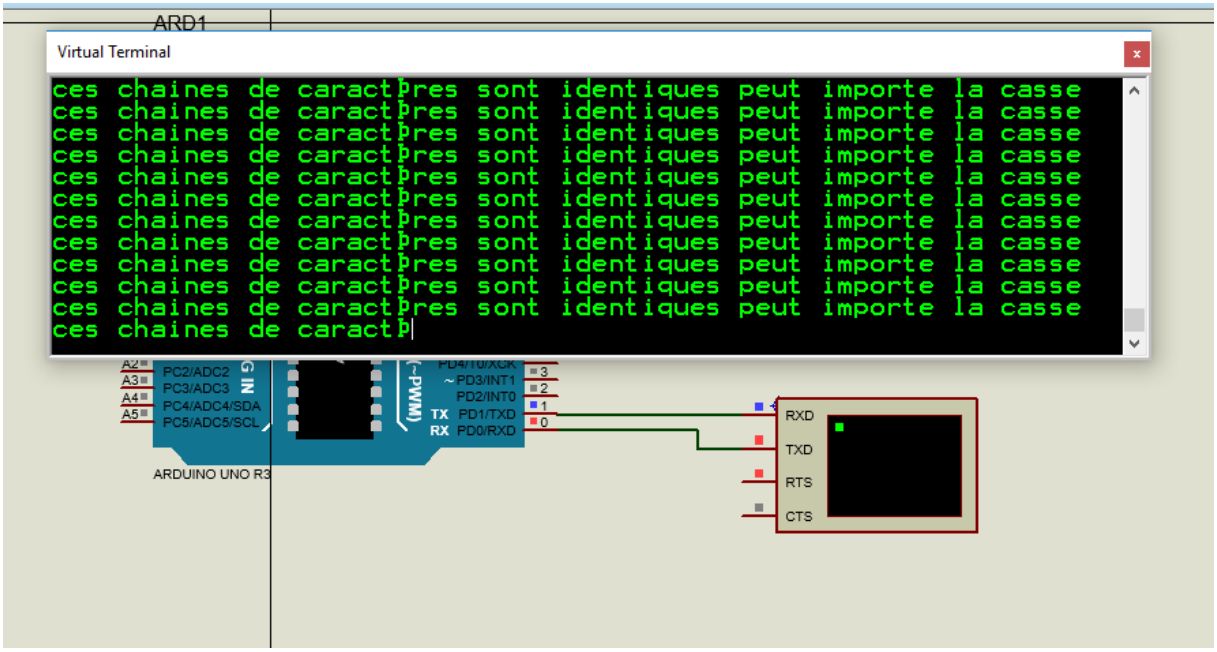


Cette fonction consiste en la comparaison si ces deux caractères sont identiques il retourne true si les deux caractères ne sont pas identiques il retourne false cette fonction est sensible à la casse

La fonction equalsIgnoreCase ()

string_string §

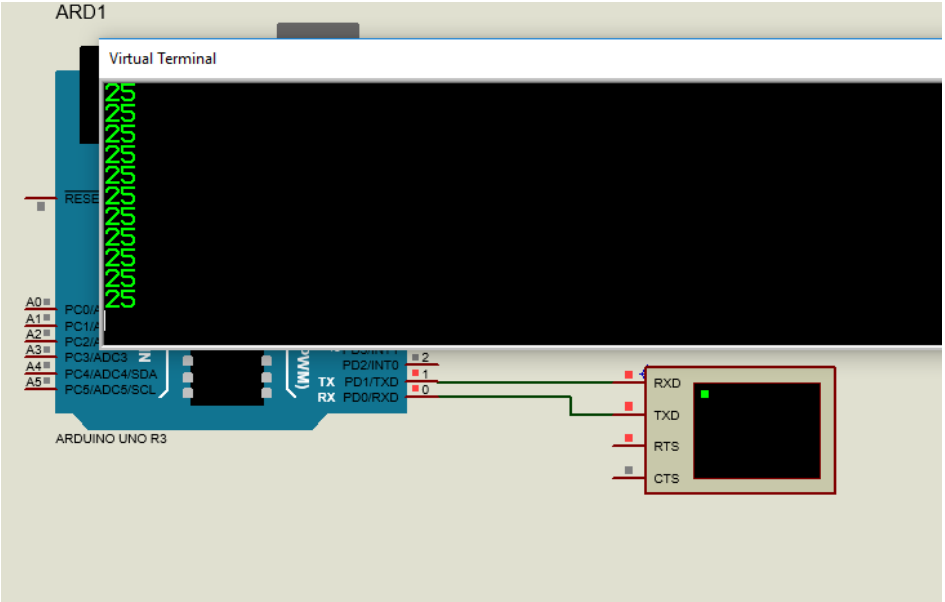
```
String mot = "anticonstitutionnellement";
String mot1 = "anticonstitutioNNEllement";

void setup() {
Serial.begin(9600);
}

void loop() {
 if(mot.equalsIgnoreCase(mot1) == true) {
  Serial.println ("ces chaines de caractères sont identiques peut importe la casse");
 }
 if(mot.equalsIgnoreCase(mot1) == false) {
  Serial.println ("ces chaines de caractères ne sont pas identiques peut importe la casse");
 }
}
```



Cette fonction fait aussi une comparaison entre deux chaines en ne tenant pas compte de la classe.

La fonction lenght

```
string_string §
1  String mot = "anticonstitutionnellement";
2  String mot1 = "anticonstitutioNNEllement";
3
4   int longueur_mot ;
5
6  void setup() {
7  Serial.begin(9600);
8  }
9
10 void loop() {
11
12   longueur_mot = mot.length();
13   Serial.println (longueur_mot);
14
15
16 }
17
```

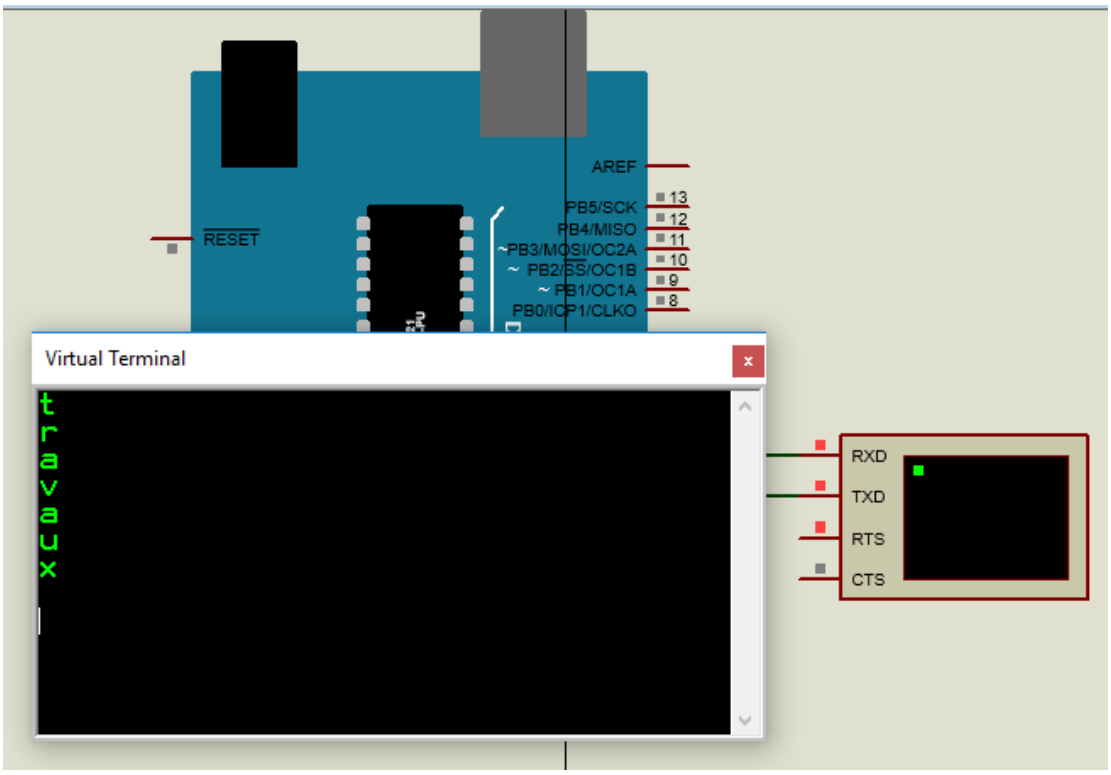


Cette fonction permet d'obtenir la longueur de la chaine d'une chaine de caractère par exemple ici on peut que cette longueur est de 25 dont 25 caractères.

getBytes()

Cette fonction permet de copier les caractères de la chaîne fournit dans un tampon fourni ici dans notre cas elle copie les caractères de la chaîne dans un tableau. On doit choisir la taille du tableau en comptant le nombre de caractères de la chaîne plus un pour être capable de stocker tous les caractères.

```
string_fin §
char   receptionstring[8];
String chaine = "travaux";
 int i = 0;
void setup() {
  Serial.begin(9600);
}

void loop() {
  chaine.getBytes(receptionstring , 8);
   while(i<8) {
    Serial.println (receptionstring[i]);
    i++;
   }
}
```

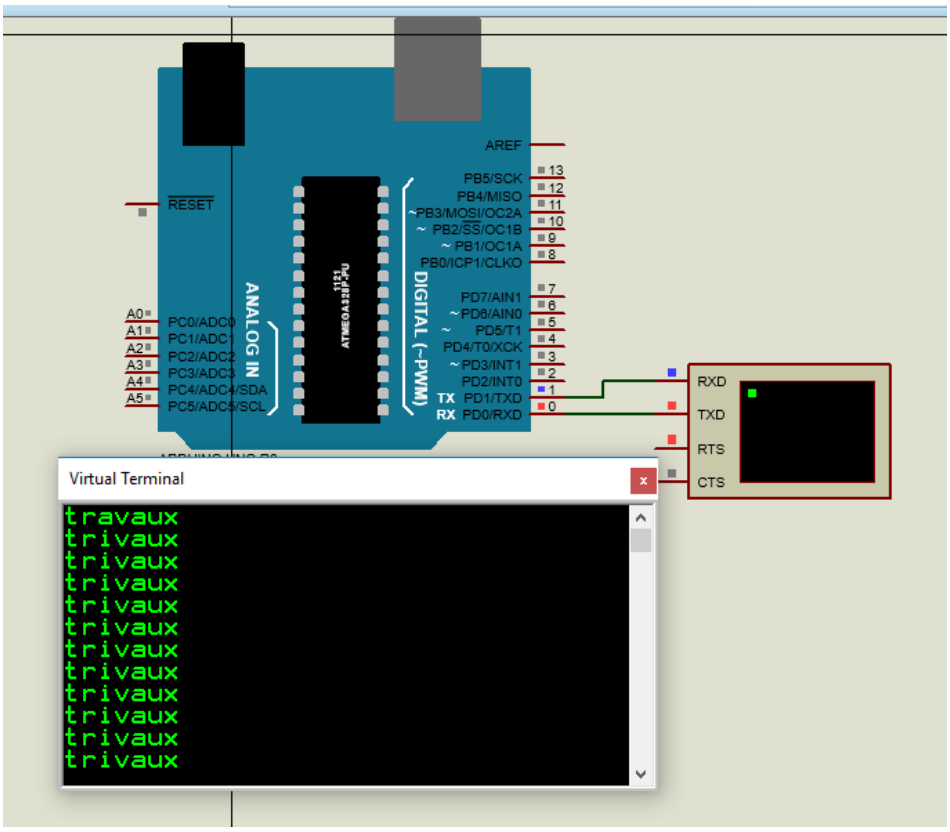


replace ()

Cette fonction permet de remplacer les occurrences d'une chaine de caractères en une autre chaine de caractères cette fonction a deux paramètres la chaine qu'on veut remplacer et son remplaçant.

string_fin §

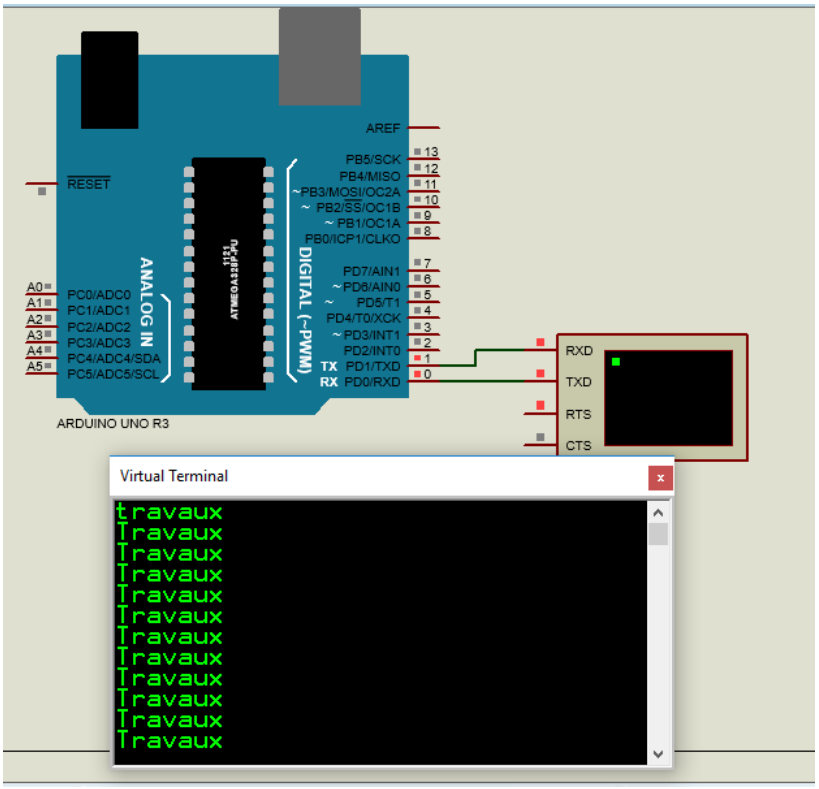
```
char   receptionstring[8];
String chaine = "travaux";
 int i = 0;
void setup() {
  Serial.begin(9600);
  Serial.println (chaine);
}

void loop() {
  chaine.replace("tra" , "tri");
  Serial.println(chaine);

}
```



setCharAt()

Cette fonction permet de changer un caractère dans une chaine de caractères elle est constituée par deux paramètres l'un déterminant la position du caractère à remplacer et l'autre donnant le caractère remplaçant.

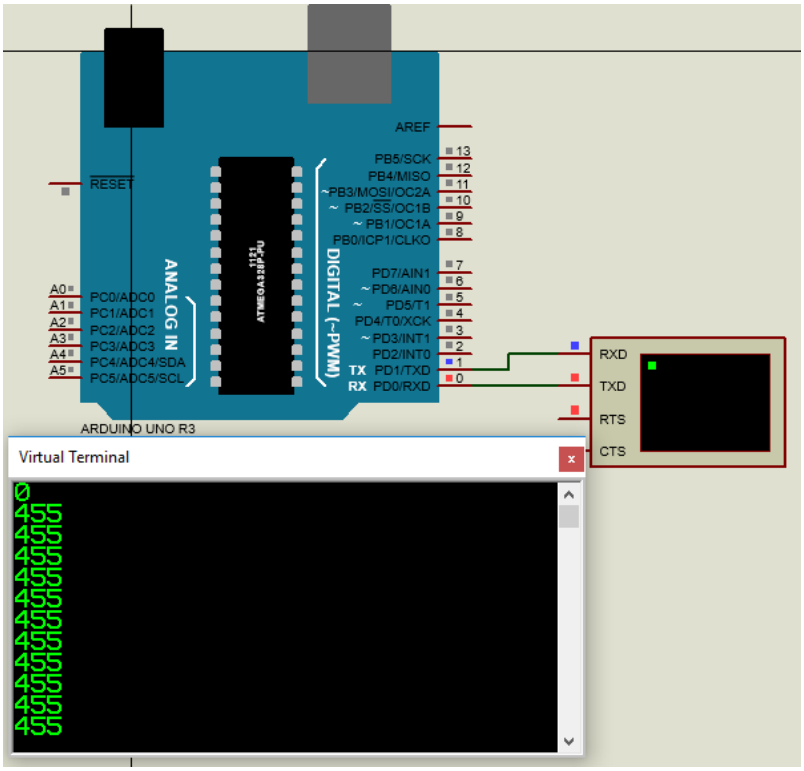
```
string_fin §
char   receptionstring[8];
String chaine = "travaux";
 int i = 0;
void setup() {
  Serial.begin(9600);
  Serial.println (chaine);
}

void loop() {

  chaine.setCharAt(0 , 'T');
  Serial.println(chaine);

}
```



toInt()

Cette fonction permet de faire la conversion d'une chaîne de caractères en une variable entière de type int. Cette conversion n'est possible que si la chaine de caractères ne contient que de chiffres dans le cas contraire rien ne se passe.

