Les premiers pas en électronique

Ateliers Open-Handicap et Option Maker
FabLab EirLab
ENSEIRB-MATMECA

27/04/2025 - rév. 0.94 - version longue

Adrien Boussicault
Enseignant-Chercheur – Fonctionnaire d'état

adrien.boussicault@u-bordeaux.fr

Code source associé au cours : code.zip

DOCUMENT EN COURS D'ÉCRITURE ET DE RELECTURE !

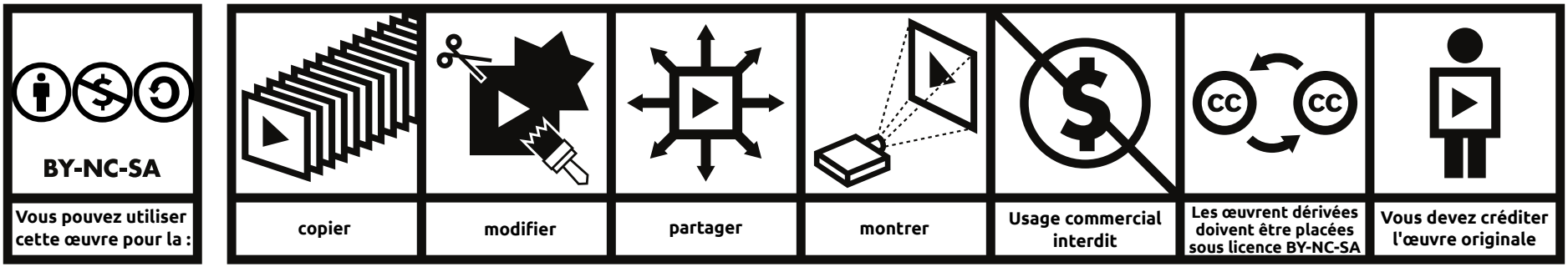
Téléchargez la version la plus récente ici : cours_elec.pdf

Plan

(version longue)

Licences et dépôt

Ce cours est sous licence Creative Commons BY-NC-SA dans sa version 4.0. Les droits et devoirs associés à cette licence sont résumés dans l'image[1] suivante :



Attribution - Pas d'utilisation commerciale - Partage dans les mêmes conditions 4.0 – Cette licence autorise autrui à copier, modifier, partager votre œuvre, sauf pour des utilisations commerciales. Les personnes utilisant votre œuvre s'engagent à la créditer, à intégrer un lien vers la licence et à indiquer si des modifications ont été effectuées à l'œuvre. Si elles réalisent des modifications, l'œuvre dérivée devra être diffusée sous licence CC-BY-NC-SA également. **https://creativecommons.org/licenses/by-nc-sa/4.0/deed.fr**

La dernière version longue de ce document est téléchargeable ici : https://www.labri.fr/perso/boussica/archives/cours/option_maker/cours_elec.pdf

Le dépot git du code source est accessible à cette adresse : https://gitub.u-bordeaux.fr/openhandicap/cours_electronique

Le code source des scripts, des programmes arduino et esp32 associés au cours est sous licence libre MIT (à l'exception d'un fichier sous licence libre LGPLv3). Concernant ces fichiers là, il n'y a pas de restriction commerciale liée à leur utilisation dans vos projets. Le code se télécharge via le dépôt git précédent ou directement à cette adresse : https://www.labri.fr/perso/boussica/archives/cours/option_maker/code.zip.

(présent dans la version courte)

Mise en garde !

Ce document est en cours d'écriture et de relecture !

Bien que l'auteur essaye d'être le plus rigoureux possible et essaye de justifier les schémas et affirmations donnés, ses domaines de spécialité sont les mathématiques et l'informatique.

Vous devez donc être très critique en lisant ce document. Vous devez comprendre le fonctionnement des circuits et multiplier et croiser les sources d'informations.

Ce document est conçu pour aider des ingénieurs à développer et concevoir de nouveaux objets. Les circuits doivent donc être validés par le lecteur avant d'être construit et utilisés. Ils doivent être testés intensivement avant d'être diffusés au grand public. L'auteur ne peut pas être tenu pour responsable des dégâts occasionnés par les circuits conçus, produits ou reproduits par le lecteur.

Si vous rencontrez des erreurs à la lecture de ce document, n'hésitez pas à contacter l'auteur pour qu'il le corrige et l'améliore.

(version longue)

Plan

(présent dans la version courte)

L'électronique et l'électrotechnique

L'électronique (resp. électrotechnique) c'est l'étude des transferts d'information (resp. d'énergie) entre différents systèmes utilisant comme support les électrons et le champs électromagnétique.

L'électronique et l'électrotechnique

L'électronique (resp. électrotechnique) c'est l'étude des transferts d'information (resp. d'énergie) entre différents systèmes utilisant comme support les électrons et le champs électromagnétique.

Énergie : valeur conservative (on ne peut pas créer ou détruire de l'énergie) qui sert à décrire l'interaction entre différents systèmes physiques. Unité : Joule (J).

La puissance : la quantité d'énergie par seconde transférée d'un système à l'autre. Unité Watt : $1W = 1J/s$.

L'électronique et l'électrotechnique

L'électronique (resp. électrotechnique) c'est l'étude des transferts d'information (resp. d'énergie) entre différents systèmes utilisant comme support les électrons et le champs électromagnétique.

Énergie : valeur conservative (on ne peut pas créer ou détruire de l'énergie) qui sert à décrire l'interaction entre différents systèmes physiques. Unité : Joule (J).

La puissance : la quantité d'énergie par seconde transférée d'un système à l'autre. Unité Watt : $1W = 1J/s$.

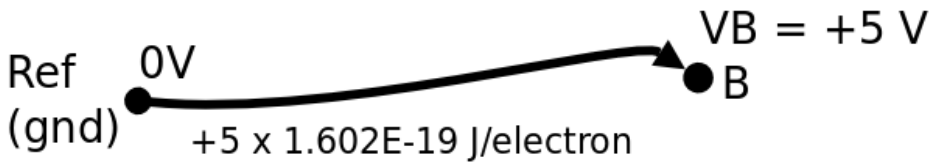
Consommation grille pain : $700W$.
Consommation d'un humain actif : 2700 KCalorie $= 130W$.
Consommation d'un humain peu actif : 2300 KCalorie $= 111W$.
Puissance voiture : 40 à 100 kW.

Le travail d'une petite voiture $\approx$ le travail de $2105 = \frac{40000}{130-111}$ humains.

(présent dans la version courte)

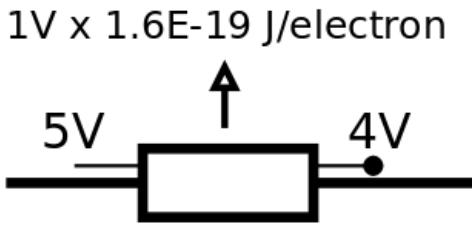
L'énergie et la tension

La tension au point B c'est l'énergie électromagnétique (e.m.) nécessaire pour déplacer un électron d'un point de référence au point B. Elle se mesure en Volt ($1V = 1.602 \times 10^{-19}$ Joules par électron $\times\, \alpha$).



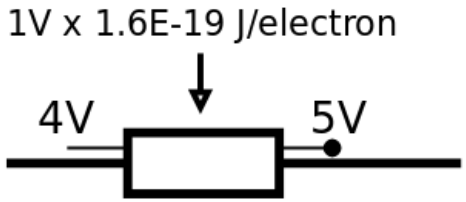
Si un électron, possédant une énergie e.m. de 5V, et se déplace vers un point à 4V, il cède $5V - 4V = 1V$ d'énergie au système traversé.

