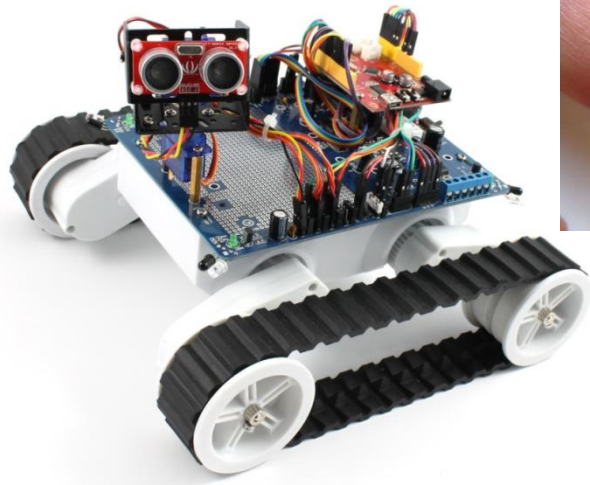
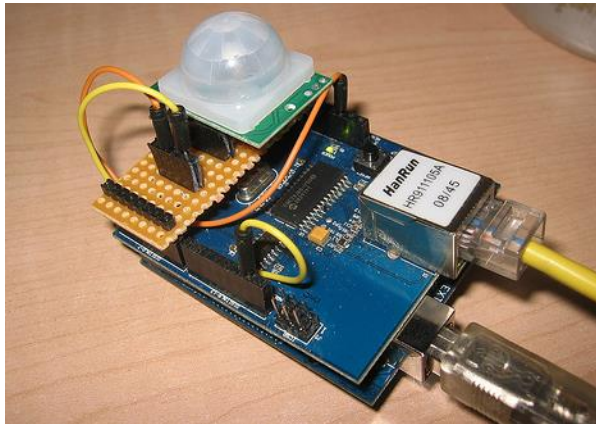


Electronique avec Arduino

Pascal MASSON
(*pascal.masson@unice.fr*)



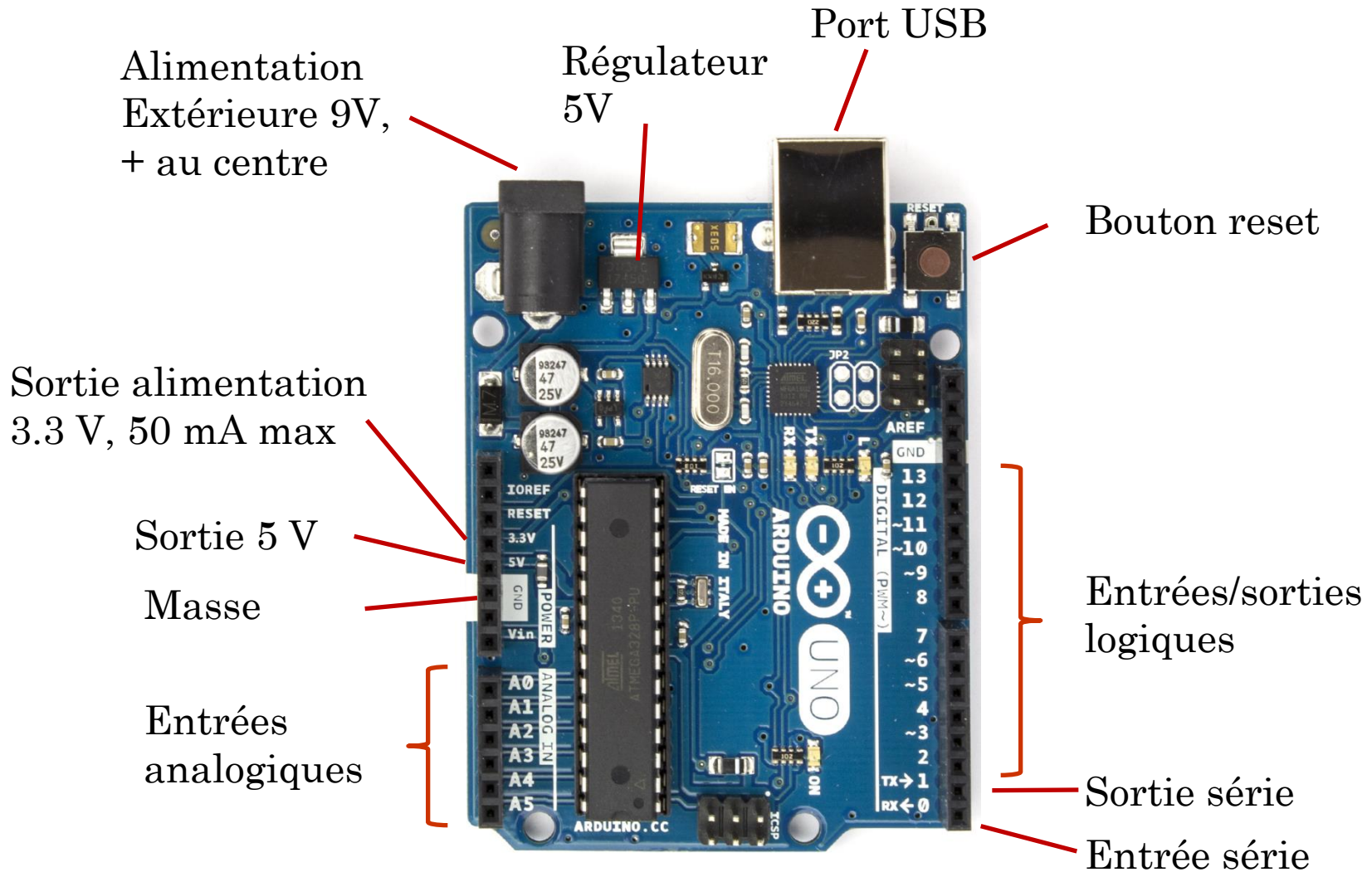
*Version projection
Edition 2015-2016-V32*

*Cours sponsorisé
par la société*

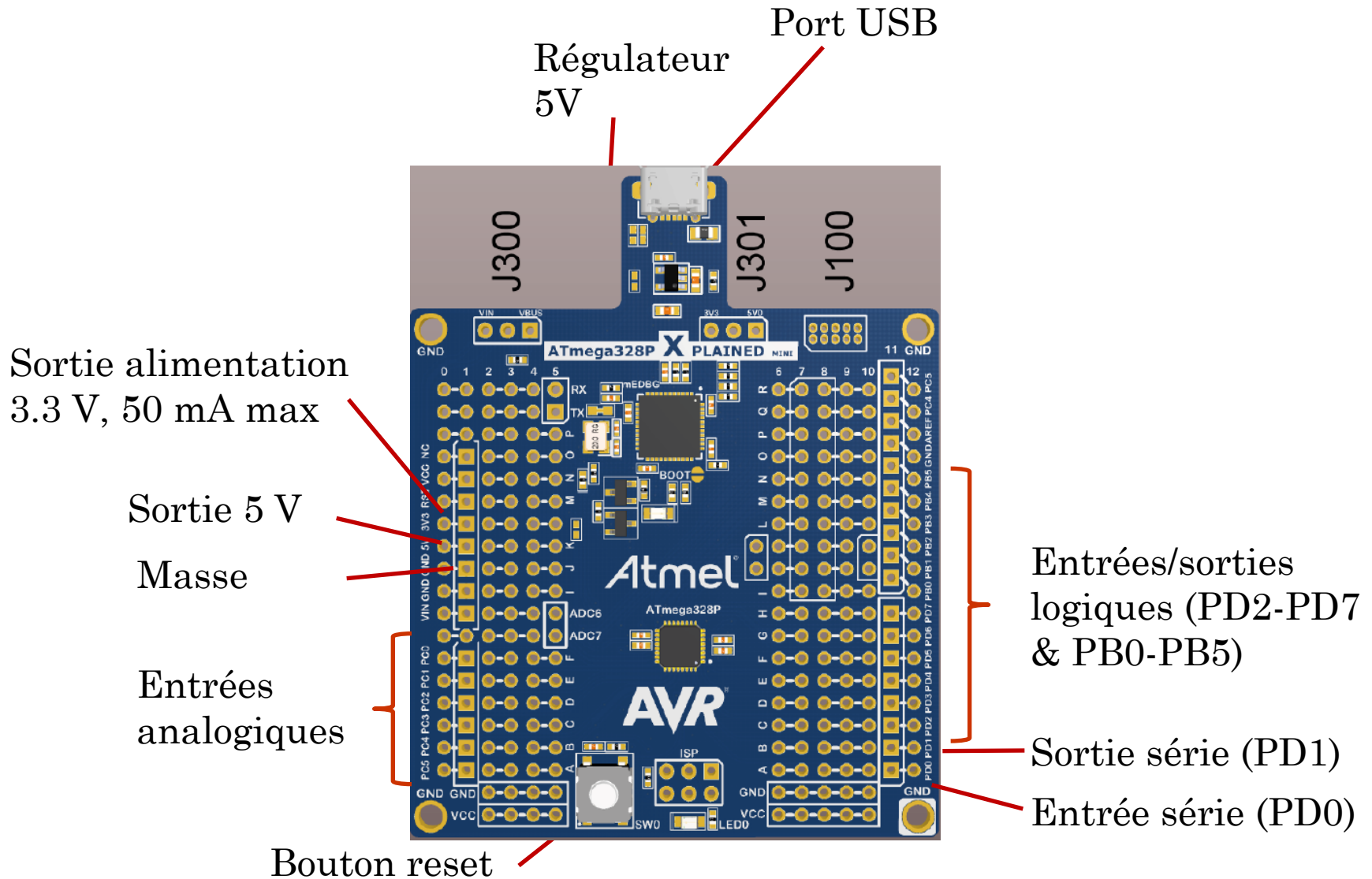


1. Généralités
2. Commande d'une LED
3. LED + bouton poussoir
4. LEDs + entrée analogique
5. Voltmètre avec LCD
6. Luminosité d'une LED
7. Mesure de distance
8. Communications IR

1.1. Présentation de la carte ARDUINO UNO



1.1. Présentation de la carte Xplained MINI



1.1. Présentation de la carte Xplained MINI

- Pour accéder à cette carte il faut :

1. Un câble USB micro
2. Installer le logiciel « driver-atmel-bundle-7.0.712.exe »

que vous trouverez sur mon site :

<http://users.polytech.unice.fr/~pmasson/Enseignement-arduino.htm>

sous l'intitulé :

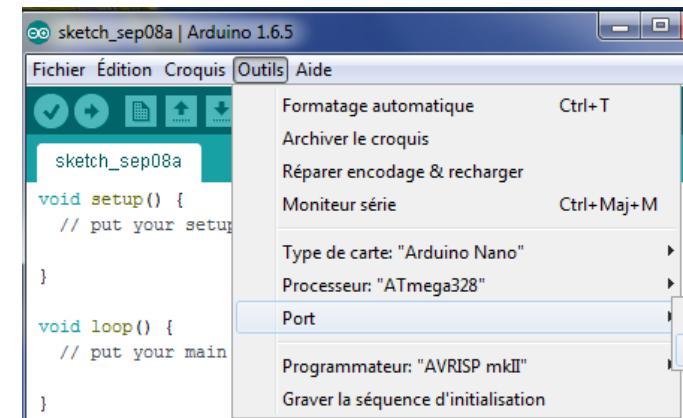
[ATMEL USB driver pour pouvoir utiliser la carte Xplained mini](#)



- Dans le logiciel IDE il faut sélectionner :

1. Une carte de type « Arduino Nano »
2. Un processeur de type : ATmega328

- Très important, un numéro de port doit apparaître dans l'arborescence Outil/Port



1.1. Présentation de la carte Xplained MINI

- Pour accéder à cette carte il faut :

1. Un câble USB micro
2. Installer le logiciel « driver-atmel-bundle-7.0.712.exe »

que vous trouverez sur mon site :

<http://users.polytech.unice.fr/~pmasson/Enseignement-arduino.htm>

sous l'intitulé :

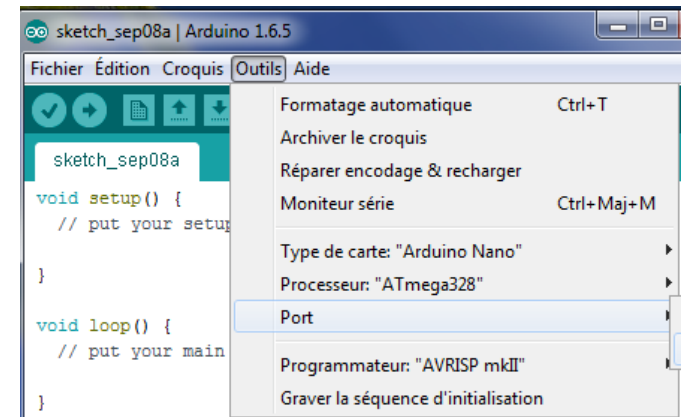
[ATMEL USB driver pour pouvoir utiliser la carte Xplained mini](#)



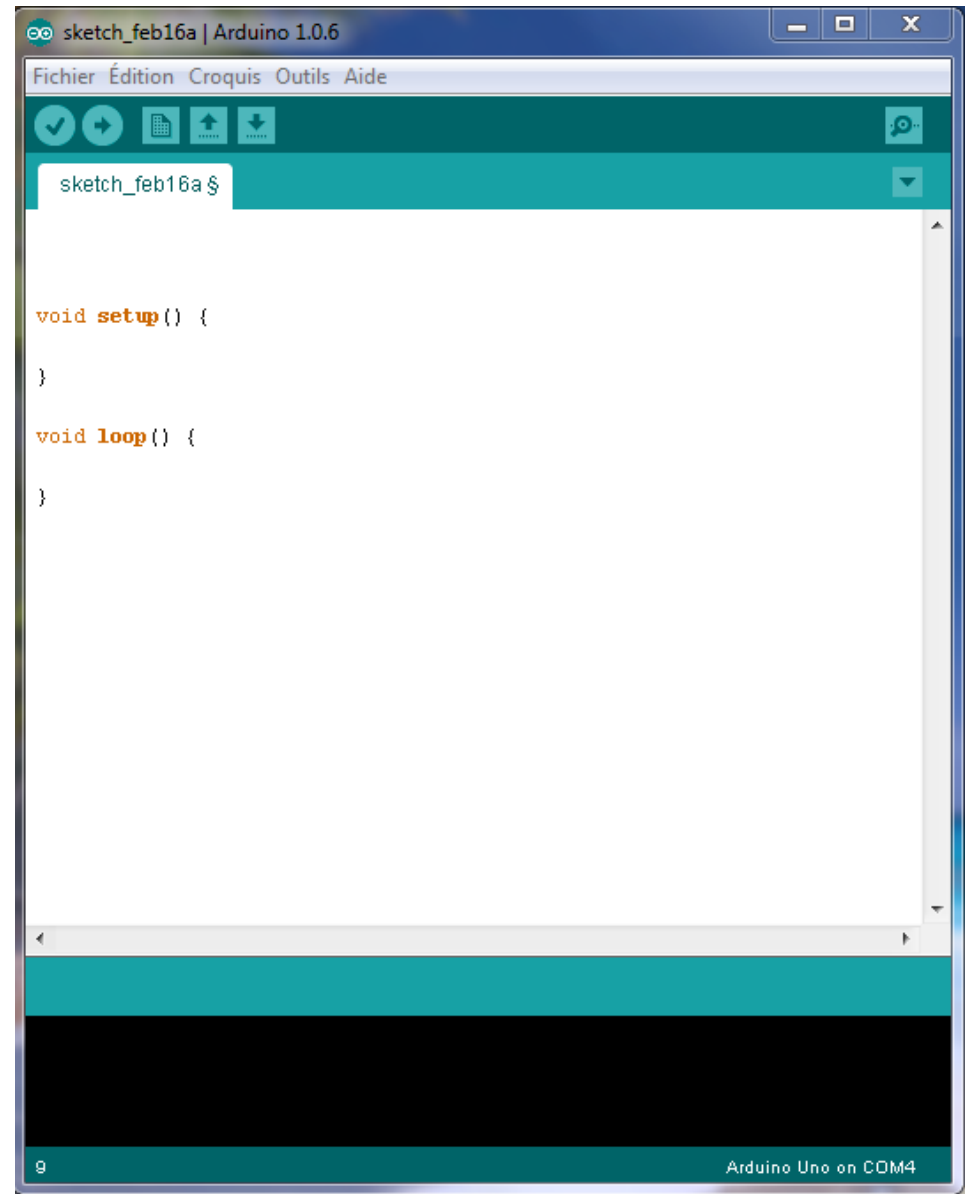
- Dans le logiciel IDE il faut sélectionner :

1. Une carte de type « Arduino Nano »
2. Un processeur de type : ATmega328

- Très important, un numéro de port doit apparaître dans l'arborescence Outil/Port



1.3. Présentation du logiciel



<http://arduino.cc/en/Main/Software>

2.1. Objectifs

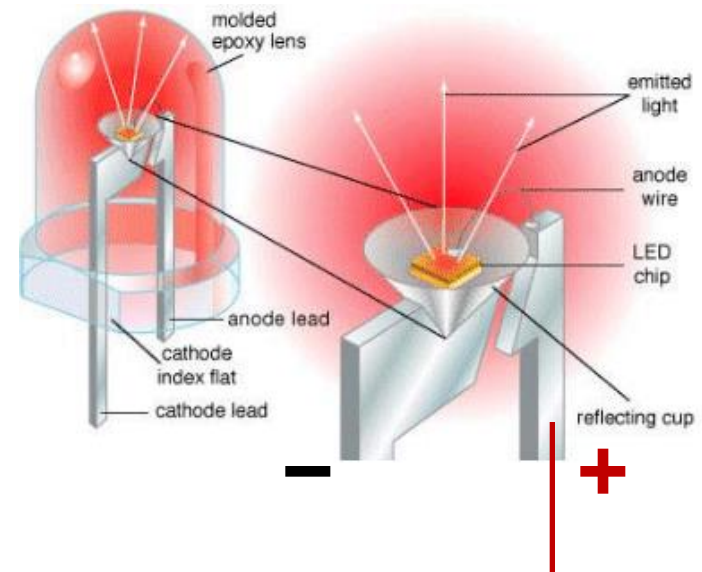
- 1) Faire clignoter une LED
- 2) Apprendre à réaliser un montage
- 3) Apprendre à écrire un programme simple en langage arduino
- 4) Apprendre à dépanner le montage et le programme

2.2. Montage

□ Rappel sur la diode

- Une diode ne laisse passer le courant que dans un sens. Elle peut aussi recevoir ou émettre de la lumière

Color	Material	Bandgap	V_r
Blue	SiC	2.64 eV	3.2–4.9
Green	GaP	2.19 eV	2.2–2.5
Yellow	GaP ₈₅ As ₁₅	2.11 eV	2.1–2.5
Orange	GaP ₆₅ As ₃₅	2.03 eV	1.9–2.2
Red	GaP ₄ As ₆	1.91 eV	1.7–2.7

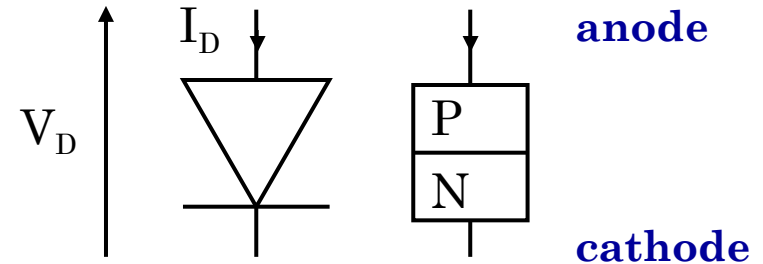


Patte plus longue

2.2. Montage

□ Rappel sur la diode

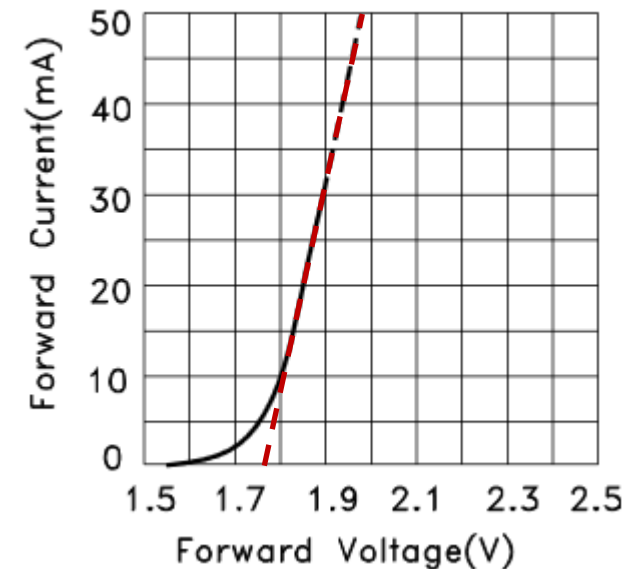
- Vue schématique :



- Modélisation simple de la diode : $\left\{ \begin{array}{l} \text{Diode bloquée : } V_D < V_S \text{ et } I_D = 0 \\ \text{Diode passante : } V_D > V_S \text{ et } V_D = V_S + R_S \cdot I_D \end{array} \right.$

- Exemple de courbe :

avec : $V_S = 1.77 \text{ V}$
 $R_S = \Delta V_D / \Delta I_D = 4.4 \Omega$



2.2. Montage

□ Polarisation de la LED

- On constate que pour la LED rouge, que si on fait passé un courant de 20 mA la tension à ses bornes vaut typiquement 2 V

ITEMS	Color	Symbol	Condition	Min.	Typ.	Max.	Unit
Forward Voltage	Red	V_F	$I_F=20\text{mA}$	1.8	2.0	2.2	V
	Green			3.0	3.2	3.4	
	Blue			3.0	3.2	3.4	
Luminous Intensity	Red	I_v	$I_F=20\text{mA}$	— — —	— — —	800	mcd
	Green			— — —	— — —	4000	
	Blue			— — —	— — —	900	
Wavelength	Red	$\Delta \lambda$	$I_F=20\text{mA}$	620	623	625	nm
	Green			515	517.5	520	
	Blue			465	466	467.5	
Light Degradation after 1000 hours	Red	-4.68% ~ -8.27%					
	Green	-11.37% ~ -15.30%					
	Blue	-8.23% ~ -16.81%					

2.2. Montage

□ Polarisation de la LED

- La résistance série de la LED étant de l'ordre de $4\ \Omega$, la chute de tension a ses bornes est d'environ 80 mV (pour $I_D = 20\text{ mA}$)
- On peut donc faire raisonnablement l'approximation que $V_S = 2\text{ V}$
- Cette approximation est raisonnable car la tension aux bornes de la diode est en faite comprise entre 1.8 et 2.2 V ce qui implique une marge d'erreur de 0.4 V
- Les sorties de l'arduino donnent une tension de 0 V (0 logique) ou de 5 V (1 logique) et si on branche directement la diode elle laissera passer un courant de :

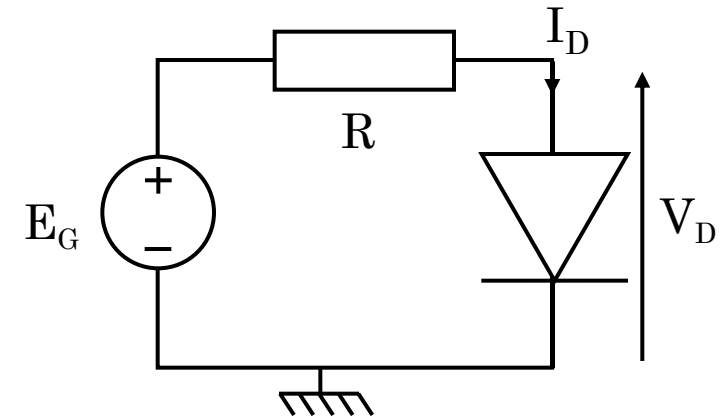
$$E_G = V_S + R_S.I_D \quad I_D = \frac{5 - 1.77}{4.4} \approx 730\text{mA}$$

- Ce courant grille la diode
- **Donc on place toujours une résistance de protection en série de la diode**

2.2. Montage

□ Choix de la résistance

- Le schéma électrique se résume à celui étudié au début du cours sur la diode en PeiP1 où E_G représente une des sorties de l'arduino



- Il faut déterminer la valeur de la résistance R et pour cela on peut se placer dans les conditions typiques indiquées par le constructeur soit $I_D = 20 \text{ mA}$ et $V_D = 2 \text{ V}$

$$R = \frac{E_G - V_D}{I_D} = 150\Omega$$

- Pour les valeurs extrêmes on trouve que R est compris entre 140Ω et 160Ω .