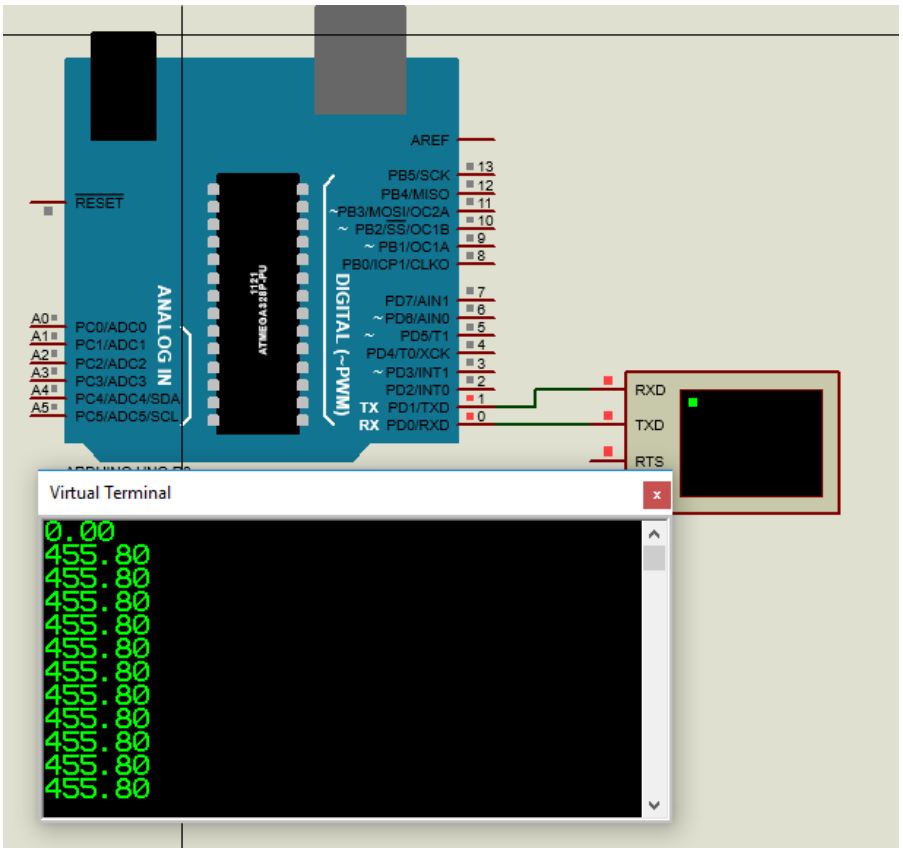
```
string_fin
char   receptionstring[8];
String chainenombre = "455.89" ;
int nombre ;
 int i = 0;
void setup() {
  Serial.begin(9600);
  Serial.println (nombre);
}

void loop() {
  nombre = chainenombre.toInt();
  Serial.println(nombre);
}
```



toDouble()

Cette fonction permet de faire la conversion d'une chaîne de caractères en une variable entière de type double. Cette conversion n'est possible que si la chaine de caractères ne contient que de chiffres dans le cas contraire rien ne se passe.

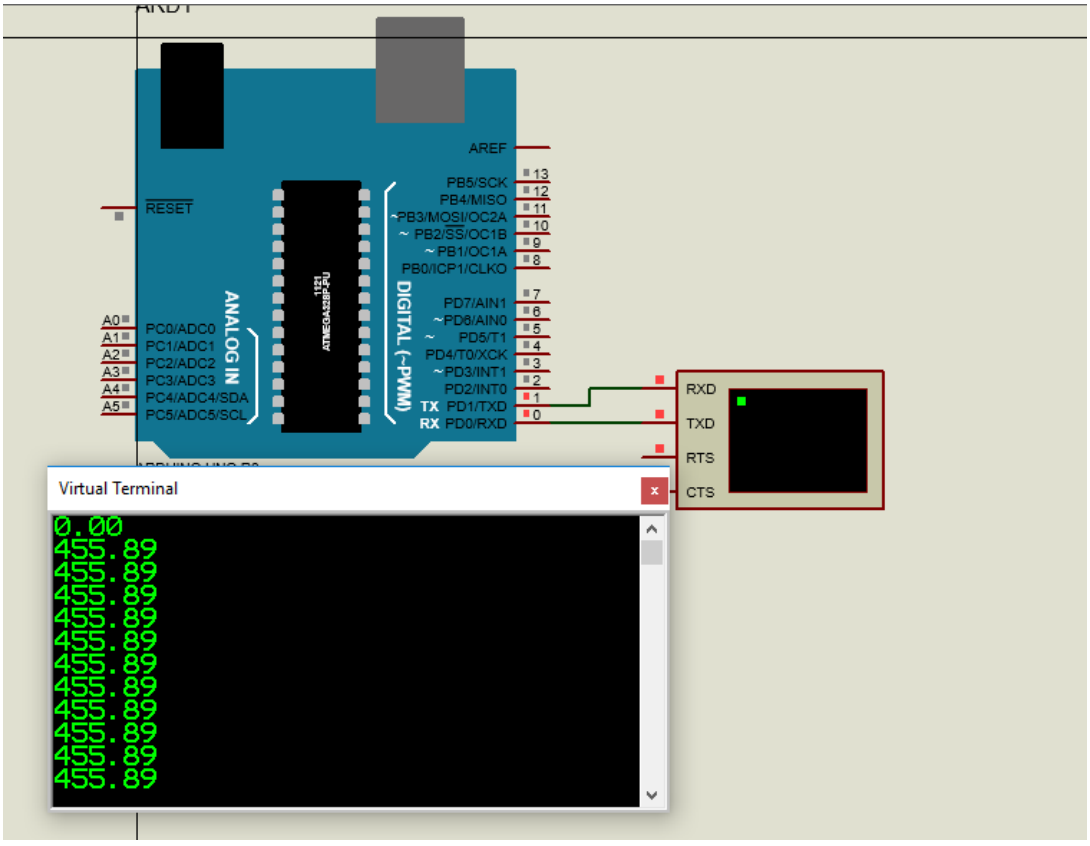
```
string_fin
char   receptionstring[8];
String chainenombre = "455.8" ;
double nombre ;
 int i = 0;
void setup() {
  Serial.begin(9600);
  Serial.println (nombre);
}

void loop() {
  nombre = chainenombre.toDouble();
  Serial.println(nombre);
}
```



toFloat()

Cette fonction permet de faire la conversion d'une chaîne de caractères en une variable entière de type float. Cette conversion n'est possible que si la chaine de caractères ne contient que de chiffres dans le cas contraire rien ne se passe.

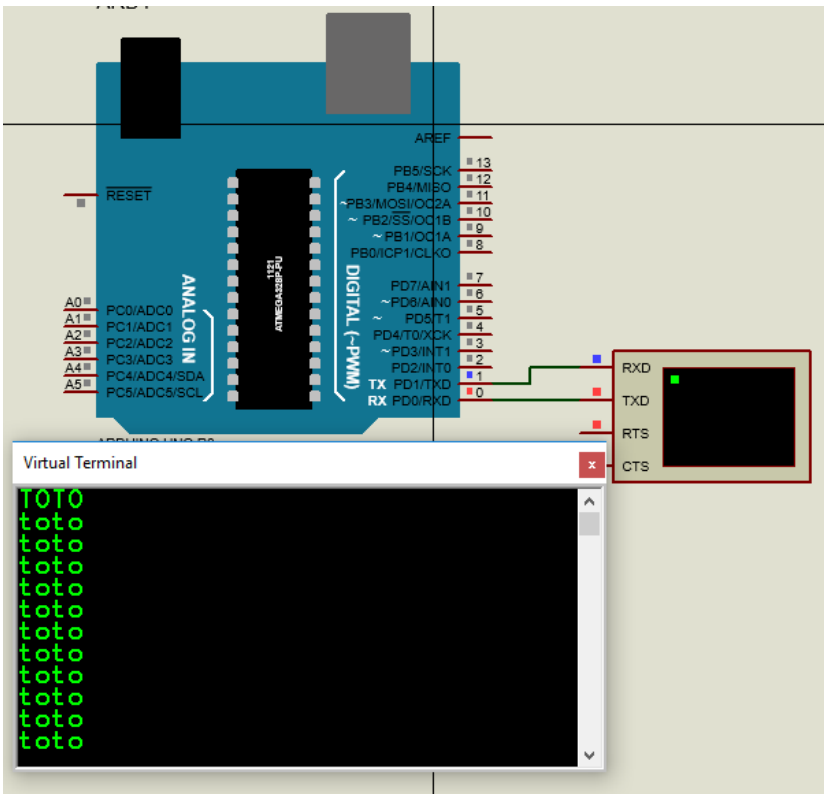
```
string_fin §
char   receptionstring[8];
String chainenombre = "455.89" ;
float nombre ;
 int i = 0;
void setup() {
  Serial.begin(9600);
  Serial.println (nombre);
}

void loop() {
  nombre = chainenombre.toFloat();
  Serial.println(nombre);
}
```



toLowerCase()

Cette fonction permet de mettre une chaine de caractères en minuscule

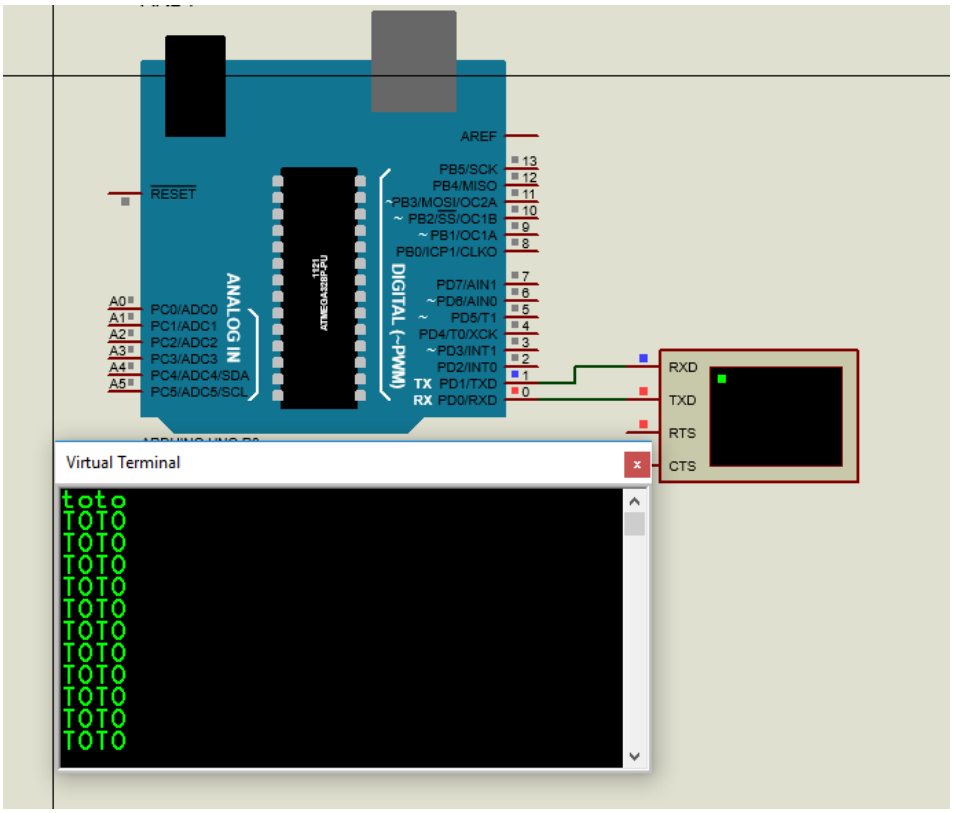
```
string_fin §
char   receptionstring[8];
String chaine = "TOTO" ;
float nombre ;
 int i = 0;
void setup() {
  Serial.begin(9600);
  Serial.println (chaine);
}

void loop() {
  chaine.toLowerCase();
  Serial.println(chaine);
}
```



toUpperCase()

Cette fonction permet de mettre une chaine de caractères en majuscule.

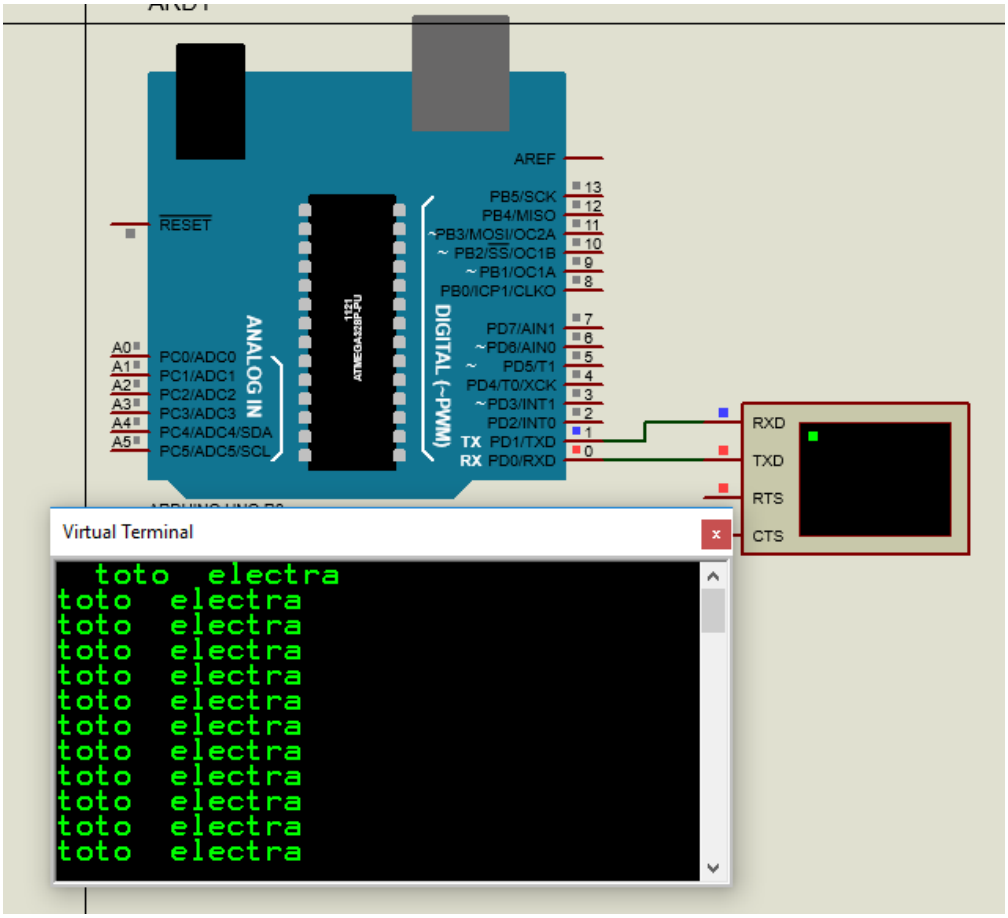
```
string_fin §
char   receptionstring[8];
String chaine = "toto" ;
float nombre ;
 int i = 0;
void setup() {
  Serial.begin(9600);
  Serial.println (chaine);
}

void loop() {
  chaine.toUpperCase();
  Serial.println(chaine);
}
```



trim()

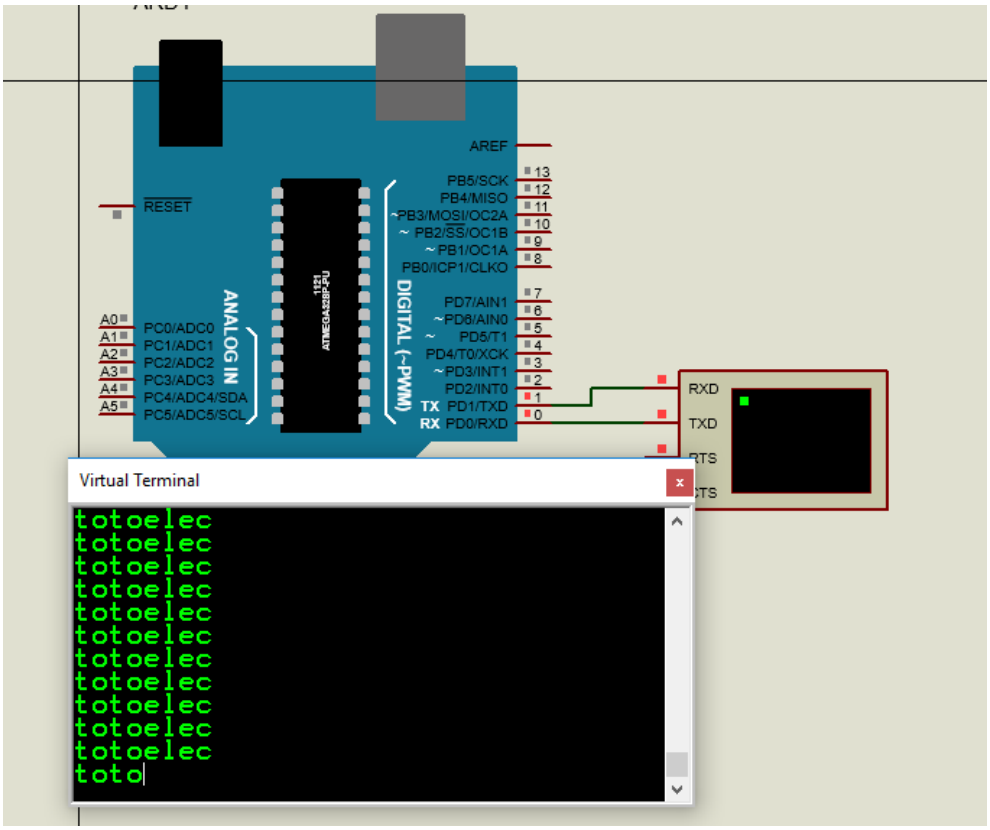
Pour une chaine de caractère cette fonction permet d'enlever les espaces au début et la fin de la chaîne de caractères.

```
string_fin §
char   receptionstring[8];
String chaine = "  toto  electra   " ;
float nombre ;
 int i = 0;
void setup() {
  Serial.begin(9600);
  Serial.println (chaine);
}

void loop() {
  chaine.trim();
  Serial.println(chaine);
}
```



remove()

```
string_fin §
1  char   receptionstring[8];
2  String chaine = "totoelectra" ;
3  float nombre ;
4   int i = 0;
5  void setup() {
6    Serial.begin(9600);
7    Serial.println (chaine);
8  }
9
10 void loop() {
11   chaine.remove(8);
12   Serial.println(chaine);
13 }
```