Système passif :



Les électrons cèdent de l'énergie

Système actif :



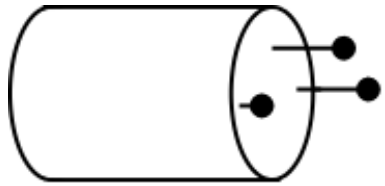
Les électrons récupèrent de l'énergie

Comment fonctionne vraiment l'électricité au niveau microscopique :
www.youtube.com/watch?v=-oRYuFFSokc (consulté le 21/03/2025)

(présent dans la version courte)

Le courant et la puissance

Le courant est le nombre d'électron par seconde qui traverse un système. Il se mesure en Ampère ($1A = 6.241 \times 10^{18}$ électrons par seconde $\times \alpha^{-1}$)



La puissance est la quantité d'énergie par seconde transférée de l'électron au système. Il se calcule ainsi :

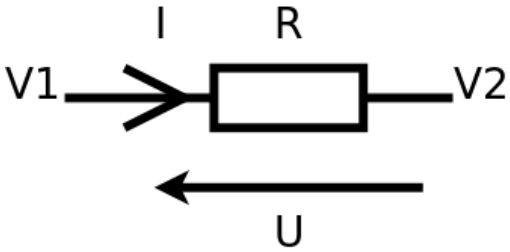
$$\underset{\frac{\text{quantité d'énergie}}{\text{seconde}}}{P} = \underset{\frac{\text{quantité d'énergie}}{\text{électron}}}{U} \times \underset{\frac{\text{nb d'électrons}}{\text{seconde}}}{I}$$

La puissance se mesure en Watt ($1W = 1J/s = 1V.A$)

Comment fonctionne vraiment l'électricité au niveau microscopique :
www.youtube.com/watch?v=-oRYuFFSokc (consulté le 21/03/2025)

(présent dans la version courte)

La loi d'ohm et la loi des noeuds



Loi d'ohms :

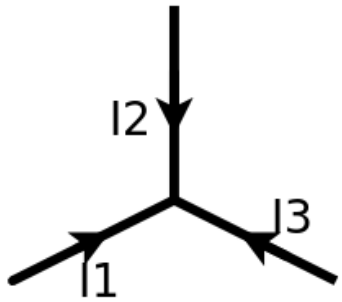
$$U = R \times I$$

$U = V_1 - V_2$: tension au borne de la résistance.

V_1, V_2 : potentiels - énergie des électrons.

I : courant

R : Résistance en Ohms (Ω)



Lois des noeuds :

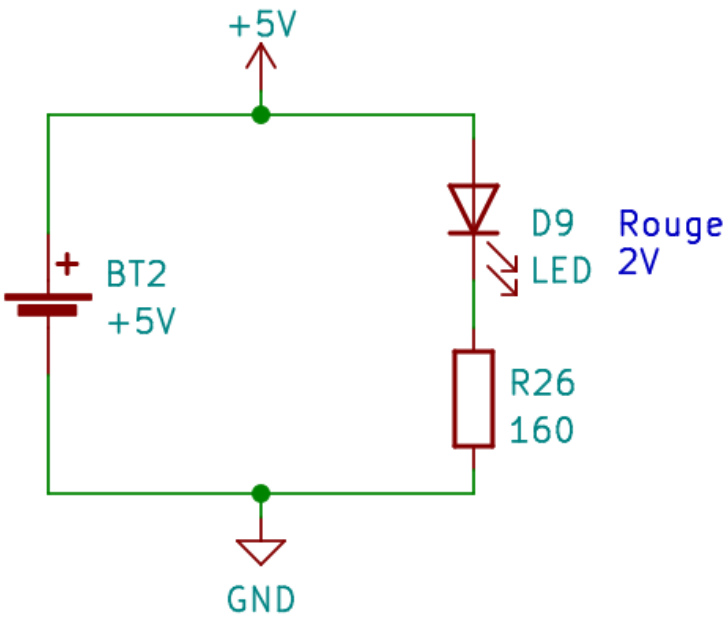
$$I_1 + I_2 + I_3 = 0$$

I_1, I_2, I_3 : courants

(présent dans la version courte)

Un premier exemple avec une diode électroluminescente

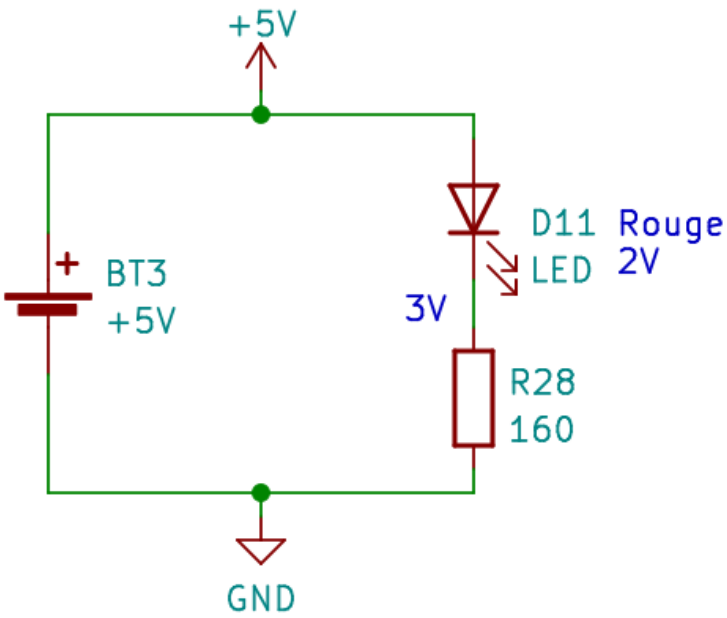
Loi d'ohms : $U = R \times I$



(présent dans la version courte)

Un premier exemple avec une diode électroluminescente

Loi d'ohms : $U = R \times I$

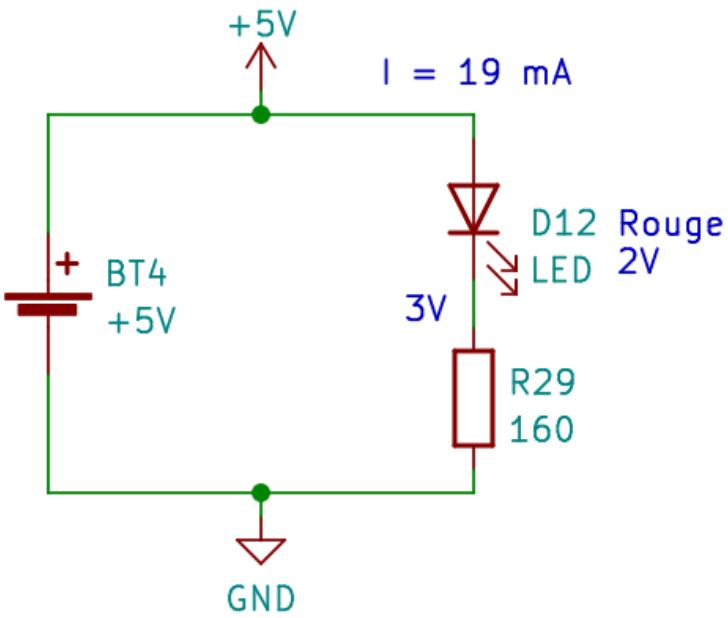


(présent dans la version courte)

