Entrée Bouton - Résistance pull-up, pull-down et déparasitage suivi de Boutons, contacts et déparasitage matériel

Dominique Meurisse (MCHobby) mercredi 18 mai 2011

https://arduino103.blogspot.fr/2011/05/entree-bouton-resistance-pull-up-pull.html

Dominique Meurisse (MCHobby) samedi 31 décembre 2011

https://arduino103.blogspot.fr/2011/12/boutons-contacts-et-deparatisage.html

Entrée Bouton - Résistance pull-up, pull-down et déparasitage

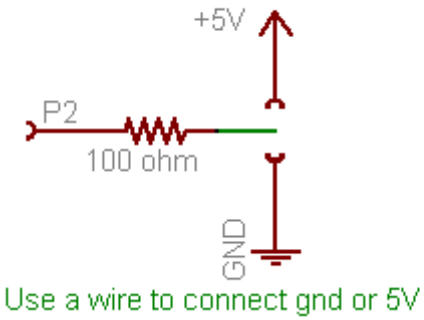
Entrée digitale

Avec Arduino, il est possible de configurer une pin en entrée ou en sortie.
Lorsqu'elle est configurée en entrée, il est possible de lire l'état haut/bas dans le programme.
Un état "haut" (HIGH) correspond au raccordement vers le +5 volts.
Un état "bas" (LOW) correspond au raccordement à la masse (GND, 0 Volts).

Il est fortement déconseillé d'appliquer plus de 5 volts sur une entrée...ce qui aurait pour effet d'endommager l'entrée, voire d'envoyer le micro-contrôleur "Ad Patres".

Protéger l'entrée digitale

Comme précisé, l'entrée digitale est configurée dans le programme.
Si par malheur cette entrée est raccordée au +5Volts et que le programme configure la pin en sortie (OUTPUT), on a toutes les chances de produire un court-circuit franc!
Le micro-contrôleur n'appréciera pas et encore une fois, il a de forte chance de rendre visite à Saint Pierre.
Pour protéger l'entrée d'un tel risque, l'on insère généralement une résistance de 100 Ohms entre l'entrée et le reste du circuit.



Source: www.ladyada.net

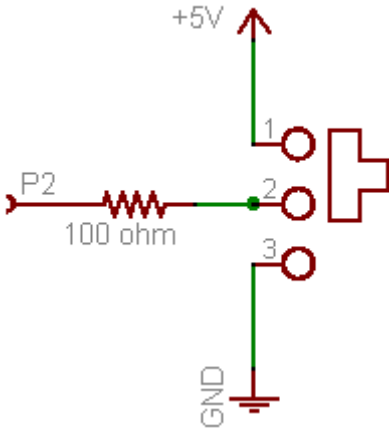
Raccordement du bouton

Maintenant, il ne reste plus qu'a raccorder l'entrée (protégée) soit au +5Volts, soit à la masse.
C'est bien entendu possible à l'aide d'un bouton deux directions ou un bouton a bascule (deux direction) comme celui présenté ci-dessous.
Les principaux désavantages d'une telle option sont:

1. Le prix beaucoup plus élevé qu'un simple bouton poussoir.

2. L'encombrement général.
3. D'une façon générale, les gens préfèrent pousser des boutons plutôt que de basculer des leviers :-)

Le raccordement ressemble alors à ceci:



Source: www.ladyada.net

Résistance de Pull-up et de Pull-down

Il est pourtant possible d'utiliser un simple bouton poussoir miniature pour atteindre exactement le même résultat qu'avec un bouton à levier deux directions.

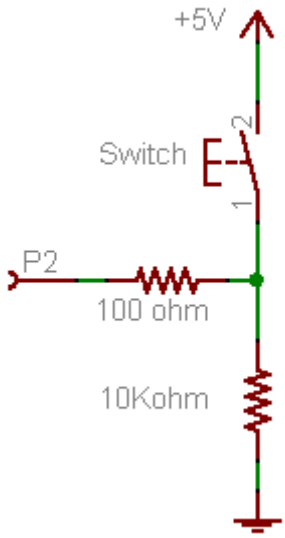
On utilise alors un montage dit "pull-down resistor".

Pull-down resistor

Dans un montage "pull-down resistor" (résistance pull-down), une résistance supplémentaire de 10KOhms est utilisée pour amener l'entrée à la masse (par défaut).

Si l'utilisateur presse le bouton, 5Volts sont alors appliqués sur l'entrée. Si l'utilisateur relâche le bouton, la résistance pull-down ramène l'entrée à la masse.

Notez que la résistance de protection de 100 Ohms est toujours insérée dans le circuit.