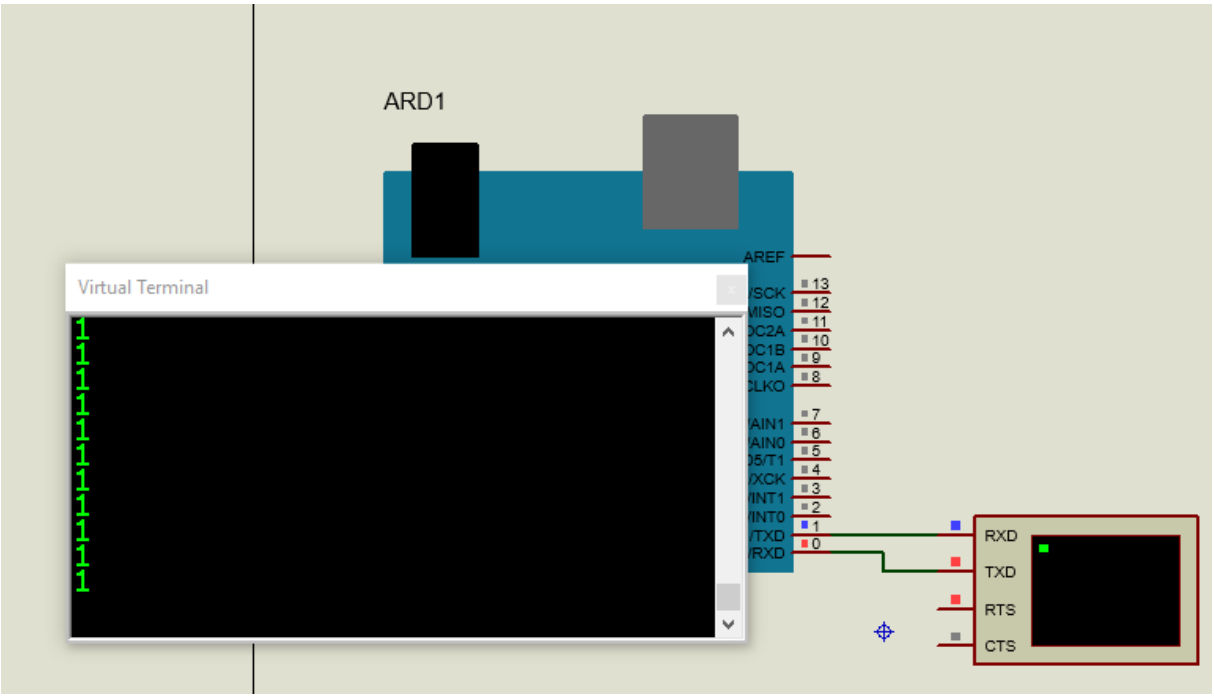


indexOf

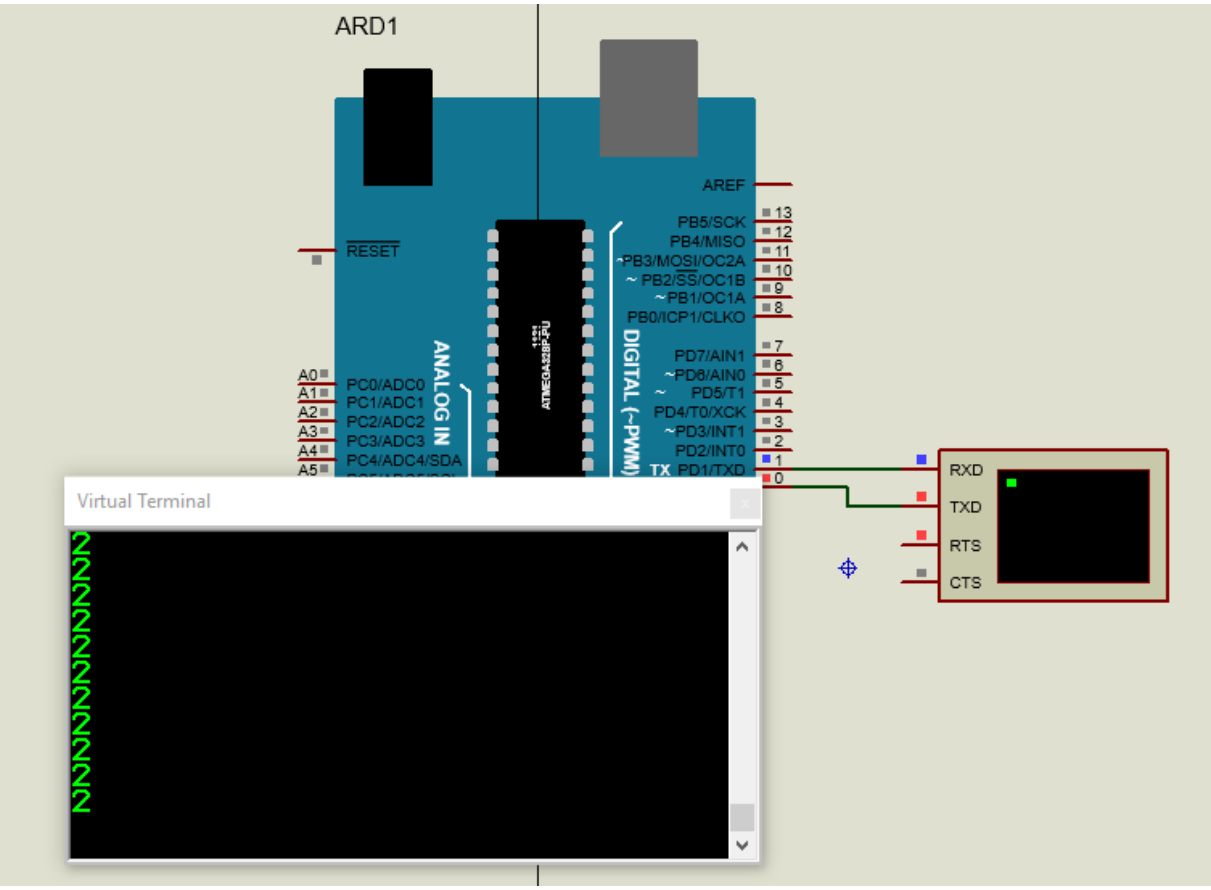
Cette fonction permet de renvoyer la position d'un caractère ou une chaine de caractère dans une autre qui est principale ça prend le premier caractère de la chaine. L'index commence par 0.

```
string_str §
 String vartext = "anticonstitutionnellement";
  int n ;
void setup() {

  Serial.begin(9600);

}

void loop() {
  n = vartext.indexOf('n');
 Serial.println(n);

}
```

Ici la valeur de n correspond à 1 car le caractère prend la deuxième position de la variable String vartext on peut voir cela dans la simulation.



```
string_str §
 String vartext = "anticonstitutionnellement";
  int n ;
void setup() {

  Serial.begin(9600);

}

void loop() {
 n = vartext.indexOf('t');
 Serial.println(n);

}
```



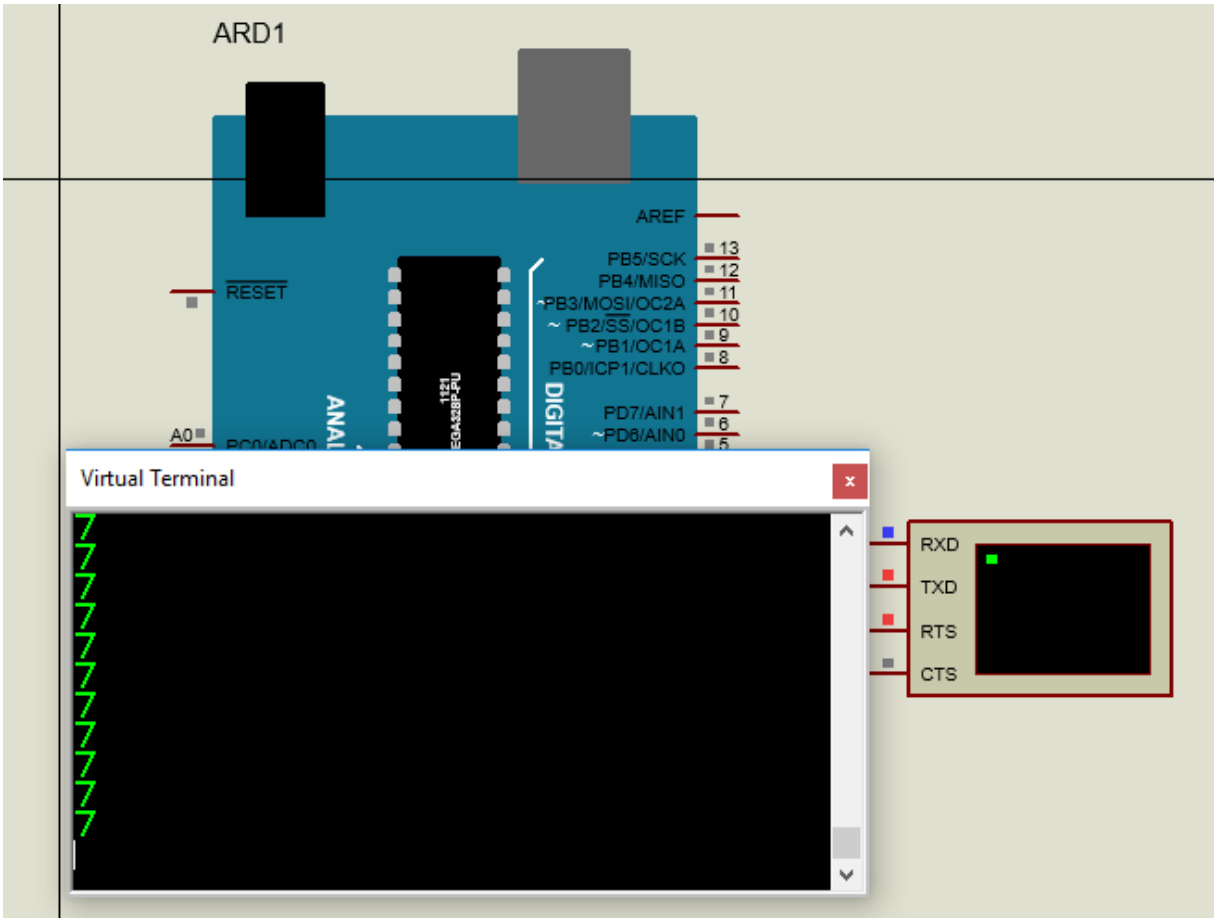
Ici on voit qu'en partant du début de la chaine de caractère la variable t est à la troisième position sachant le comptage commence à partir de 0 cela fait que la fonction indexOf retourne une valeur de

2. Mais on peut constater que cette variable il y a beaucoup de t la fonction indexOf ; elle retourne la position du premier t de la chaine de caractères

La fonction indexOf peut aussi repérer une chaine de caractères. Comme on peut voir dans le code suivant :

string_str §

```
 String vartext = "anticonstitutionnellement";
  int n ;
void setup() {

  Serial.begin(9600);

}

void loop() {
 n = vartext.indexOf("stitu");
 Serial.println(n);

}
```



Ici on voit que la chaine de caractère se trouve sur la position numéro 7.

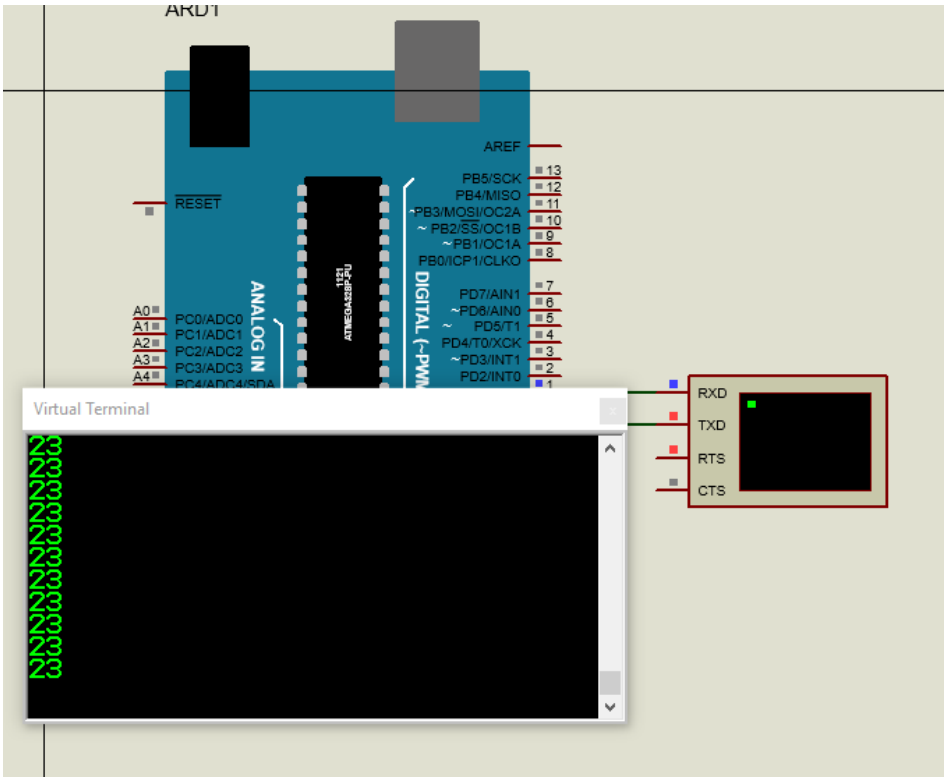
<u>lastIndexOf</u>

Cette fonction permet de renvoyer la position d'un caractère ou une chaine de caractère dans une autre qui est principale ça prend le dernier caractère de la chaine. L'index commence par 0.

Code :

```
string_str §
 String vartext = "anticonstitutionnellement";
  int n ;
void setup() {

  Serial.begin(9600);

}

void loop() {
 n = vartext.lastIndexOf("n");
 Serial.println(n);

}
```

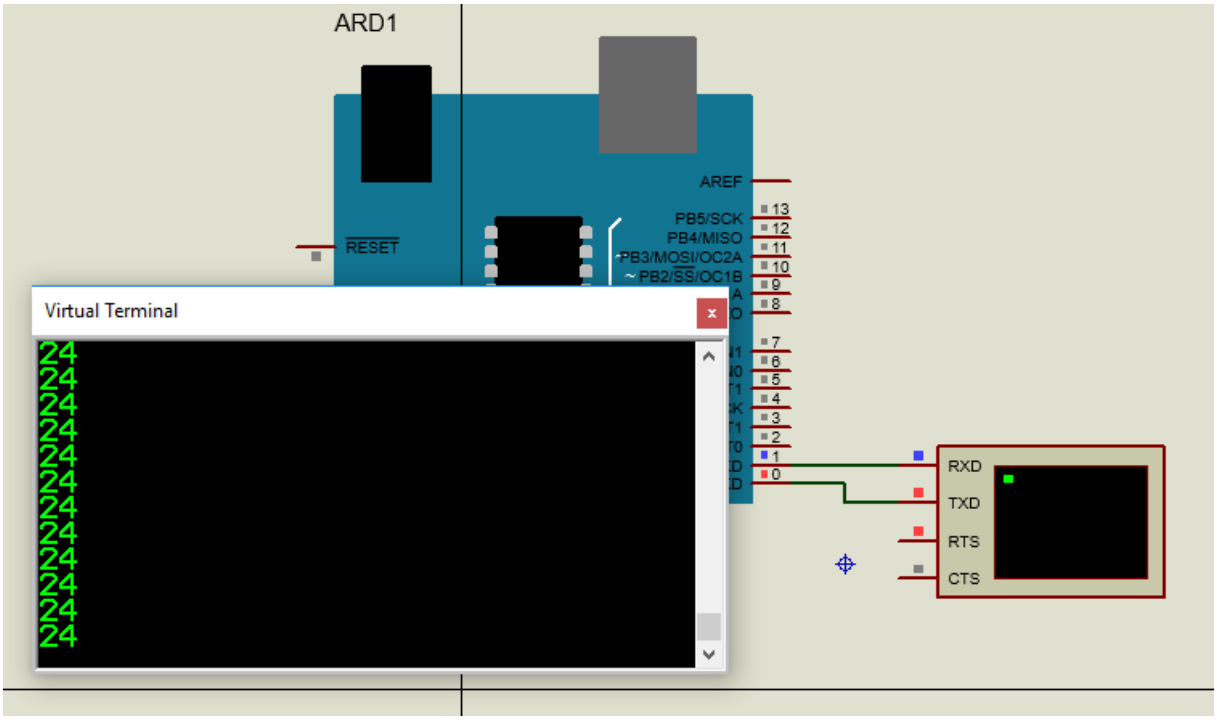
Résultat :



Code :

```
string_str §
 String vartext = "anticonstitutionnellement";
  int n ;
void setup() {

  Serial.begin(9600);

}

void loop() {
 n = vartext.lastIndexOf("t");
 Serial.println(n);

}
```

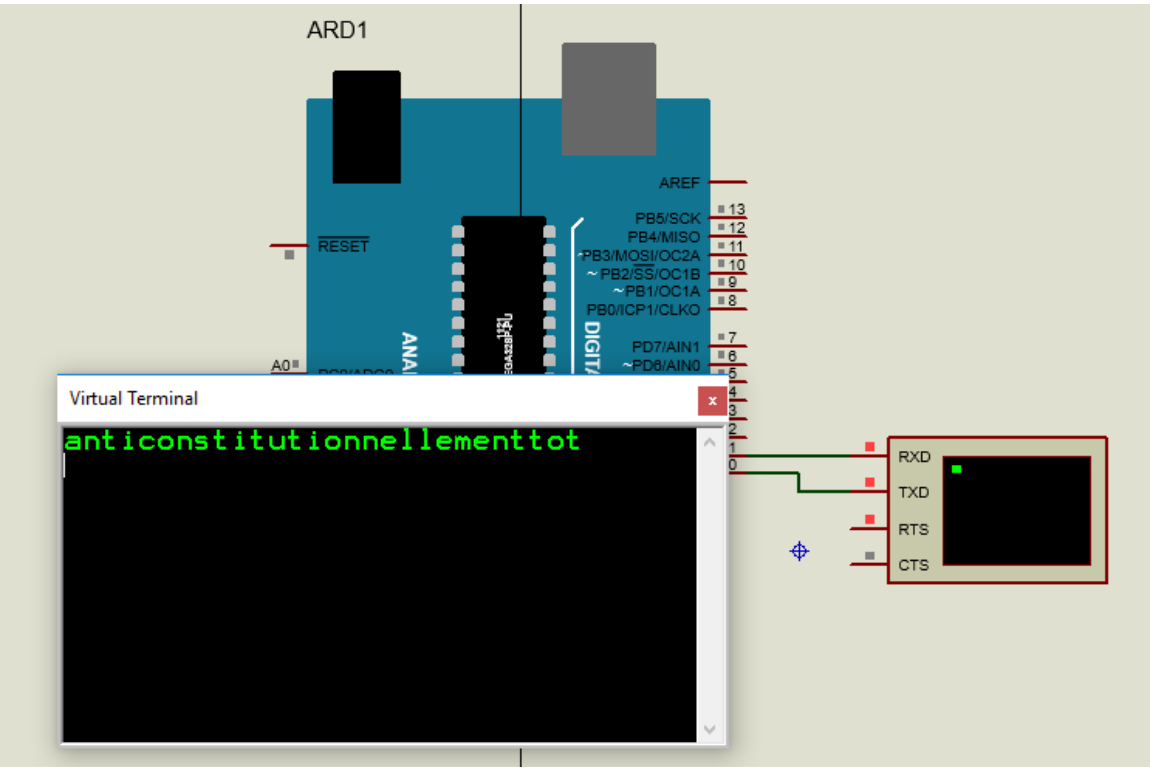
Résultat :



concat

Cette fonction permet de concaténer deux chaines de caractères ou additionner deux chaines de caractères.

```
string_str §
#include <String.h>

String vartext = "anticonstitutionnellement";
String reception  = "tot";
String recu ;

 int n ;
void setup() {
 Serial.begin(9600);
recu = vartext.concat(reception) ;
 Serial.println(vartext);

}

void loop() {


}
```



Ici on voit que les variables String vartext et reception sont additionner en termes de programmation on parle de la concaténation.