Un premier exemple avec une diode électroluminescente

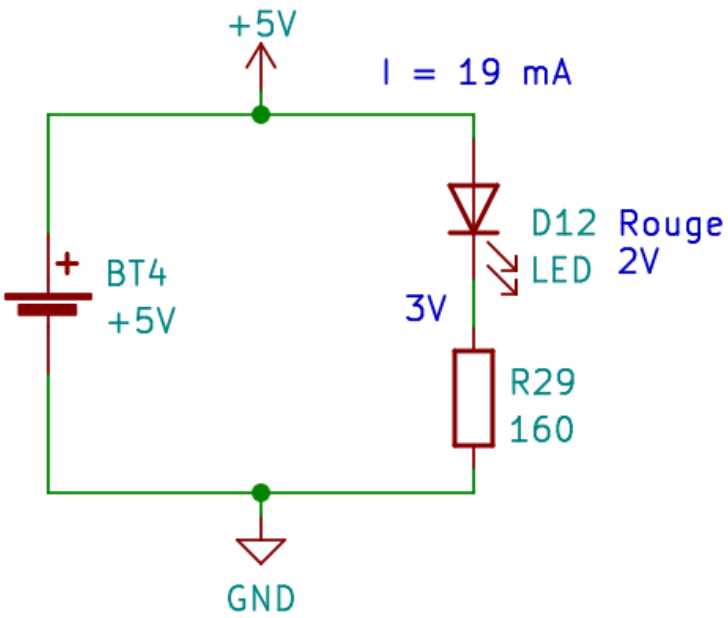
Loi d'ohms : $U = R \times I$

Courant : $I = 3V/160 = 0.0187$



(présent dans la version courte)

Un premier exemple avec une diode électroluminescente

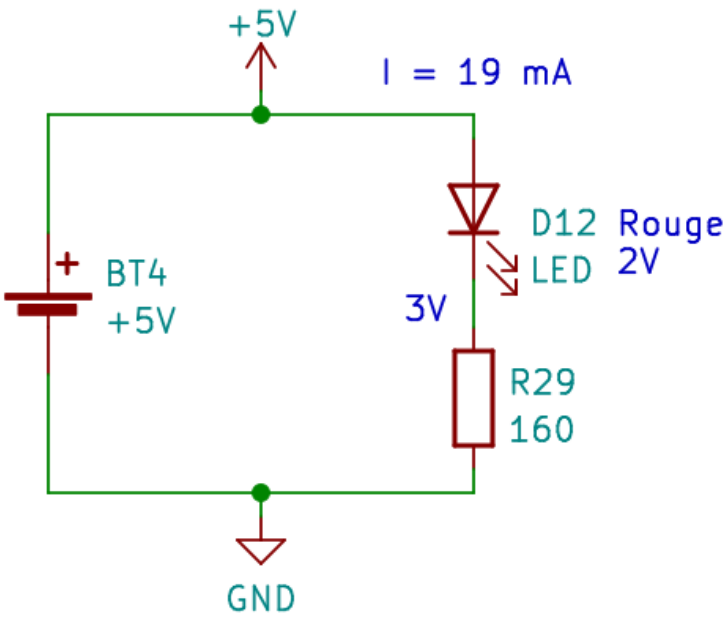


Loi d'ohms : $U = R \times I$

Courant : $I = 3V/160 = 0.0187$

Puissances dissipées : $P = U \times I$

Un premier exemple avec une diode électroluminescente



Loi d'ohms : $U = R \times I$

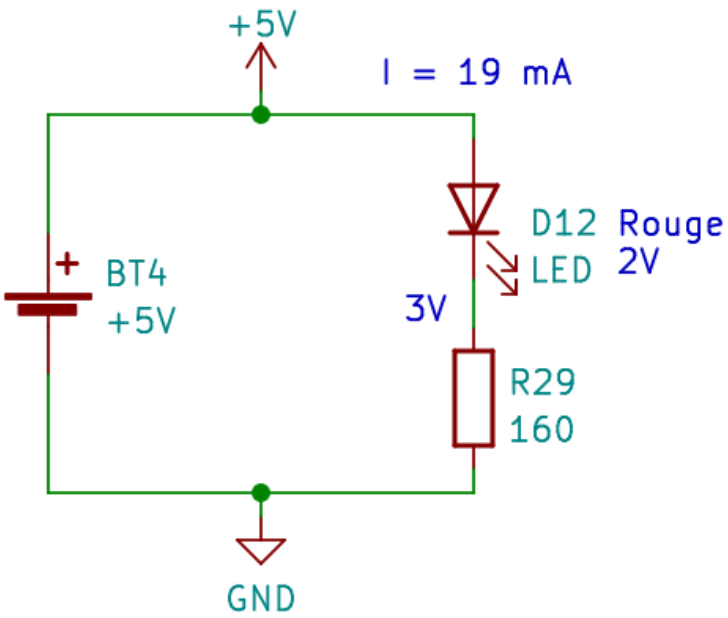
Courant : $I = 3V/160 = 0.0187$

Puissances dissipées : $P = U \times I$

$$P_R = U \times I = 3V \times 0.0187 = 56mW$$

La résistance est un $1/4W$, elle ne brûlera pas,

Un premier exemple avec une diode électroluminescente



Loi d'ohms : $U = R \times I$
Courant : $I = 3V/160 = 0.0187$

Puissances dissipées : $P = U \times I$

$$P_R = U \times I = 3V \times 0.0187 = 56mW$$

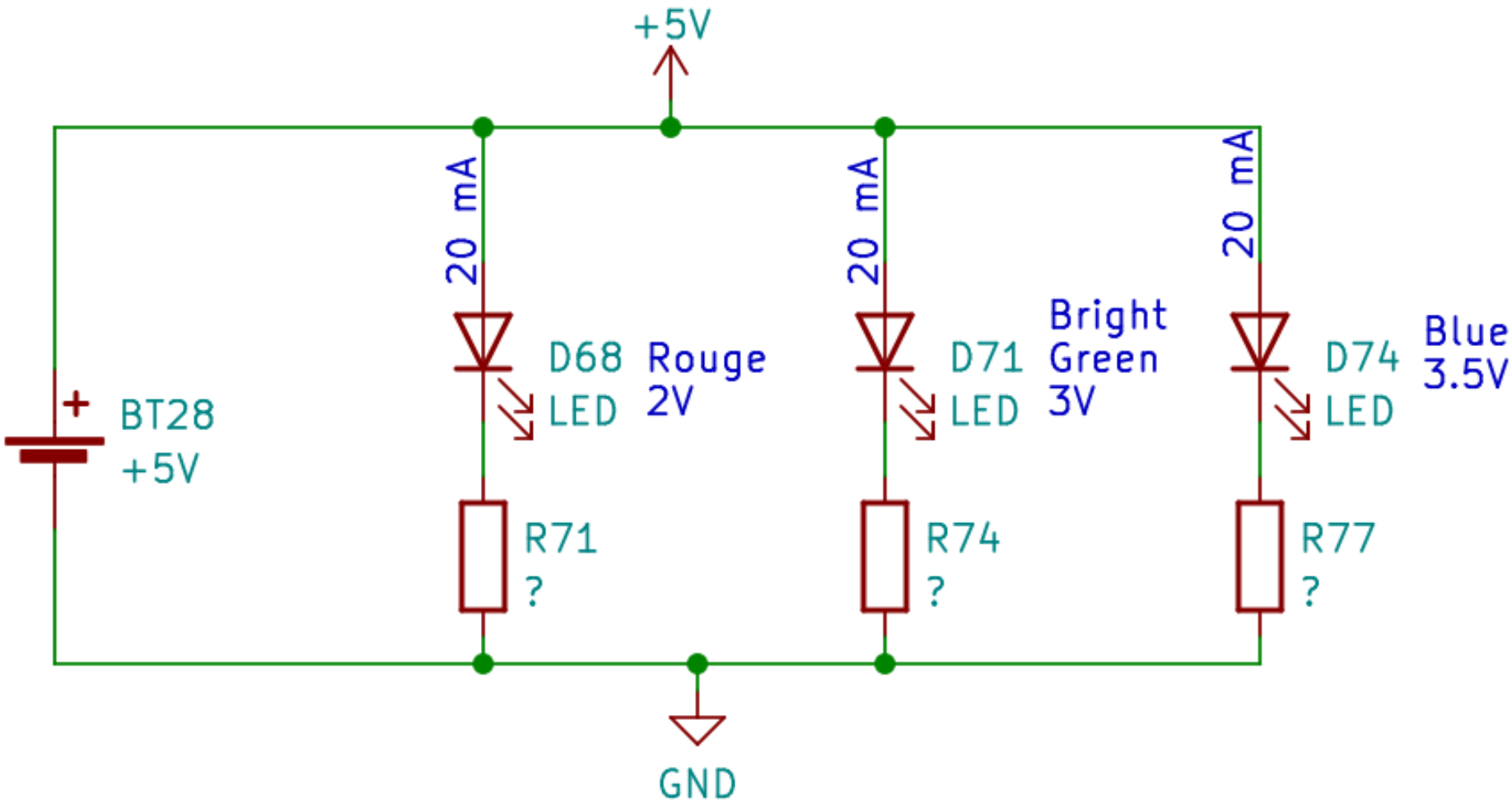
La résistance est un $1/4W$, elle ne brûlera pas,

$$P_{led} = U \times I = 2V \times 0.0187 = 37mW$$

La led doit au plus consommer en continue $20mA$, elle ne brûlera pas.