Source: www.ladyada.net

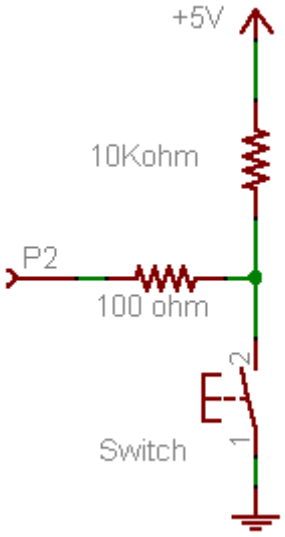
Pull-up resistor

Le montage Pull-Up resistor est le montage complémentaire du Pull-Down Resistor.

Dans son fonctionnement par défaut, un Pull-Up resistor applique 5Volts sur l'entrée tant que l'utilisateur ne presse pas le bouton.

Lorsque l'utilisateur presse le bouton, l'entrée est raccordée à la masse.

C'est un cas typique des commandes de Reset, le Reset n'étant effectif/signalé que lorsque le signal appliqué passe à la masse (c'est en autre le cas d'Arduino).



Source: www.ladyada.net

Résistance de Pull-up/Pull-Down

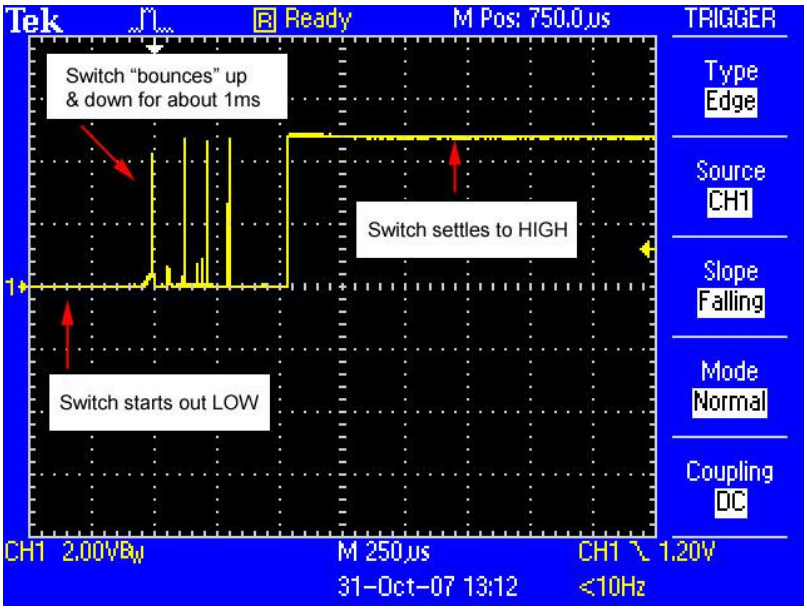
La résistance est ici de 10KOhms, c'est une valeur assez commune permettant d'identifier facilement un circuit de pull-up/down.

Il est possible de diminuer cette résistances à 4.7 KOhms mais cela consommera aussi plus de courant.

Il n'est pas conseillé d'utiliser une résistance de pull-up de plus de 10KOhms.... à partir de 100 KOhms, la résistance de pull-up/down interfère avec la circuiterie interne du micro-contrôleur et le résultat (la détection) deviendrait incertain.

Déparasitage des boutons

Lorsque l'on presse/relâche un bouton, il y a souvent l'apparition de parasites pendant quelques milisecondes.
Ce parasites n'apparaissent que durant les moments où l'on enfonce/relâche le bouton poussoir.



Source: le tutorial de www.ladyada.net

Si l'on compte le nombre de pressions (pour faire un compteur), ces parasites viennent justement perturber le bon fonctionnement du logiciel. Le problème est matériel... et iles parasites viennent ajouter des pressions fantômes (voir leçon 5 ci-avant).

Il y a deux façons de corriger le problème:

1. Utiliser une capacité de déparasitage
2. Introduire un délai logiciel dans le programme.

Voici un article d'IkaLogic.com expliquant comment déparasiter des circuits (façon logiciel et matérielle)

Déparasitage logiciel

Puisque le phénomène transitoire ne dure que quelques micro-secondes, il suffirait de faire deux lectures successives de l'entrée (après un délai de quelques millisecondes) et de s'assurer qu'il ne s'agit pas d'une phase transitoire.
S'il ne s'agit pas d'une phase transitoire, la lecture de l'état de l'entrée 10 ms plus tard doit être identique à celle 10 ms plus tôt.
Personne n'arrivant à presser et relâcher un bouton en moins de 10 ms.