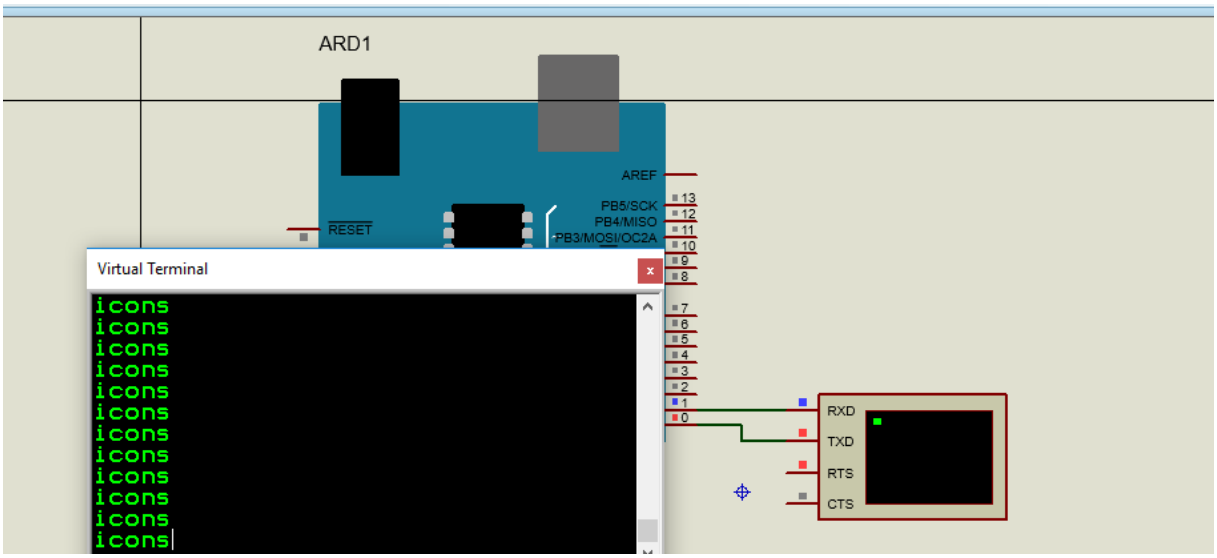
substring

Cette méthode permet d'extraire une chaine de caractère dans une autre de caractère il est constitué par deux paramètres l'un étant inclus et l'autre exclus.

Code :

```
string_str §
#include <String.h>

String vartext = "anticonstitutionnellement";


 int n ;
void setup() {
 Serial.begin(9600);
}

void loop() {

  Serial.println(vartext.substring(3,8));

}
```

Résultat :



Pour la fonction substring
On commence aussi par l'index 0
Cette fonction est constituée par deux paramètres il permet d'obtenir une chaine de caractères dans une autre ici lors du triage le premier élément est inclus tandis que le second est exclus.

compareTo

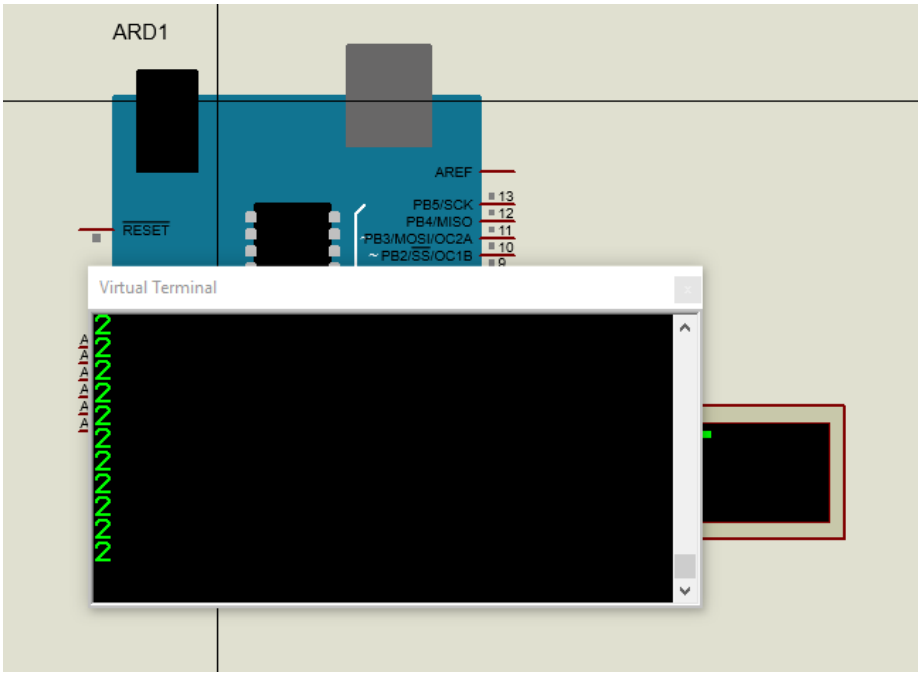
Pour cette fonction la comparaison est bonne quand les deux chaines de caractères ont une casse de début qui est le même.

Tous doivent commencer par une lettre minuscule ou tous commencent par une lettre majuscule

Fichier Édition Croquis Outils Aide

string_str §

```
#include <String.h>

String vartext = "anti";
String  vartext2 = "cob";

 int n ;
void setup() {
 Serial.begin(9600);
}

void loop() {
  n = vartext2.compareTo(vartext);
  Serial.println(n);

}
```



En français et en anglais les lettres vont de A à Z

A ou a correspond à 0

Et Z ou z correspond à 25

Alors la fonction compareTo retourne un int résultant de la différence entre la position alphabétique de premier caractère

Par exemple ici on voit deux cela veut dire

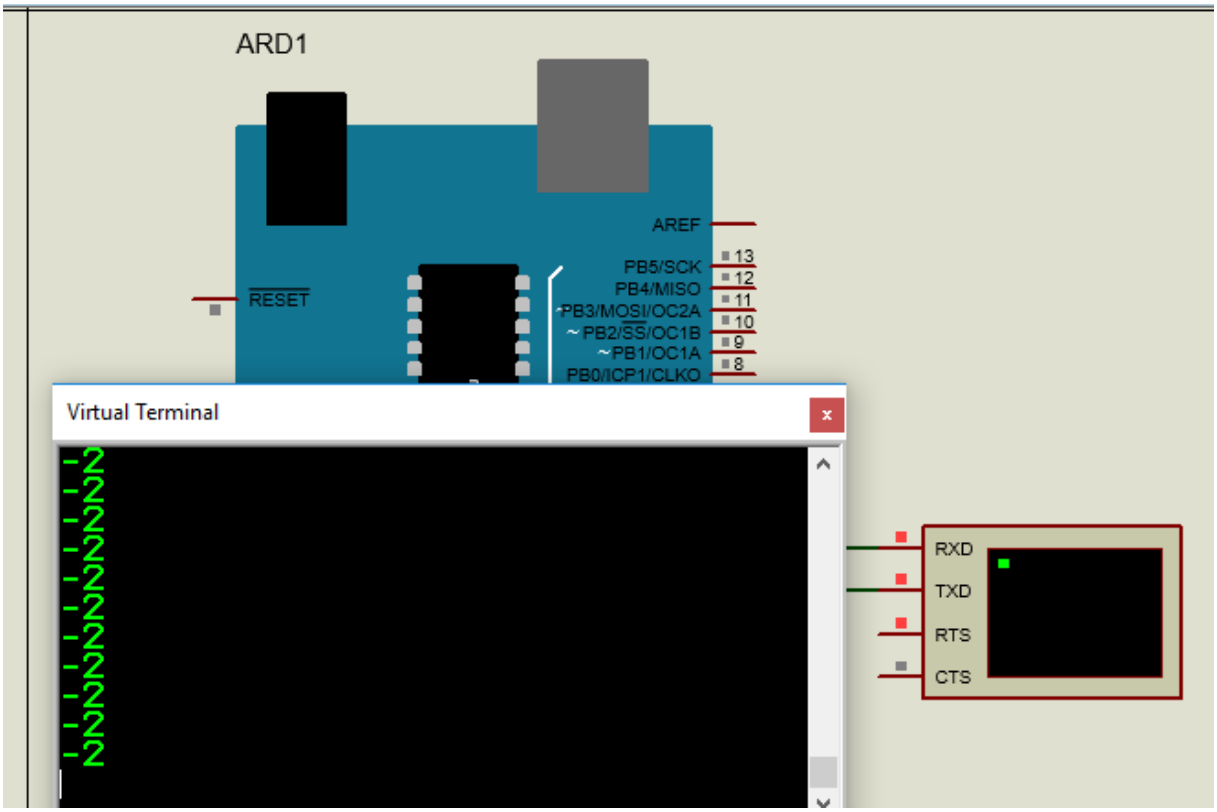
La lettre c occupe la position 2

Et la lettre a occupe la position 0

Ce qui donne 2-0=2

string_str §

```
#include <String.h>

String vartext = "anti";
String  vartext2 = "cob";

 int n ;
void setup() {
 Serial.begin(9600);
}

void loop() {
  n = vartext.compareTo(vartext2);
  Serial.println(n);

}
```




```
void loop() {
  n = vartext.compareTo(vartext2);
  Serial.println(n);
```

Si vartext est alphabétiquement en avant de vartext2 ça retourne un nombre négatif

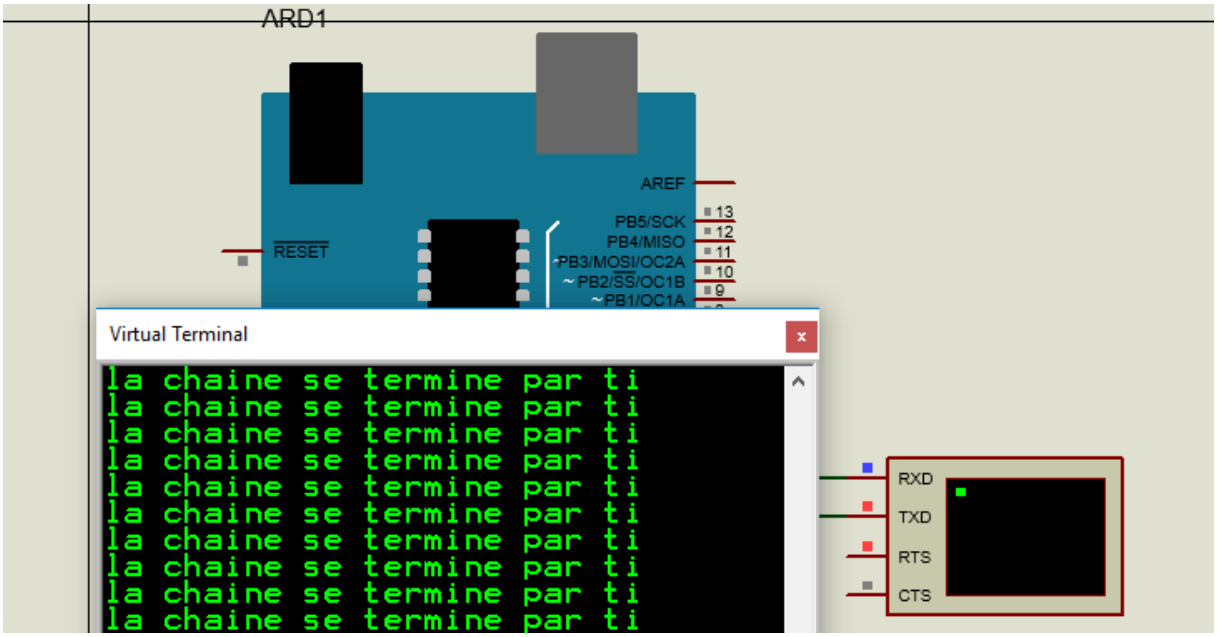
Si vartext est alphabétiquement en arrière de vartext2 ça retourne un nombre positif

endsWith

Cette fonction permet vérifier si une chaine de caractère se termine par une autre

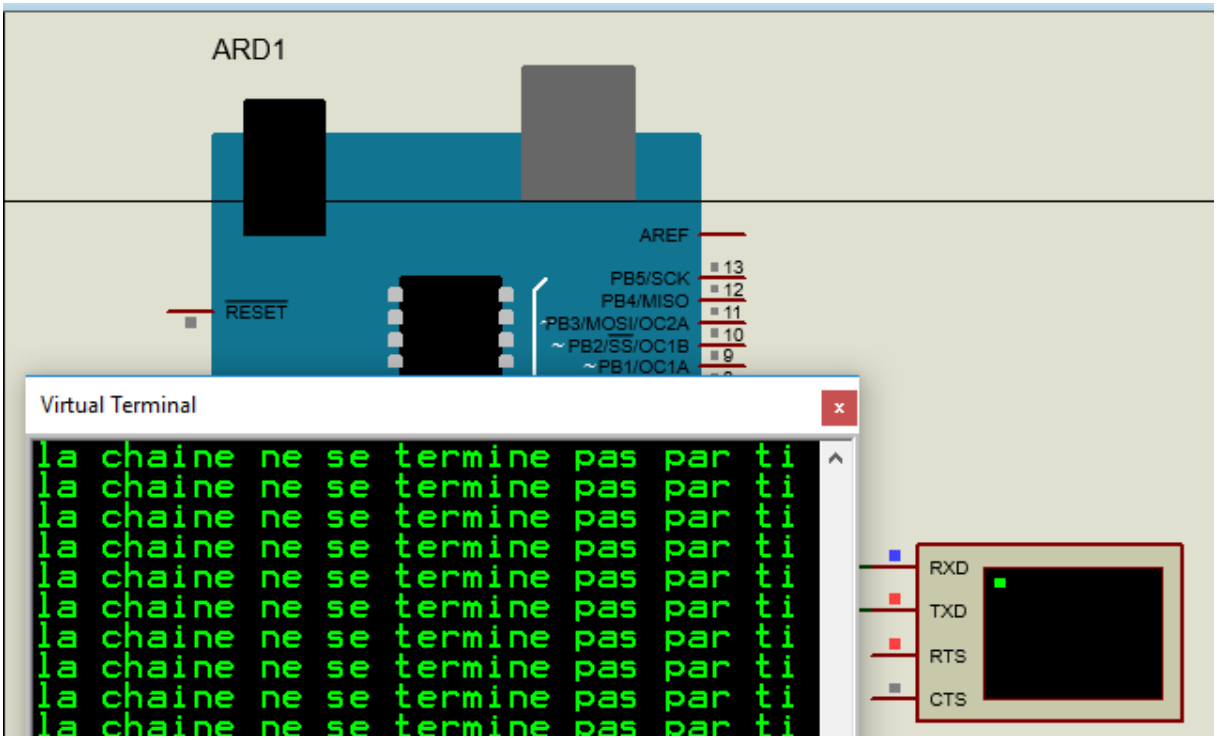
Code :

```
string_str §

#include <String.h>

String vartext = "anti";
String  vartext2 = "cob";

 int n ;
void setup() {
 Serial.begin(9600);
}

void loop() {
 if (vartext.endsWith("ti") ) {
  Serial.println ("la chaine se termine par ti");

 } else {
  Serial.println ("la chaine ne se termine pas par ti");

 }
}
```



Ici on voit effectivement que la chaine de caractère qui fait l'objet de l'étude se termine par «ti »

```
string_str §
#include <String.h>

String vartext = "anti";
String  vartext2 = "cob";

 int n ;
void setup() {
 Serial.begin(9600);
}

void loop() {
 if (vartext.endsWith("to") ) {
  Serial.println ("la chaine se termine par ti");

 } else {
  Serial.println ("la chaine ne se termine pas par ti");

 }
}
```



Il exécute l'instruction else car la chaine de caractère qui fait l'objet de l'étude ne se termine pas par « to ».