2.2. Montage

□ Choix de la résistance

- Il n'existe pas toutes les valeurs possibles de résistances et il faut en choisir une qui s'approche : 120, **150**, 180, 220 Ω

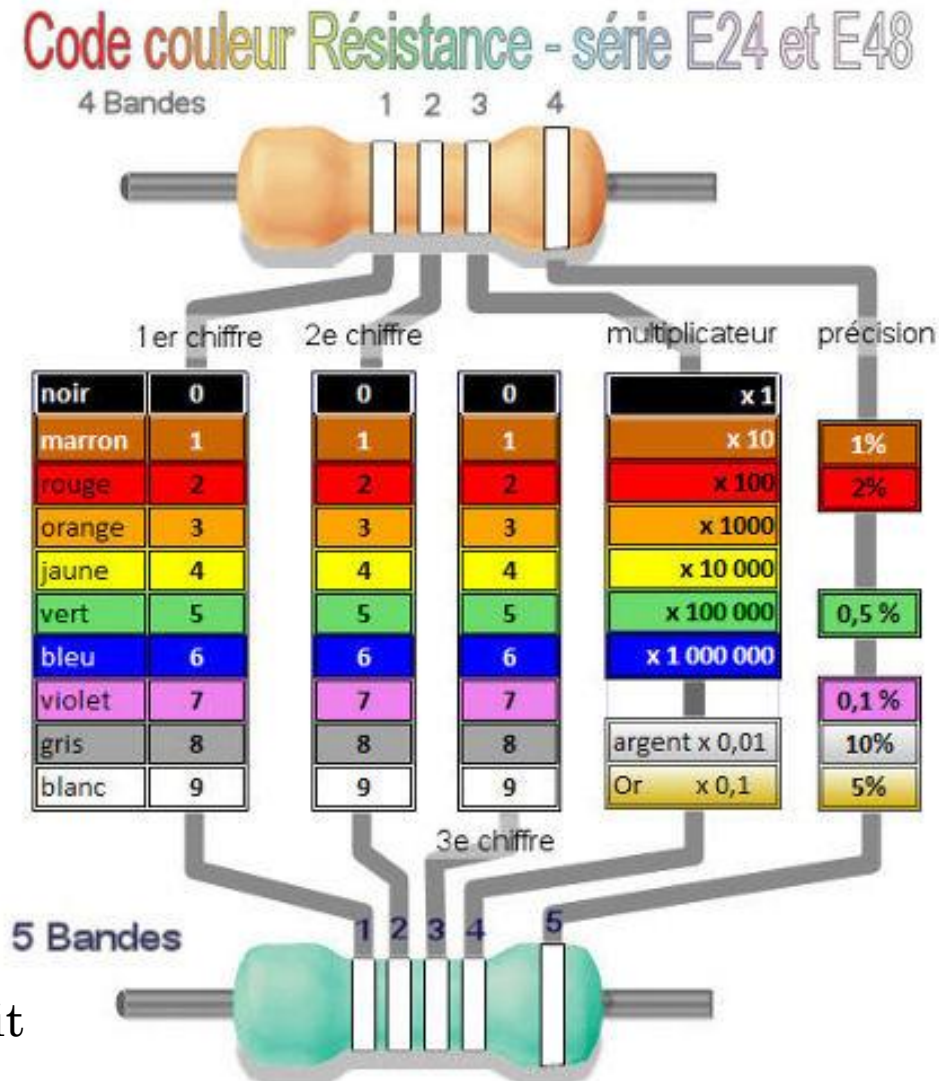
- Donc pour 150 Ω il faut le code : marron / vert / marron



- Il faut aussi vérifier la puissance à dissiper

$$P = R.I_D^2 = 0.06W$$

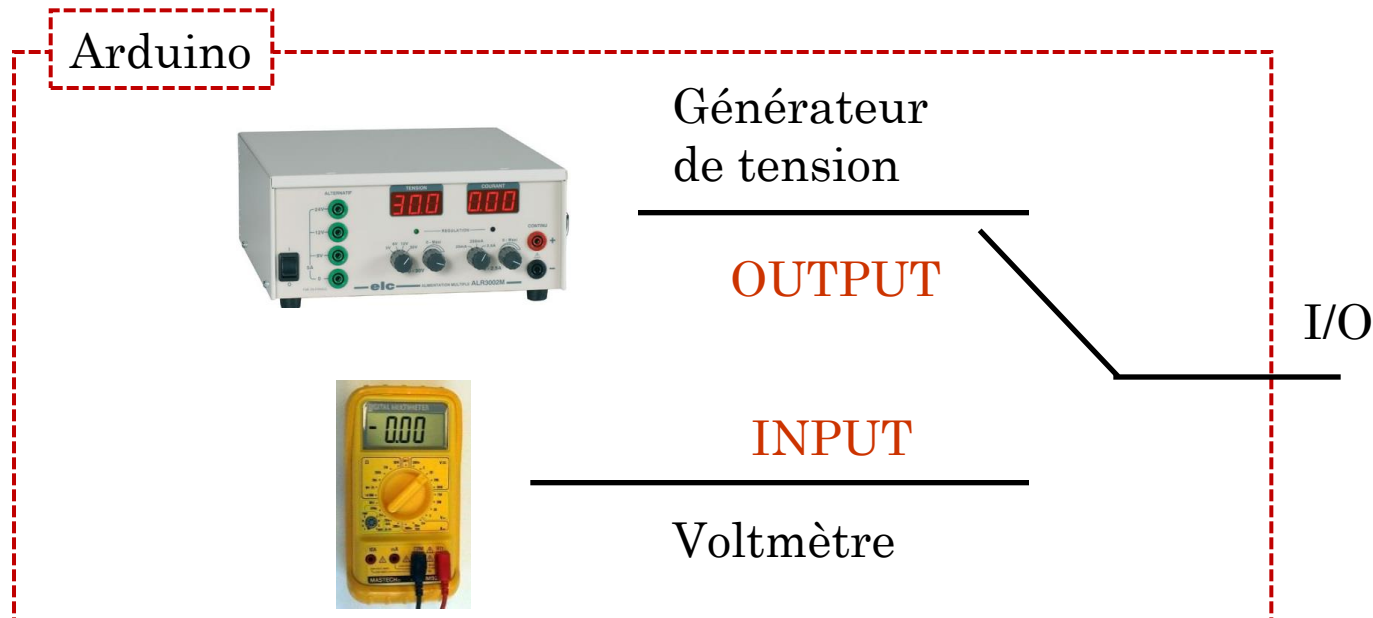
- Donc une résistance 1/4 de Watt suffit



2.2. Montage

❑ Sortie/entrée numérique de l'arduino

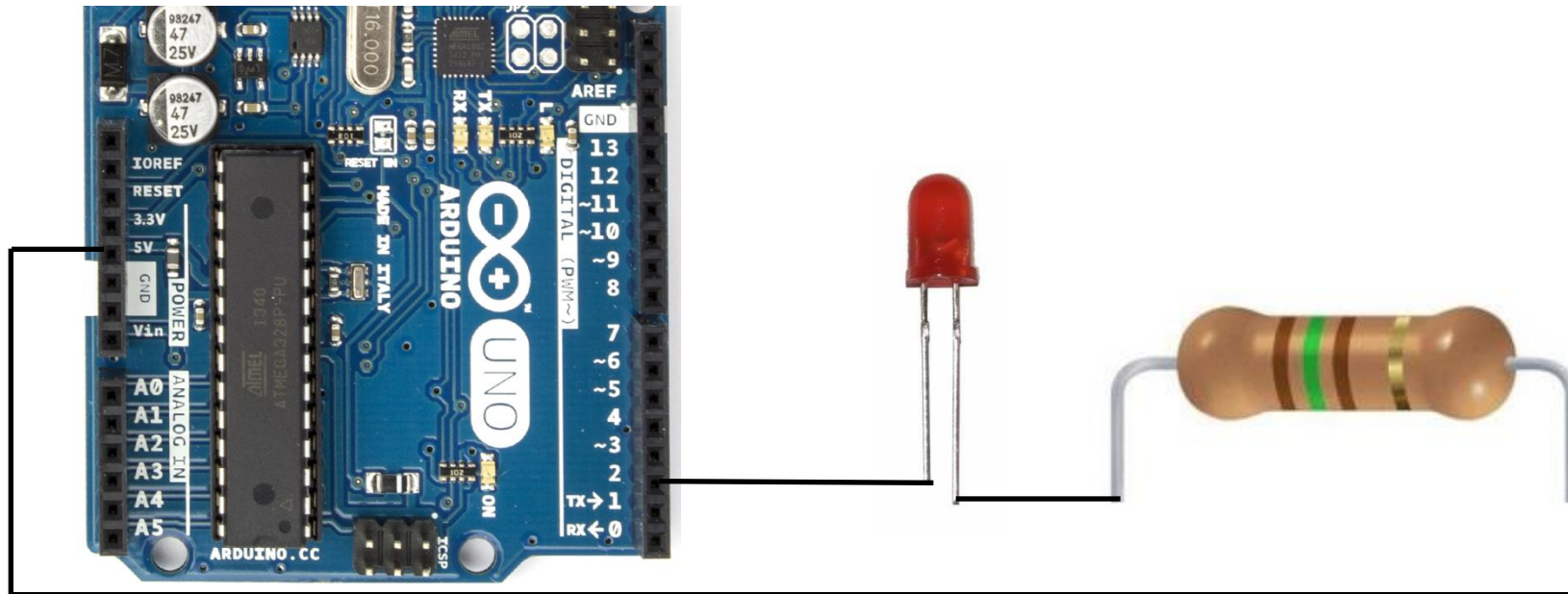
- Les sorties numériques de l'arduino peuvent aussi être configurées comme des entrées (I/O) et cela devra être précisé lors de l'écriture du programme.
- Si l'I/O est configurée comme une sortie, un « interrupteur » bascule dans l'arduino et sélectionne un générateur de tension (0 et 5 V) sinon l'interrupteur sélectionne un voltmètre qui ne lie que les tensions 0 et 5 V.



2.2. Montage

❑ Sortie/entrée numérique de l'arduino

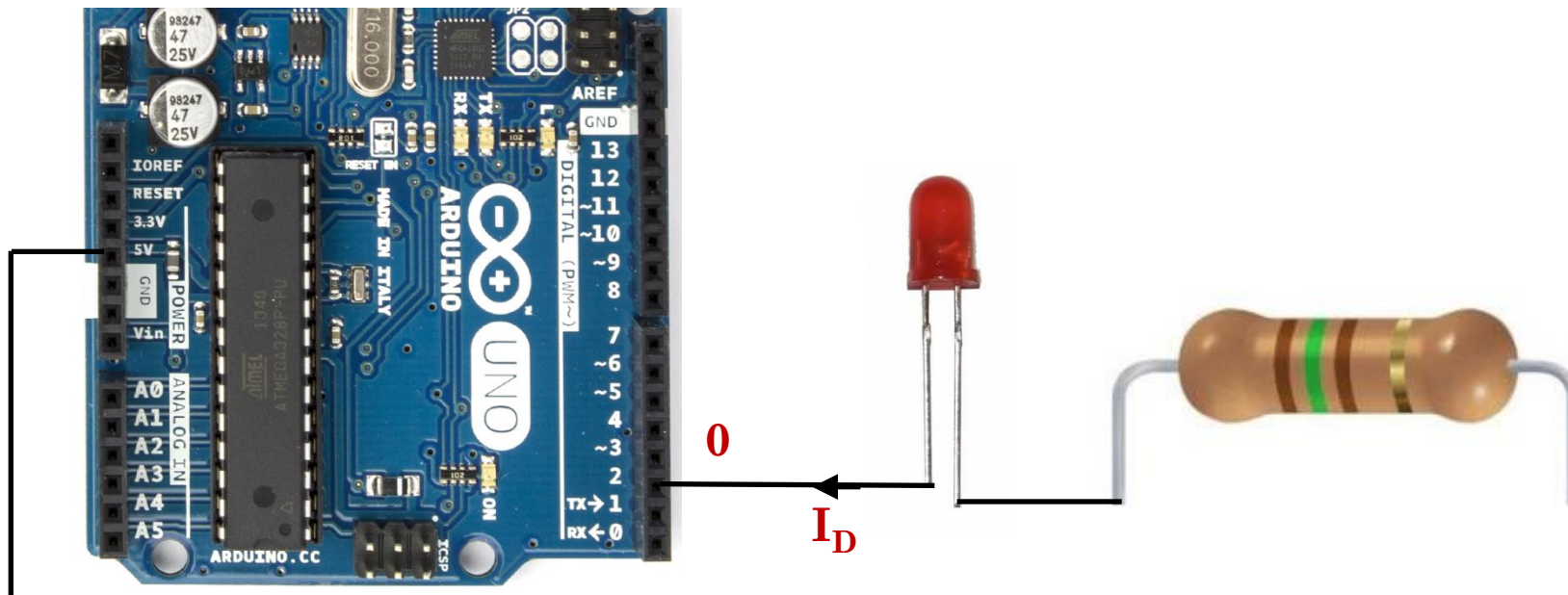
- Les sorties peuvent délivrer ou absorber du courant mais il est préférable de ne pas demander de courant au micro-contrôleur. Une I/O peut délivrer 40 mA
- On choisit (arbitrairement) l'I/O n°2



2.2. Montage

□ Sortie/entrée numérique de l'arduino

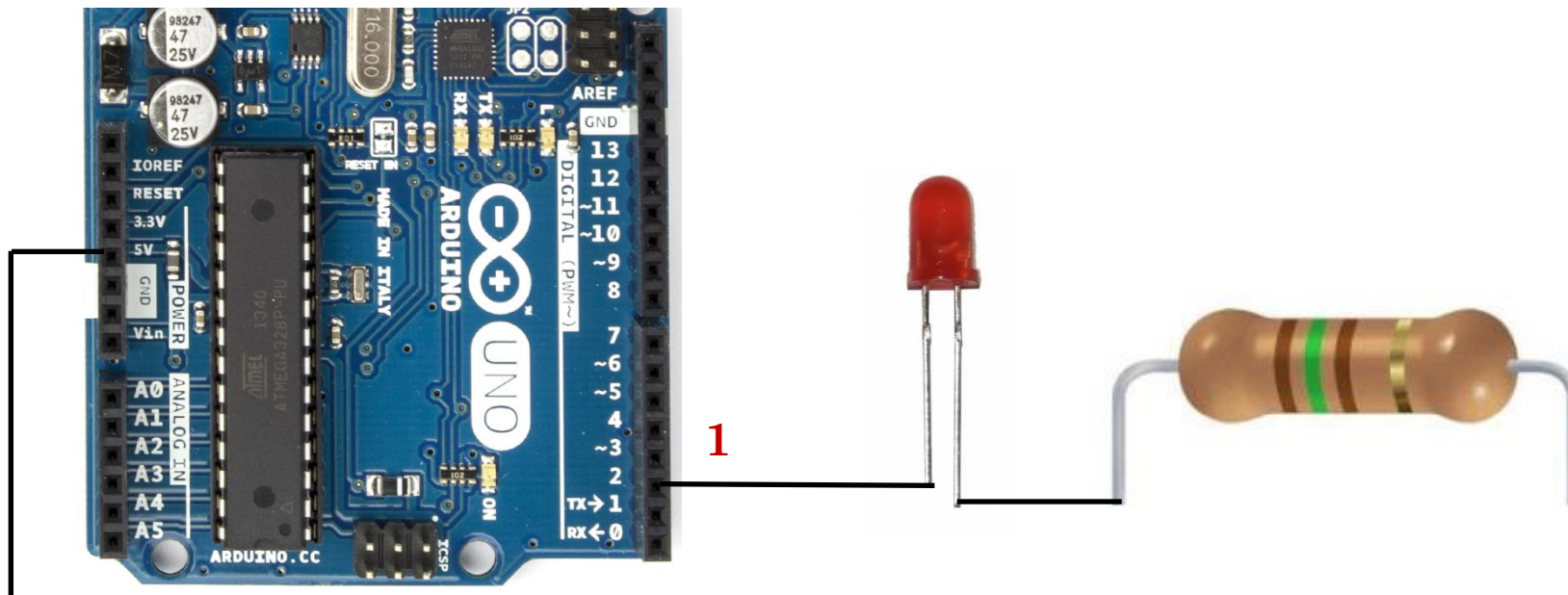
- Si la sortie n°2 est à 0, alors un courant circule et la diode s'allume



2.2. Montage

□ Sortie/entrée numérique de l'arduino

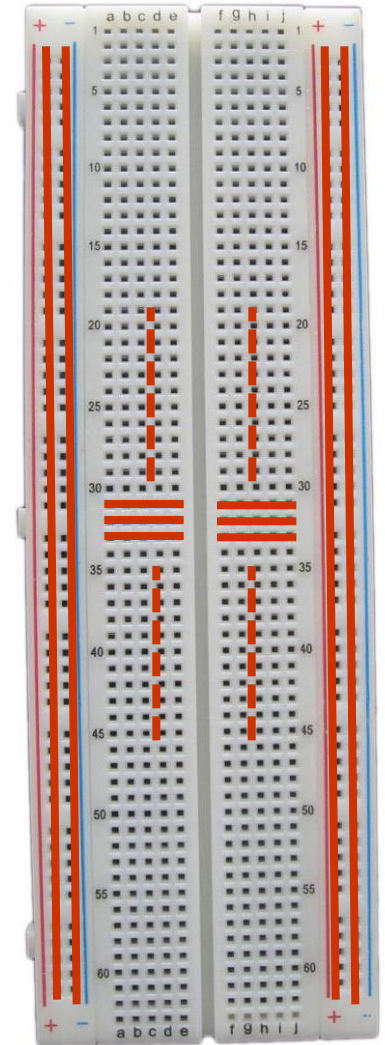
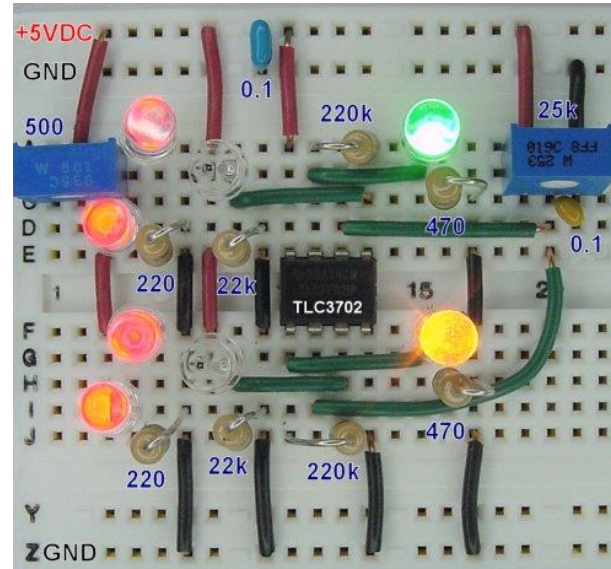
- Si la sortie n°2 est à 1 (donc 5 V), alors aucun courant circule et la diode est éteinte



2.2. Montage

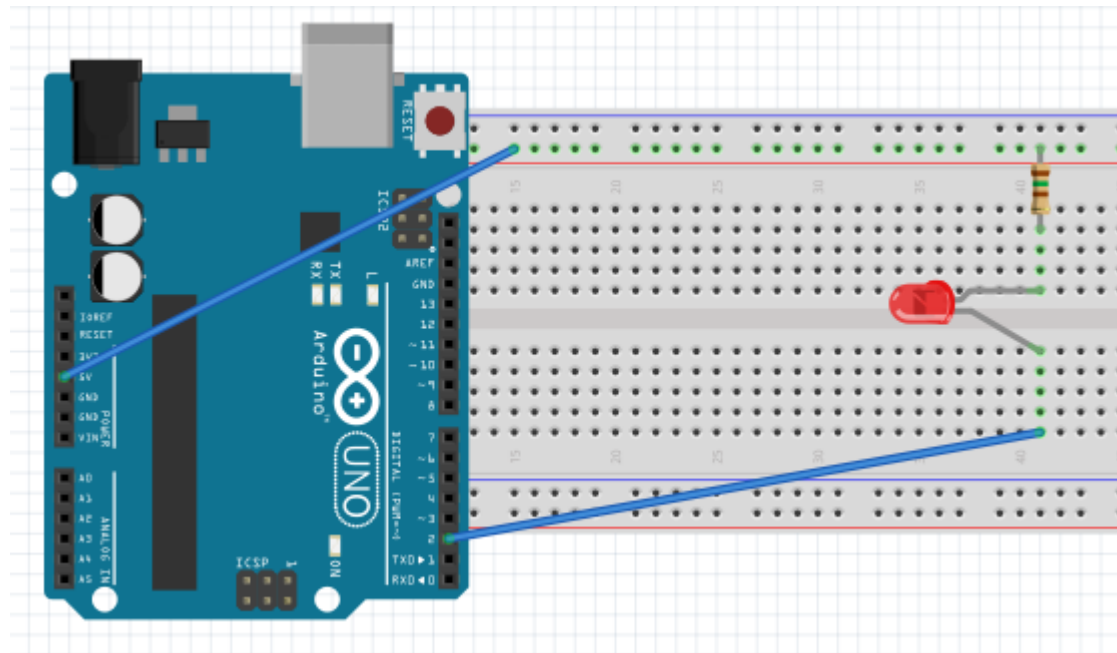
□ Circuit sur la plaque de test

- La plaque utilisée dans ce cours comporte 830 points de connexion dans lesquels on enfiche les pattes des composants.
- Une partie des points de connexions sont reliés entre eux
- Les montages que vous allez réaliser devront être soigné en sans risque de court circuit.



2.2. Montage

□ Montage final



2.3. Ecriture du programme

❑ A ne pas oublier

// indique une ligne de commentaire

; indique la fin d'une instruction

❑ Structure d'un programme

// Nom du programme et description de ce qu'il fait

// Déclaration des constantes et variables

void setup() { //début du programme

} // fin de définition du début de programme avec les variables

void loop() { // écriture de ce que fait le programme

} // indique la fin du programme qui recommencera au début

2.3. Ecriture du programme

▣ Les nouvelles fonctions

- **const** : on définit une « variable » qui est en fait « constante »
- **int** : correspond à une variable de type « entier » dont la valeur est comprise entre – 2147483648 et + 2147483647
- **pinMode (X,Y)** : permet de configurer une entrée/sortie (I/O) en sortie ou en entrée. X correspond au numéro de l'I/O (2, 3, 4) et Y au mode : **INPUT** ou **OUTPUT**
- **digitalWrite (X,Y)** : permet d'affecter la valeur 0 ou 1 à la sortie numérique X, Y prend la valeur **LOW** ou **HIGH**
- **delay(X)** : permet d'arrêter le déroulement du programme durant X millisecondes

2.3. Ecriture du programme

□ Programme final

//LED qui clignote

const int led_rouge=2; //on définit une constante de type entier

void setup(){ //début du programme

pinMode(led_rouge, **OUTPUT**); // l'I/O 2 est utilisée comme une sortie

} // fin de définition du début de programme avec les variables

void loop() { // écriture de ce que fait le programme

digitalWrite(led_rouge, **LOW**); // mettre la sortie 2 à l'état bas

delay(1000); // ne rien faire pendant 1000 ms = 1s

digitalWrite(led_rouge, **HIGH**); // mettre la sortie 2 à l'état haut

delay(200); //attendre à nouveau

} // indique la fin du programme qui recommencera au début

2.3. Ecriture du programme

□ Programme final

- Il n'est pas obligatoire de définir une variable `led_rouge` et on peut directement écrire `digitalWrite(2, LOW);`
- Par contre si vous écrivez un programme avec plusieurs centaines de lignes et que vous décidez de changer de sortie, il faudra changer le numéro dans le programme à chaque apparition

2.3. Ecriture du programme

□ Envoi du programme à l'arduino

Compilation

Compilation
+ envoi à l'arduino

Rapport de fin de
compilation et
d'envoi à l'arduino

The screenshot shows the Arduino IDE interface. The title bar indicates 'LED_qui_clignote_V2 | Arduino 1.0.6'. The menu bar includes 'Fichier', 'Édition', 'Croquis', 'Outils', and 'Aide'. The toolbar contains icons for opening, saving, and uploading. The main text area displays the following code:

```
//LED qui clignote
const int led_rouge=2; //on définit une constante de type entier
void setup(){ //début du programme
    pinMode(led_rouge,OUTPUT); // l'I/O 2 est utilisée comme une sortie
} // fin de définition du début de programme avec les variables

void loop() { // écriture de ce que fait le programme
    digitalWrite(led_rouge,LOW); // mettre la sortie 2 à l'état haut
    delay(1000); // Ne rien faire pendant 1000 ms = 1s
    digitalWrite(led_rouge,HIGH); //mettre la sortie 2 à l'état bas
    delay(1000); //attendre à nouveau
} // indique la fin du programme qui recommencera au début
```

Below the code editor, a status bar shows 'Enregistrement terminé.' and 'Taille binaire du croquis : 1 082 octets (d'un max de 32 256 octets)'. The bottom status bar indicates '12' and 'Arduino Uno on COM4'.