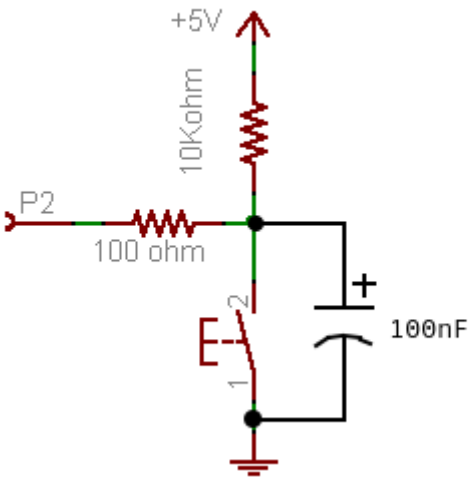
Voici donc un bout de code (francisé) issus du tutoriel de de Ladyada.

```
/*
 *  Eclairage Velo avec bouton déparasité
 *  Presser une fois
 */

  int switchPin = 2; // Bouton connecté à la pin 2 (pull-up, LOW=bouton
pressé)
  int led1Pin = 12;
  int led2Pin = 11;
  int led3Pin = 10;
  int led4Pin = 9;
  int led5Pin = 8;

  int val;        // variable pour lire l'état de l'entrée
  int val2;       // variable pour lire l'état de l'entrée (après un delai)
  int buttonState;   // variable pour mémoriser l'état du bouton
  int lightMode = 0;              // La lampe est elle allumée?

void setup() {
  pinMode(switchPin, INPUT);    // Le bouton est une entrée

  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
  pinMode(led3Pin, OUTPUT);
  pinMode(led4Pin, OUTPUT);
  pinMode(led5Pin, OUTPUT);

  buttonState = digitalRead(switchPin);   // lecture de l'état initial
}

void loop(){
  val = digitalRead(switchPin);      // Lecture de la valeur d'entrée
  delay(10);                         // 10 millisecondes
  val2 = digitalRead(switchPin);     // Relecture de l'entrée pour
vérification de parasitage

  if (val == val2) {                 // S'assurer que l'on a 2 lectures
successives consistante!

    if (val != buttonState) {          // Le bouton a changer d'état!
      if (val == LOW) {                // Le bouton est il pressé?
        if (lightMode == 0) {          // La lumière est-elle éteinte?
          lightMode = 1;               // Allumer la lumière!
          digitalWrite(led1Pin, HIGH);
```

```
          digitalWrite(led2Pin, HIGH);
          digitalWrite(led3Pin, HIGH);
          digitalWrite(led4Pin, HIGH);
          digitalWrite(led5Pin, HIGH);
        } else {
          lightMode = 0;                // Éteindre la lumière!
          digitalWrite(led1Pin, LOW);
          digitalWrite(led2Pin, LOW);
          digitalWrite(led3Pin, LOW);
          digitalWrite(led4Pin, LOW);
          digitalWrite(led5Pin, LOW);
        }
      }
    }
    buttonState = val;                  // Sauver le nouvel état du bouton
dans une variable
  }
}
```

Déparasitage matériel - version 1

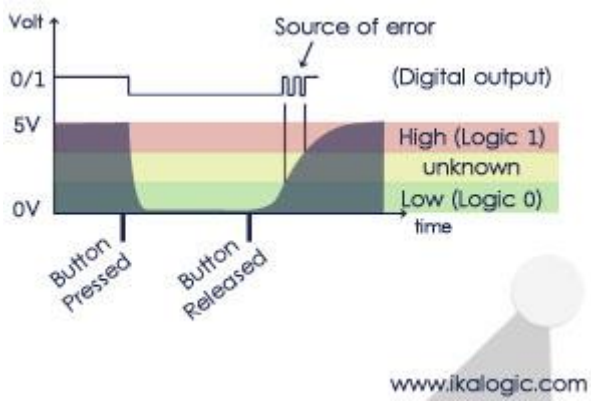
Le déparasitage se fait à l'aide d'une capacité qui absorbera les impulsions parasites.
Mais à elle seule, elle n'est pas suffisante.
Dans le cas d'un montage pull-up, l'on place une capacité de 100nF comme suit (100 nF pour une résistance de pull-up de 10KOhm)



Lorsque l'on presse le bouton:
La capacité et instantanément déchargée (court-circuité par le bouton).
L'entrée passe en LOW.
Si il y a des parasites (pics a +5 volts) au niveau du bouton, ces derniers seront absorbés par la capacité.

Lorsque l'on relâche le bouton:
La tension est appliquée sur la capacité et cette dernière va se charger assez vite.
Cependant, la charge n'est pas instantanée mais progressive.

Cela veut dire que les 5 volts sont appliqués progressivement à l'entrée comme le montre le graphique issu de IkaLogic.



Source: IkaLogic

Sur le graphique, l'on voit clairement que la courbe de charge passe un certain temps dans la zone d'incertitude où le micro-contrôleur ne sait pas si c'est toujours un 0 ou déjà un 1 logique.
Le parasitage existe donc toujours au moment ou l'on relâche le bouton (mais sous une autre forme).
Pour éviter ce nouveau type de parasitage, il fait insérer un trigger de Schmitt sur l'entrée. Ainsi, tant que la tension sera en dessus du seuil minimal du 1 logique, le trigger ne change pas d'état vers 1... et inversement tant que la tension ne sera pas passée sous le seuil minimum du trigger (0 logique), le trigger ne repassera pas à 0.