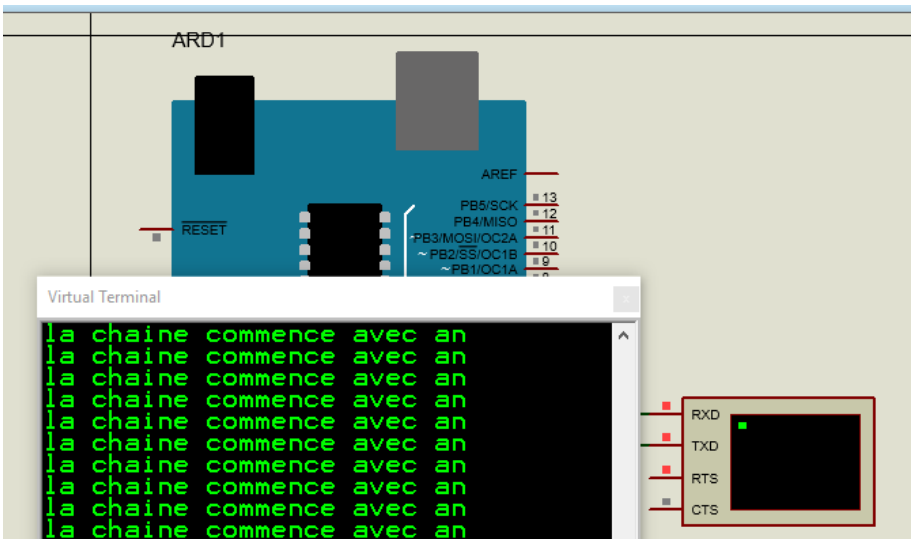
startWith

Cette fonction permet vérifier si une chaine de caractère commence par une autre

Code :

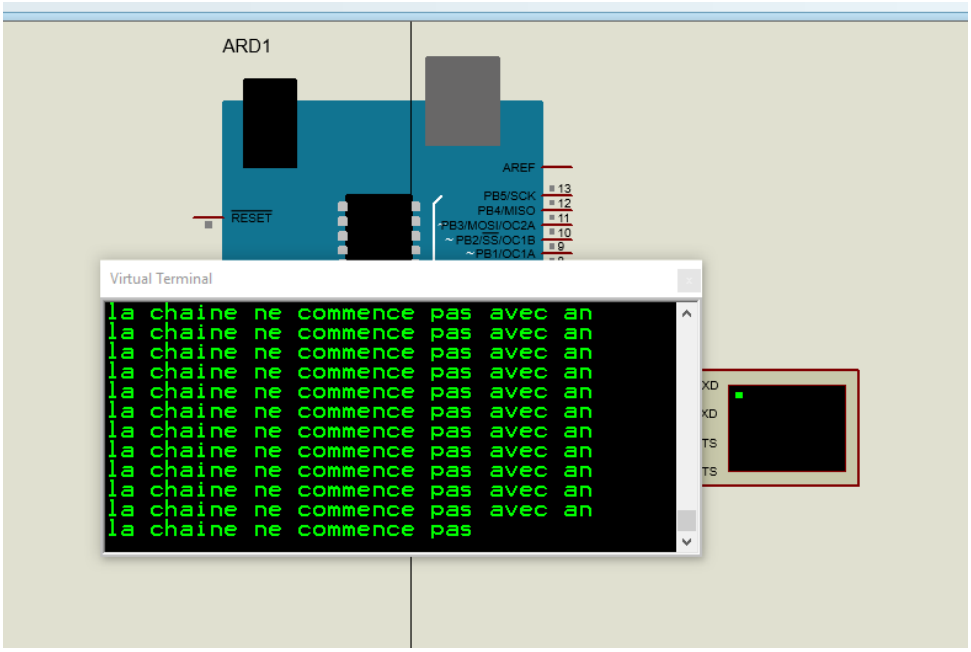
```
string_str §
#include <String.h>

String vartext = "anti";
String  vartext2 = "cob";

 int n ;
void setup() {
 Serial.begin(9600);
}

void loop() {
 if (vartext.startsWith("an") ) {
  Serial.println ("la chaine commence avec an");

 } else {
  Serial.println ("la chaine ne commence pas avec an");

 }
}
```

Résultat :



Ici on voit réellement que la chaine de caractère qui fait l'objet de cette étude est terminée par « an »

```
string_str §
#include <String.h>

String vartext = "anti";
String  vartext2 = "cob";

 int n ;
void setup() {
 Serial.begin(9600);
}

void loop() {
 if (vartext.startsWith("yf") ) {
  Serial.println ("la chaine commence avec an");

 } else {
  Serial.println ("la chaine ne commence pas avec an");

 }
}
```



Ici on voit que la chaine de caractère ne commence pas avec y

C’est pourquoi on constate l’exécution d’else.

CHAPITRE 2 LES INTERRUPTIONS

INTRODUCTION

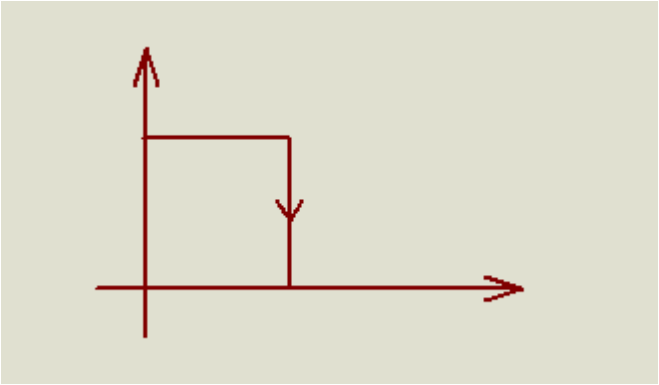
Imaginer que vous avez un téléphone entrain de suivre une vidéo puis on vous appelle la vidéo va d'abord s'arrêter à cause de l'appelle car pour un téléphone l'appelle est prioritaire. En effectuant l'appelle une fois l'appelle terminer la vidéo revient là où elle était pour continuer. En quelque sorte c'est ça l'interruption il s'agit d'une action prioritaire coupant le déroulement normal du programme pour s'exécuter puis le programme retourne à son déroulement normal.

On distingue deux sortes d'interruptions internes et externes dans ce présent travail nous allons parler sur les interruptions externes.

FORME LOGIQUE DE L'INTERRUPTION

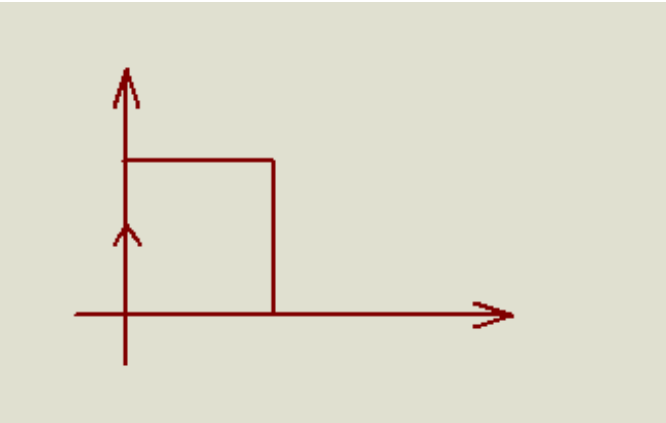
En ARDUINO il y a principalement quatre formes logiques d'interruptions

1. **FALLING** : il s'agit d'une interruption sur le front descendant sur les broches 3 ou 2.



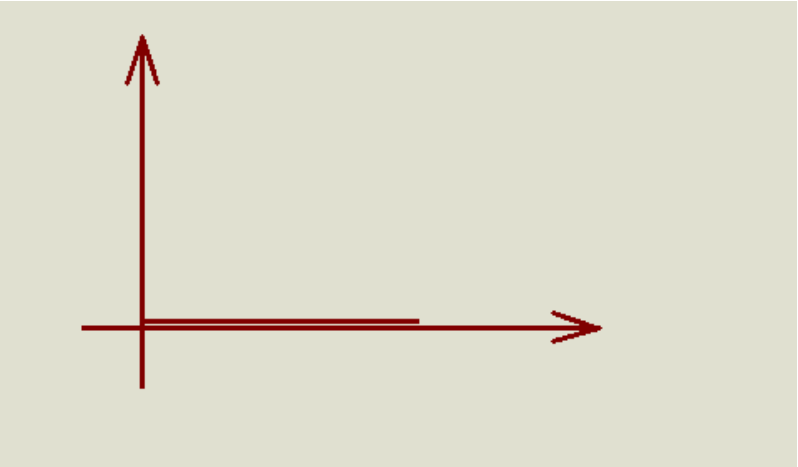
Dans ce cas il y a interruption si la broche passe de 1 à 0. Il s'agit d'une interruption externe sur front descendant.

2. **RISING** : il s'agit d'une interruption sur front montant sur les broches 3 ou 2



Dans ce cas il y a interruption si la broche passe de 0 à 1. Il s'agit d'une interruption externe sur front montant.

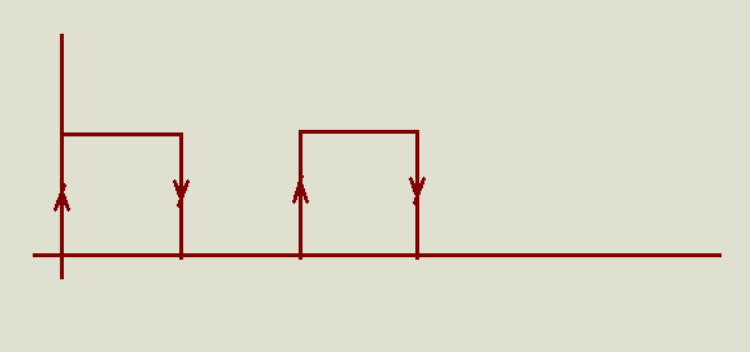
3. **<u>LOW</u>** : il s'agit d'une interruption qui s'effectue lorsque la broche est à l'état 0



Ici si la broche est à 0 cela est source d'interruption

4. **<u>CHANGE</u>**

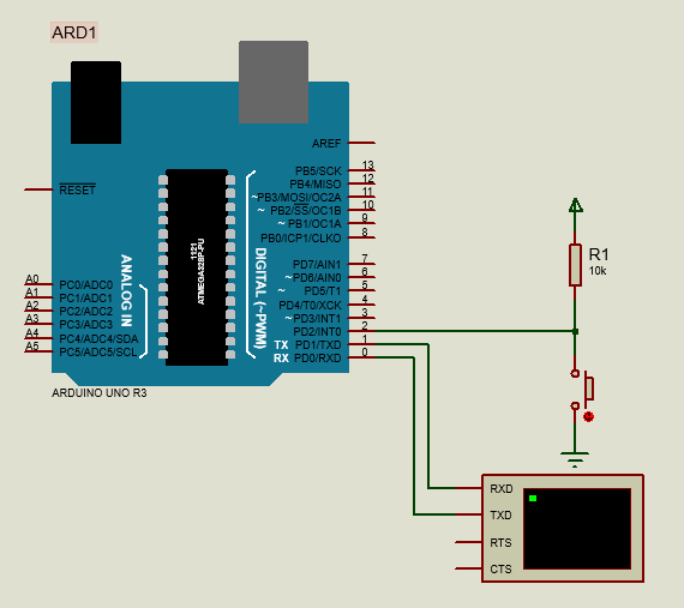
Dans ce cas il y a interruption lorsqu'il y a un changement d'état de la broche de 0 à 1 ou de 1 à 0.



<u>PROGRAMMATION DE L'INTERRUPTION EN ARDUINO</u>

1. **UTILISATION DE L'INTERRUPTION FALLING**
   Il est utile à signaler que cette interruption ne peut s'effectuer que sur la borne 3 et la borne 2 de la carte ARDUINO. On peut voir le code pour l'implémentation de cette interruption :

```
1  int compteur ;
2 void setup(){
3  Serial.begin(9600);
4  attachInterrupt(0,impulsioncompte,FALLING);
5 }
6
7 void loop() {
8   Serial.print("la valeur de compte est :");
9   Serial.println (compteur);
10 }
11  void impulsioncompte(){
12    compteur++;
13     if (compteur == 100){
14        compteur = 0;
15     }
16  }
17
```




```
4  attachInterrupt(0,impulsioncompte,FALLING);
```

La fonction <u>attachInterrupt</u> comprend trois paramètres

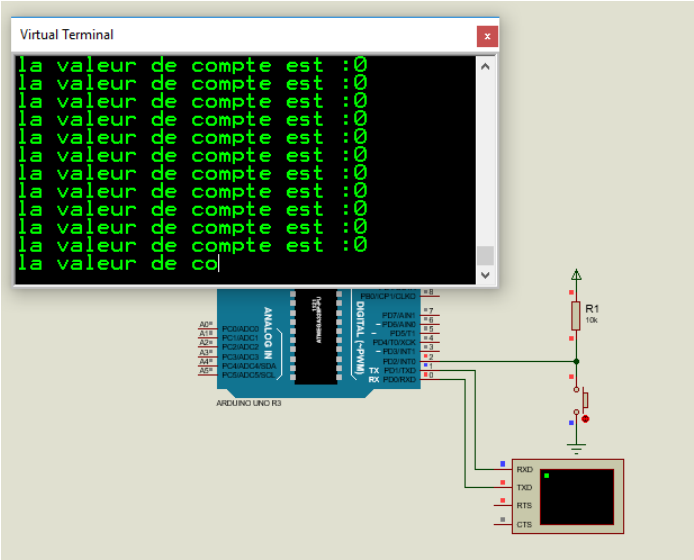
- <u>Le premier paramètre</u> détermine la broche de la carte arduino qui est sera objet de l'interruption
  - ✓ Si ce paramètre est égal à 0 donc l'interruption se fera sur la broche 2 de la carte ARDUINO.

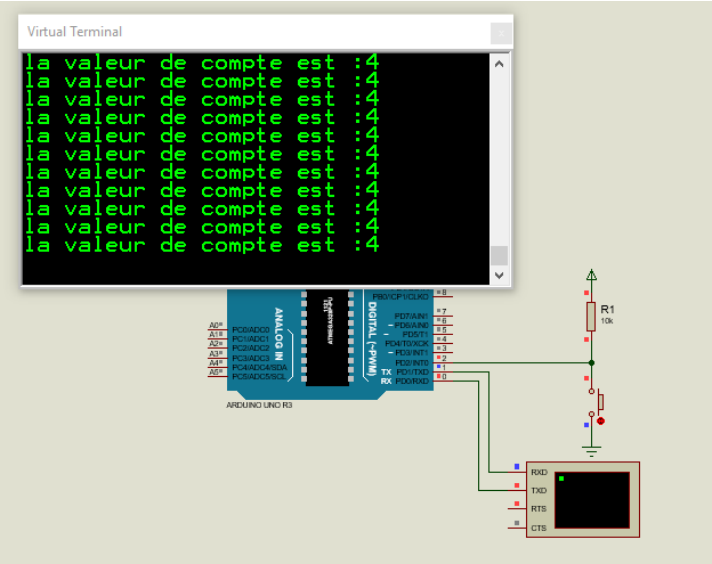
- ✓ Si ce paramètre est égal à 1 donc l'interruption se fera sur la broche 3 de la carte ARDUINO.

- <u>Le deuxième paramètre</u> détermine la fonction qui pourra s'exécuter lorsqu'il y a interruption ici dans notre cas à chaque interruption la fonction qui s'exécute c'est la fonction impulsioncompte () cette fonction permet d'incrémenter la variable compte donc à chaque entré dans l'interruption la variable compte est incrémentée.

- <u>Le troisième paramètre</u> détermine la nature de l'interruption par exemple ici dans notre cas il y aura interruption quand l'état de la broche 2 de la carte ARDUINO passe de 1 à 0.

IMAGE DE SIMULATION





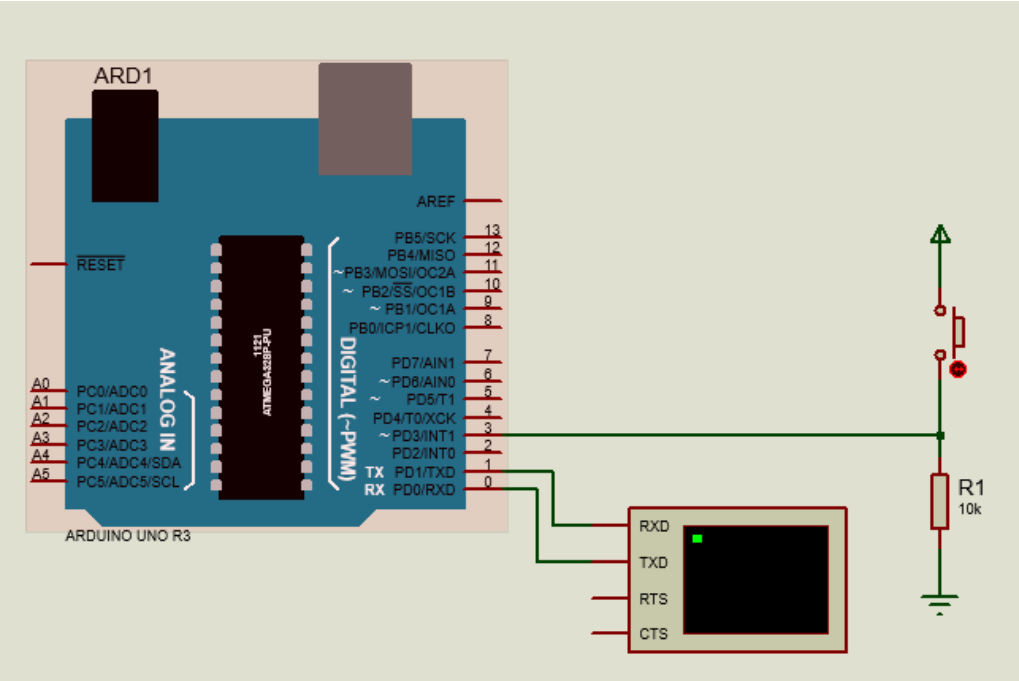
A chaque appuie sur le bouton la variable compte se voit incrémenter quand la valeur compte arrive à 100 elle retourne à 0.

Il est utile à signaler que l'interruption s'exécute juste pendant un temps. Donc quand il y a interruption le programme s'arrête pour aller exécuter la fonction d'interruption après cette exécution il retourne dans son déroulement normal.