2.4. Compilation

- L'ordinateur converti le programme en une succession d'instructions que l'arduino peut comprendre.

```
0000000 313a 3030 3030 3030 3030 3943 3634 3031
0000010 3030 3943 3734 3033 3030 3943 3734 3033
0000020 3030 3943 3734 3033 4230 0d36 3a0a 3031
0000030 3030 3031 3030 4330 3439 3337 3030 4330
0000040 3439 3337 3030 4330 3439 3337 3030 4330
0000050 3439 3337 3030 3439 0a0d 313a 3030 3230
0000060 3030 3030 3943 3734 3033 3030 3943 3734
0000070 3033 3030 3943 3734 3033 3030 3943 3734
0000080 3033 3830 0d34 3a0a 3031 3030 3033 3030
0000090 4330 3439 3337 3030 4330 3439 3337 3030
00000b0 3437 0a0d 313a 3030 3430 3030 3030 3943
00000c0 3134 3037 3031 3943 3734 3033 3030 3943
00000d0 3734 3033 3030 3943 3734 3033 4230 0d46
```

2.5. Déroulement du programme

- On voit la LED clignoter avec une période de 2 s.
- Vous pouvez modifier la durée des temps haut et bas
- Vous pouvez aussi changer d'I/O
- Le programme étant dans la mémoire ROM de l'arduino, si on débranche la carte et qu'on la rebranche, le programme recommence du début

2.5. Déroulement du programme

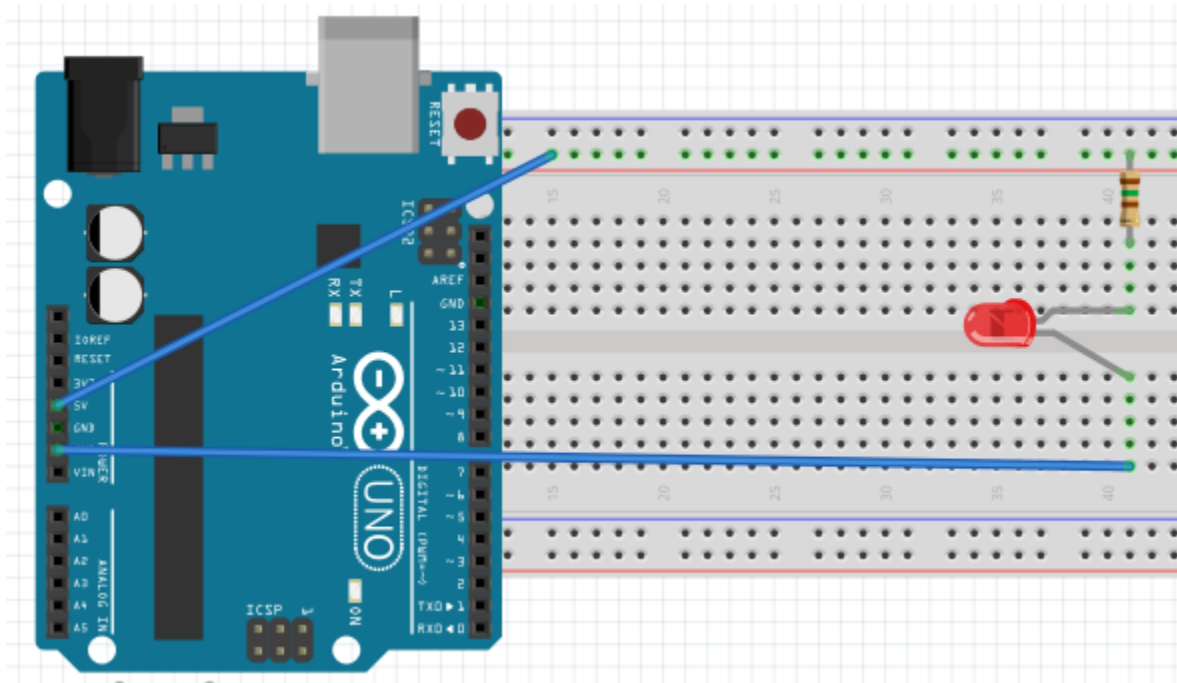
❑ Les pannes fréquentes

- Le montage ne présente des coupures du circuit au niveau de la plaquette (erreur de branchement)
- La diode est branchée à l'envers
- La diode a été branchée directement sur le 5 V Il faut la remplacer
- Les pattes des composants sont coupées trop court et ne touchent pas les pistes métalliques de la plaque de test

2.5. Déroulement du programme

❑ Les pannes fréquentes

- Pour vérifier si le montage fonctionne, il faut mettre 0 V sur l'entrée 2 et le moyen le plus simple (sans passer par le programme) est de prendre le fil relié à l'entrée 2 et de le mettre à la masse (GND)



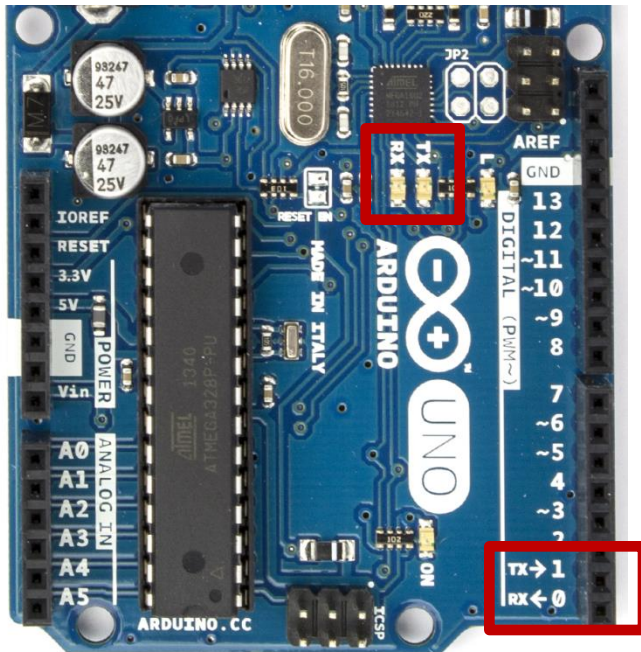
2.5. Déroulement du programme

❑ Les pannes fréquentes

- Branchement sur la deuxième sortie (i.e. sortie 1) et non sur la sortie 2
- Le programme n'a pas été téléversé
- Le programme présente des erreurs. Pour ce cas, nous allons demander à l'arduino d'envoyer des informations à l'ordinateur sur les étapes qu'il effectue

2.6. Communication par liaison série

- Il existe plusieurs moyens pour faire communiquer des systèmes électroniques entre eux et la carte arduino est équipée d'une liaison série (appelée aussi RS 232) de type full-duplex. Cela signifie que les 2 systèmes connectés peuvent parler et écouter en même temps.
- La liaison série nécessite 3 fils :



- _____ réception
 - _____ émission
 - _____ masse (qui sert de référence)
- L'échange d'informations est géré par un protocole qui définit le langage utilisé

2.6. Communication par liaison série

- Lorsque on demande l'envoi d'une information, elle est stockée en mémoire jusqu'à ce qu'elle puisse être envoyée
- Lorsque la carte arduino reçoit une information, elle est stockée en mémoire (buffer de réception) jusqu'à ce qu'on l'utilise. Les informations sont stockées dans l'ordre d'arrivée et on ne peut les lire que caractère par caractère.

2.6. Communication par liaison série

❑ Nouvelles fonctions

- **Serial.begin(X)** : définit la vitesse X (en bits par seconde) de transfert des données. Cette fonction doit apparaitre dans le setup du programme. Il existe plusieurs vitesse de transmission des données : 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 250000
- **serial.print(X)** : permet d'envoyer l'information X qui doit être entre " " si c'est du texte
- **serial.println(X)** : même action que print mais avec un retour à la ligne à la fin du message

2.7. Ecriture du programme

□ Programme final

//LED qui clignote

const int led_rouge=2; //on définit une constante de type entier

void setup(){ //début du programme

pinMode(led_rouge, **OUTPUT**); // l'I/O 2 est utilisée comme une sortie

Serial.begin(9600); // initialisation de la transmission série

} // fin de définition du début de programme avec les variables

void loop() { // écriture de ce que fait le programme

digitalWrite(led_rouge, **LOW**); // mettre la sortie 2 à l'état bas

Serial.println(" **LOW** "); //envoi de l'état de la diode à l'ordinateur

delay(1000); // ne rien faire pendant 1000 ms = 1s

digitalWrite(led_rouge, **HIGH**); // mettre la sortie 2 à l'état haut

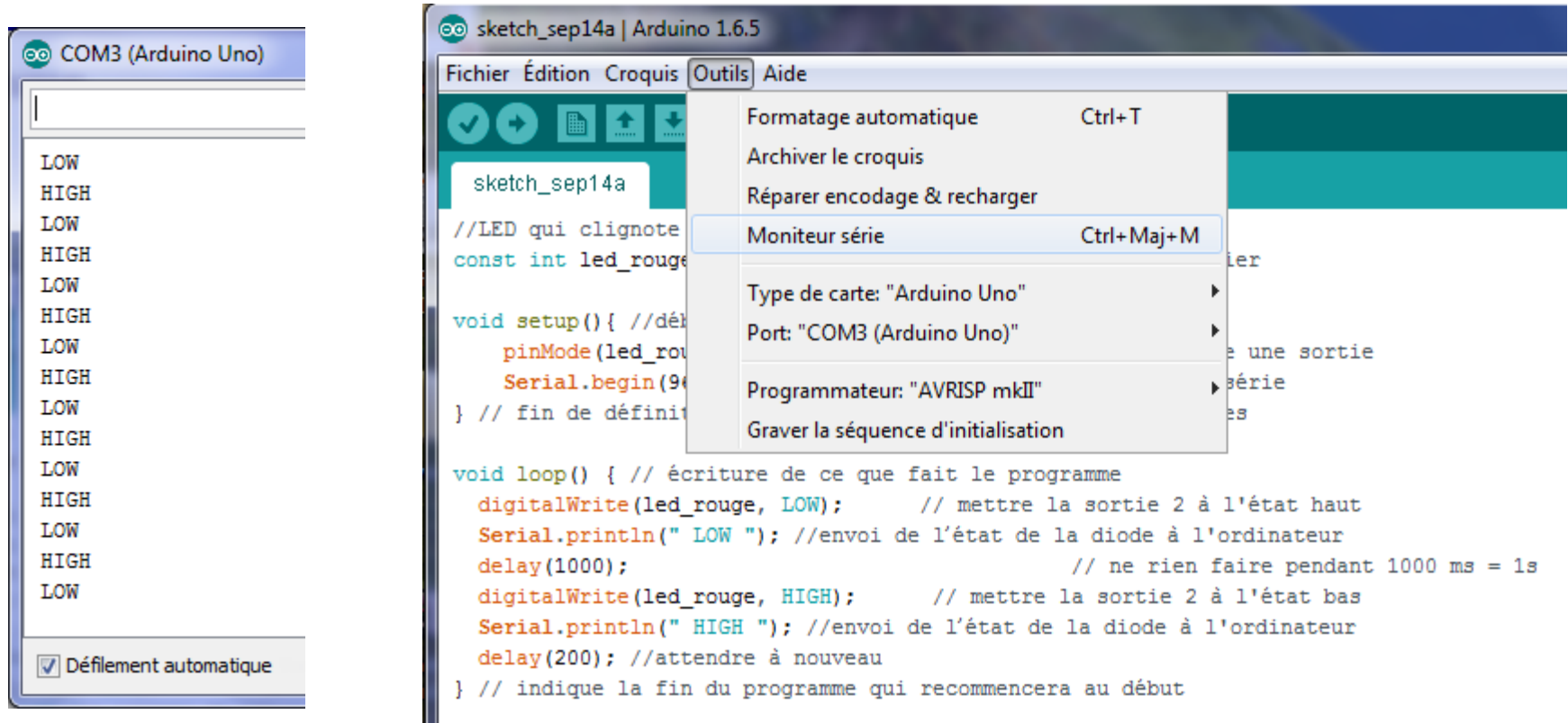
Serial.println(" **HIGH** "); //envoi de l'état de la diode à l'ordinateur

delay(200); //attendre à nouveau

} // indique la fin du programme qui recommencera au début

2.8. Déroulement du programme

- Ouvrir le moniteur série
- Vous devriez voir défiler l'état de la sortie 2
- La LED orange nommée TX (absente sur la carte Xplained) doit clignoter à chaque envoi d'information



3.1. Objectifs

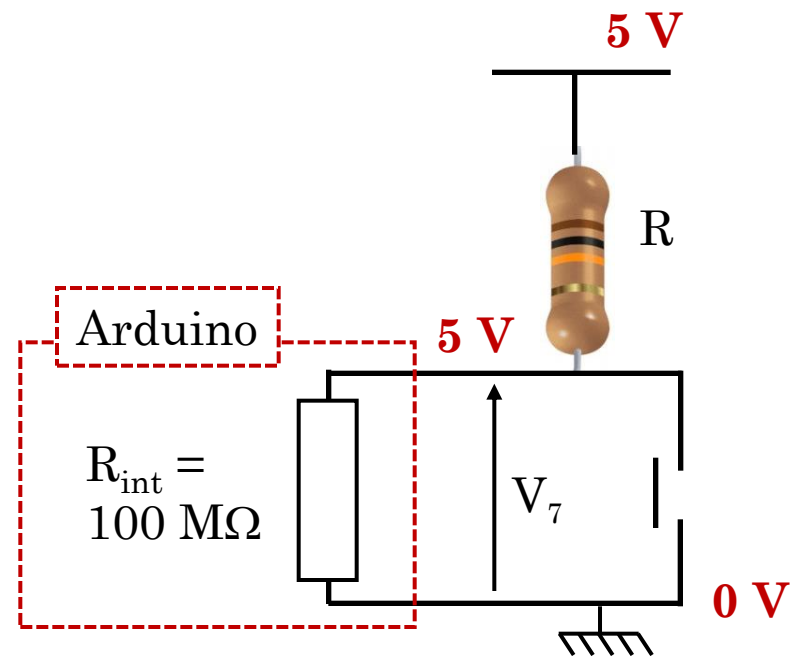
- 1) Allumer une LED lorsque l'on appuie sur un bouton et l'éteindre lorsqu'on relache le bouton
- 2) Allumer/éteindre une LED lorsque l'on appuie sur un bouton (mémorisation)

3.2. Montage

□ Prise en compte du bouton poussoir

- L'entrée I/O de l'arduino présente une impédance de $100\text{ M}\Omega$ et se comporte donc comme l'entrée d'un oscilloscope mais avec une résistance plus grand (oscilloscope = $1\text{ M}\Omega$)
- On considère ici l'I/O n°7
- Quelle est la valeur de la résistance ?
- Quand on appuie pas sur le bouton la valeur de la tension V_7 s'obtient avec un simple diviseur de tension

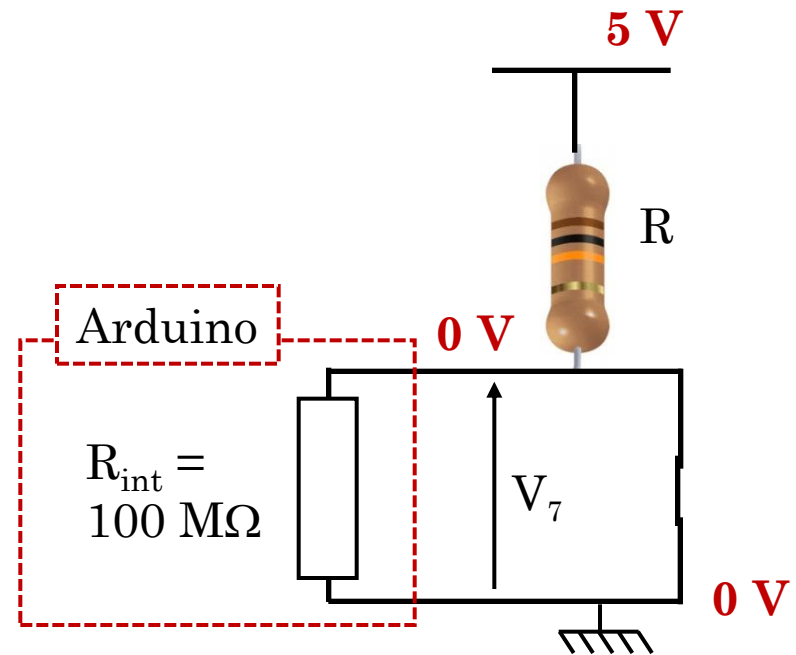
$$V_7 = \frac{R_{\text{int}}}{R + R_{\text{int}}} \cdot 5 \approx 5\text{V}$$



3.2. Montage

□ Prise en compte du bouton poussoir

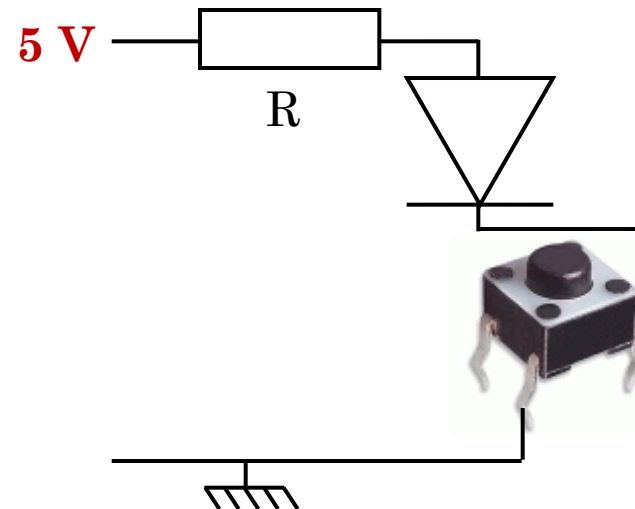
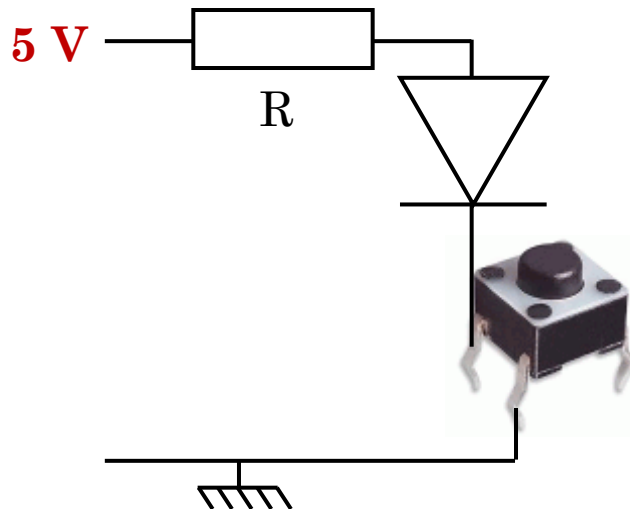
- Quand on appuie sur le bouton poussoir, on force V_7 à 0 V
- Le courant qui circule dans la résistance R est donc de 0.5 mA ce qui est faible par rapport au courant que peut délivrer le régulateur de tension qui donne le 5 V



3.2. Montage

□ Prise en compte du bouton poussoir

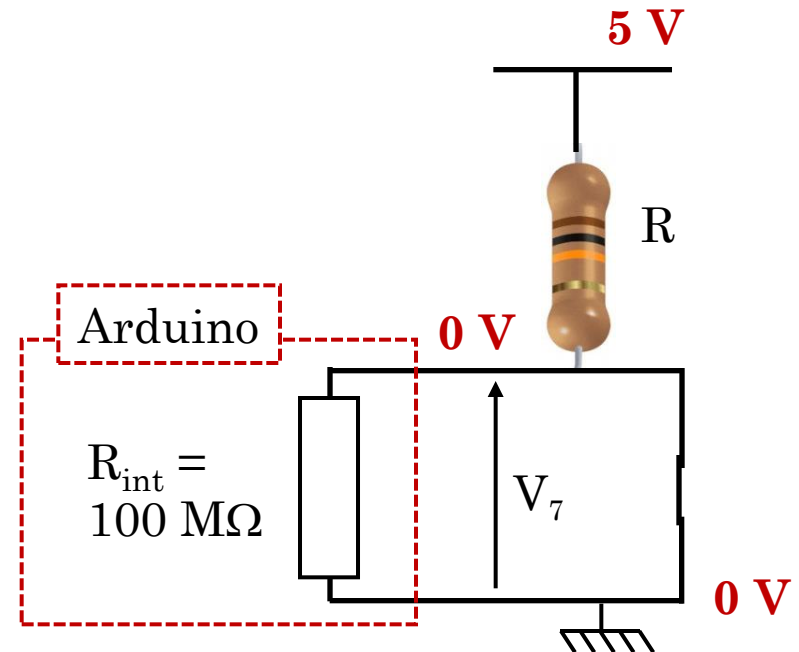
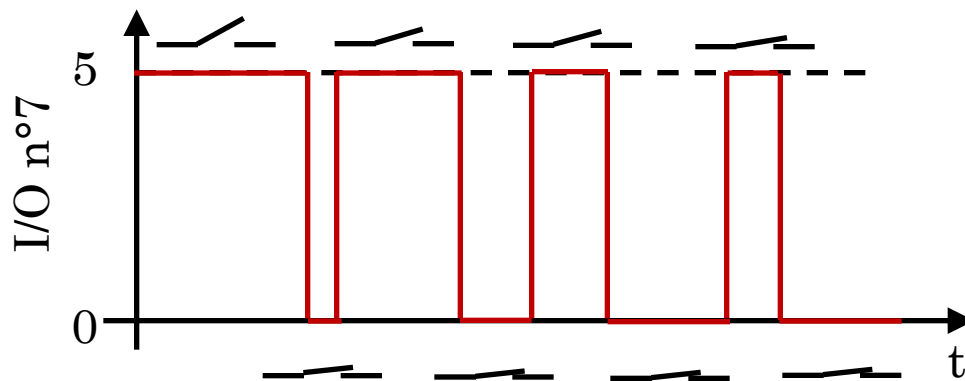
- Le bouton poussoir possède 4 pattes et on ne sait pas à priori où se trouve l'interrupteur entre ces pattes
- Un moyen simple pour le savoir est d'utiliser un Ohmmètre mais si on en a pas, on peut utiliser un circuit simple avec une diode en branchant 2 des pattes du bouton :



3.2. Montage

❑ Problème de rebond du bouton poussoir

- Un bouton poussoir ne passe pas « proprement » de l'état ouvert à l'état fermé.
- En effet, lorsque l'on appuie sur le bouton, les fils en métal que l'on met en contact vont rebondir l'un sur l'autre avant de se stabiliser. On obtient ainsi plusieurs fois la transition ouvert/fermé.



3.2. Montage

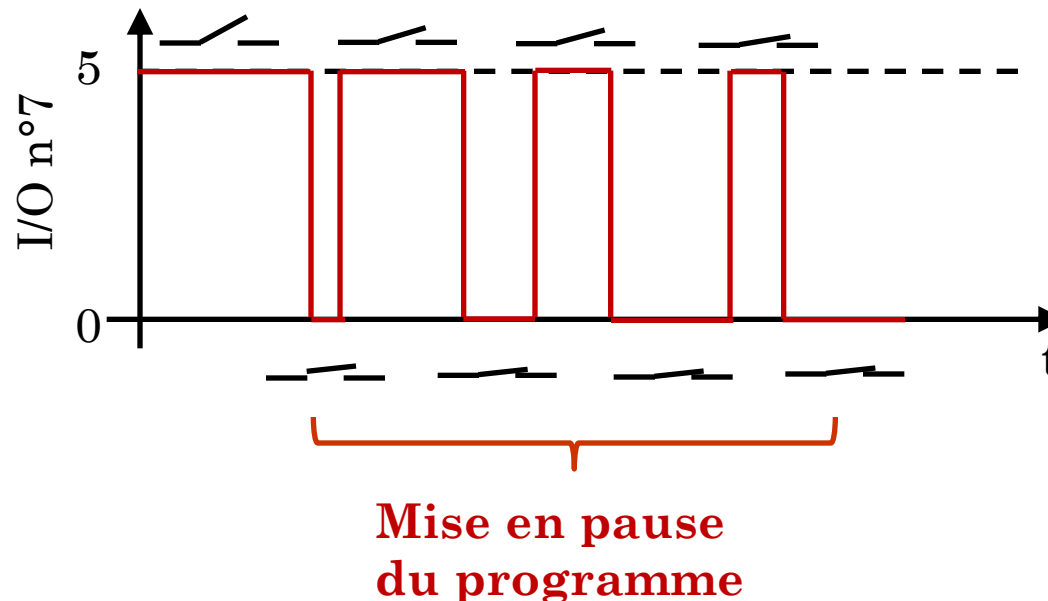
❑ Problème de rebond du bouton poussoir

- L'arduino fonctionne à 16 MHz ce qui signifie que le programme peut effectuer plusieurs fois la partie `void loop()` durant les rebonds et donc faire comme si on avait appuyé plusieurs fois sur le bouton
- Il existe 2 méthodes pour éliminer l'influence de ces rebonds : une méthode logicielle et une méthode matérielle.