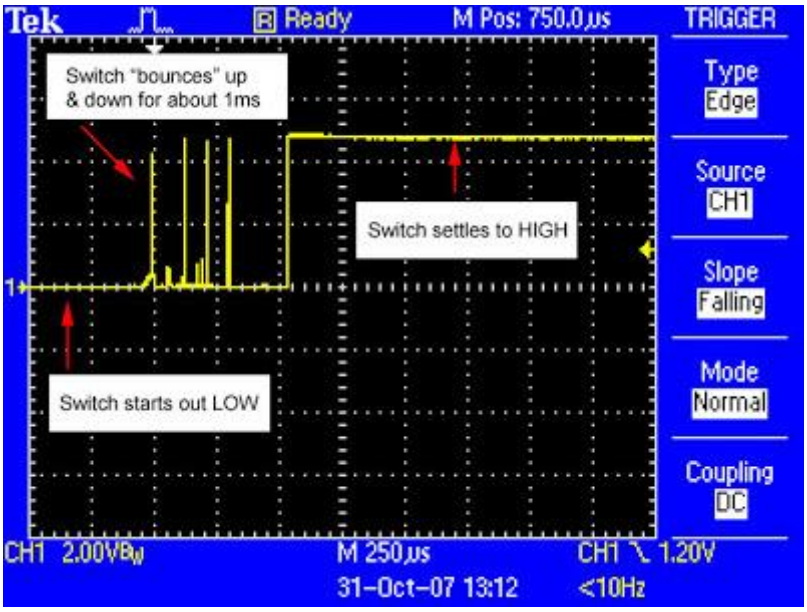
Déparasitage matériel - version 2

Un deuxième article beaucoup plus précis est paru sur le déparasitage matériel.
Vous pouvez le consulter ici: Boutons, contacts et déparasitage matériel.

Boutons, contacts et déparasitage matériel

Introduction

Dans le précédent article "Entrée Bouton - Résistance pull-up, pull-down et déparasitage", j'abordais la problématique du parasitage des contacts ainsi que la méthode de déparasitage logiciel qui pouvait être utilisée pour corriger ce problème.



Source: le tutorial de www.ladyada.net

Quand le déparasitage logiciel est interdit

Cependant, il existe des cas où l'utilisation du déparasitage logiciel est interdit.
C'est le cas, en autre, du traitement des interruptions sur Arduino.
En effet, la fonction `delay()` utilisée durant pour déparasité logiciel est basée sur les timers (horloge interne d'Arduino), ces mêmes timers qui sont également utilisé pour le traitement des interruptions. Raison technique de son interdiction dans le traitement des interruptions.
Il faut donc utiliser un déparasitage matériel.... méthode assez simple.

Le traitement des interruptions

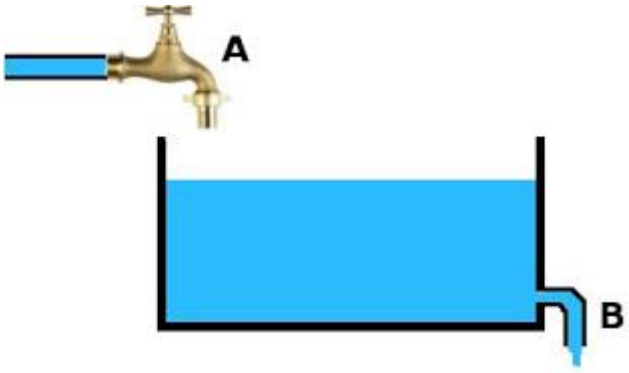
Le traitement des interruptions fera l'objet d'un autre article à venir (*voir Les interruptions sur Arduino*).
Le présent article se concentrera uniquement sur le déparasitage matériel.

Principe de base

Le principe de base du déparasitage matériel est basé sur l'utilisation d'un circuit RC (résistance + condensateur).
En effet, lorsque l'on branche un condensateur, il emmagasine progressivement de l'énergie et lorsque l'on débranche un condensateur il restitue cette énergie (progressivement aussi).
En gros, le condensateur fait office de réservoir, c'est cette capacité d'emmagasiner (ou restituer) progressivement l'énergie qui sera mis à contribution pour le déparasitage.

La métaphore du réservoir pour aider la compréhension

Imaginons un réservoir d'eau qui fera office de le condensateur.
Un robinet de remplissage (A) sera utilisé pour symboliser notre interruption (car nous pouvons l'ouvrir et le fermer à notre guise).