Dimensionner un circuit avec trois leds

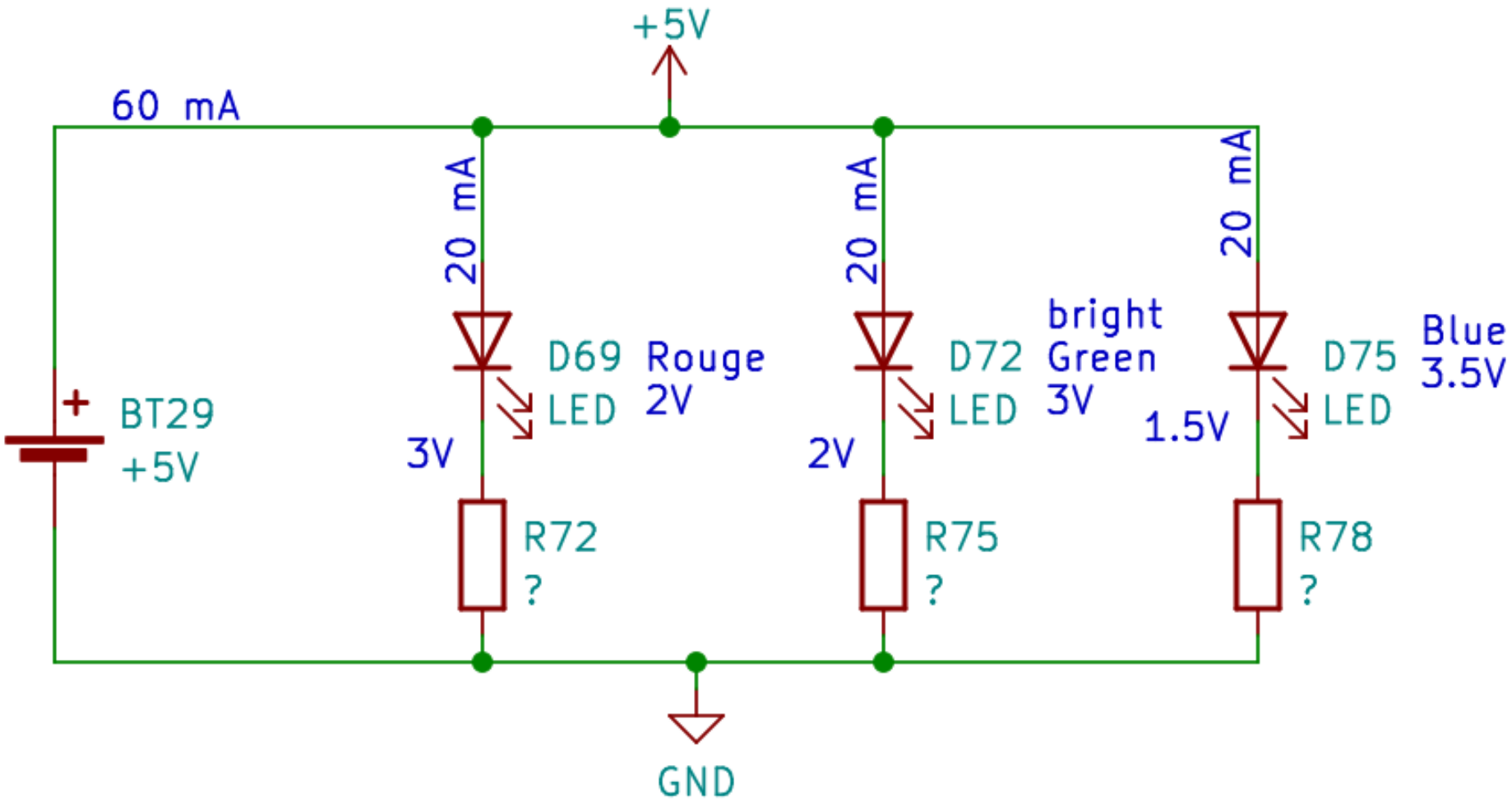
Pour des diodes de 20 mA, on obtient



(présent dans la version courte)