3.2. Montage

❑ Problème de rebond du bouton poussoir : solution logicielle

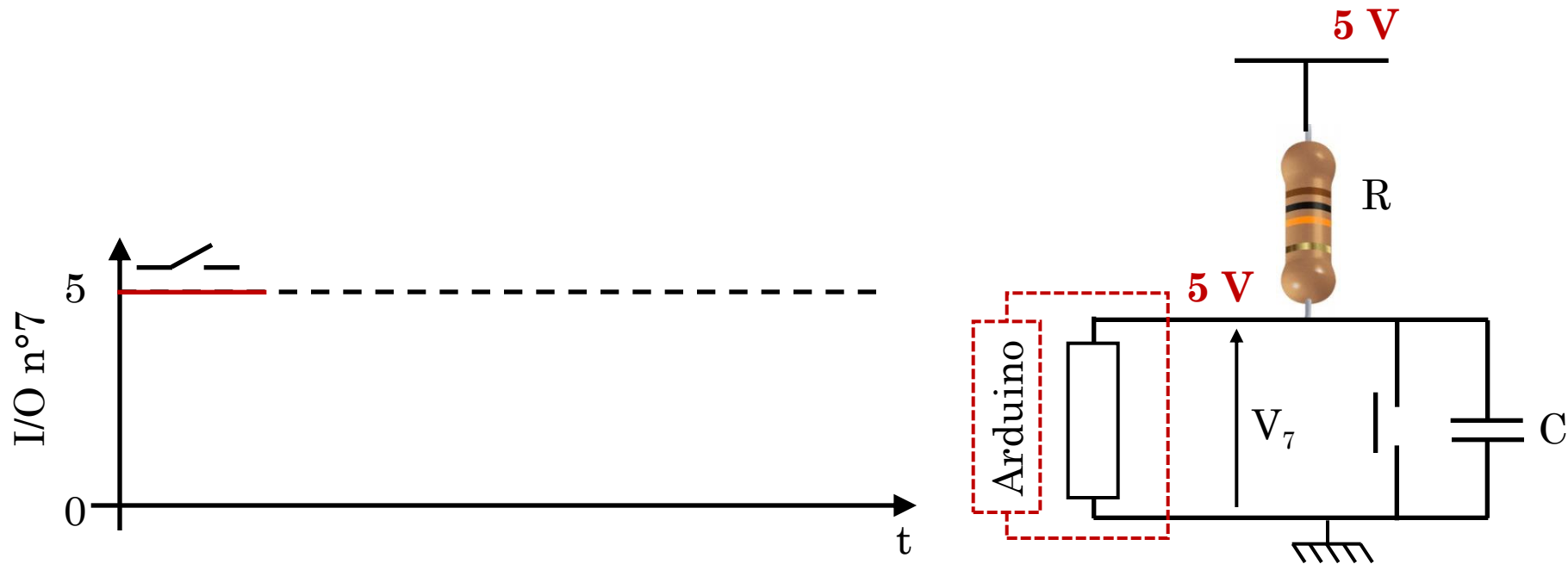
- Après avoir détecté la « première » transition du bouton poussoir, on peut demander au programme de faire une pause pour permettre au bouton de se stabiliser
- Un delay de quelques milli-secondes est suffisant



3.2. Montage

❑ Problème de rebond du bouton poussoir : solution matérielle

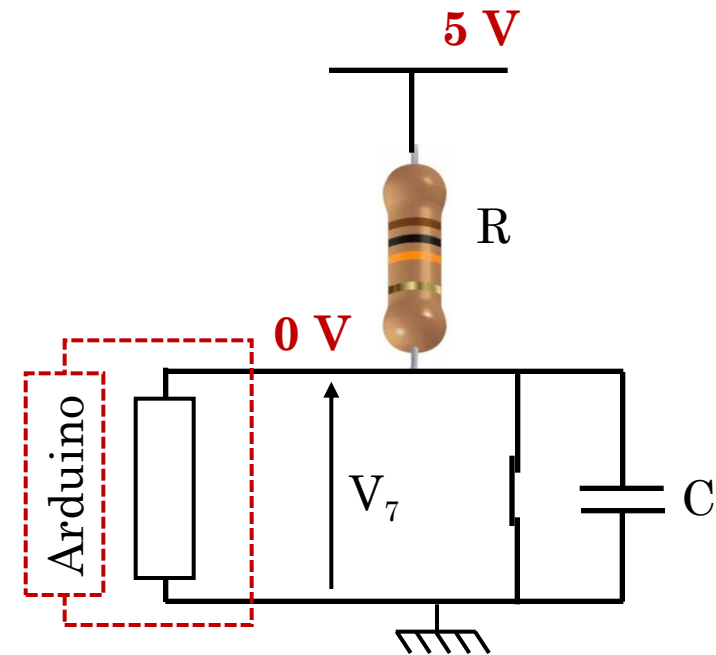
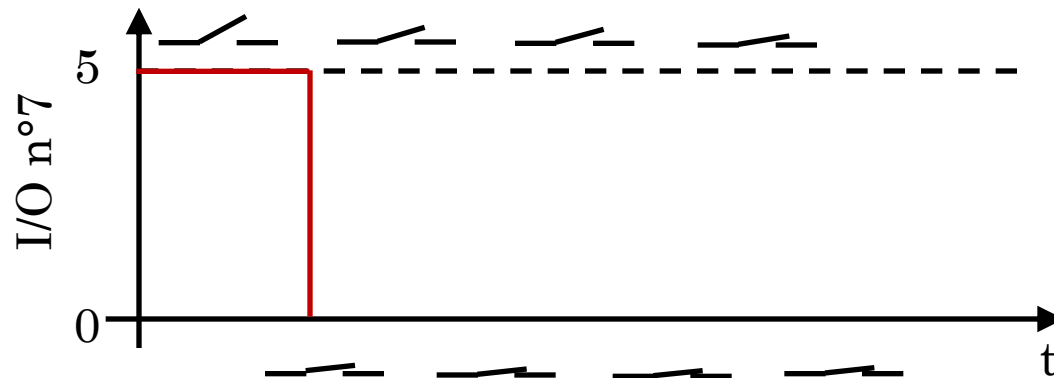
- On place un condensateur en parallèle avec le bouton
- Initialement le condensateur est chargé à la tension 5 V



3.2. Montage

❑ Problème de rebond du bouton poussoir : solution matérielle

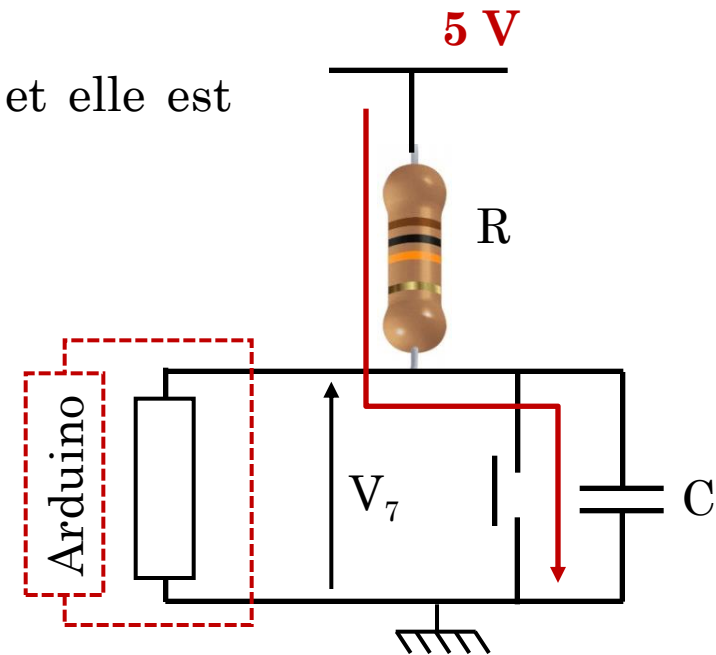
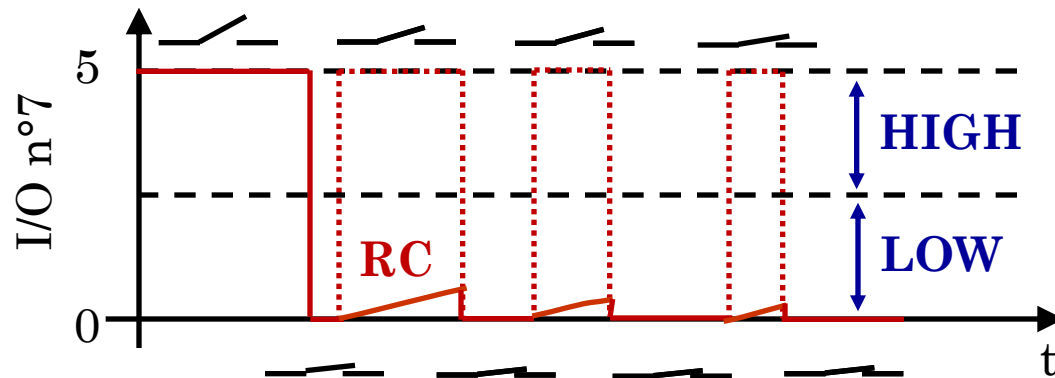
- On place un condensateur en parallèle avec le bouton
- Initialement le condensateur est chargé à la tension 5 V
- Lorsque l'on appuie sur le bouton, il se décharge instantanément et la tension à ses bornes devient nulle



3.2. Montage

❑ Problème de rebond du bouton poussoir : solution matérielle

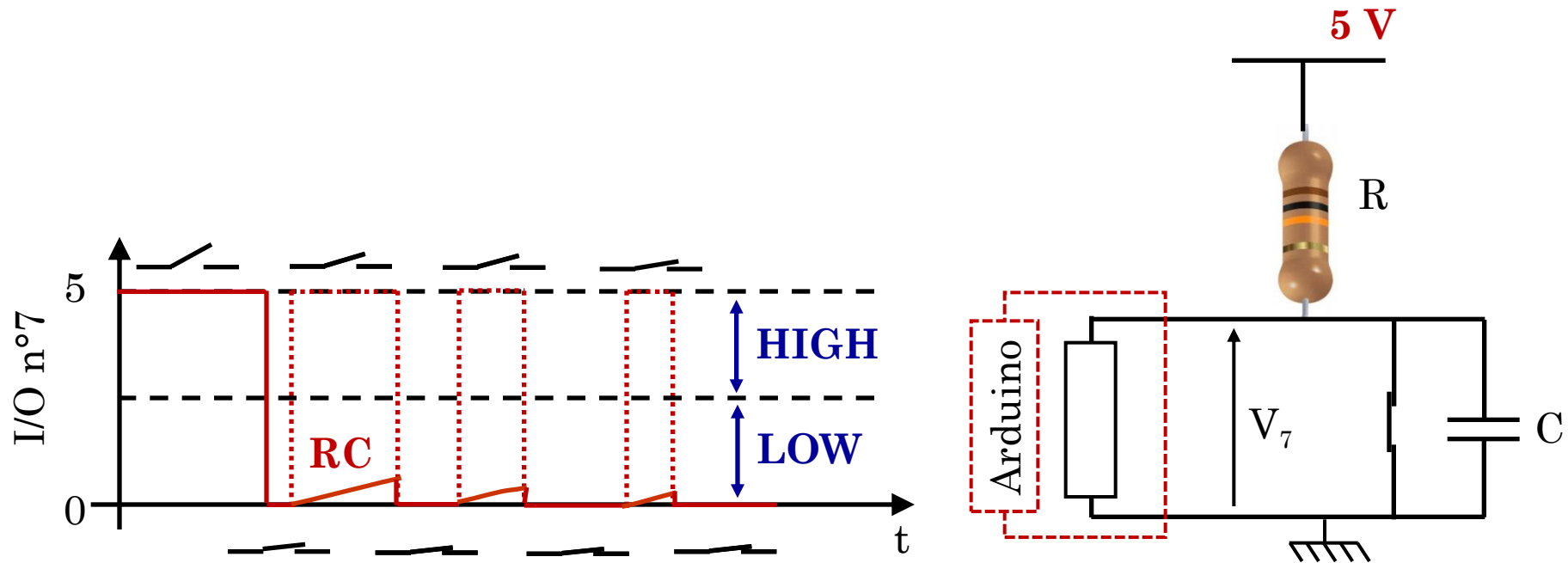
- Lors du premier rebond, le condensateur n'est plus court-circuité et il va se charger via la résistance de 10 k Ω donc avec une constante de temps RC
- Cette constante de temps doit être très longue par rapport au temps des rebond
- Ainsi la tension V_7 reste très proche de 0 V et elle est assimilée par l'entrée à un état LOW



3.2. Montage

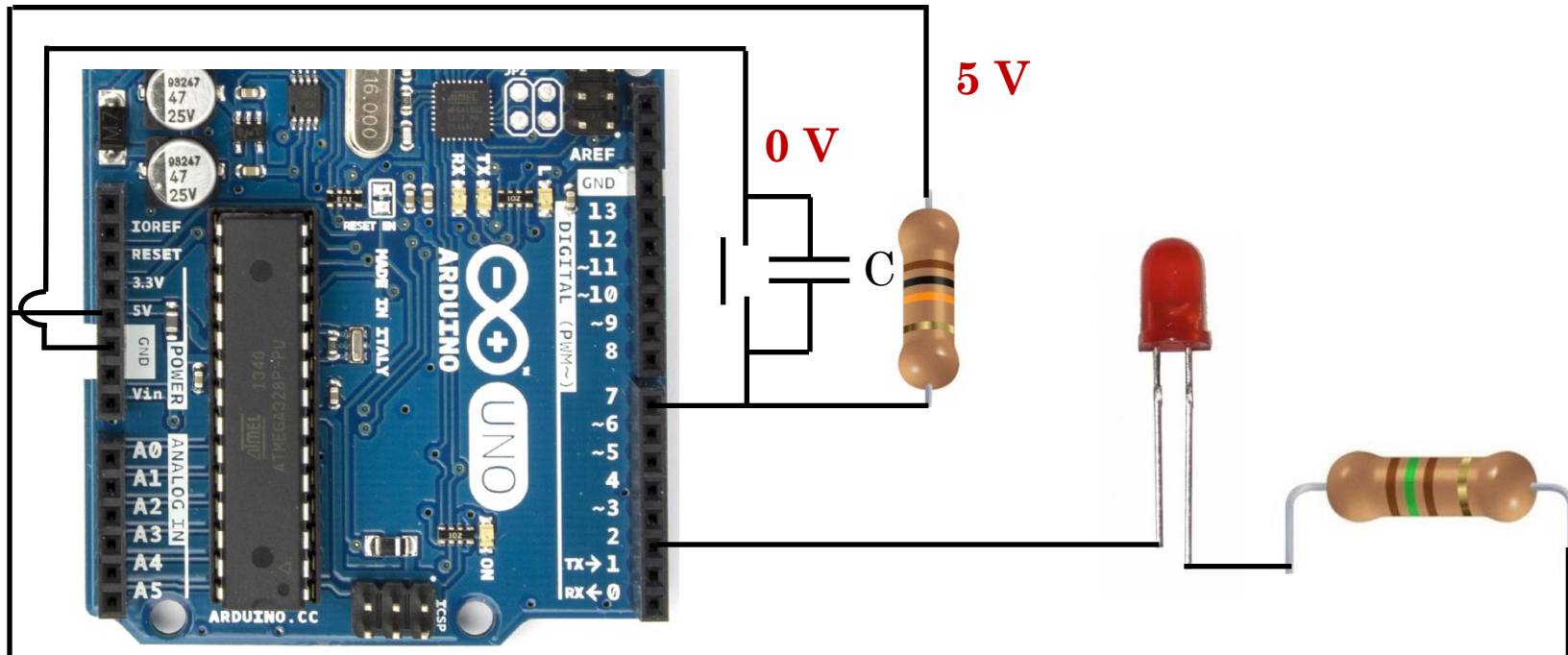
❑ Problème de rebond du bouton poussoir : solution matérielle

- Si on choisit un condensateur de 10 nF, la constante de temps a pour valeur $RC = 0.1 \text{ ms}$



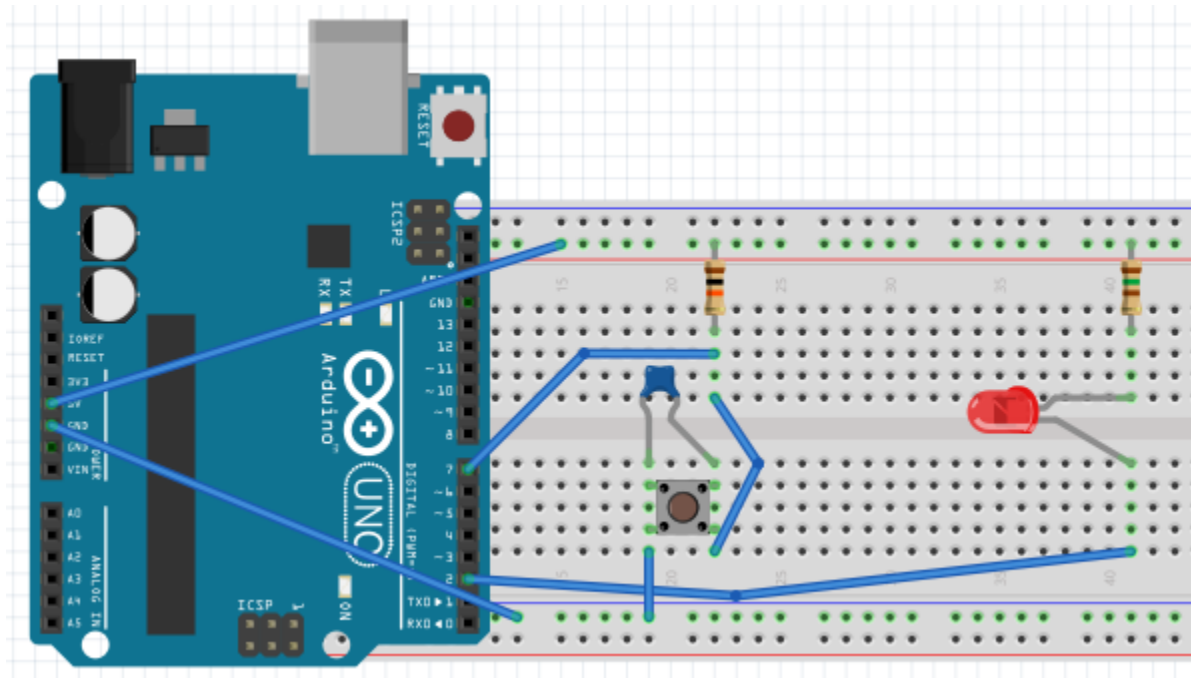
3.2. Montage

- Montage complet avec l'arduino



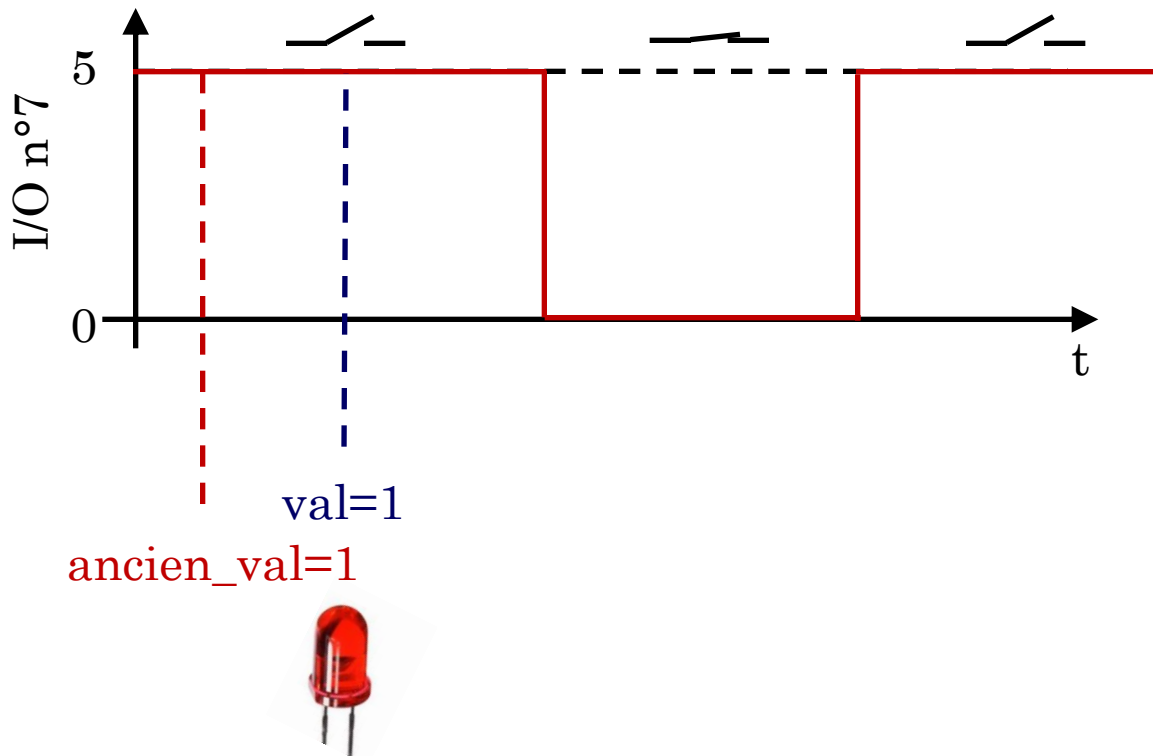
3.2. Montage

- Montage complet avec l'arduino



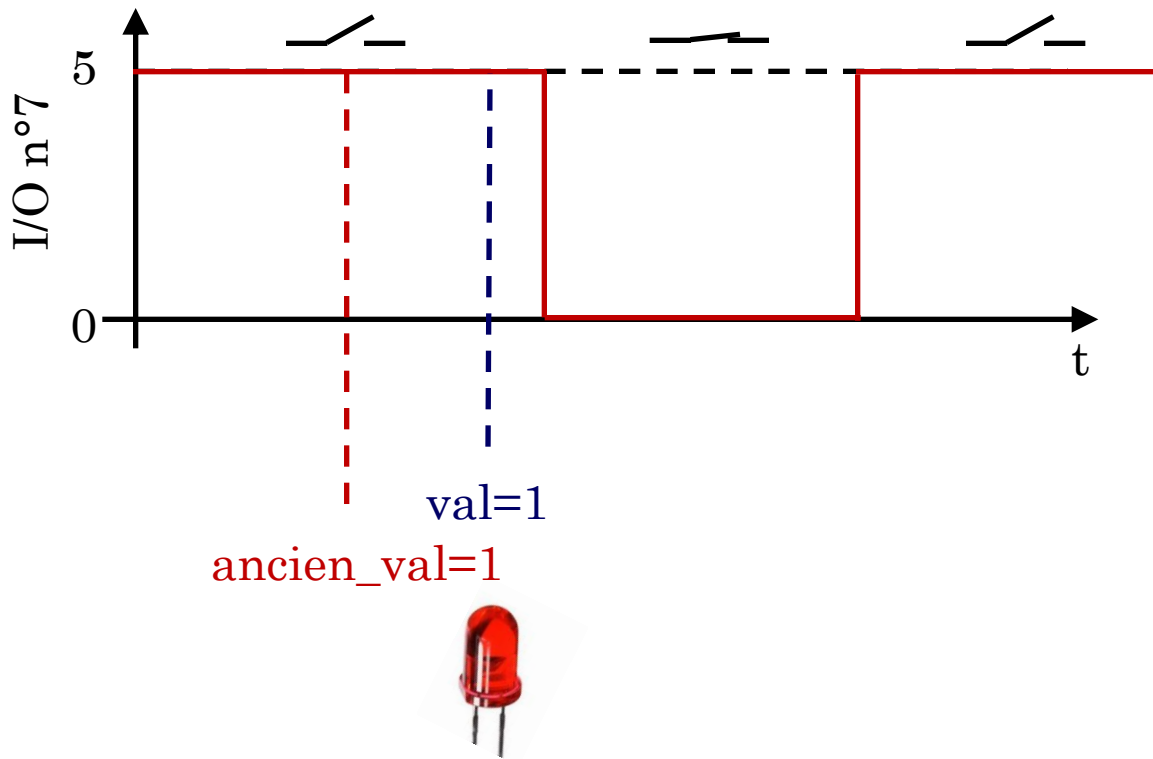
3.3. Détection d'un front descendant

- Pour éviter de prendre plusieurs fois en compte le fait que le bouton poussoir est enfoncé, on peut faire une détection de son changement d'état
- Pour cela on conserve en mémoire l'état dans lequel était le bouton poussoir dans la boucle d'avant