Lorsqu'il y a de l'eau dans le réservoir, la sortie B à un débit constant.
Peu importe que que le robinet A soit ouvert normalement ou si l'on joue avec lui pour faire varier son débit... le réservoir se vide à "vitesse constante" par l'évacuation B. Cela est possible parce que le réservoir à accumulé de l'eau.
Les perturbations sur le robinet A n'ont pas d'impact direct sur l'évacuation B, celle-ci réagit avec un certain temps de retard (car bien entendu, tôt ou tard le réservoir sera vide où presque vide et les perturbation arriverons jusque là).

C'est un principe similaire qui s'applique avec le condensateur et le bouton poussoir.

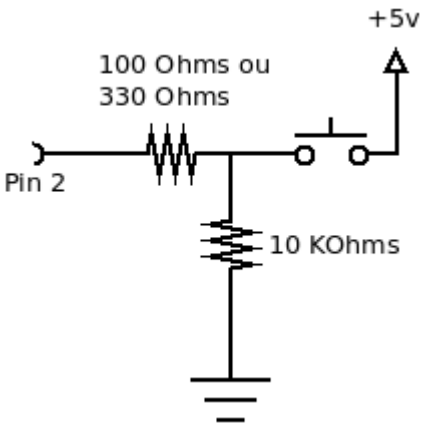
Comme il est plus facile d'imaginer la situation en cours de fonctionnement, imaginons donc que le bouton poussoir est enfoncé, le condensateur chargé et que l'on s'apprête à relâcher le bouton poussoir.
En gros, c'est comme refermer le robinet A et les parasites seraient simulés par une série d'ouverture/fermeture rapide du robinet avant sa fermeture définitive.
Le condensateur pendant toute la période des parasites, le condensateur relâchera "avec constance" l'energie accumulé... peu importe les perturbations à l'entrée. Et lorsque le condensateur sera vide (tout comme le réservoir), il sera simplement vide.
C'est comme le réservoir lorsqu'il est vide, l'écoulement s'interrompt.

Ce qui est vrai pour la décharge l'est aussi pour la charge... que cela soit pour le réservoir ou le condensateur. Les petites perturbations ne viennent pas perturber l'écoulement en sortie.

C'est magique, c'est ce qui va permettre le déparasitage matériel.

Montage standard et montage déparasité

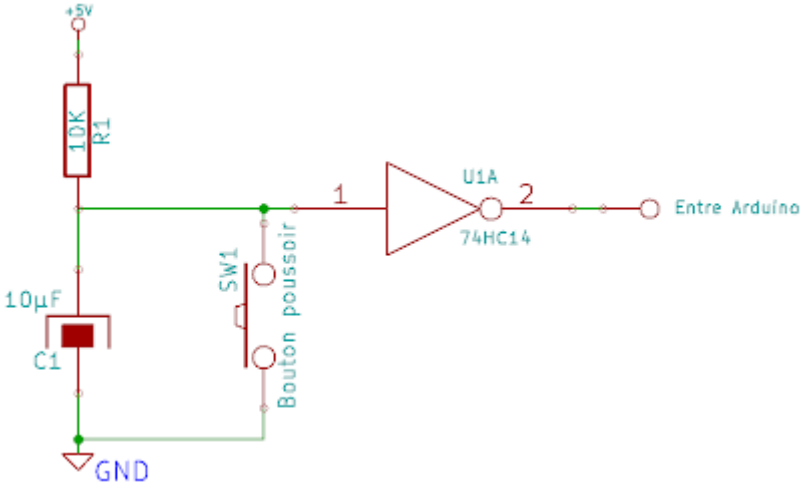
En montage standard, le bouton poussoir se monte souvent avec une résistance pull-down (pour que l'entrée Arduino soit tirée à la masse lorsque le bouton n'est pas pressé).



Montage standard - Pull-down

Voir article Entrée Bouton - Résistance pull-up, pull-down et déparasitage pour plus détails.

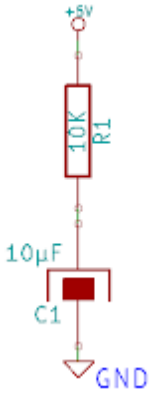
Pour déparasité matériellement une entrée, il faut utiliser un montage pull-up, un circuit RC et un trigger de schmitt inverseur.
Le principe de ce montage est détaillé tout au long de cet article.



Entré déparasité matériellement

Circuit RC

La base du montage déparasité matériellement est un circuit RC.
Un circuit RC est une résistance couplée à un condensateur.



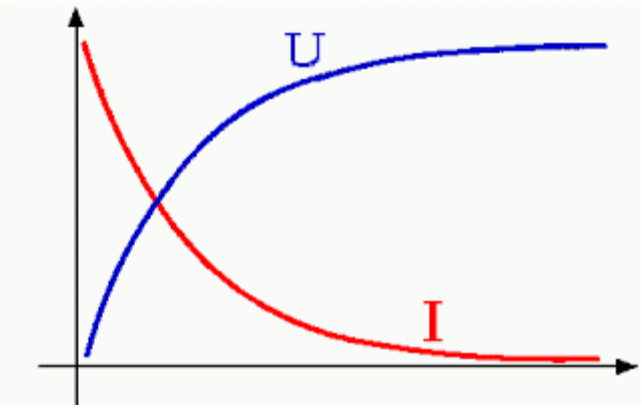
Circuit RC

Pourquoi une résistance?