Dimensionner un circuit avec trois leds

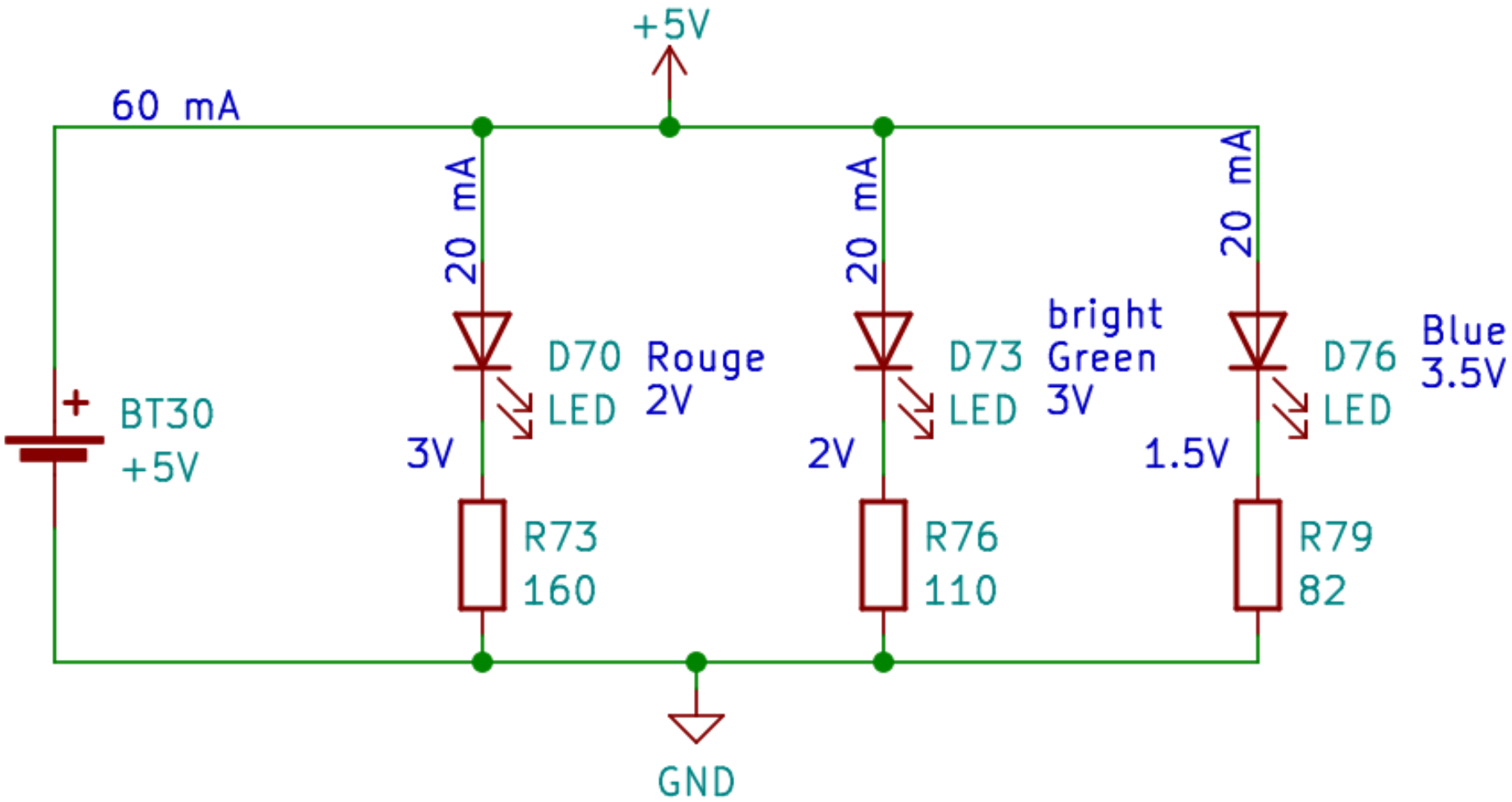
Pour des diodes de 20 mA, on obtient



(présent dans la version courte)

Dimensionner un circuit avec trois leds

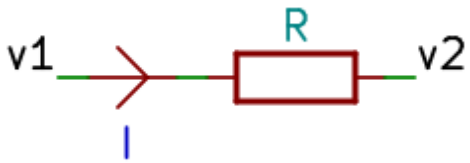
Pour des diodes de 20 mA, on obtient



(présent dans la version courte)

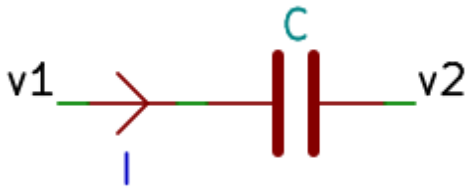
Quelques composants indispensables – 1/2

La résistance :



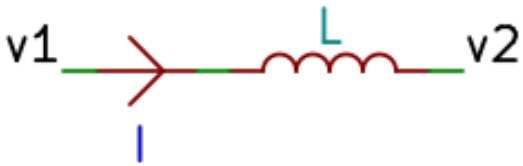
$$v_1 - v_2 = R \times I$$

Le condensateur :



$$I = \frac{\partial}{\partial t}(C \times (v_1 - v_2)) \qquad v1 - v2 \text{ est continue !}$$

La bobine :



$$v_1 - v_2 = \frac{\partial}{\partial t}(L \times I) \qquad I \text{ est continue !}$$

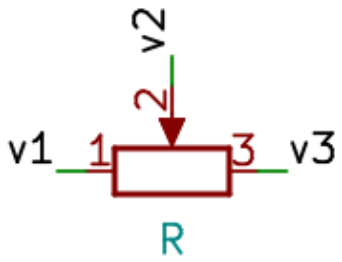
La diode :

$$\begin{cases} I = 0 & \Rightarrow & v1 - v2 < 0.6V = U_d \\ I > 0 & \Rightarrow & v1 - v2 \approx 0.6V = U_d \text{ (tension directe)} \\ I < 0 & \Rightarrow & \text{est au paradis des diodes} \end{cases}$$

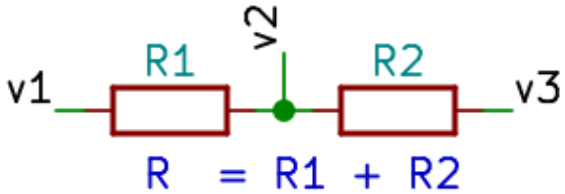
(présent dans la version courte)

Quelques composants indispensables – 2/2

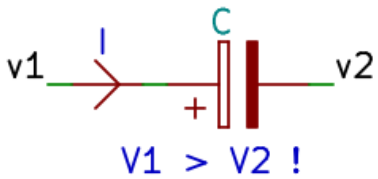
Le potentiomètre :



Les résistance $R1$ et $R1$ sont variables.

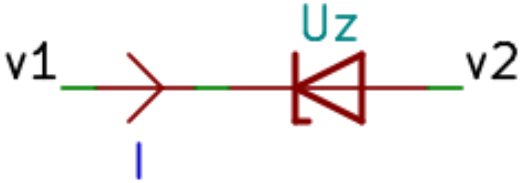


Le condensateur polarisé :



Si $v1 < v2$ alors le condensateur polarisé explose.

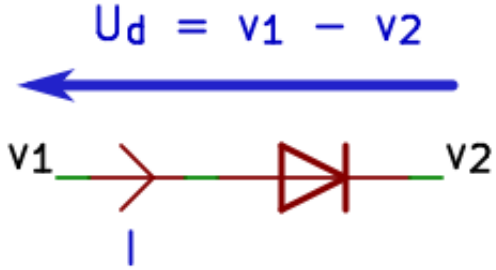
La diode zener :



$$\begin{cases} I > 0 & \Rightarrow & v1 - v2 \approx U_z \ (\text{tension zener}) \\ I = 0 & \Rightarrow & -0.6V < v1 - v2 < U_z \\ I < 0 & \Rightarrow & v1 - v2 \approx -0.6V = -U_d \ (\text{-tension directe}) \end{cases}$$

(présent dans la version courte)

Tensions directes et longueurs d'onde de différentes diodes

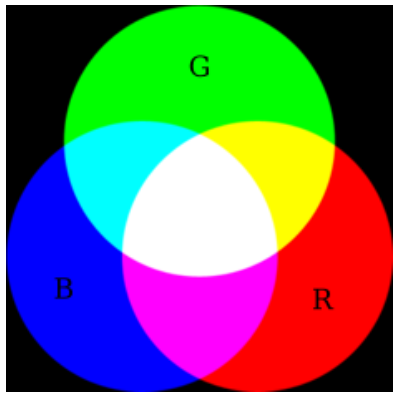


Les tensions directes U_d des différentes diodes sont données pour de faibles courants ($I \leq 200mA$).

	diodes		diode électroluminescente (led)						
	Schottky	Silicium	Infra rouge	Rouge	Jaune	Vert (GaP)	Vert clair (GaN)	Bleue	Ultra violet
tension directe (V)	0.4	0.6-0.8	1.3	2	2	2	3	3.5	3.7
longueur d'onde (nm)	/	/	> 760	615 à 650	574 à 582	500 à 570		466 à 490	< 400

Comment fonctionne les led :
https://www.youtube.com/watch?v=AF8d72mA41M
https://www.youtube.com/watch?v=l2y-w9aS98k

(consultés le 21/03/2025)