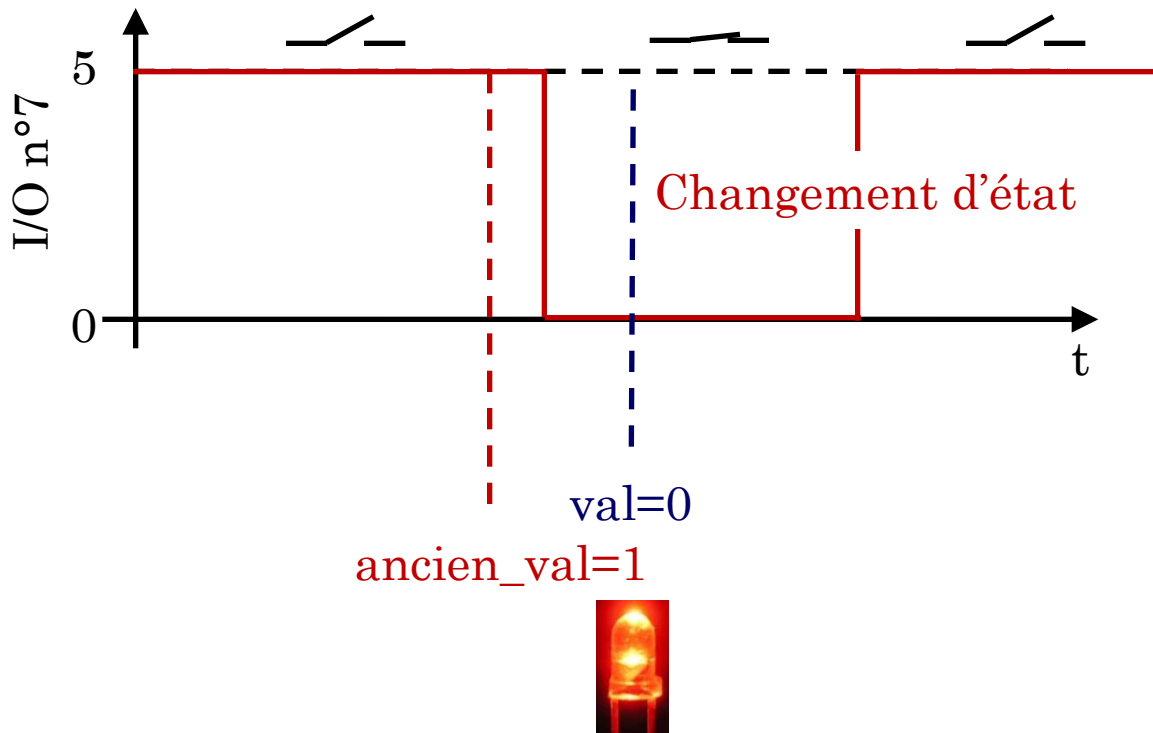
3.3. Détection d'un front descendant

- Pour éviter de prendre plusieurs fois en compte le fait que le bouton poussoir est enfoncé, on peut faire une détection de son changement d'état
- Pour cela on conserve en mémoire l'état dans lequel était le bouton poussoir dans la boucle d'avant



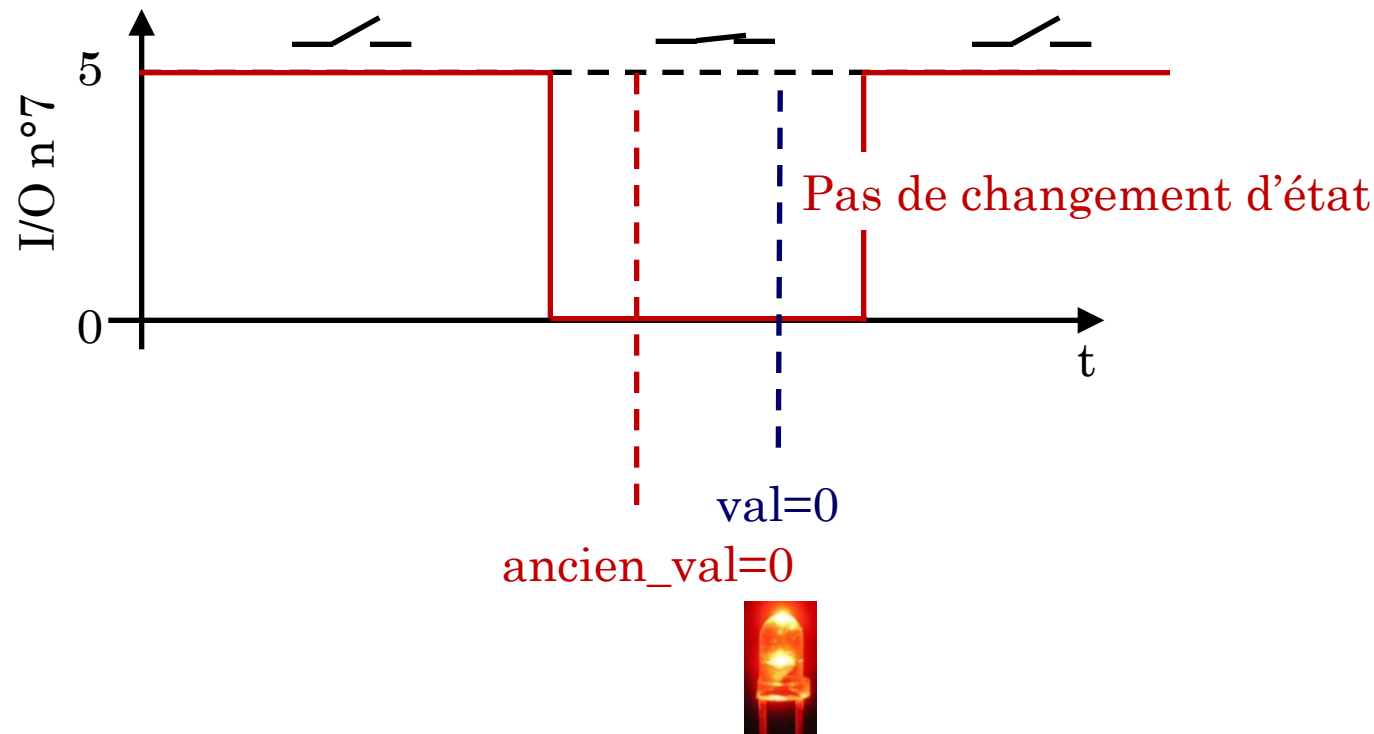
3.3. Détection d'un front descendant

- Pour éviter de prendre plusieurs fois en compte le fait que le bouton poussoir est enfoncé, on peut faire une détection de son changement d'état
- Pour cela on conserve en mémoire l'état dans lequel était le bouton poussoir dans la boucle d'avant



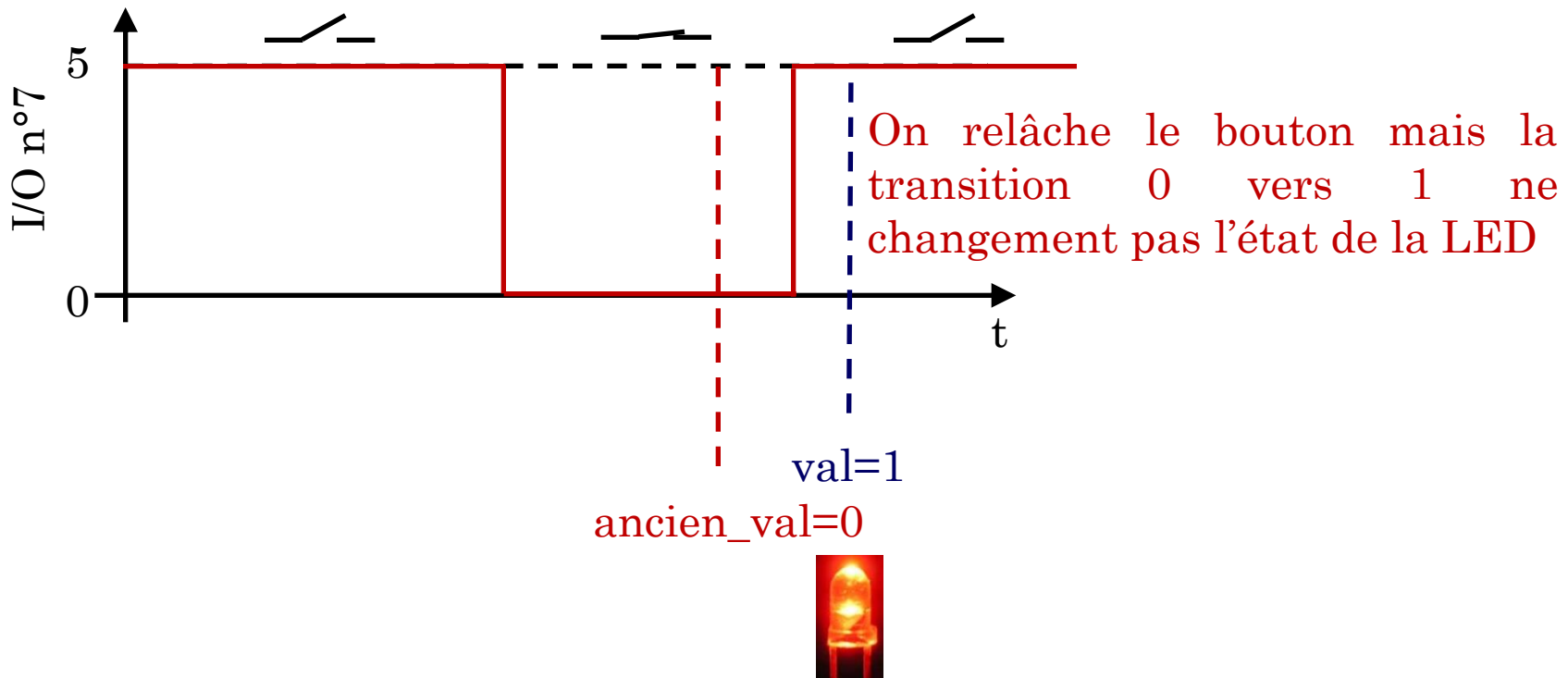
3.3. Détection d'un front descendant

- Pour éviter de prendre plusieurs fois en compte le fait que le bouton poussoir est enfoncé, on peut faire une détection de son changement d'état
- Pour cela on conserve en mémoire l'état dans lequel était le bouton poussoir dans la boucle d'avant



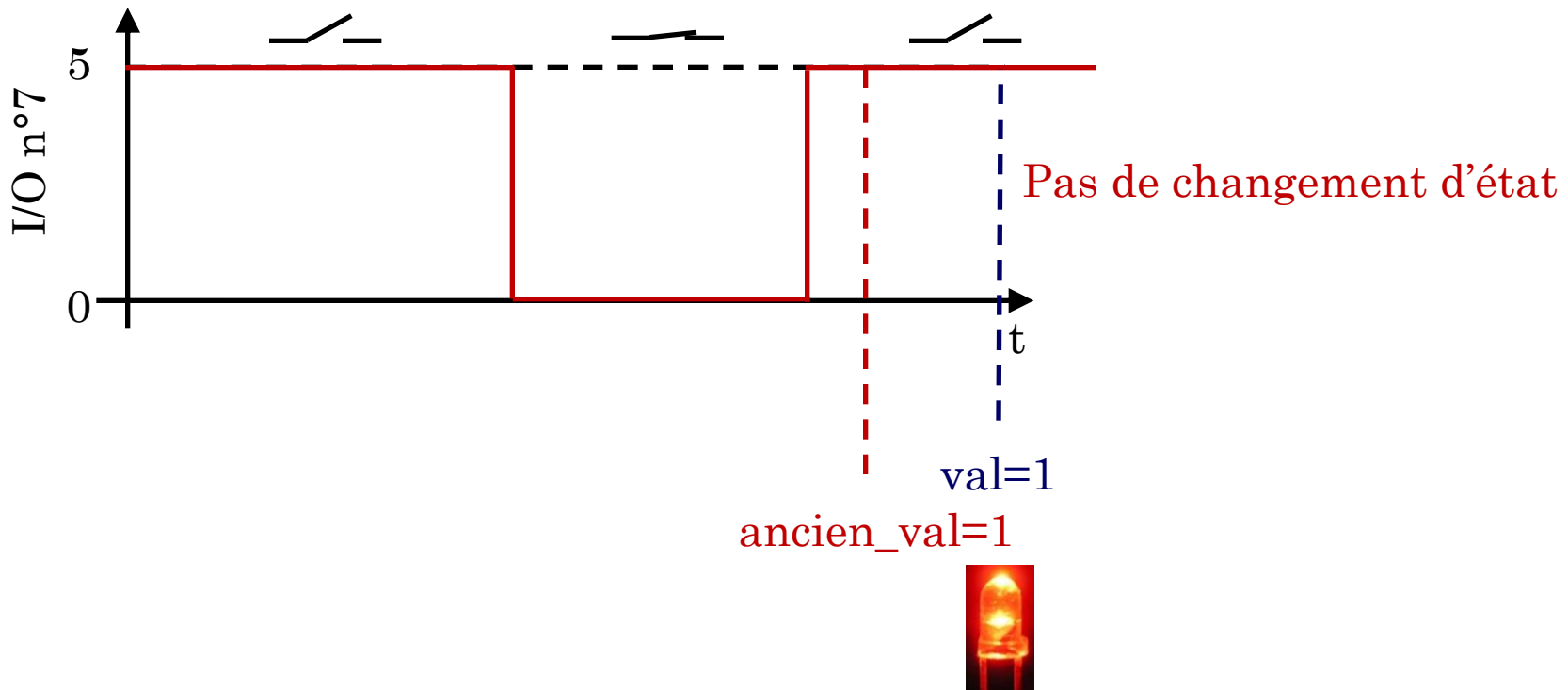
3.3. Détection d'un front descendant

- Pour éviter de prendre plusieurs fois en compte le fait que le bouton poussoir est enfoncé, on peut faire une détection de son changement d'état
- Pour cela on conserve en mémoire l'état dans lequel était le bouton poussoir dans la boucle d'avant



3.3. Détection d'un front descendant

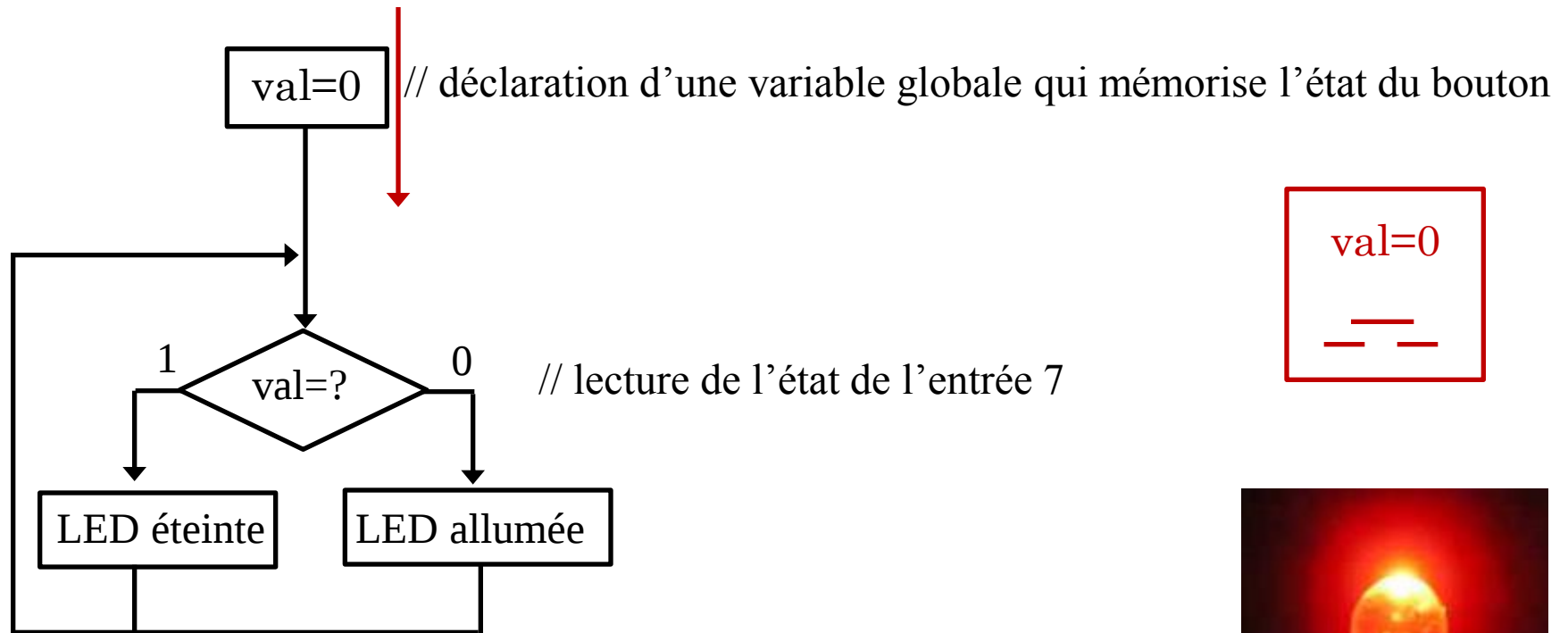
- Pour éviter de prendre plusieurs fois en compte le fait que le bouton poussoir est enfoncé, on peut faire une détection de son changement d'état
- Pour cela on conserve en mémoire l'état dans lequel était le bouton poussoir dans la boucle d'avant



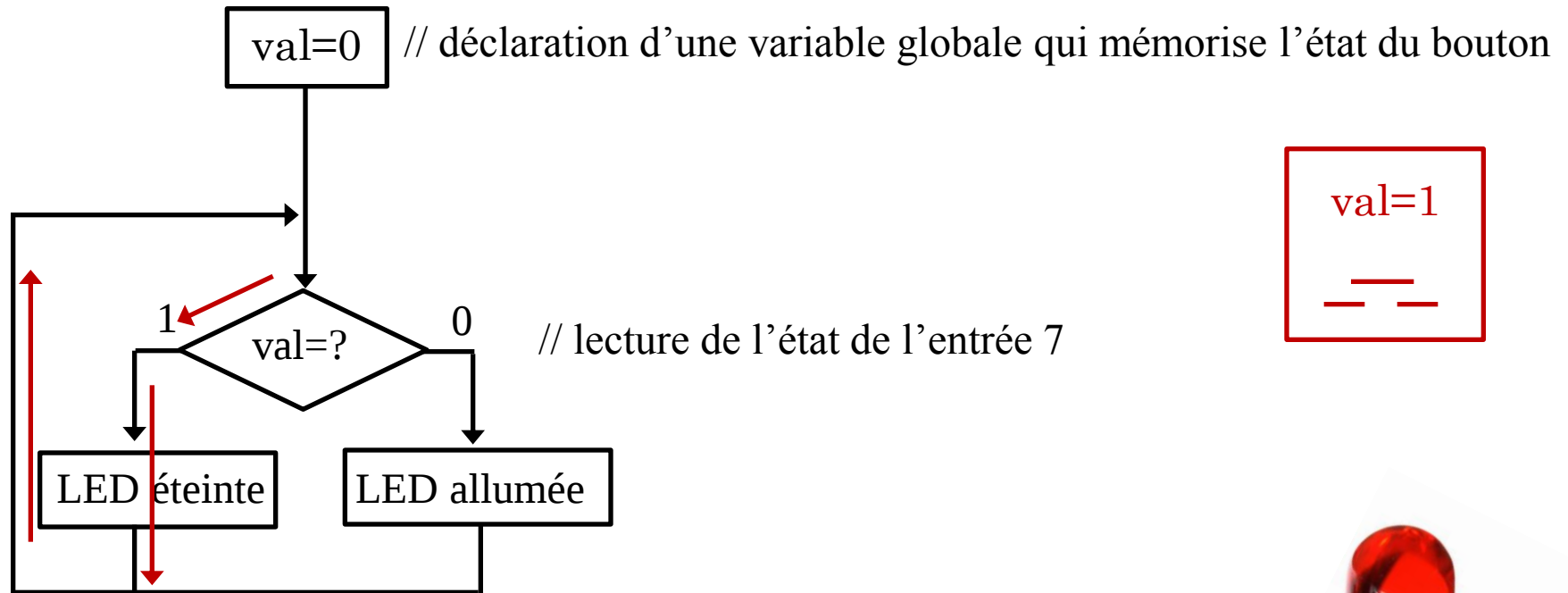
3.4. Les nouvelles fonctions

- **digitalRead (X)** : Lit l'état **LOW** ou **HIGH** de l'I/O X déclarée comme une entrée
- **if(X) {YYYYYY}** : permet d'effectuer les opérations YYYYYY si la condition X est vérifiée
- **else {ZZZZZZ}** : sinon on effectue les opérations ZZZZZZ
- **X==Y** : donne la valeur **HIGH** si la stricte égalité est vérifiée sinon cela donne la valeur **LOW**
- **(X==Y) && (Z==W)** : donne la valeur **HIGH** si les strictes égalités sont vérifiées sinon cela donne la valeur **LOW**. C'est la fonction ET.