C'est cette résistance est essentielle dans le circuit. En effet, en absence de résistance, un condensateur mis sous tension essaye d'accumuler sa charge aussi vite que possible (donc instantanément s'il en la possibilité).
En conséquence, le courant d'appel sera très important... important au point de pouvoir considérer le condensateur comme un "réel court-circuit" pendant son temps de charge.
Bien évidemment, vous imaginez fort bien qu'un tel comportement énergivore est totalement incompatible avec les limitations des alimentations.
On place donc une résistance en série avec le condensateur. Cela limite donc le courant d'appel et le condensateur se charge progressivement.

Conséquences de la résistance?

Puisque la résistance limite le courant de charge, le condensateur se chargera progressivement.
En conséquence, la tension aux bornes du condensateur évoluera de 0 volts à la tension finale (évolution logarithmique, voir graphique)... un peu comme le réservoir d'eau se remplir progressivement. La hauteur du liquide représente la tension aux bornes du condensateur.
Le temps que mettra le condensateur à charger dépend de la capacité du condensateur et de la résistance associée.



Charge du condensateur (courbe bleue)
Source: AccroDavion

La constante de temps

Comme précisé juste avant "Le temps que mettra le condensateur à charger dépend de la capacité du condensateur et de la résistance associée".
Pour un condensateur identique, plus résistance est élevée (pensez "serrer le robinet") et plus le courant sera faible (pensez "débit d'eau faible").
Le condensateur mettra donc plus de temps à charger (pensez à l'eau qui monte dans le réservoir).
Si la résistance est plus grande, la tension mettra donc plus de temps pour atteindre son maximum aux bornes du condensateur.

Il est possible d'évaluer ce temps. Pour cela, on utilise la constante de temps identifié à l'aide le la lettre grecque Tau.

```
Tau = Résistance (en Ohm) * Capacité (en farad).
```

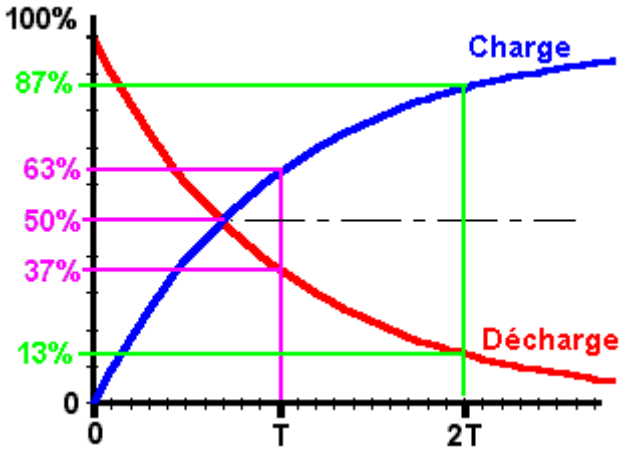
Dans le montage type qui nous concerne, nous avons donc:
`Tau = 10 k * 10 µF = 10000 * 0.000010 = 0.1 sec`

Temps de charge et constante de temps

On considère généralement que le condensateur:

- a 37% de sa charge au bout de 1 * la constante de temps.
- est presque complètement chargé au bout de 2 * la constante de temps.
- est totalement chargé au bout de 5 * la constante de temps.

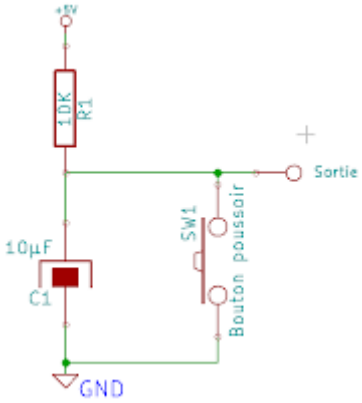
Le "manuel internet des radioamateurs" propose le graphique récapitulatif suivant qui illustre parfaitement la relation entre Tau et Charge.



Source: Manuel internet des radioamateurs

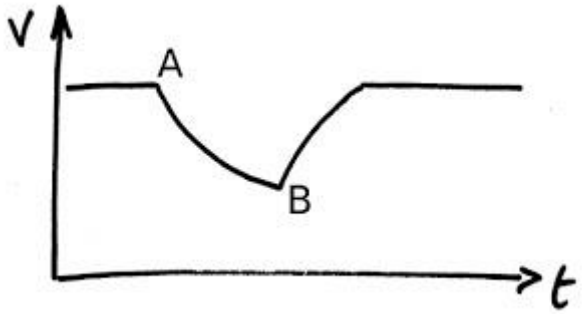
Montage RC Pull-down

Le schéma utilisé pour le déparasitage est un montage de type pull-down.