UTILISATION DE L'INTERRUPTION RISING

interruption §

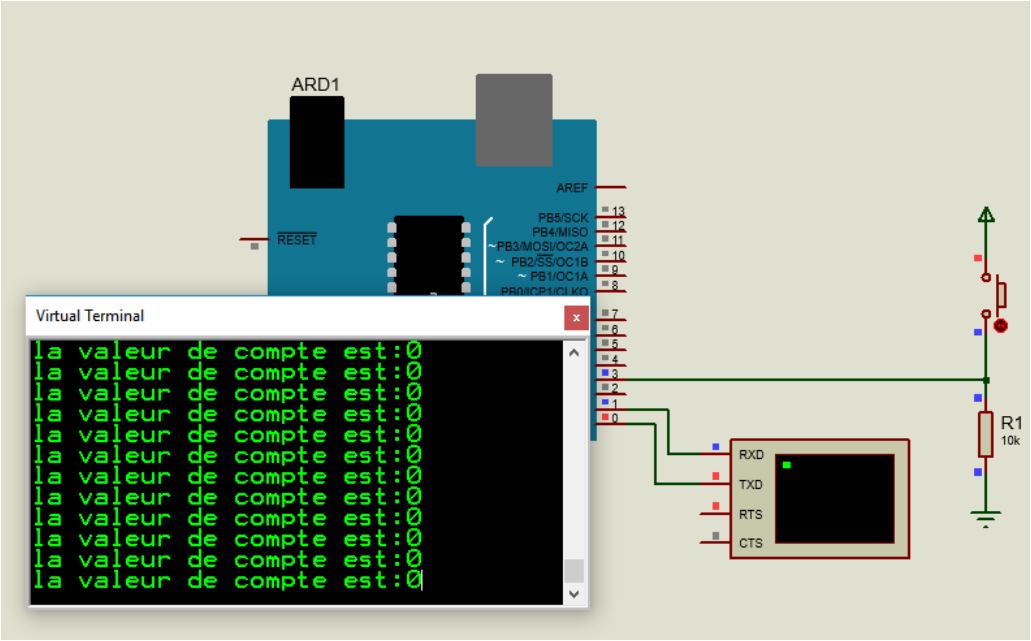
```
 int compteur ;
void setup(){
 Serial.begin(9600);
 attachInterrupt(1,impulsioncompte,RISING);
}

void loop() {
  Serial.print("la valeur de compte est:");
  Serial.println(compteur);
}
 void impulsioncompte(){
   compteur++;
    if (compteur == 100){
       compteur = 0;
    }
 }

```



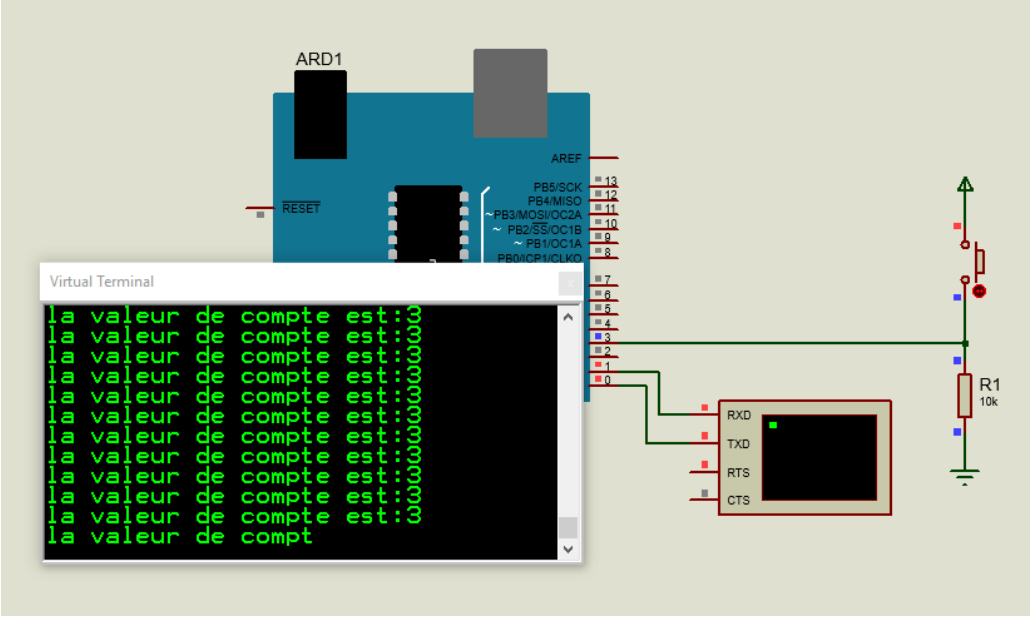
Ici l'interruption se passe sur la broche 3 de la carte ARDUINO. Il s'agit de l'interruption **RISING** dans ce cas l'interruption est déclenché lorsque l'entre passe de 0 à 1 ici il s'agit de l'interruption sur front montant.

RESULTAT DE SIMULATION

Résultant avant appuie sur le bouton poussoir :



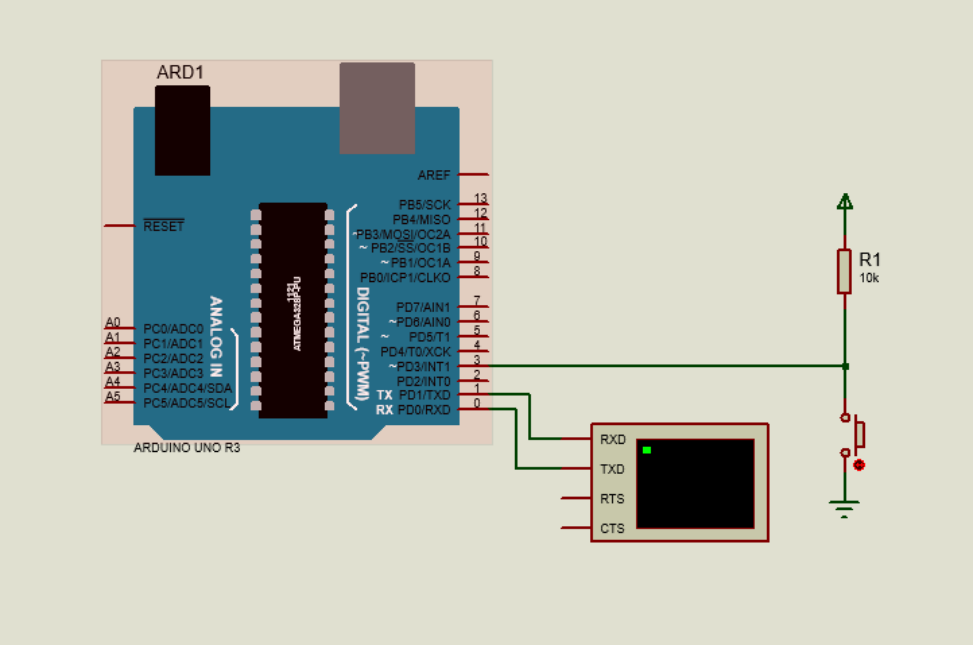
Résultat après trois appuies sur le bouton poussoir :



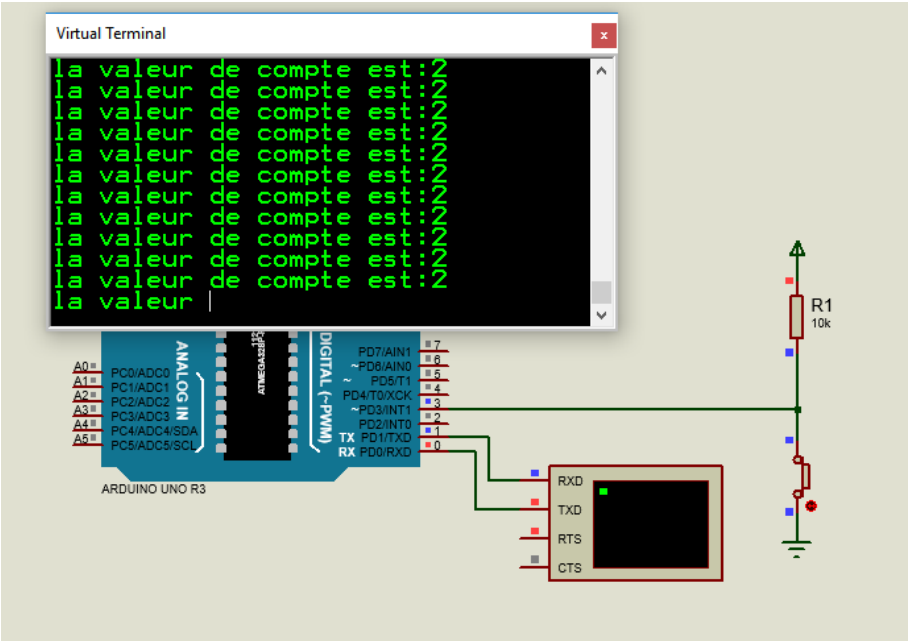
UTILISATION DE L'INTERRUPTION LOW

interruption §

```
 int compteur ;
void setup(){
 Serial.begin(9600);
 attachInterrupt(1,impulsioncompte,LOW);
}

void loop() {
  Serial.print("la valeur de compte est:");
  Serial.println(compteur);
}
 void impulsioncompte(){
   compteur++;
    if (compteur == 100){
       compteur = 0;
    }
 }
```



Ici pour ce montage il y a interruption lorsque la borne 3 est à 0.

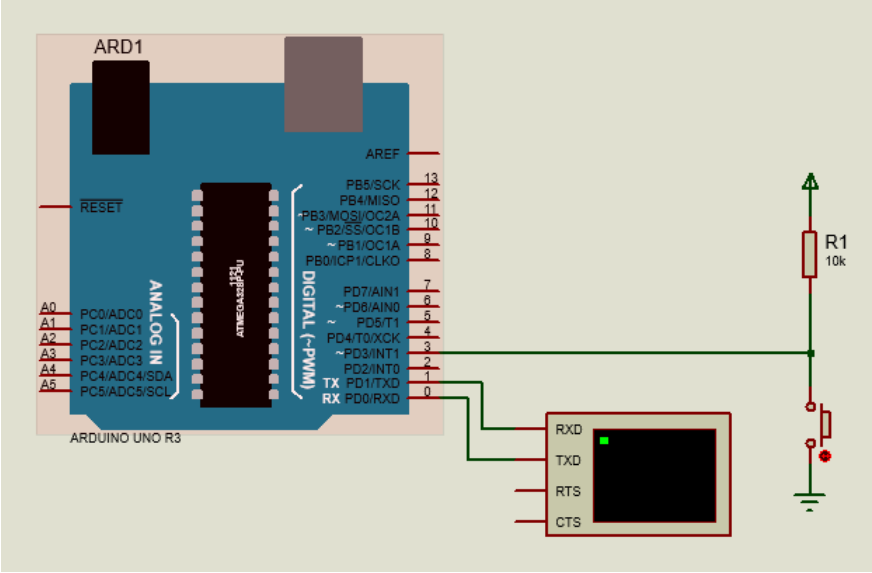


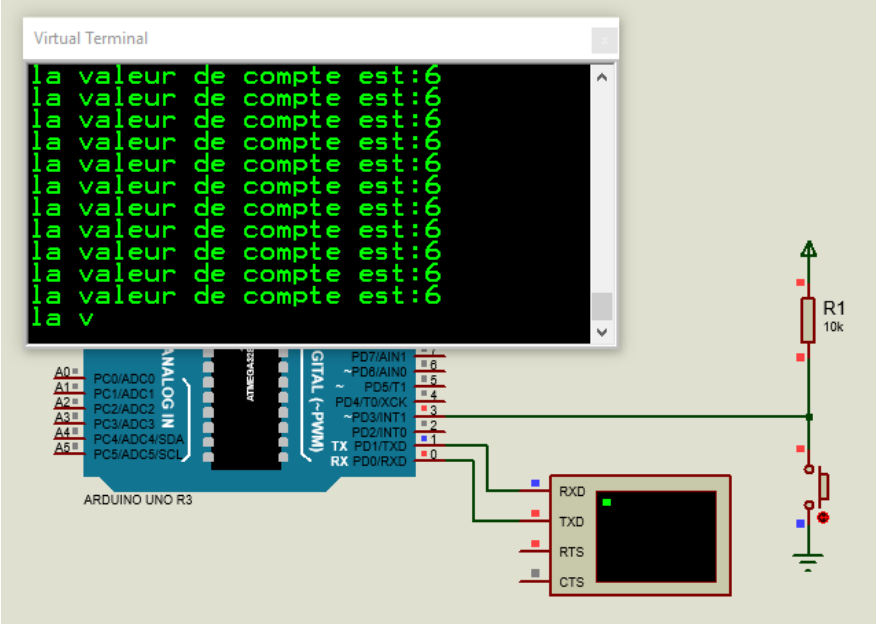
Dans ce cas il y a interruption lorsque la borne 3 passe à 0.

UTILISATION DE L'INTERRUPTION CHANGE

interruption

```
 int compteur ;
void setup(){
 Serial.begin(9600);
 attachInterrupt(1,impulsioncompte,CHANGE);
}

void loop() {
  Serial.print("la valeur de compte est:");
  Serial.println(compteur);
}
 void impulsioncompte(){
   compteur++;
    if (compteur == 100){
       compteur = 0;
    }
 }
```





Ici il y a interruption lorsqu'il y a changement d'état sur la broche 3 la variable compte incrémente lorsqu'on appuie et lorsqu'on lâche le bouton poussoir donc à chaque changement d'état il y a interruption.

L'interruption est une notion trop importante en lisant cet article je crois vous avez capable d'appréhender cette notion.

Il est utile de noter que les interruptions sont des commandes prioritaires qui s'exécutent pendant une courte durée puis le programme revient dans son déroulement normal.

CHAPITRE 3 ENVOIE DES DONNEES DE C# VERS ARDUINO

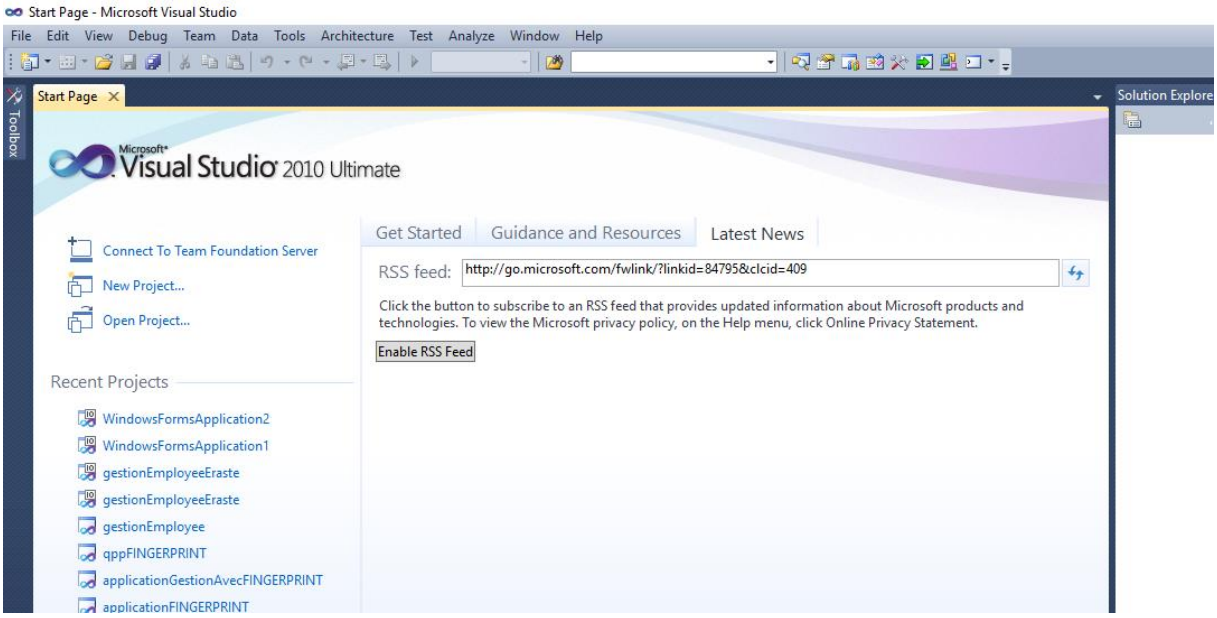
INTRODUCTION

Parfois vous pouvez avoir besoin de faire la commande de vos différents équipements avec ARDUINO mais en utilisant un ordinateur. En utilisant une application comme interface de commande pour arriver à cela vous pouvez utiliser plusieurs langages de programmation mais dans cette présente article nous utilisons le langage C#.

Comme exercice nous allons faire un programme permettant d'allumer ou d'éteindre une led selon que l'on a appuyé sur le bouton de l'application ON ou OFF

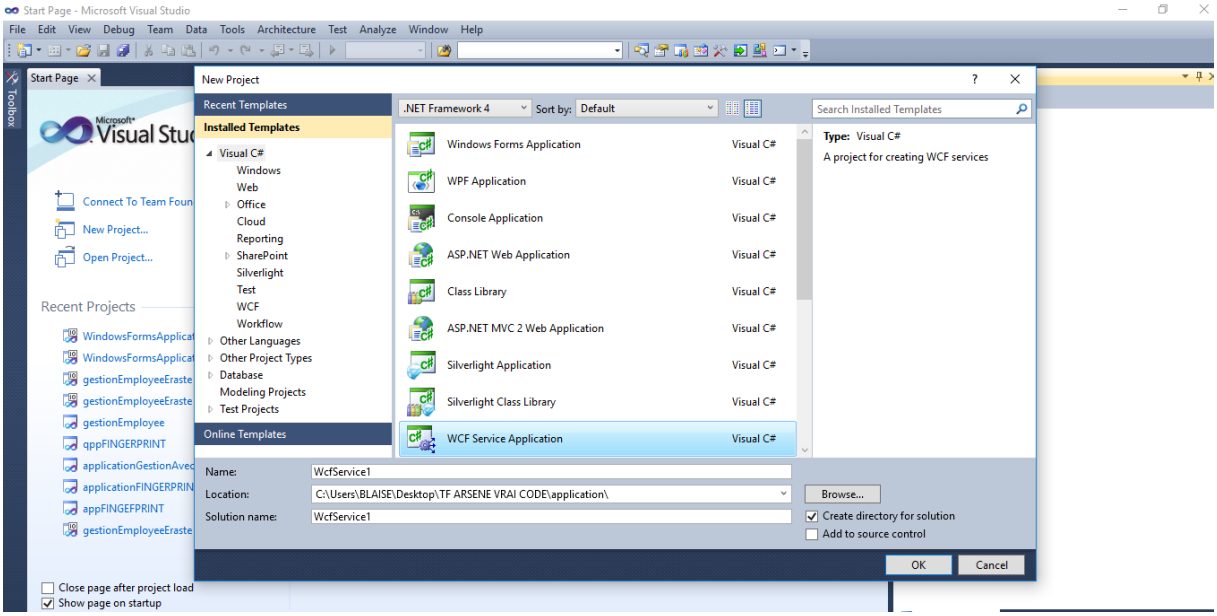
Au début nous allons commencer par la création de l'application C#

Au début lancer le programme **Microsoft visual studio** à l'accueil vous aurez :