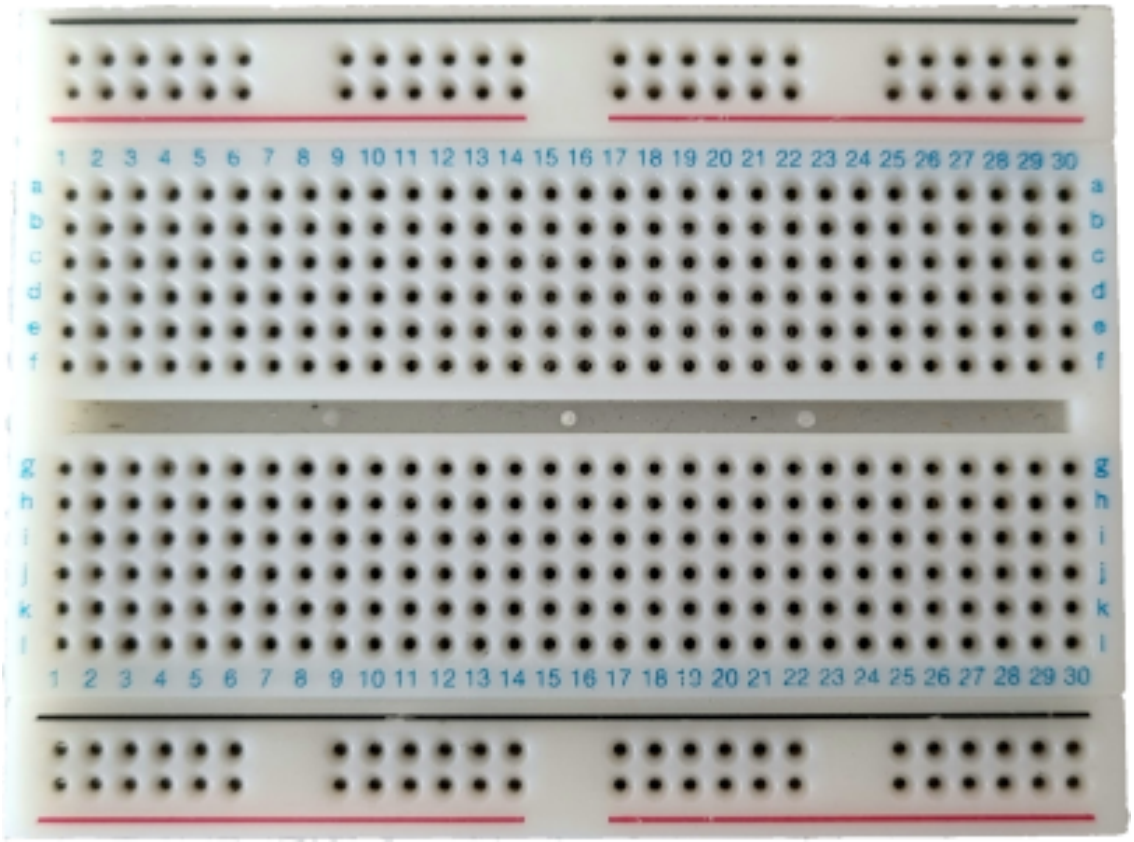
(présent dans la version courte)

Plan

(présent dans la version courte)

TD : Réaliser un montage électronique sur platine d'expérimentation – 1/3

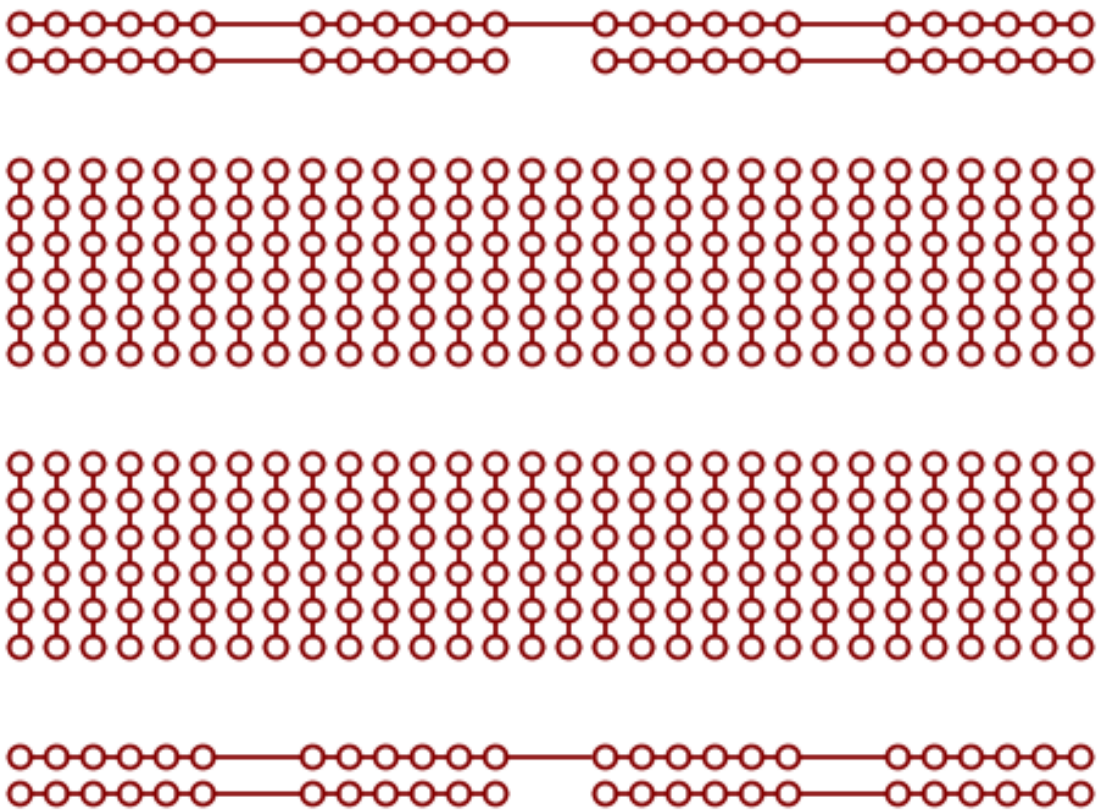
La platine d'expérimentation - 1 A max !



(présent dans la version courte)

TD : Réaliser un montage électronique sur platine d'expérimentation – 1/3

La platine d'expérimentation - 1 A max !

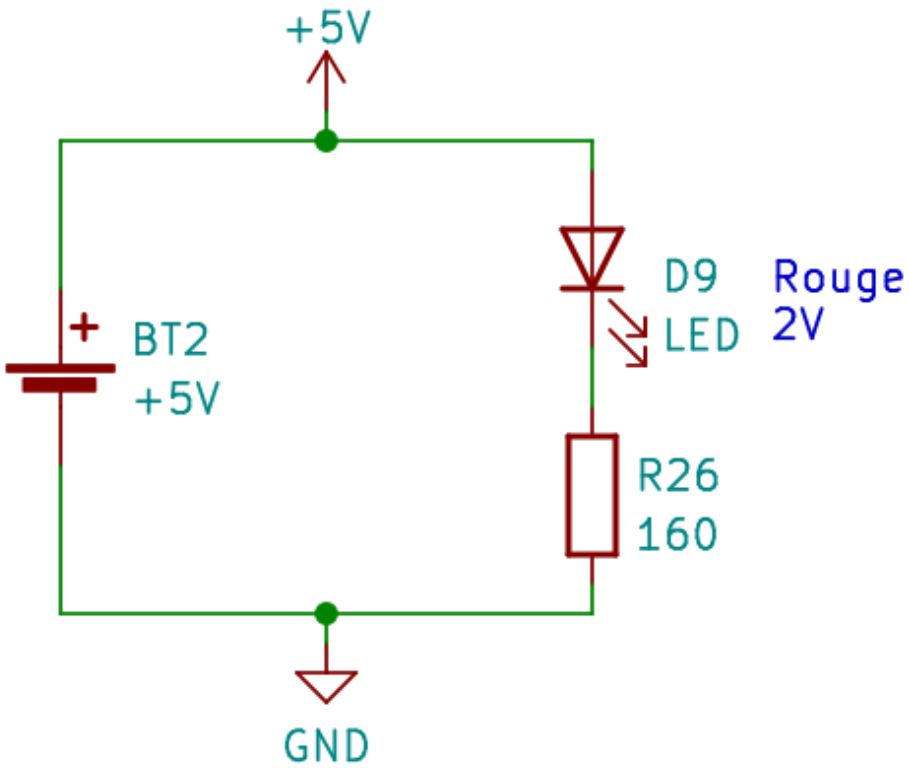


Les pistes de cuivre à l'intérieur

(présent dans la version courte)