RC Pull-down

Donc en principe, la tension de sortie est +5v et lorsque l'on presse le bouton poussoir, le circuit RC se décharge en suivant la courbe de décharge.



Charge et Décharge du RC Pull-Down

Si l'utilisateur presse le bouton poussoir (point A), le circuit commence à se décharger. Lorsqu'il relâche le bouton poussoir (point B), le circuit commence à se recharger.
Grâce à la nature même du circuit RC, tous les parasites que le bouton pourrait générer sont absorbés par le circuit.

Considération sur la constante de temps

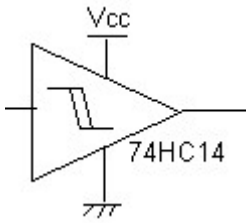
Avoir la sortie du circuit qui se charge et décharge amène une autre question.
Est-il judicieux d'utiliser un Circuit RC pour activer une entrée sur Arduino?
En effet, s'il met trop de temps à se décharger, l'utilisateur pourrait presser plusieurs fois sur le bouton sans que l'entrée Arduino n'ai détecté l'action!
C'est là qu'intervient la constante de temps.
Selon les calculs faits plus haut, une résistance de 10KOhms et un capacité de 10µF correspondent à une constante de temps de 0.1 sec.
Donc, au bout de 0.1 sec, la circuit est déchargé à 37% et au bout de 0.2 sec (2 * Tau) il peut être considéré comme entièrement déchargé.
0.1 sec (ou 0.2 sec en cas extrême) c'est:

1. Largement suffisant pour absorber toutes les perturbations de faux contact.
2. C'est suffisamment court en regard du temps nécessaire pour qu'un humain presser et relâche le bouton poussoir.

La combinaison RC 10 KOhms + 10µF est donc idéale dans le cas qui nous concerne.

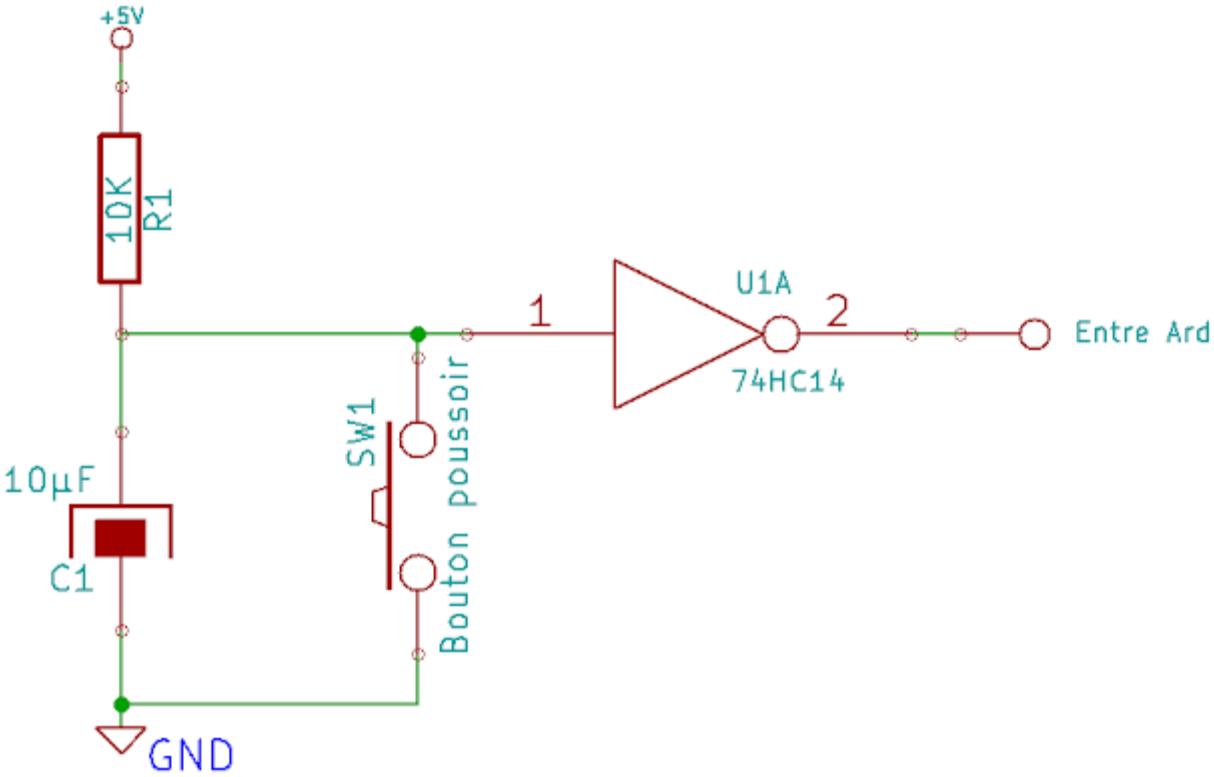
Le trigger de Schmitt

Le dernier élément de ce montage est un trigger de Schmitt inverseur (74HC14).