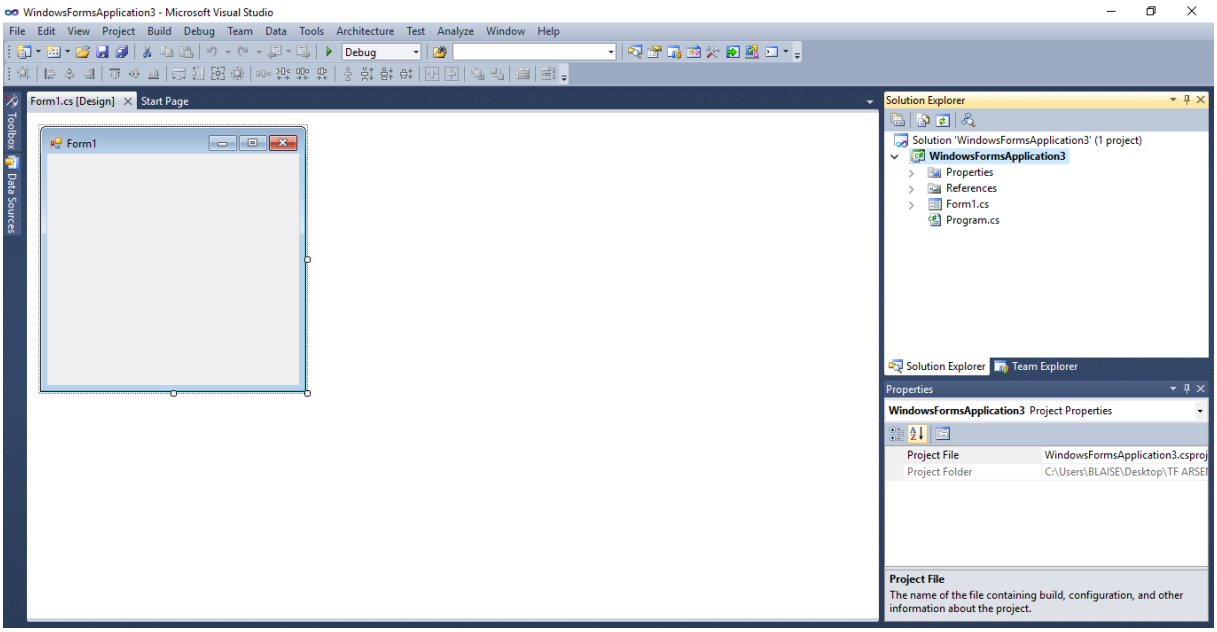
Une fois vous êtes là créer un nouveau projet :

File / new / Project

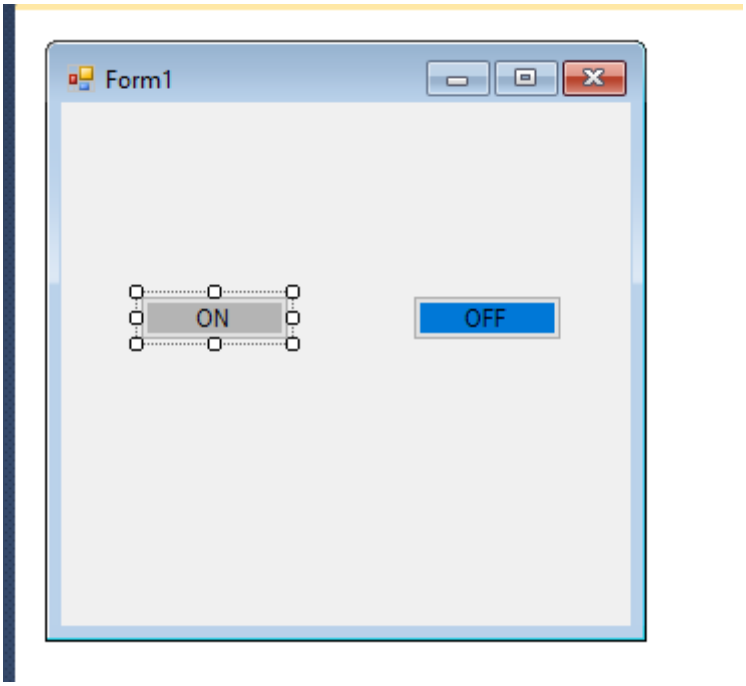


Choisissez C# puis windows applications

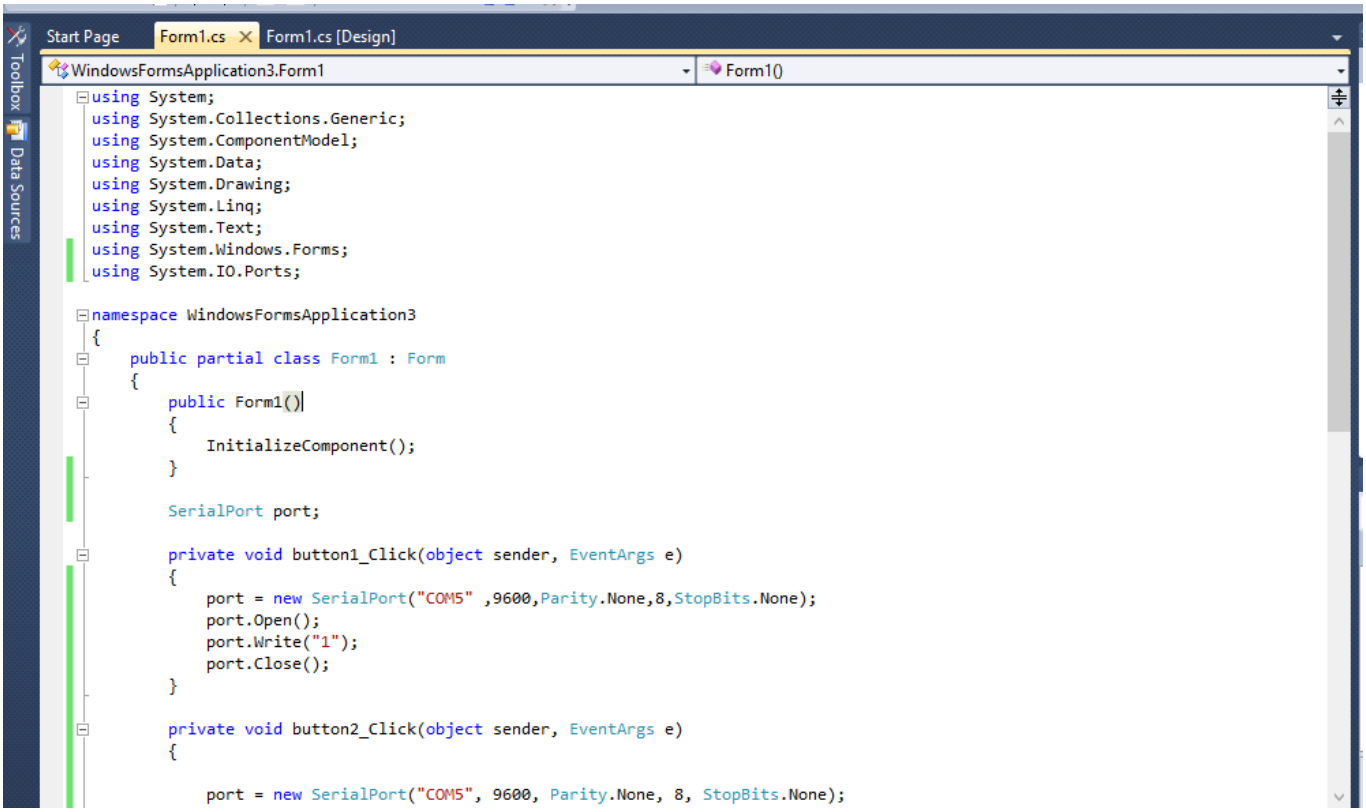
Une telle fenêtre doit normalement apparaitre



Cliquez sur toolBox puis faites le design de votre application :



Puis écrivez le code suivant : quand on appuie sur ON on envoie «1 » sur le port série et si on appuie sur OFF on envoie « 0 » sur le moniteur série.



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

namespace WindowsFormsApplication3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        SerialPort port;

        private void button1_Click(object sender, EventArgs e)
        {
            port = new SerialPort("COM5" ,9600,Parity.None,8,StopBits.None);
            port.Open();
            port.Write("1");
            port.Close();
        }

        private void button2_Click(object sender, EventArgs e)
        {

            port = new SerialPort("COM5", 9600, Parity.None, 8, StopBits.None);
```

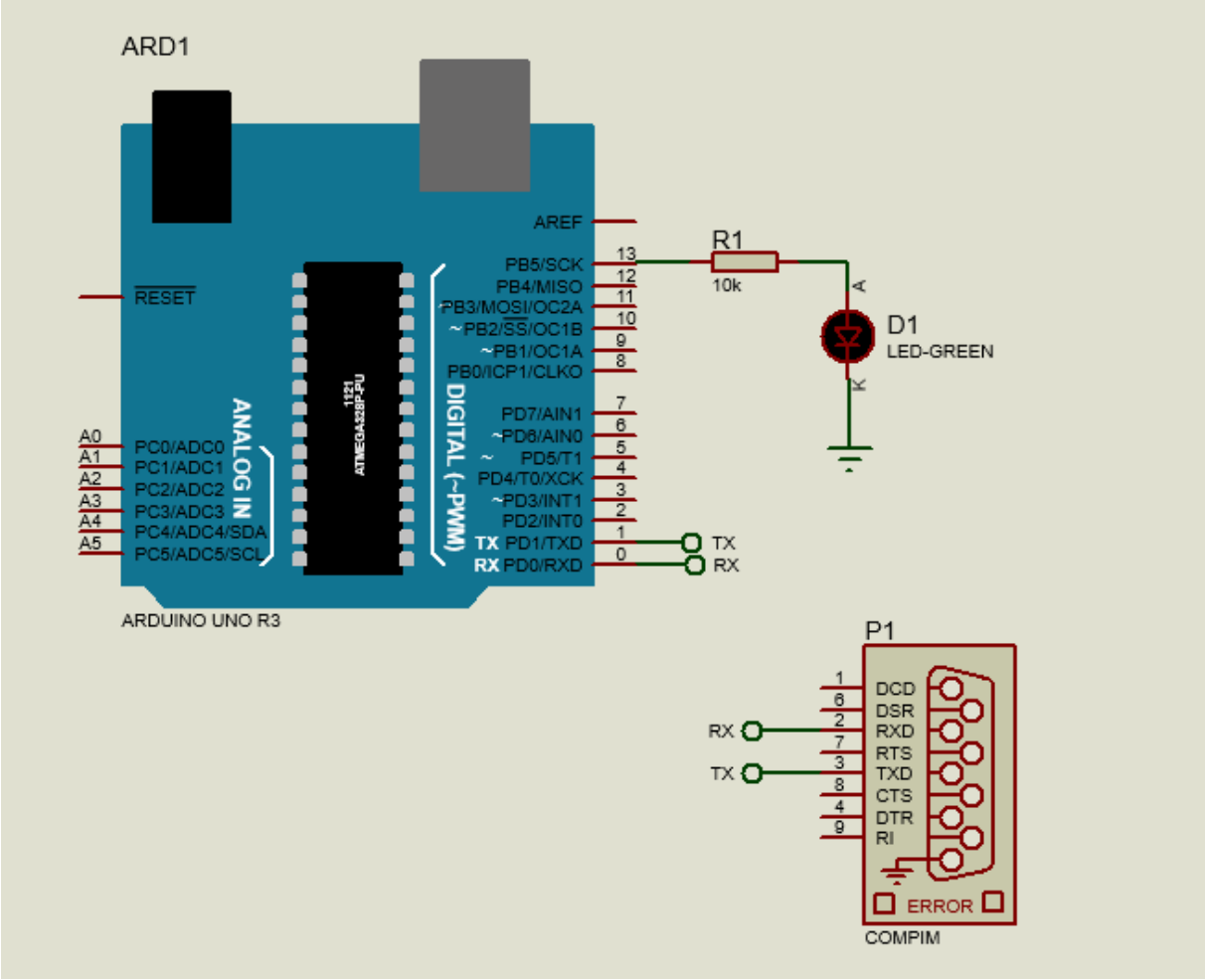
```
            port.Open();
            port.Write("0");
            port.Close();
        }
    }
}
```

Du côté ARDUINO faites le code suivant : juste une simple communication série voir code suivant :

```
int information ;

void setup() {
  Serial.begin (9600) ;
  pinMode (13 ,OUTPUT) ;
}

void loop() {
  if (Serial.available () >0) {

     information = Serial.read () ;