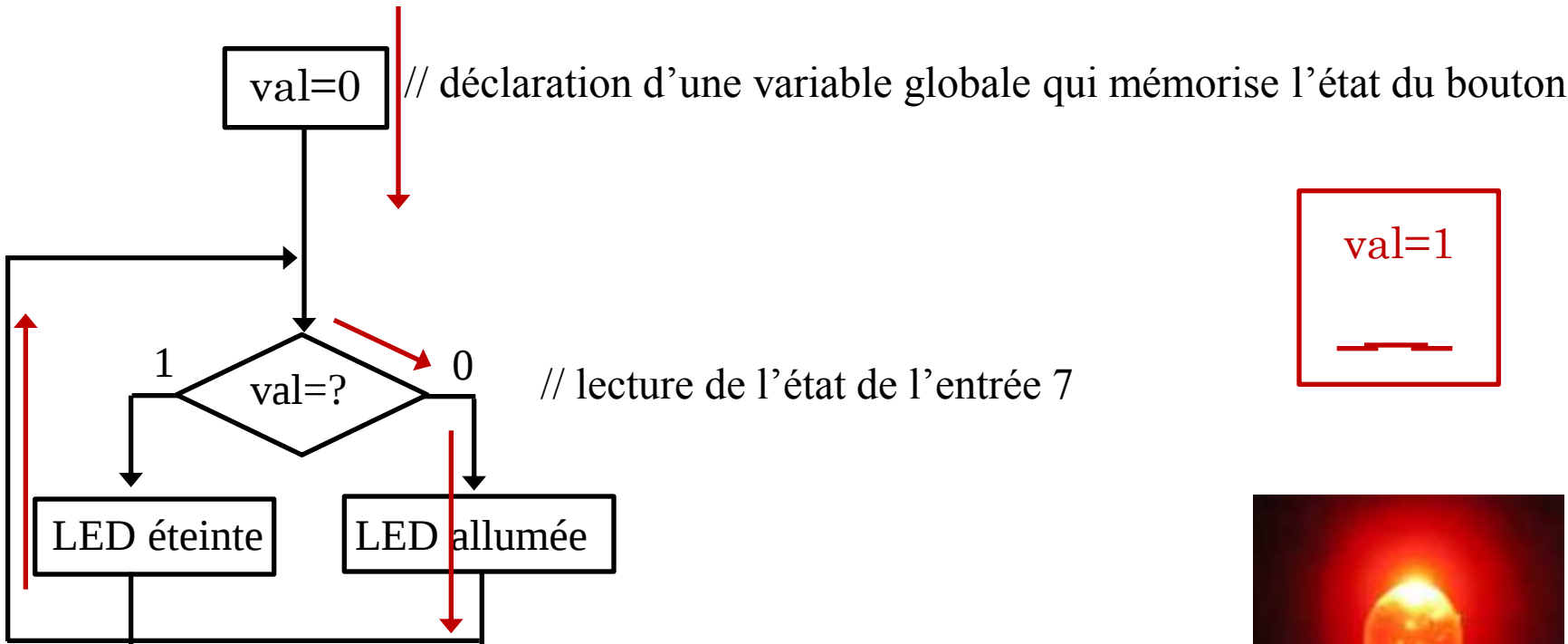
3.5. Objectif n°1 : diagramme fonctionnel



3.5. Objectif n°1 : diagramme fonctionnel



3.5. Objectif n°1 : diagramme fonctionnel



3.6. Objectif n°1 : le programme

//LED qui s'allume quand on appuie sur un bouton

const int led_rouge=2; //on définit une constante de type entier

const int bouton=7; //on définit une autre constante de type entier

int val=0; // déclaration d'une variable globale qui mémorise l'état du bouton

void setup(){ //début du programme

pinMode(led_rouge, **OUTPUT**); // l'I/O 2 est utilisée comme une sortie

pinMode(bouton, **INPUT**); // l'I/O 2 est utilisée comme une sortie

digitalWrite(led_rouge, **LOW**); // allume la LED

delay(5000); // et attendre 5 s

} // fin de définition du début de programme avec les variables

void loop() { // écriture de ce que fait le programme

 val=**digitalRead**(bouton); // lecture de l'état de l'entrée 7

if (val==**HIGH**) {**digitalWrite**(led_rouge, **HIGH**);} // éteint la LED

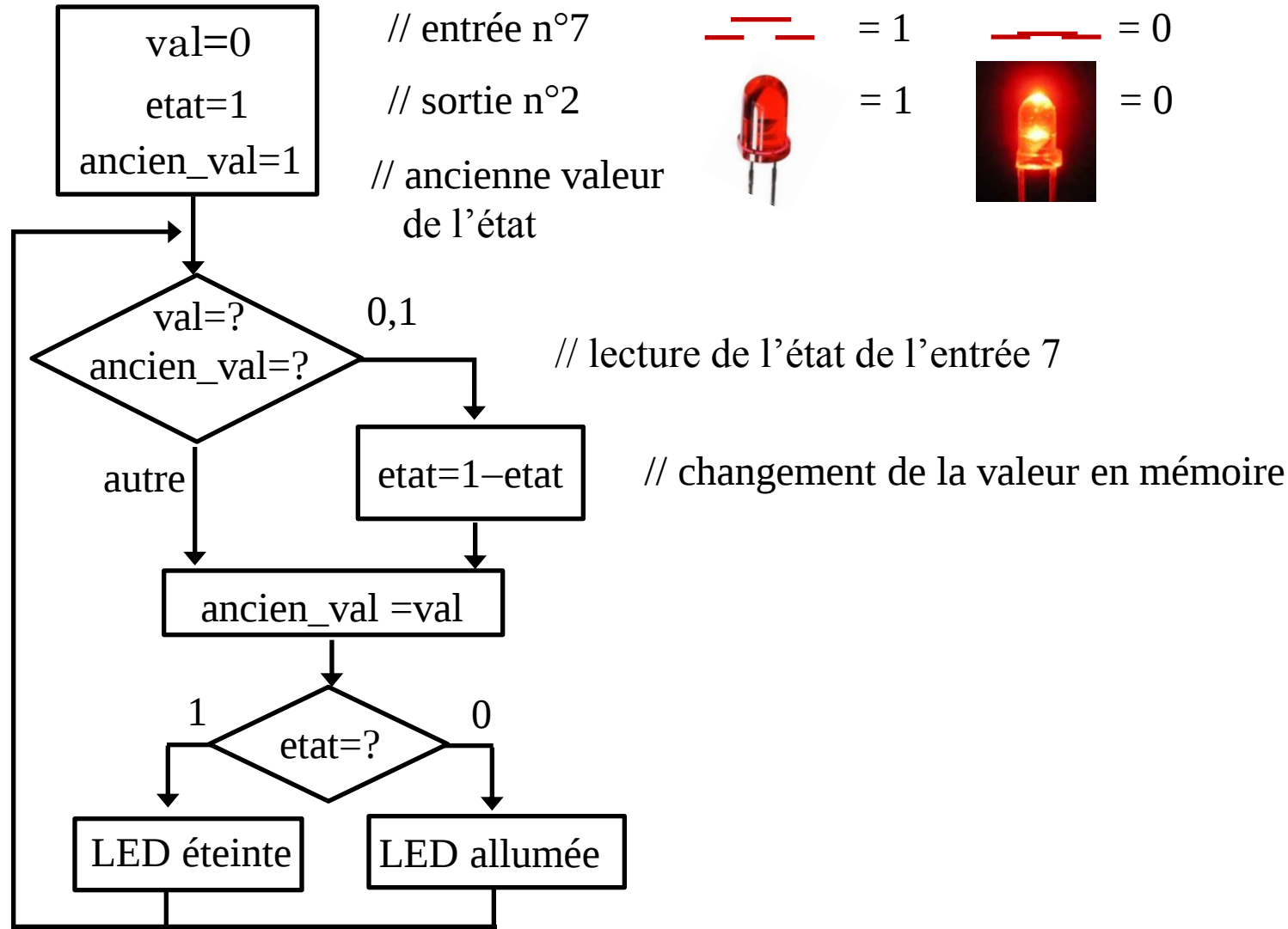
else {**digitalWrite**(led_rouge, **LOW**);} // allume la LED

} // indique la fin du programme qui recommencera au début

3.7. Objectif n°1 : déroulement

- La LED s'allume durant 5 s puis s'éteint si on appuie pas sur le bouton
- La LED s'allume quand on appuie sur le bouton
- Remarque : si on appuie sur le bouton « reset » on recommence le programme du début

3.8. Objectif n°2 : diagramme fonctionnel avec condensateur



3.9. Objectif n°2 : programme avec condensateur

```
//LED qui s'allume quand on appuie sur un bouton
const int led_rouge=2; //on définit une constante de type entier
const int bouton=7; //on définit une autre constante de type entier
int val=0; // déclaration d'une variable globale qui mémorise l'état du bouton
int etat=1; // = 0 pour LED éteinte sinon = 1
int ancien_val=1;
-----
void setup(){
    pinMode(led_rouge, OUTPUT); // l'I/O 2 est utilisée comme une sortie
    pinMode(bouton, INPUT); // l'I/O 2 est utilisée comme une sortie
    digitalWrite(led_rouge, LOW); // allume la LED durant 5 s
    delay(5000); }
-----
void loop() {
    val=digitalRead(bouton); // lecture de l'état de l'entrée 7
    if ((val==LOW)&&(ancien_val==HIGH)) {etat=1-etat;}
    ancien_val=val;
    if (etat==HIGH) {digitalWrite(led_rouge, HIGH);} // éteint la LED
    else {digitalWrite(led_rouge, LOW);} }
```

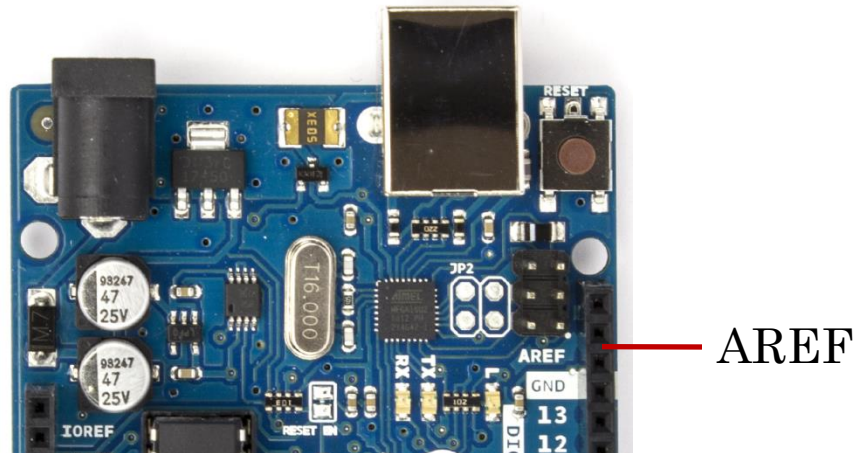
3.10. Objectif n°2 : programme avec delay

- On peut aussi supprimer l'influence des rebonds en utilisant un petit temps d'attente

```
void loop() {  
    val=digitalRead(bouton);    // lecture de l'état de l'entrée 7  
    if ((val==LOW)&&(ancien_val==HIGH)) {  
        etat=1-etat;  
        delay(100);}   
    ancien_val=val;  
    if (etat==HIGH) {digitalWrite(led_rouge, HIGH);} // éteint la LED  
    else {digitalWrite(led_rouge, LOW);} }
```

4.1. Présentation des entrées analogiques

- L'arduino possède 6 entrées analogiques numérotées A0 à A5. Elles sont numérotées PC0 à PC5 sur la carte Xplained Mini.
- Le fonctionnement de l'arduino étant numérique, les entrées vont convertir le signal analogique en signal numérique
- Il est possible de modifier la tension de référence de la conversion en appliquant sa nouvelle valeur sur l'entrée AREF



- L'impédance d'entrée des entrées analogiques est de 100 M Ω

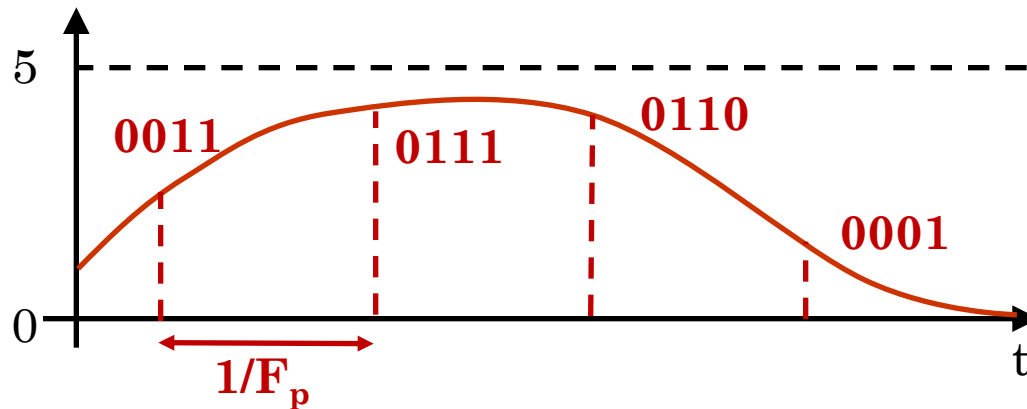
4.2. Conversion analogique/digitale

□ Méthode de conversion

- Il existe plusieurs techniques plus ou moins rapides pour convertir un signal analogique en signal numérique
- Il y a par exemple les convertisseurs à simple rampe, Sigma Delta et Flash

□ Echantillonnage

- Périodiquement (notion de fréquence d'échantillonnage, F_p) la tension est mesurée et convertie en mots binaires



4.2. Cas de l'arduino

- La fréquence d'échantillonnage est de 10 kHz donc la tension que l'on doit convertir ne doit pas varier plus vite que cela.
- La conversion se fait sur 10 bits ce qui signifie qu'il y a 1024 combinaisons pour coder la tension qui doit être comprise entre 0 et 5 V (sauf si on change la référence).
- Pour une tension de 3 V, on obtient le nombre suivant :

$$3 \text{ V} \Rightarrow \frac{1024}{5} \cdot 3 \approx 614$$

- Le minimum de variation de tension que l'on peut détecter est :

$$\Delta V = \frac{5}{1024} \approx 4.89 \text{ mV}$$

4.3. Les nouvelles fonctions

- **analogRead(X)** : lit la tension sur l'entrée X (X allant de 0 à 5) et la convertie en un nombre compris entre 0 et 1023
- Le nombre obtenue est un entier (INT) mais on peut le sauver en tant que nombre a virgule en ayant au préalable définie une variable flottante :

```
float tension_num=0;  
tension_num=analogRead(0);
```

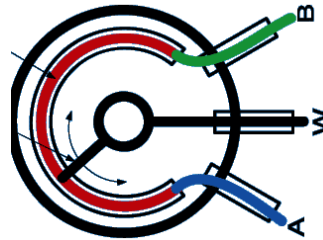
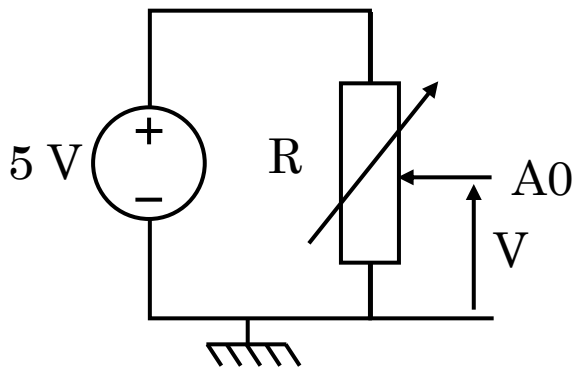
- Il est possible de faire des calculs avec la fonction analogRead mais au moins un des nombres utilisés doit comporter une virgule

```
tension=analogRead(0)/204.8;
```

- Remarque : il n'est pas nécessaire de déclarer l'utilisation d'une entrée analogique dans la partie setup. En effet et contrairement aux I/O, une entrée analogique ne peut pas fonctionner comme une sortie et il n'y a pas « d'interrupteur » à faire basculer.

4.4. Utilisation d'un potentiomètre

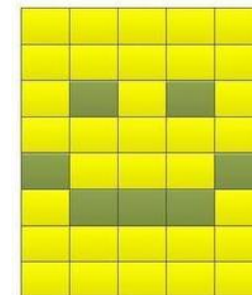
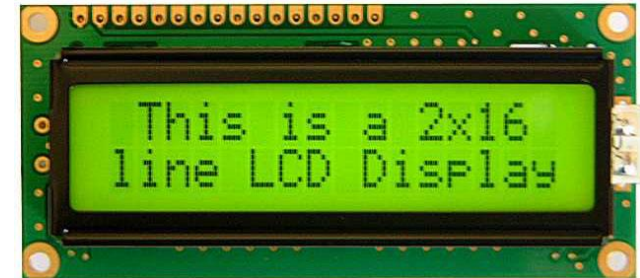
- Un potentiomètre sert à modifier la valeur d'une résistance ou à prélever une partie d'une tension (pont diviseur de tension)
- Utilisation d'un potentiomètre de $10\text{ k}\Omega$ alimenté par une tension de 5 V :



0V **A0** 5V

5.1. Présentation de l'écran LCD

- L'écran LCD que nous utilisons est un 16x2 ce qui signifie qu'il est constitué de 2 lignes de 16 caractères
- Un décodeur de caractères est déjà intégré à l'écran et permettra d'afficher ce qui sera envoyé par l'arduino et de le garder en mémoire
- Les instructions sont données en mode parallèle et donc cet écran nécessite l'utilisation de plusieurs sorties de l'arduino
- Chaque caractère est constitué d'un bloc de 8×5 pixels



5.2. Les entrées de l'écran

- Voici la description des entrées de l'écran



Masse (0 V)

5 V

Cette tension permet de régler le contraste. Comme sa valeur dépend de la personne qui regarde cet écran, on utilise un potentiomètre de 5 k Ω connecté entre 0 V et 5 V

5.2. Les entrées de l'écran

- Voici la description des entrées de l'écran



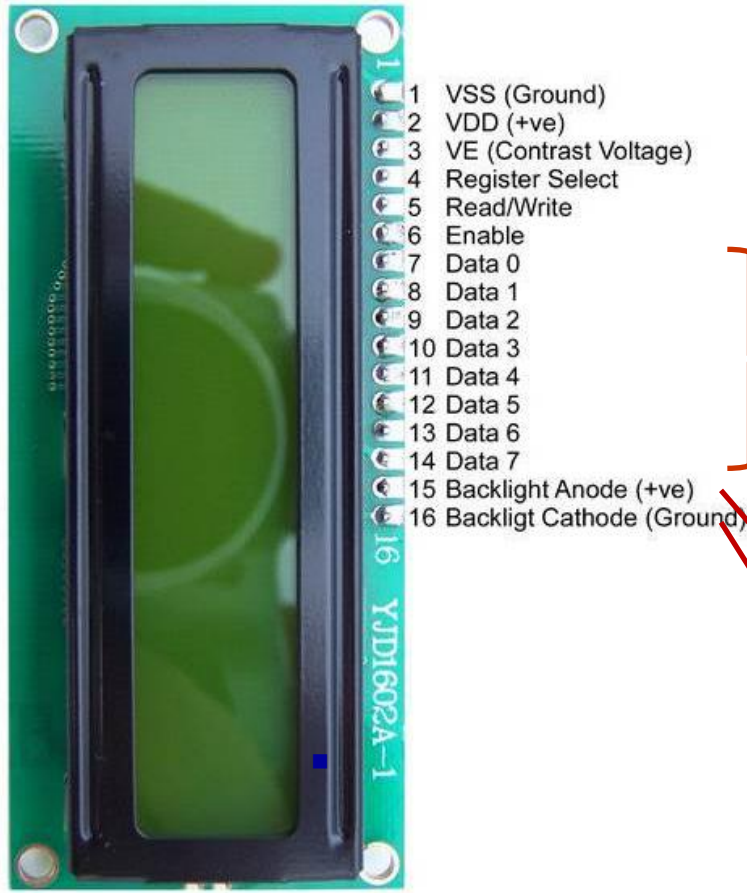
On sélectionne le type de message qu'on envoie. High : envoi de données. Low : envoi d'instructions

High : on lit la mémoire de l'écran. Low : on écrit dans la mémoire de l'écran. Dans ce cours on ne fait qu'écrire dans la mémoire donc on connecte cette entrée à la masse (0 V)

L'écran prend en compte les données lorsque cette entrée passe de l'état haut à l'état bas.

5.2. Les entrées de l'écran

- Voici la description des entrées de l'écran



Pour écrire les caractères il est nécessaire de donner 8 bits (Data0 à Data7) mais pour les caractères usuels, 4 bits sont suffisants (Data4 à Data7)

Alimentation du rétro-éclairage : 5 V

Alimentation du rétro-éclairage : 0 V

5.3. Librairie et fonctions de l'écran

- Des fonctions ont déjà été développées pour gérer cet écran et elles sont présentes dans ce que l'on appelle une « bibliothèque ». Celle qui nous concerne s'appelle « **LiquidCrystal** » et il sera nécessaire de l'appeler en début de programme avec la fonction :

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

- On peut aussi aller chercher cette librairie dans : croquis => importer bibliothèque => LiquidCrystal
- Il faut initialiser les commandes en indiquant les sorties utilisées

```
LiquidCrystal(rs, enable, D4, D5, D6, D7)
```

- Il faut initialiser le nombre de colonnes et de lignes de notre écran

```
lcd.begin(X, Y)
```

Pour l'écran du cours nous avons X = 16 et Y = 2

5.3. Librairie et fonctions de l'écran

- Pour imprimer le texte XYZ, on écrit :

```
lcd.print("XYZ")
```

- Pour effacer un texte de la mémoire de l'écran on écrit :

```
lcd.clear()
```

- Pour faire disparaître le texte sans l'effacer de la mémoire :

```
lcd.noDisplay()
```

- Pour faire ré-apparaître le texte :

```
lcd.display()
```

5.3. Librairie et fonctions de l'écran

- Pour commencer à écrire un texte à une colonne (X) et une ligne (Y) données, il faut d'abord écrire :

```
lcd.setCursor(X, Y)
```

- On peut faire clignoter le curseur en écrivant avant :

```
lcd.blink()
```

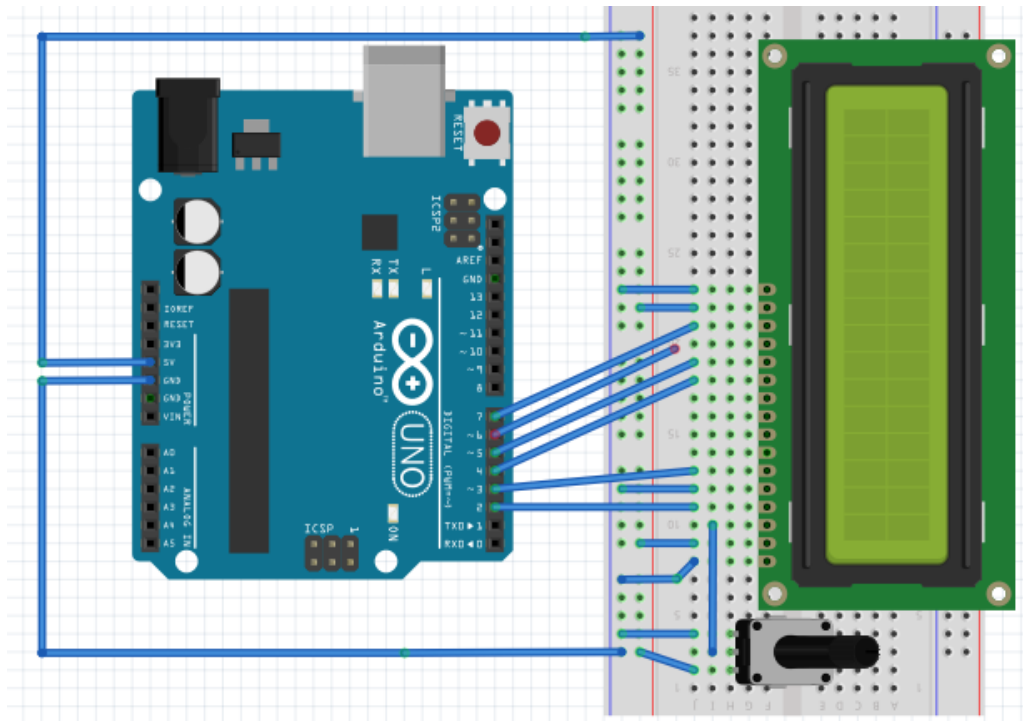
- Et pour arrêter son clignotement :

```
lcd.noBlink()
```

5.4. Exemple de montage

- Dans cet exemple nous déclarons :

`LiquidCrystal(2, 3, 4, 5, 6, 7)`



5.5. Exemple de programme

```
// affichage d'un texte sur un LCD
```

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

```
LiquidCrystal lcd(2, 3, 4, 5, 6, 7); // initialise les commandes avec les n° de broches
```

```
void setup() {
```

```
  lcd.begin(16, 2); // initialiser le nombre de colonnes et de lignes
```

```
  lcd.print("Bonjour");
```

```
  lcd.setCursor(0, 1); //le curseur se positionne à la 1ère colonne, 2ième ligne
```

```
  lcd.print("je suis en PeiP2");
```

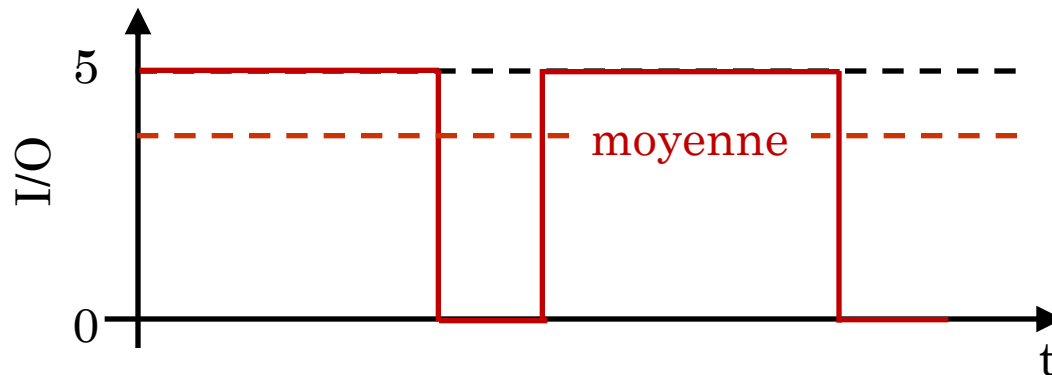
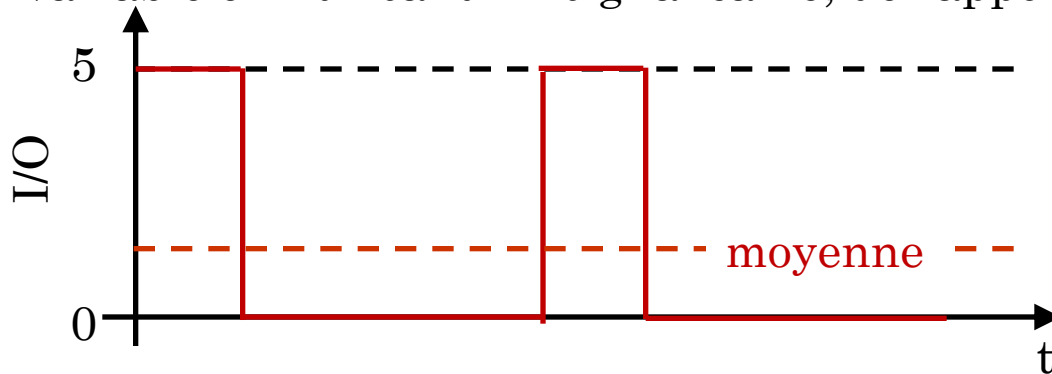
```
}
```

```
void loop() {
```

```
}
```