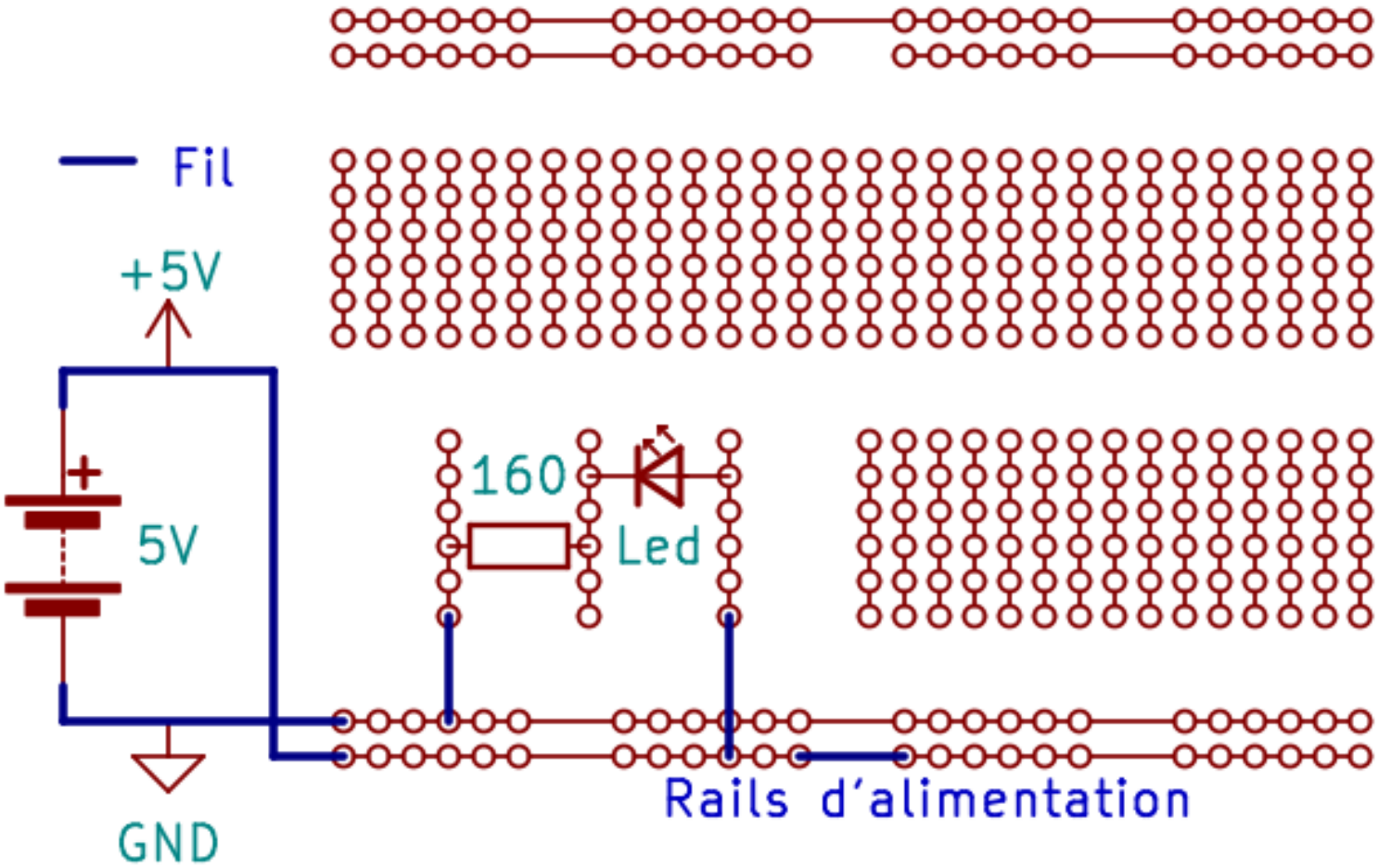
TD : Réaliser un montage électronique sur platine d'expérimentation – 2/3

Nous allons réaliser le montage suivant:



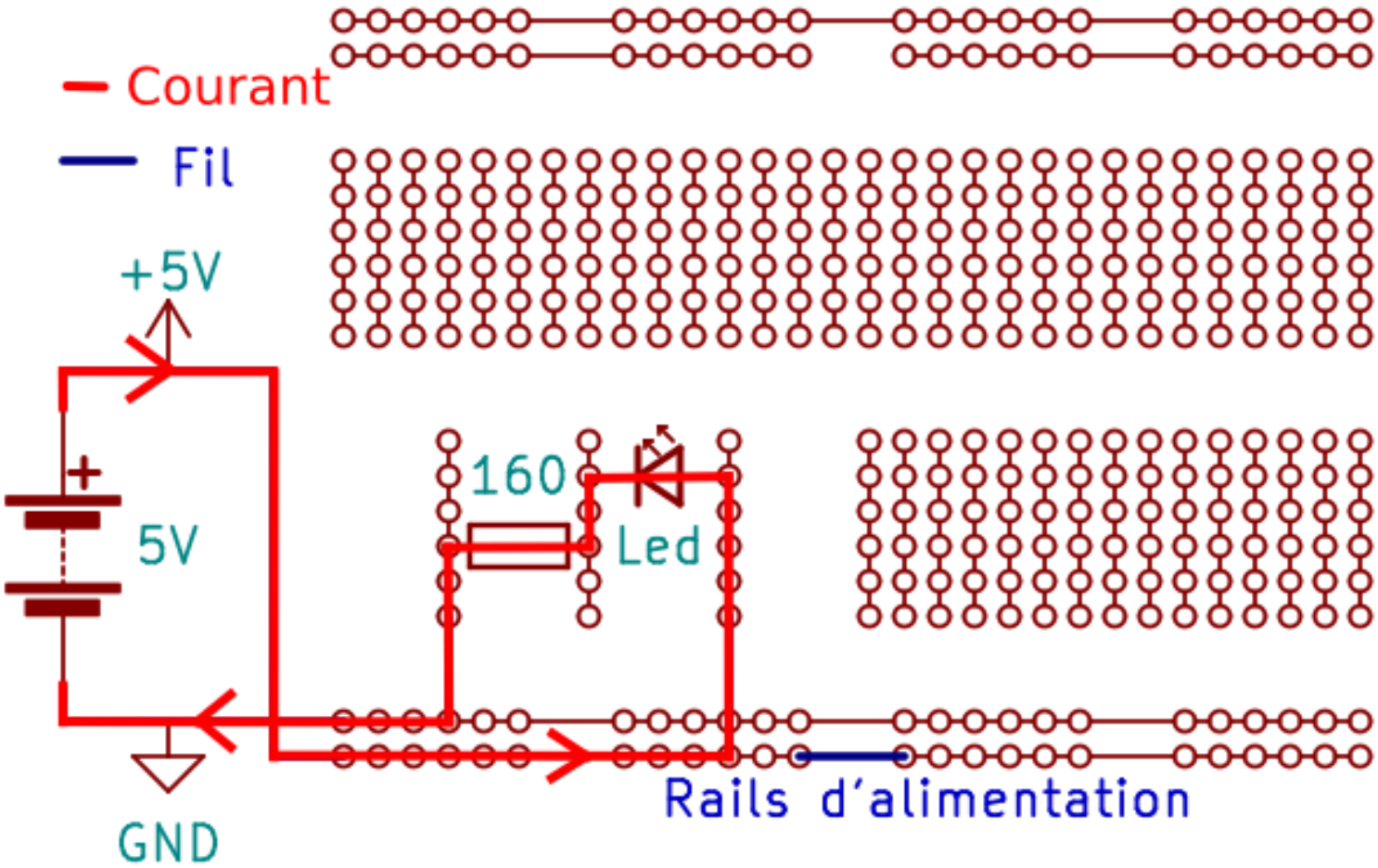
(présent dans la version courte)