  }

   if (information == '1') {
     digitalWrite (13, HIGH) ;
   }
   if (information == '0') {
     digitalWrite (13, LOW) ;
   }
}
```

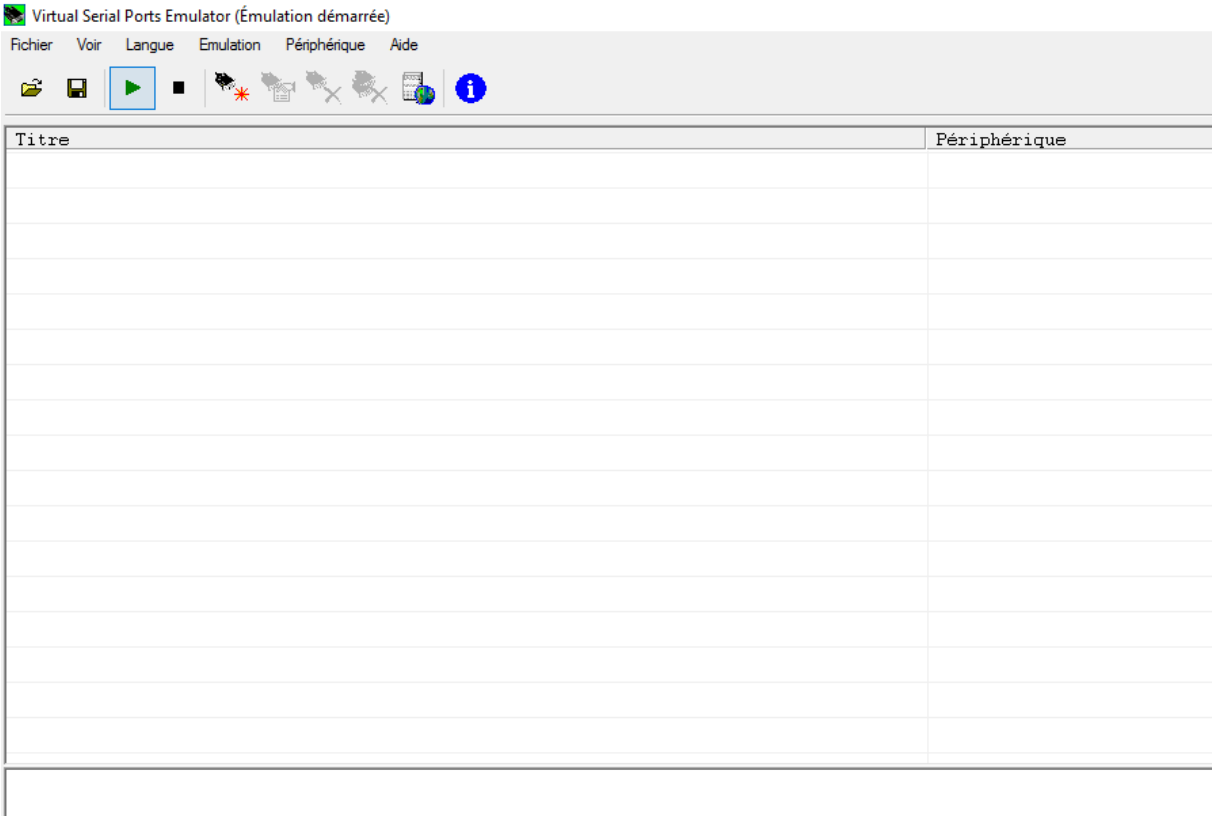
NB : dans le code C# il y a un endroit où on change le nom du port. Veuillez changer cela en fonction du port sur lequel se trouve votre arduino.

SCHEMA

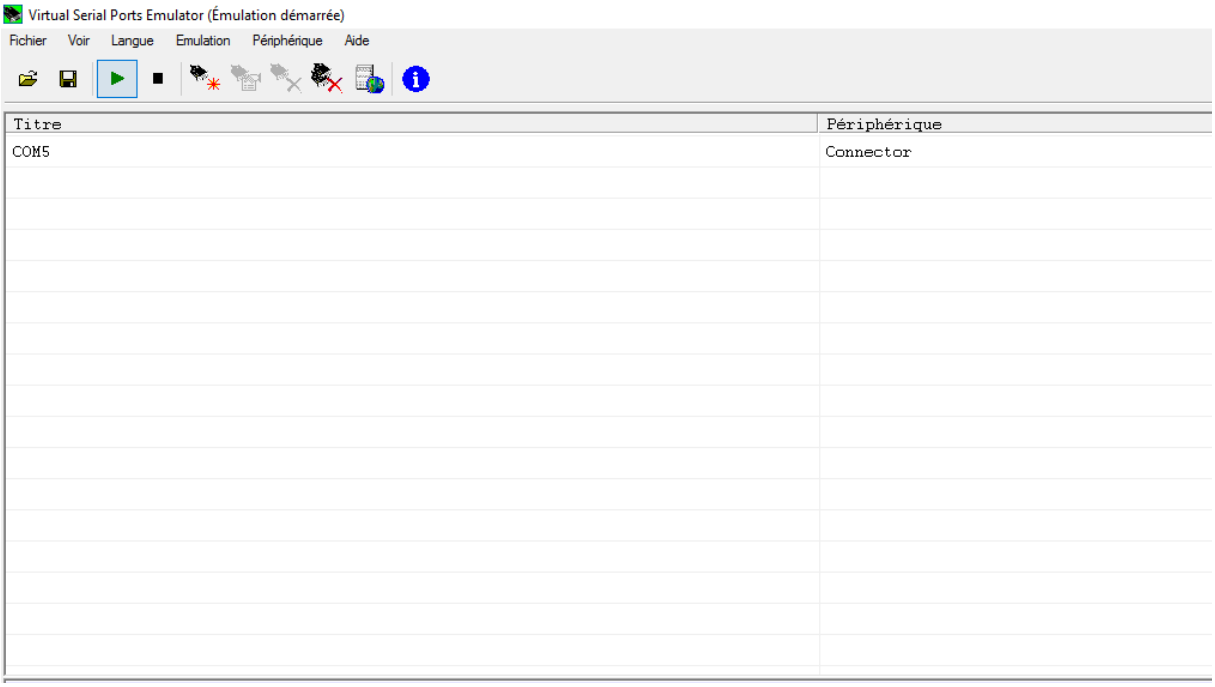


SIMULATION

Pour la simulation vous devez avoir un logiciel de création de port virtuel dans notre cas nous avons utilisé VSPE

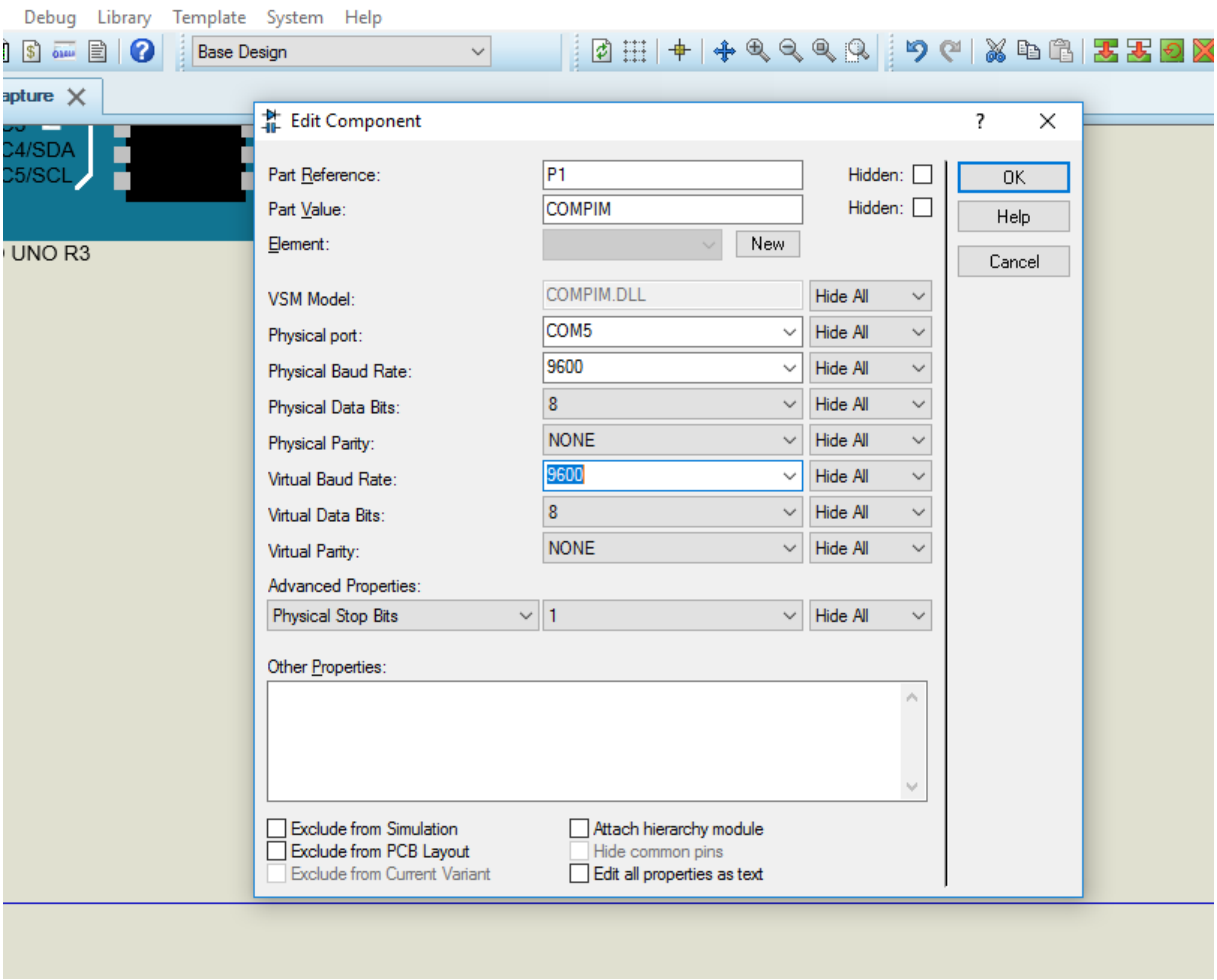


Grace à ce logiciel nous allons créer un port virtuel COM5

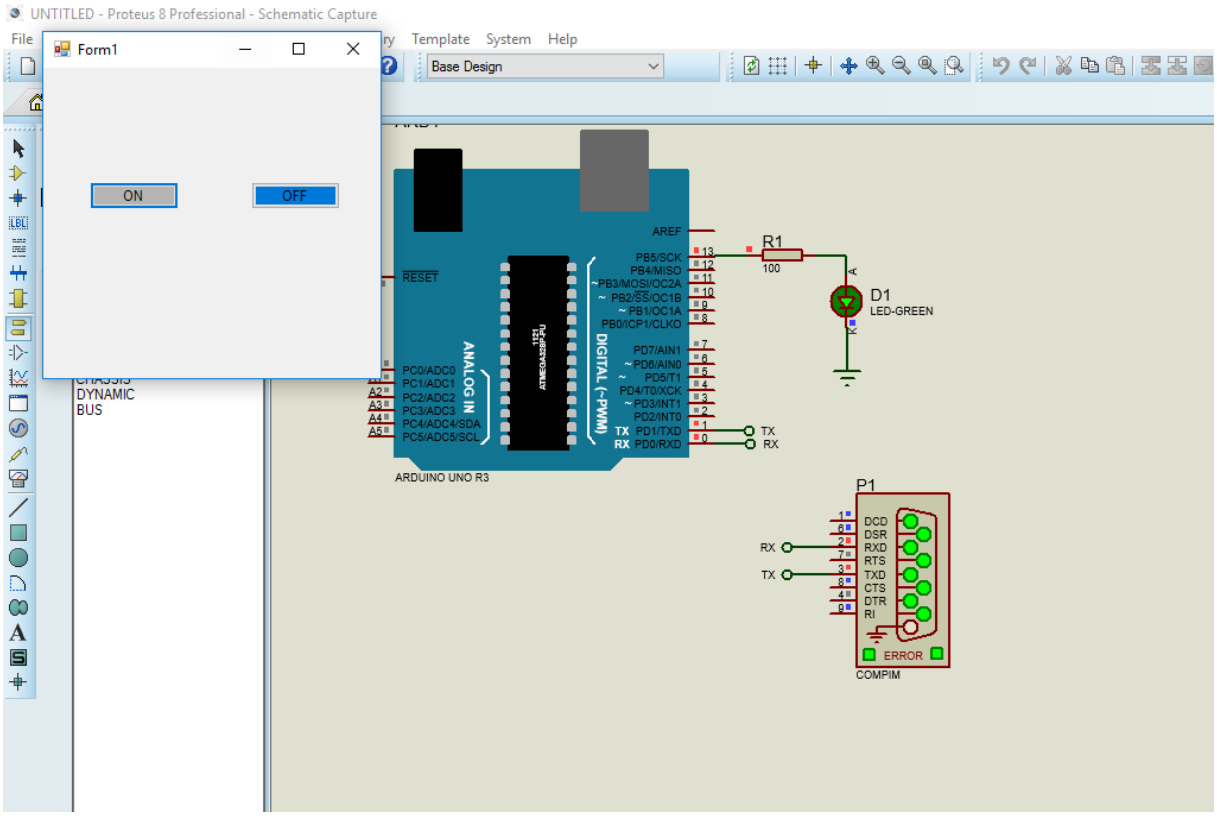


Ce port est déjà disponible.

En proteus fait clic droit sur le port puis configurer les propriétés suivant :

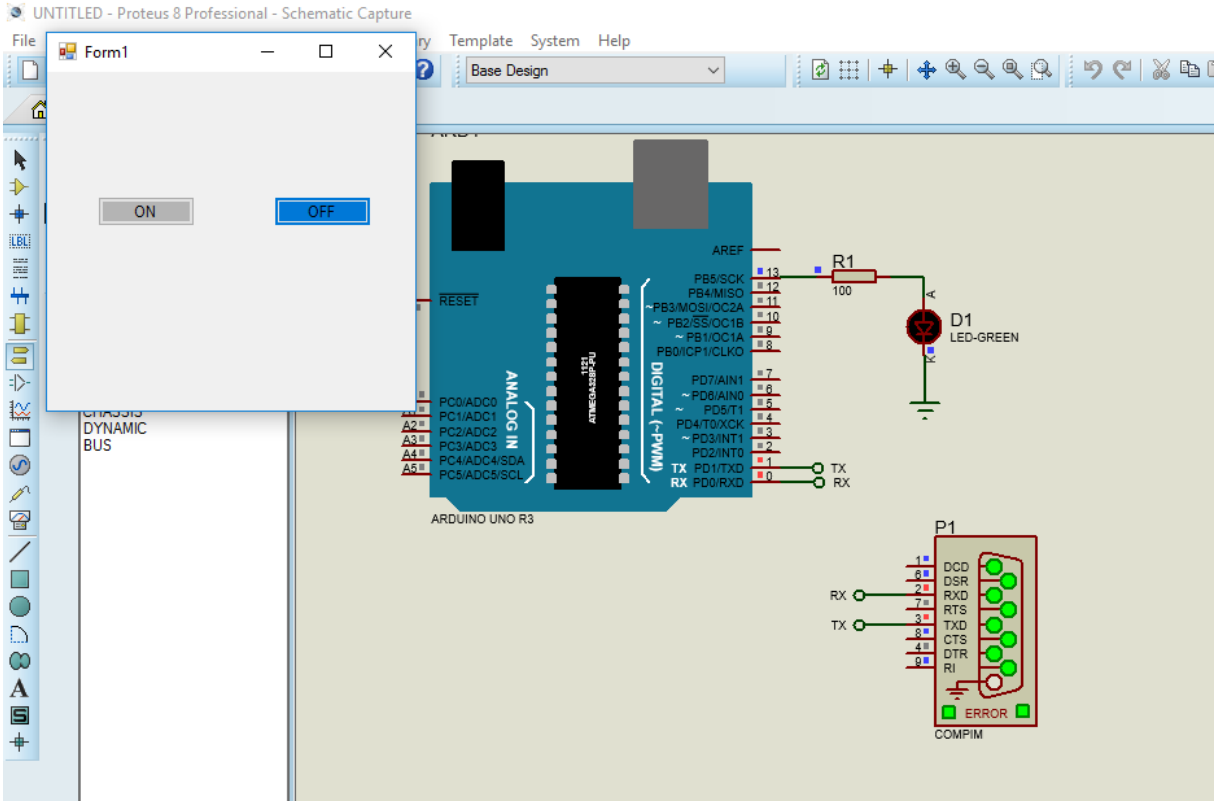


Enfin on peut simuler en lançant PROTEUS puis l'application que nous venons de créer :



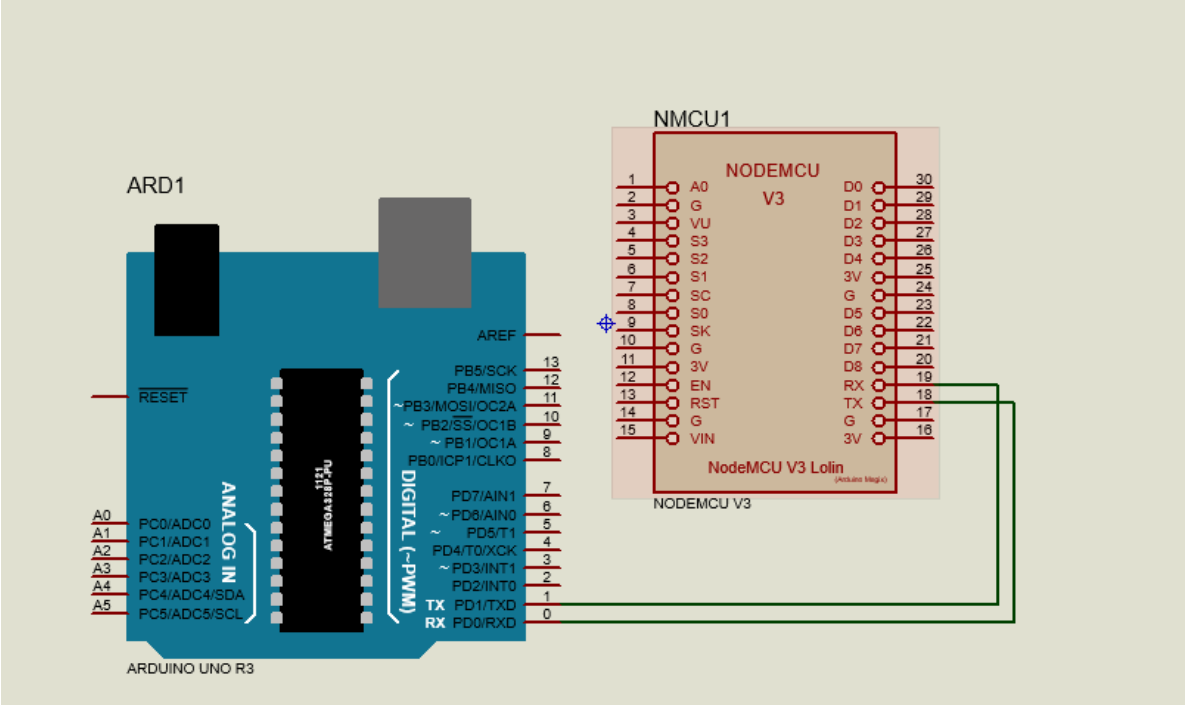
Ici en appuie sur le bouton ON la lampe s’allume

En appuyant sur le bouton OFF la lampe s’éteint



CHAPITRE 4 COMMUNICATION ARDUINO ET NODEMCU

Dans cette présente partie nous allons traiter sur la communication de la carte ARDUINO et le NODEMCU.
Dans certains projet il devient évident de faire communiquer la carte arduino et le NODEMCU dans cette partie du livre nous allons faire une communication série UART de la carte ARDUINO et le NODEMCU.



Comme vous pouvez voir ici on câble le montage de sorte que le RX de la carte ARDUINO puisse aller vers le TX de la carte NODEMCU et le TX de la carte ARDUINO puisse aller vers le TX du NODEMCU. C'est la cablage de base de la communication série UART.
On va un code qui envoie bonjour quand on appuie sur un bouton de l'ARDUINO.

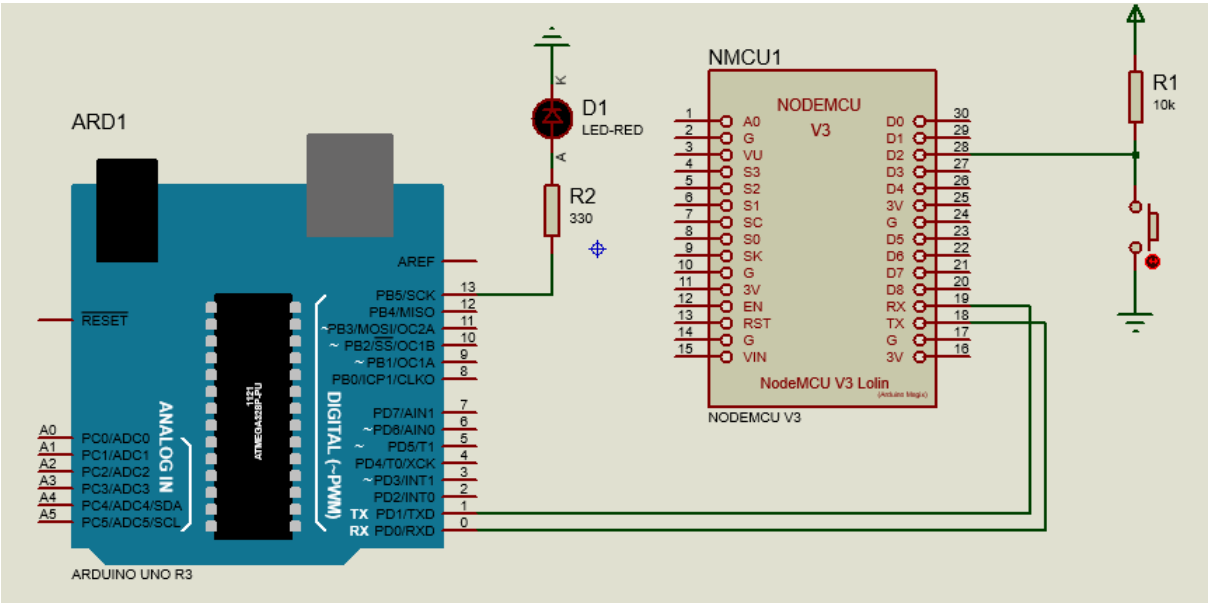
CODE ARDUINO

```
String info;
void setup() {
 Serial.begin(9600);
 pinMode(13,OUTPUT);
 Serial.print ("bonjour");
}

void loop() {
 if(Serial.available()>0){
     info = Serial.readString();
 }
  if (info=="ALLUMER"){
     digitalWrite(13,HIGH);
  }
   if(info=="ETEINDRE"){
     digitalWrite(13,LOW);
   }
}
```

CODE NODEMCU

```
String info;
void setup() {
 Serial.begin(9600);
 Serial.println("code test");
 pinMode(D2,INPUT);
}

void loop() {
  if(digitalRead(D2)==LOW){
     Serial.print("ALLUMER");
  } else {
     Serial.print("ETEINDRE");
  }
}
```

SCHEMA ELECTRONIQUE GENERAL



Sur ce schéma on voit une communication UART entre ARDUINO et NODEMCU
Quand on appuie sur le bouton poussoir le NODEMCU envoie à l'ARDUINO une information ici dans notre cas il s'agit de la chaine de caractères « ALLUMER » l'arduino reçoit cette information puis allume la led se trouvant sur la borne 13 de la carte ARDUINO.
Quand on n'appuie pas sur le bouton poussoir le NODEMCU envoie la chaine de caractères « ETEINDRE » puis l'envoie à la carte ARDUINO au travers la communication série puis l'ARDUINO à son tour éteint la led se trouvant à la borne 13 de la carte ARDUINO.
NB : il est possible de faire aussi cette communication en utilisant la communication série virtuelle côté ARDUINO et côté NODEMCU.