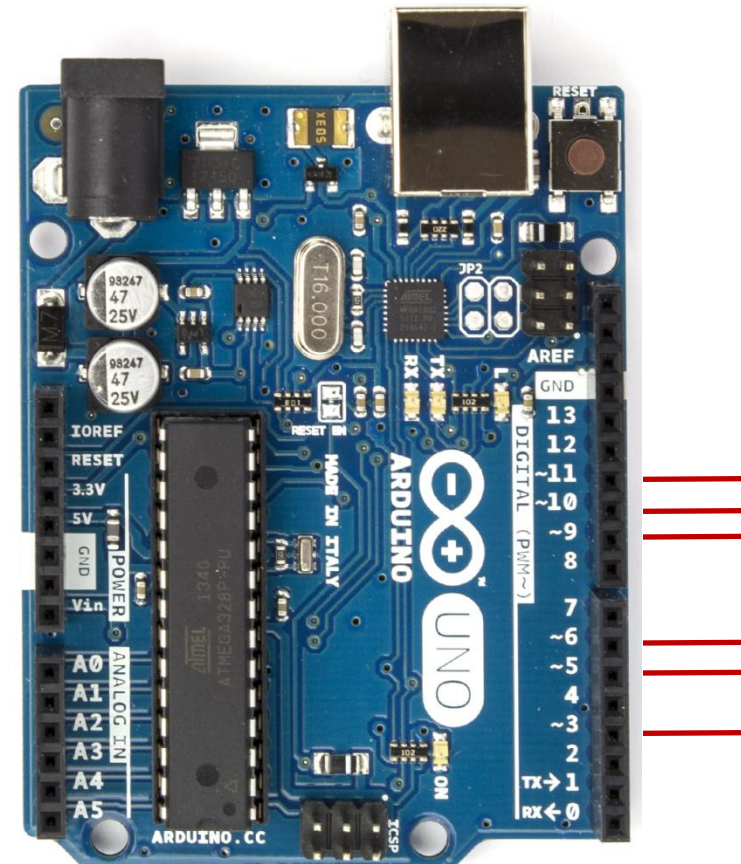
6.1. Notion de PWM

- Il n'est pas possible d'obtenir une tension analogique en sortie de l'arduino (nous sommes restreint aux valeurs 0 V et 5 V)
- Dans certains cas il est possible de faire comme si on avait une tension variable en utilisant un signal carré, de rapport cyclique variable



6.2. Sorties compatibles PWM de l'arduino

- Les I/O qui permettent d'utiliser les PWM sont précédées du signe « ~ »
- Il s'agit des I/O 3, 5, 6, 9, 10 et 11
- La fréquence d'horloge du PWM est de 490 Hz sauf pour l'I/O n°5 qui a une fréquence de 980 Hz.



6.3. Les nouvelles fonctions

- **analogWrite(X, Y)** : c'est la fonction dédiée au PWM de l'arduino. Il faut spécifier l'I/O X ainsi que la valeur du rapport cyclique Y. Y doit avoir une valeur comprise entre 0 et 255. Y = 0 : la sortie est toujours à 0. Y = 255 : la sortie est toujours à 5 V.

```
analogWrite(led_rouge, 100);
```

- **for (X=a; X<b; X++) {YYYYYY}** : cela s'appelle une boucle FOR et permet d'exécuter plusieurs fois les opérations YYYYYY. Pour compter les boucles, on utilise la variable X que l'on initialise à la valeur « a » et que l'on incrémente de 1 à chaque début de boucle (X++). On effectue la boucle tant que X est inférieure à « b »

```
for (i=0; i<=255; i++) {  
    analogWrite(led_rouge, i);  
    delay(10); }  

```

6.3. Les nouvelles fonctions

- **millis()** : la carte arduino est dotée d'un chronomètre qui enregistre le temps écoulé depuis l'allumage de la carte. Le temps est obtenu en milliseconde et le temps maximum enregistrable est 50 jours soit $4.32 \cdot 10^9$ ms. Si le temps maximum n'est pas suffisant, il est possible de connecter à l'arduino une horloge extérieure alimentée par une pile.



```
chrono=millis(); //on mémorise la valeur du chronomètre
```

```
.....
```

```
if ((millis()-chrono)>500) { // durant plus de 500
    etat--; // on enlève 1 à la variable « état »
    delay(20);} // on règle la sortie de la boucle if
```

6.3. Les nouvelles fonctions

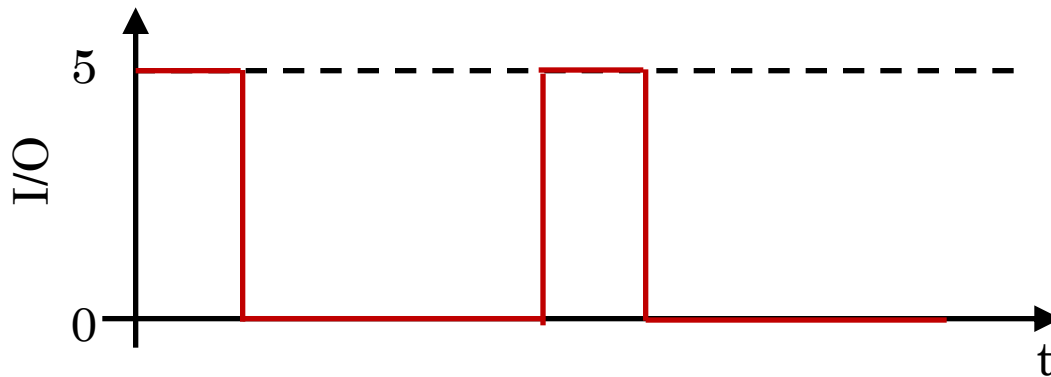
- **long** : en raison de la longueur du chiffre renvoyé par la fonction **millis()**, il faut stocker cette valeur dans une variable de type **long**. Cette variable est de type entier et va de $-2\,147\,483\,648$ à $2\,147\,483\,647$.
- **unsigned** : pour arriver aux 50 jours, on enlève le signe de la variable **long**. Dans ce cas **unsigned long** va de 0 à $4\,294\,967\,295$.

6.4. Exemple : luminosité d'une LED

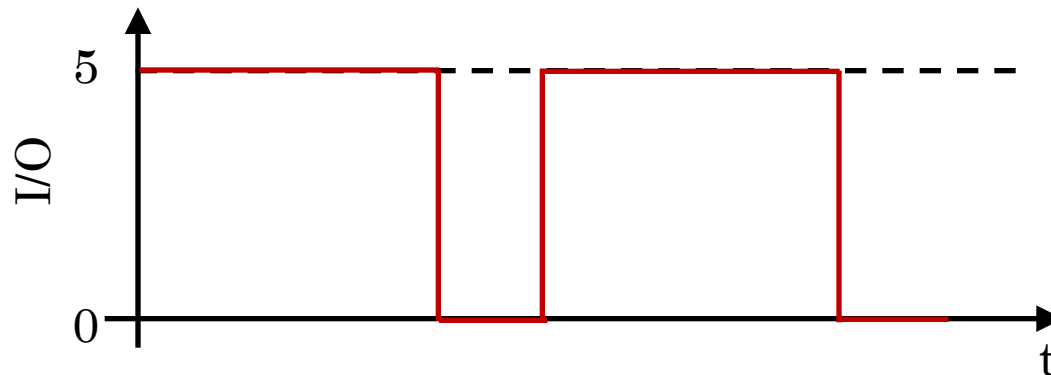
- Pour modifier la luminosité d'une LED il faut soit modifier la valeur de la résistance mise en série avec la LED soit modifier la tension d'alimentation.
- Ces deux solutions ne sont pas envisageables simplement avec l'arduino
- Une solution consiste à utiliser la persistance rétinienne (utilisée par la Télévision). En effet si on allume et on éteint la LED très rapidement, on la verra toujours éclairée. Si on augmente le temps durant lequel la LED est allumée, on verra la LED plus brillante.

6.4. Exemple : luminosité d'une LED

- La variation du rapport cyclique du signal module la luminosité de la LED



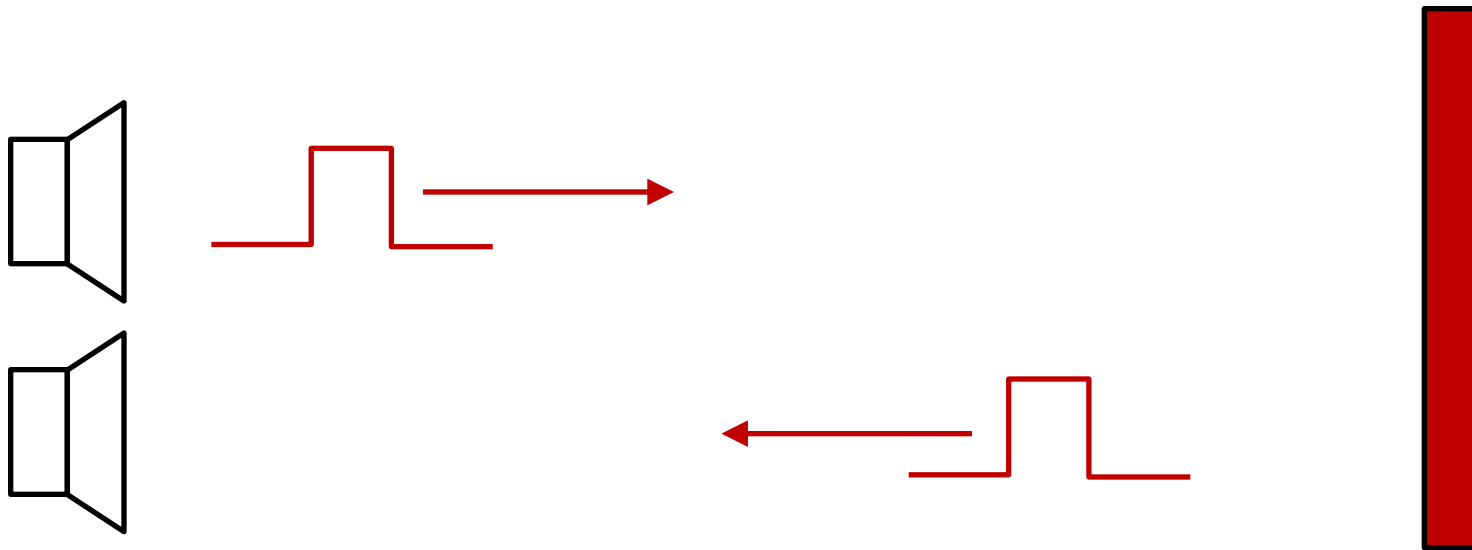
LED très lumineuse



LED peu lumineuse

7.1. Mesure d'une distance par le ultra-son

- Une onde acoustique est envoyée avec un haut-parleur piézo-électrique.
- Elle est réfléchiée sur l'obstacle revient sur un autre (ou le même) haut-parleur piézo-électrique.
- En prenant en compte la vitesse de propagation dans l'air (environ 340 m/s), le temps que met l'onde pour faire l'aller-retour donne la distance de l'objet.

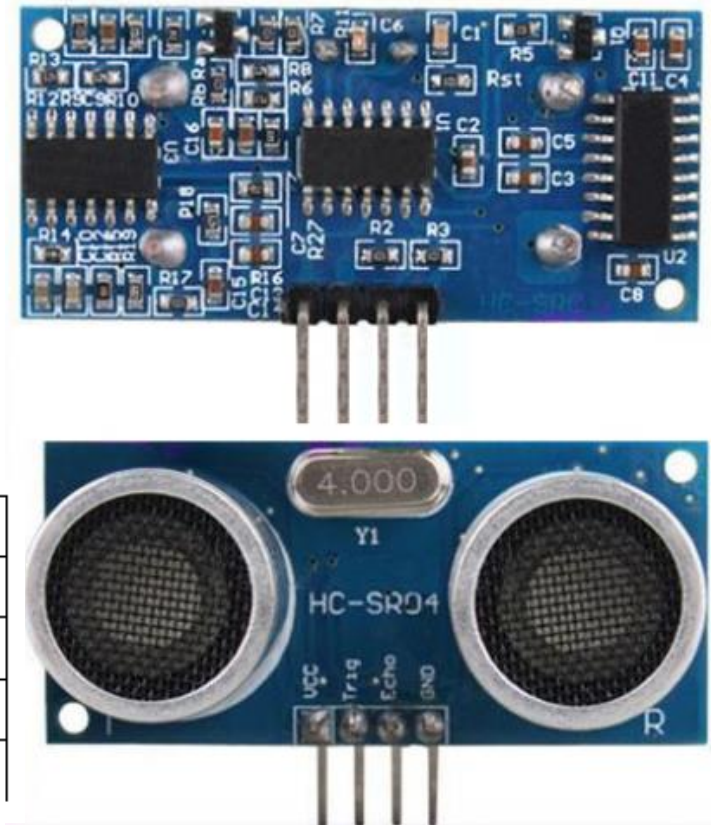


7.2. Présentation du module HC-SR04

□ Matériel

- Il existe plusieurs modules qui permettent de mesurer une distance et nous utiliserons le HC-SR04.
- Il comprend 2 haut-parleurs et l'onde utilisée a une fréquence de 40 kHz.
- L'électronique de gestion des haut-parleurs et de mesure de la distance est déjà intégrée au module

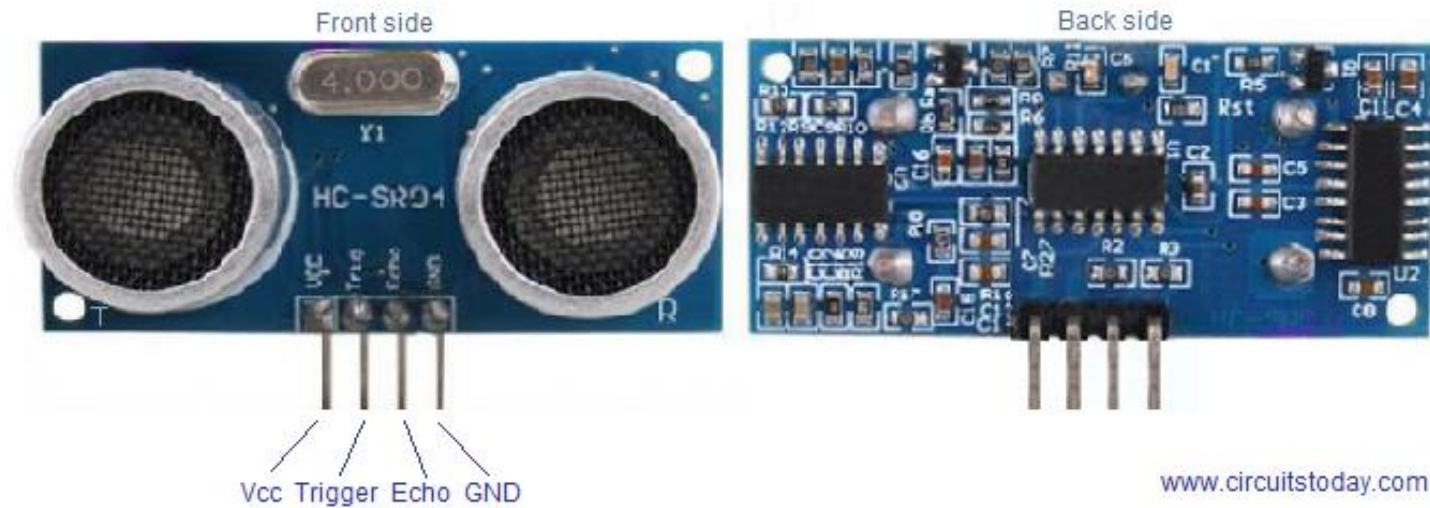
Parameter	Min	Typ.	Max	Unit
Operating Voltage	4.50	5.0	5.5	V
Quiescent Current	1.5	2	2.5	mA
Working Current	10	15	20	mA
Ultrasonic Frequency	-	40	-	kHz



7.2. Présentation du module HC-SR04

□ Matériel

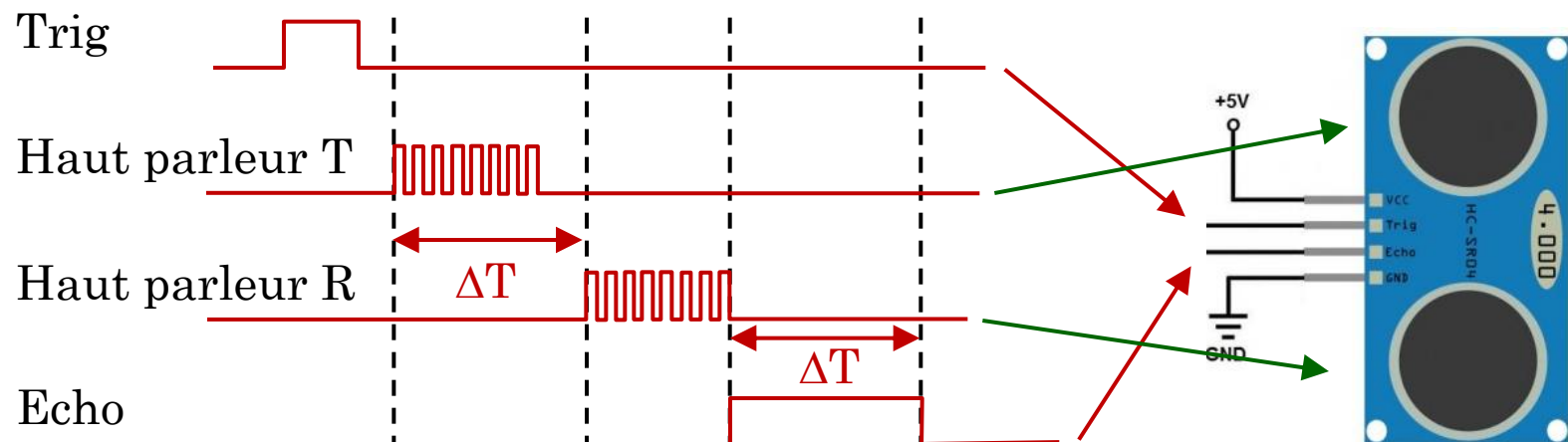
- Il y a 3 entrées dans le module : VCC, GND et Trig
- Il y a 1 sortie : Echo



7.2. Présentation du module HC-SR04

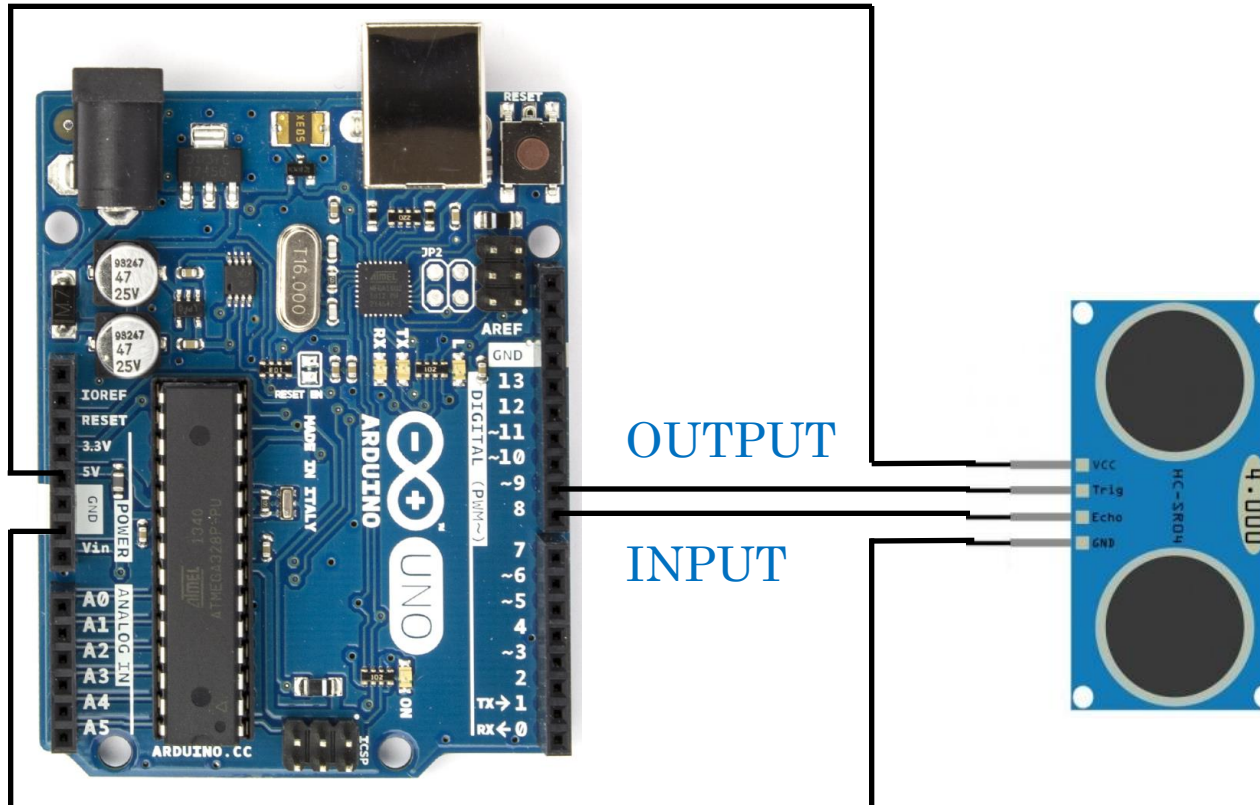
□ Fonctionnement

- Pour déclencher une mesure il faut appliquer une impulsion d'au moins 10 μ s sur l'entrée Trig
- Le module envoie alors 8 impulsions sonores
- On obtient en sortie (Echo) une impulsion dont la longueur correspond au temps que mettent les 8 impulsions pour faire l'aller/retour
- Il est préférable d'attendre 60 ms avant de demander une nouvelle mesure



7.2. Présentation du module HC-SR04

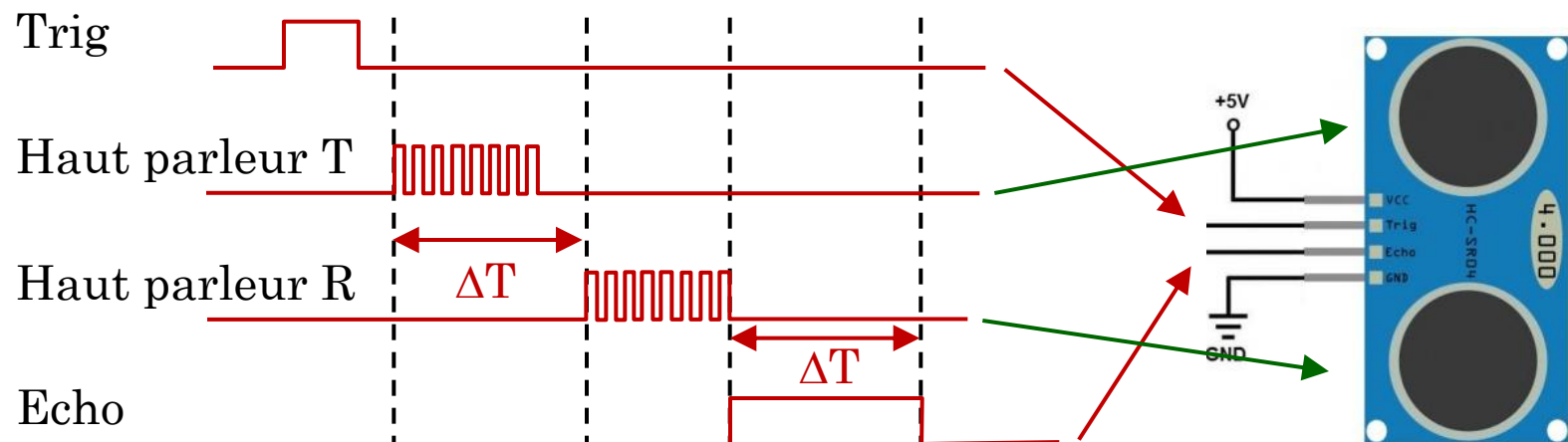
❑ Exemple de montage



7.2. Présentation du module HC-SR04

□ Fonctionnement

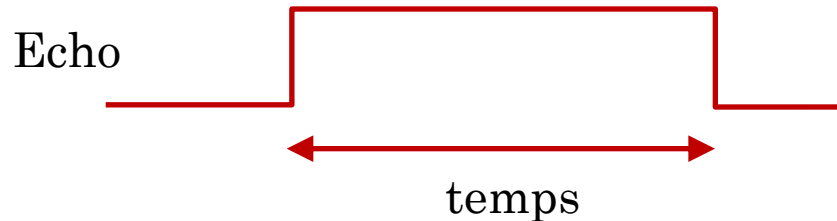
- Pour déclencher une mesure il faut appliquer une impulsion d'au moins 10 μ s sur l'entrée Trig
- Le module envoie alors 8 impulsions sonores
- On obtient en sortie (Echo) une impulsion dont la longueur correspond au temps que mettent les 8 impulsions pour faire l'aller/retour
- Il est préférable d'attendre 60 ms avant de demander une nouvelle mesure