Symbole d'un Trigger de Schmitt (non inverseur)

Ce Trigger s'insère entre la sortie du circuit RC et l'entrée d'Arduino



Entrée déparasité matériellement

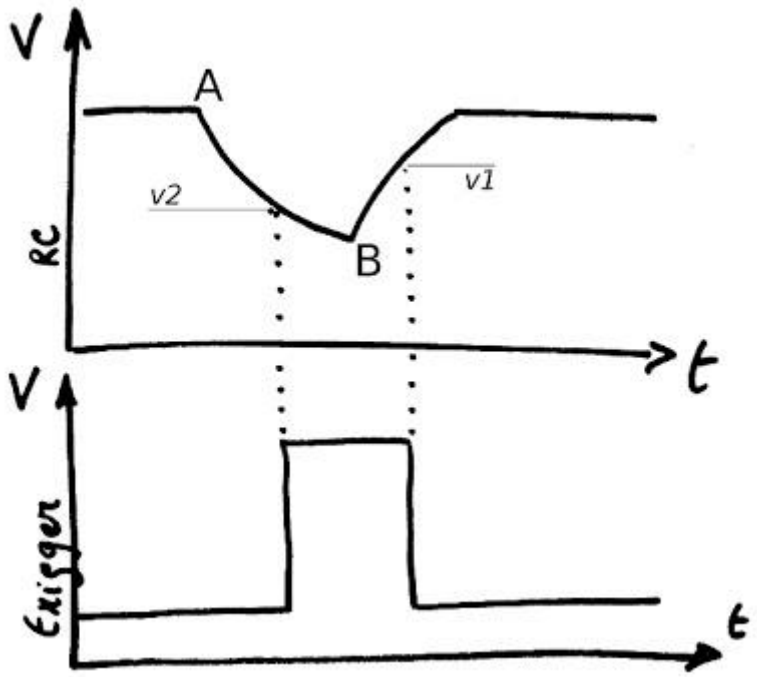
L'inverseur du trigger

"Inverseur" signifie simplement qu'il inverse l'état logique de sa sortie.
C'est bien pratique dans notre cas puisque lorsque le bouton pressoir est pressé, la tension de sortie diminue.

En conclusion, quand la tension de sortie du circuit RC sera BAS, la sortie de trigger sera HAUT.
Et lorsque le bouton sera relâché, la tension de sortie du circuit RC sera HAUT et la sortie du trigger sera BAS.
C'est particulièrement pratique pour nous, il suffira de brancher la sortie du Trigger sur une entrée Arduino et de simplement tester les valeurs HIGH (bouton pressé) et LOW (bouton relâché).

Les seuils du trigger

Un des autres avantages du trigger de Schmitt est qui utilise des tensions de seuils sur son entrée.
Pour résumé, il faudra que la tension d'entrée soit descendue en dessous d'un certain seuil pour que la sortie change d'état. Et a l'inverse, durant la recharge du circuit RC, il faudra que la tension ait dépassé un certains seuil pour que la sortie change encore une fois d'état.
Le trigger de Schmitt va donc transformer l'onde du circuit RC en onde Carrée.



Résultat en sortie du trigger inverseur
(par rapport au signal RC)

Note sur le trigger de Schmitt

Un des avantages indéniable du trigger de Schmitt est qu'il utilise des tensions de seuil différentes pour passer d'un état à l'autre.... évitant ainsi au circuit de basculer entre deux états lorsque la tensions d'entrée avoisine une tension de seuils qui aurait été commune.
Ainsi, pour un trigger NON INVERSEUR:

- v1 serait la tension de seuil pour passer à l'état haut
- v2 serait la tension de seuil pour passer à l'état bas.
- v1 > v2.

Donc quand la tension d'entrée augmente, elle doit au moins atteindre la tension v1 (supérieure à v2) pour que le trigger passe à l'état haut.
Et inversement, lorsque la tension d'entrée redescend, elle doit retomber en dessous de v2 (inférieure à v1) avant que le trigger le passe à l'état bas.
Le trigger ne change jamais d'état entre v1 et v2, c'est une zone de sécurité où il ne se passe rien. On parle aussi de cycle d'hystérésis pour identifier cette particularité.

Conclusion

Voila, ce fut un très long article par ailleurs laborieux à écrire durant ces périodes de fêtes (*je remercie mon épouse pour le temps qu'elle a concédé à l'écriture de cet article*)
J'espère qu'il est suffisamment clair et précis car il servira bientôt dans l'article concernant la programmation des interruptions sur Arduino :-) .