7.2. Présentation du module HC-SR04

□ Détermination de la distance

- En une 1 s, le son parcourt 340 m donc en 1 μ s il parcourt 0.034 cm



- On en déduit la distance en cm :

$$\text{dis tan ce} = \frac{\text{temps}}{2} 0.034 = 0.017.\text{temps}$$

7.3. Les nouvelles fonctions

- **delayMicroseconds(X)** : permet de bloquer le déroulement du programme durant $X \mu s$
- **pulseIn(X,Y)** : retourne la durée d'une impulsion Y sur l'I/O X. Le temps retourné est en μs et va de $10 \mu s$ à 3 mn. Si Y = HIGH alors on attend une impulsion LOW-HIGH-LOW. Si Y = LOW, on attend une impulsion HIGH-LOW-HIGH. Le temps d'attente pour l'apparition de l'impulsion est de 1s. Au-delà de ce temps, la fonction retourne la valeur 0.

```
digitalWrite(trig, HIGH); // génération de l'impulsion Trig de 10  $\mu s$   
delayMicroseconds(10);  
digitalWrite(trig, LOW);  
lecture_echo = pulseIn(echo, HIGH); // lecture de la longueur temporelle de l'impulsion Echo
```

7.3. Les nouvelles fonctions

- **pulseIn(X,Y,Z)** : Z permet de modifier le temps (en μs) d'attente de l'apparition de l'impulsion. Si le temps d'attente est dépassé, la valeur retournée est 0

7.4. Librairie du HC-SR04

- Des personnes trouvent que les librairies de certains modules ne sont pas assez performantes et décident de faire mieux. C'est le cas pour le module HC-SR04 (dont nous avons pas utilisé la librairie arduino)
- Cette librairie s'appelle « NewPing » et il faut installer son répertoire dans le répertoire « librairie » du logiciel arduino
- Cette librairie s'accompagne de nouvelles fonctions

□ Nouvelles fonctions

- **NewPing sonar(X,Y,Z)** : initialise l'utilisation du sonar. X = trigger, Y = echo, Z est la distance maximale à mesurer (en cm). Il n'est pas obligatoire de donner une valeur pour Z qui par défaut est de 500 cm

```
NewPing sonar(trig, echo, 200);
```

- **sonar.ping_cm()** : donne la distance en cm ou 0 si il n'y a pas d'écho

```
cm=sonar.ping_cm();
```

7.5. Réception de données sur le port série

- Il est possible d'envoyer un ordre ou plus globalement des données à la carte arduino via le port série.
- Ces données sont stockées dans le buffer série de la carte arduino avant d'être lues. Elles peuvent être envoyées par le moniteur série du logiciel

□ Nouvelles fonctions

- **Serial.read()** : lit le premier caractère du buffer et renvoi le caractère -1 si il est vide

```
Serial.println("Souhaitez-vous la valeur en cm de la distance Y/N ?");
```

```
.....
```

```
test = Serial.read(); //on lit le 1er caractère du buffer
```

```
if (test == 'Y') {
```

```
    Serial.print(cm); //envoi de la distance à l'ordinateur
```

```
    Serial.println(" cm");}
```

- **Serial.available()** : permet d'obtenir le nombre de caractères dans le buffer

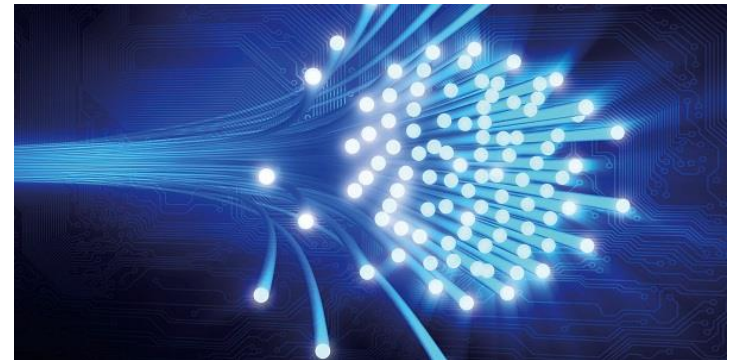
8.1. Communications optiques

- Envoi d'un code MORSE entre bateaux



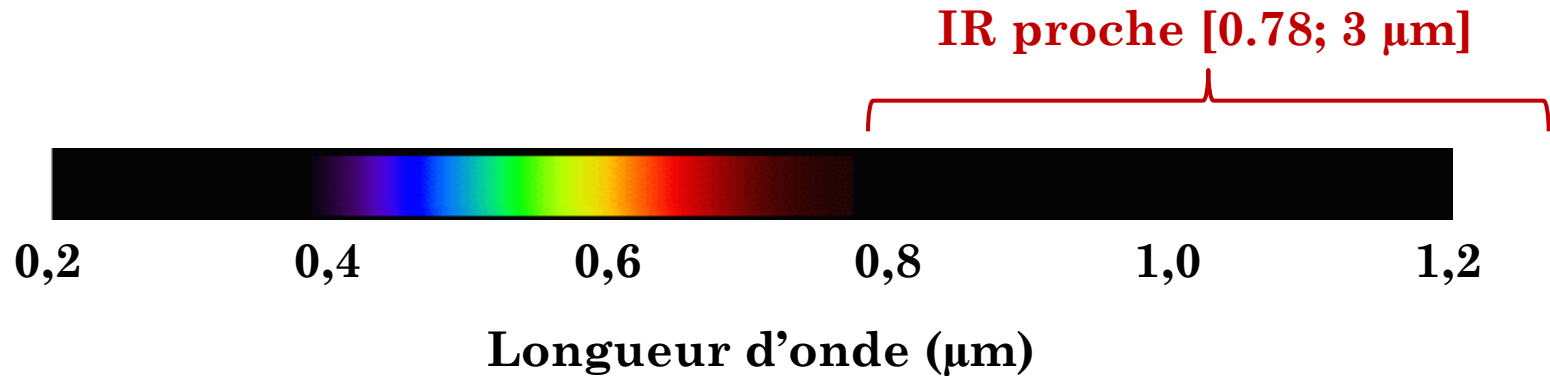
- Envoi d'un code entre télécommande et télé

- Envoi d'un code via fibre optique



8.2. Les diodes IR

- Les LED IR émettent dans l'Infra Rouge (infrarouge) qui est une longueur d'onde qui s'étale de $0.78 \mu\text{m}$ à 5 mm

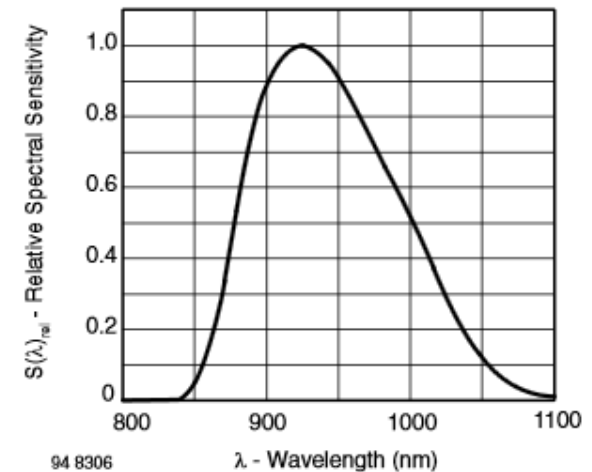


- Les diodes utilisées dans ce cours émettent une onde à 940 nm



8.3. Les phototransistor IR

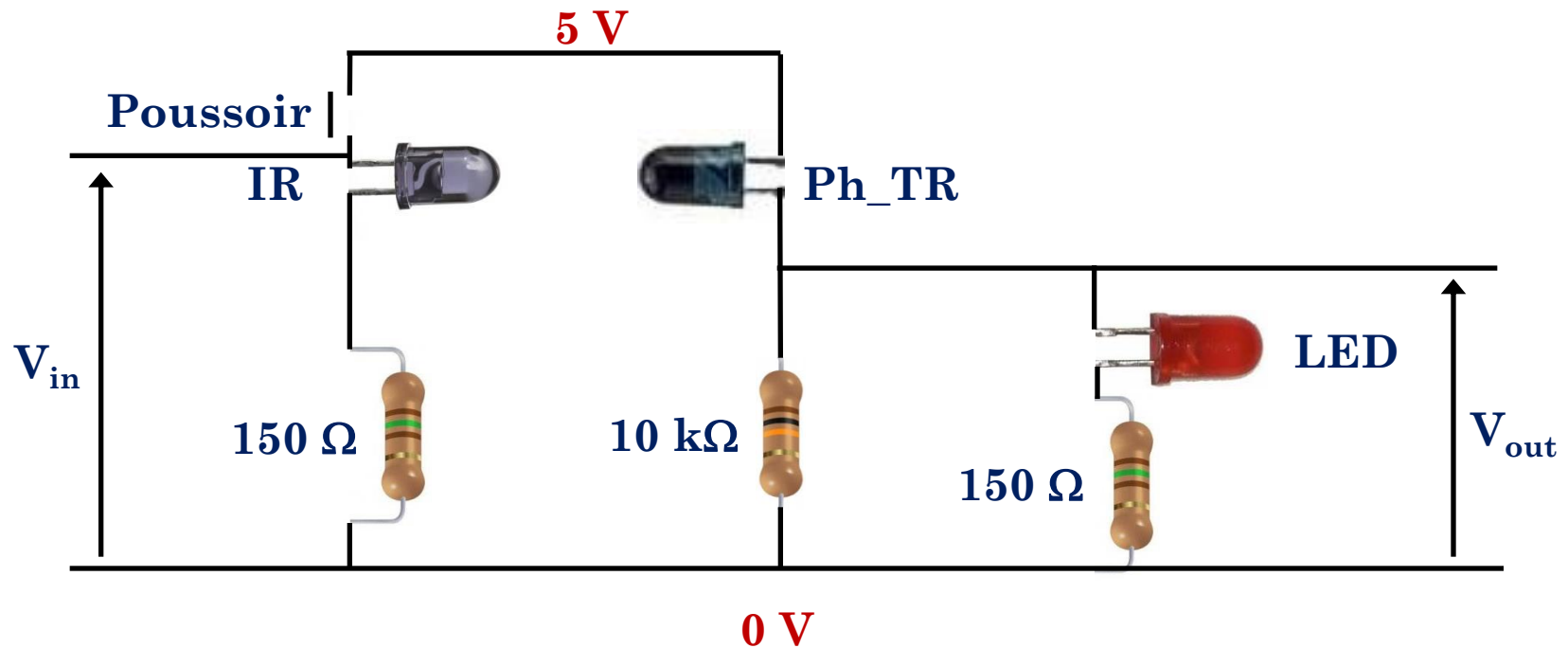
- Les phototransistors sont des transistors bipolaires dont la base est sensible à la lumière.
- Dans le cas des phototransistors IR, la base n'est sensible qu'à (ou presque) une longueur d'onde
- Les phototransistors utilisés dans ce cours sont sensibles à la longueur d'onde 940 nm



8.4. Prise en main des composants : code MORSE

□ Montage de base

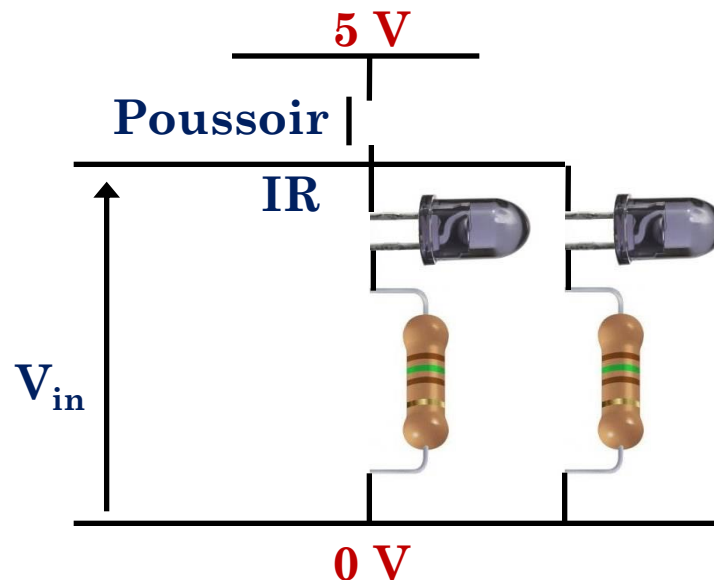
- Pour prendre en main la diode et le phototransistor IR, on peut commencer par un montage simple qui ne fait pas intervenir l'arduino (sauf pour l'alimentation)
- Si $V_{in} = "1"$ alors $V_{out} = "1"$ et si $V_{in} = "0"$ alors $V_{out} = "0"$



8.4. Prise en main des composants : code MORSE

□ Amélioration du montage

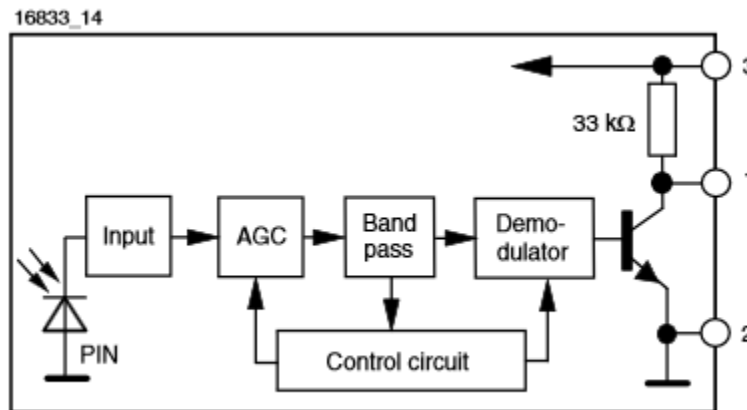
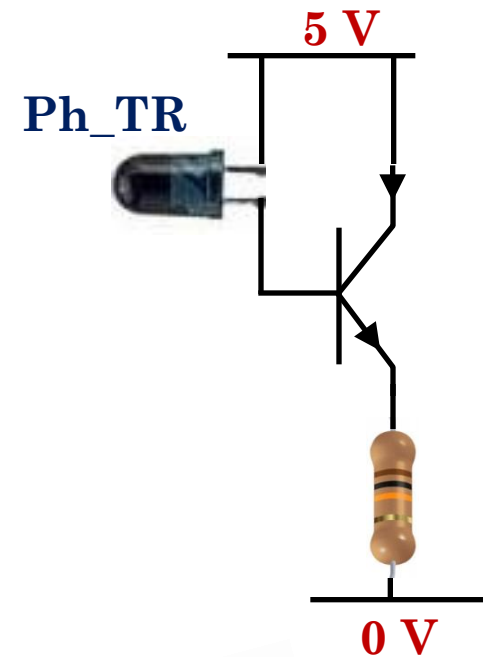
- Plus la diode IR s'éloigne, moins le phototransistor reçoit de photons. A partir d'une certaine distance la valeur de V_{out} n'est plus suffisamment grande pour être comprise comme un "1" logique
- On peut augmenter le nombre de photons en utilisant plusieurs LED IR



8.4. Prise en main des composants : code MORSE

□ Amélioration du montage

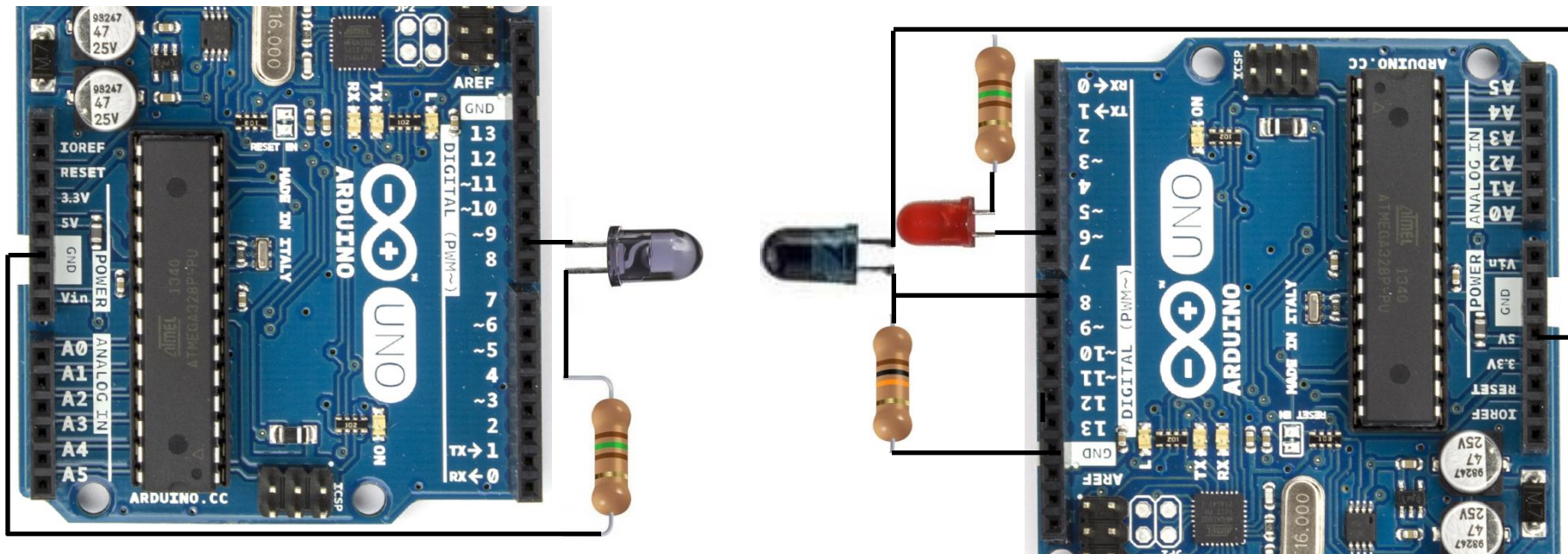
- On peut aussi augmenter le courant qui sort du phototransistor avec un montage Darlington
- Sur ce principe, il existe des composants qui intègrent déjà des amplificateurs.



8.5. Envoi d'un code entre 2 Arduino

□ Montage

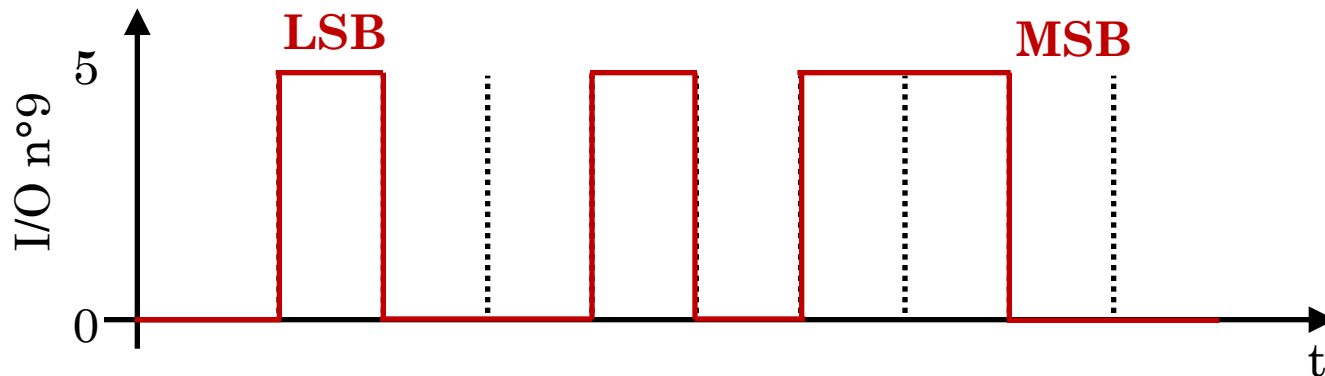
- Pour atteindre plus facilement les 5 V à la réception, on retire la LED rouge
- Bien que le montage ne fait apparaître la liaison que dans un sens, on peut ajouter le coupe diode/phototransistor pour émettre dans l'autre sens.
- On peut alors définir l'I/O n°9 pour la transmission et l'I/O n°8 pour la réception.



8.5. Envoi d'un code entre 2 Arduino

□ Communication

- Il est nécessaire de définir un protocole de discussion entre les 2 arduino
- On peut par exemple choisir d'envoyer un mot de 8 bits dont le premier sera un 1 pour que le récepteur détecte qu'on lui envoie quelque chose

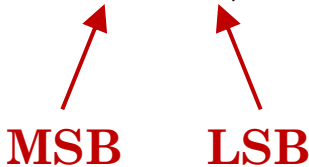


- Il faut aussi définir si le premier bit est un bit de poids faible (LSB : Least Significant Bit) ou un bit de poids fort (MSB : Most Significant Bit)
- La vitesse de discussion (bits par seconde) doit aussi être définie

8.6. Les nouvelles fonctions

- **byte** : définition d'un nombre binaire

byte b = B10010;



MSB LSB

- **bitRead**(X, N) : permet de lire un bit en particulier du mot binaire X. N est le n° du bit avec 0 pour le LSB
- **bitWrite**(X, N, Y) : permet d'écrire un bit en particulier du mot binaire X. N est le n° du bit où 0 correspond au LSB. Y correspond à la valeur qu'il faut écrire
- **bitSet**(X, N) : permet d'écrire un 1 sur le bit n°N du mot binaire X.
- **bitClear**(X, N) : permet d'écrire un 0 sur le bit n°N du mot binaire X.

8.7. La librairie SoftwareSerial

- Cette librairie permet de gérer le protocole de communication entre les deux arduino de façon aussi simple qu'avec l'ordinateur
- Cette librairie s'accompagne de nouvelles fonctions et il faut d'abord préciser au programme qu'il faut l'utiliser :

```
#include <SoftwareSerial.h>
```

- **SoftwareSerial** NOM(rxPin, txPin) : permet de définir un jeu de fonctions de communication qui porte le nom « NOM ». rxPin correspond à l'I/O de réception et txPin à l'I/O de transmission

```
SoftwareSerial mySerial(8,9); //RX TX
```

- NOM.**begin**(X) : définition de la vitesse de communication

```
mySerial.begin(4800)
```

8.7. La librairie SoftwareSerial

- `NOM.available()` : renvoi le nombre de bit disponibles dans le buffer
- `NOM.read()` : retourne un caractère lu sur l'entrée rxPin
- `NOM.print(X)` : permet d'envoyer la variable X sur la sortie txPin
- `NOM.println(X)` : permet d'envoyer la variable X sur la sortie txPin avec un « enter »
- Comme il peut être amené à manipuler des caractères, il faut pouvoir les définir
- `char` : définit une variable de type caractère

```
char c;
```

8.7. La librairie SoftwareSerial

- `NOM.available()` : renvoi le nombre de bit disponibles dans le buffer
- `NOM.read()` : retourne un caractère lu sur l'entrée rxPin
- `NOM.print(X)` : permet d'envoyer la variable X sur la sortie txPin
- `NOM.println(X)` : permet d'envoyer la variable X sur la sortie txPin avec un « enter »
- Comme il peut être amené à manipuler des caractères, il faut pouvoir les définir
- `char` : définit une variable de type caractère

```
char c;
```