TD : Réaliser un montage électronique sur platine d'expérimentation – 3/3



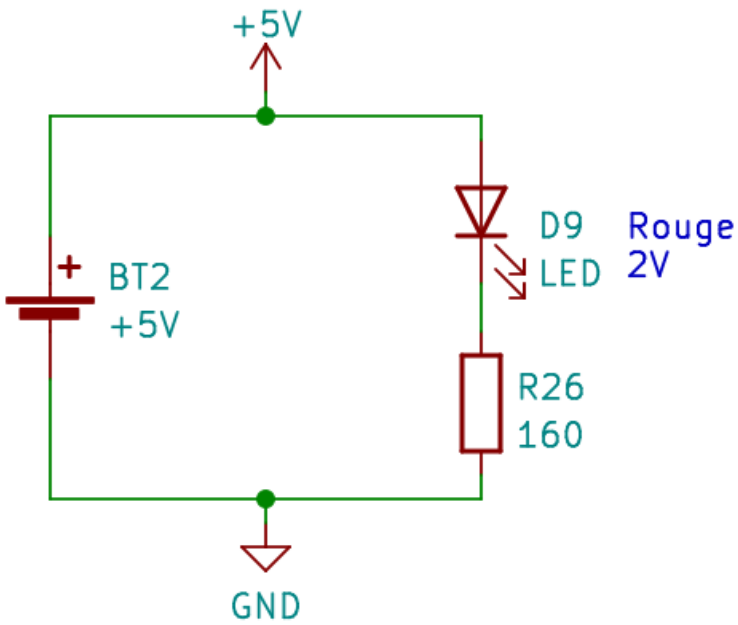
(présent dans la version courte)

TD : Réaliser un montage électronique sur platine d'expérimentation – 3/3



(présent dans la version courte)

TD : Alimenter un circuit avec une alimentation de laboratoire – 1/5



A FAIRE : Mettre des images.

(présent dans la version courte)

TD : Vérifier son circuit avant de l'alimenter – 2/5

Il vous faut un voltmètre :

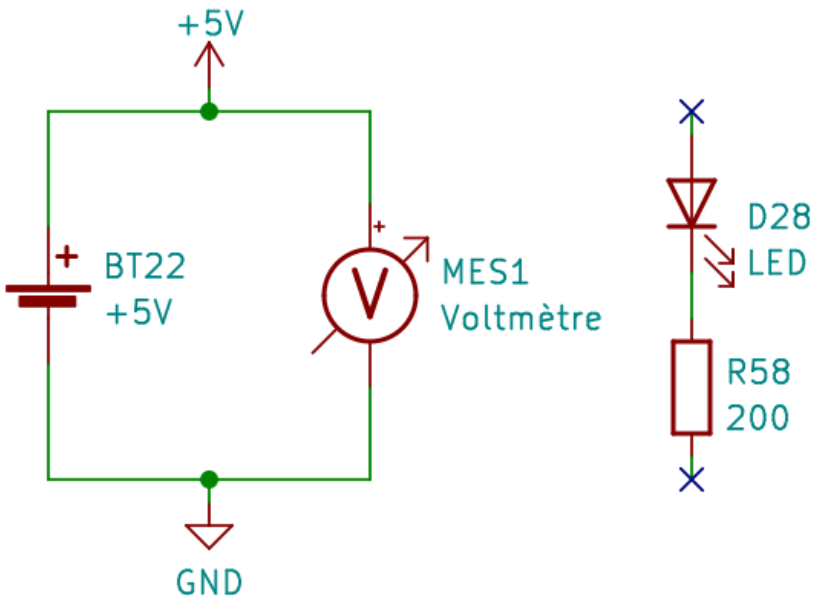
On utiliseras deux modes :

- Le mode Ohmmètre.
  Symbole : Ω
- Le mode Voltmètre en mode DC. Symbole : $\overline{\overline{V}}$

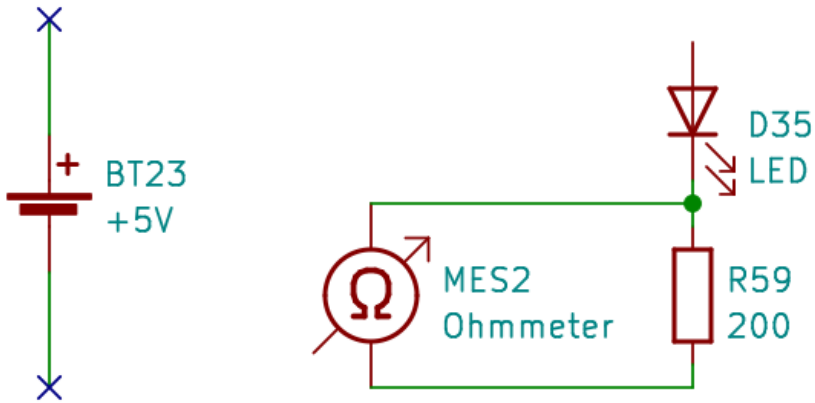
TD : Vérifier son circuit avant de l'alimenter – 3/5

Ne connectez pas l'alimentation à votre circuit !

Vérifier la tension de l'alimentation :



Vérifier la résistance :

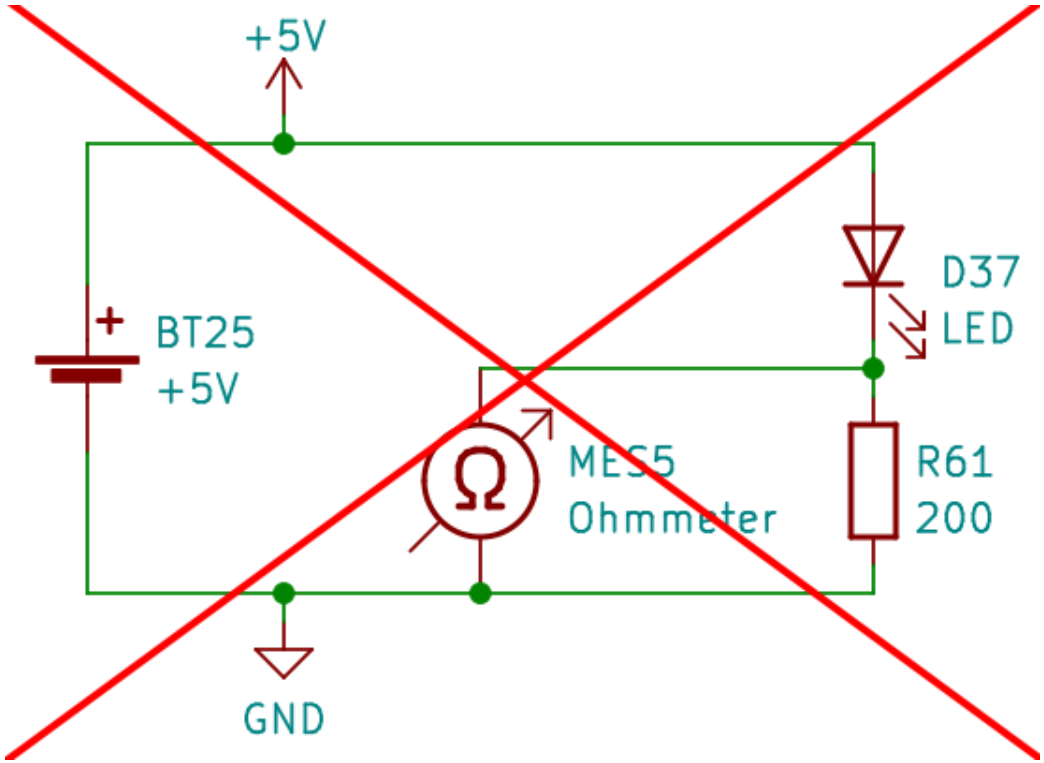


(présent dans la version courte)

TD : Ce qu'il ne faut jamais faire ! – 4/5

L'ohmmètre génère une tension pour faire sa mesure.

Ne jamais utiliser un multimètre en mode ohmmètre quand le circuit est branché !

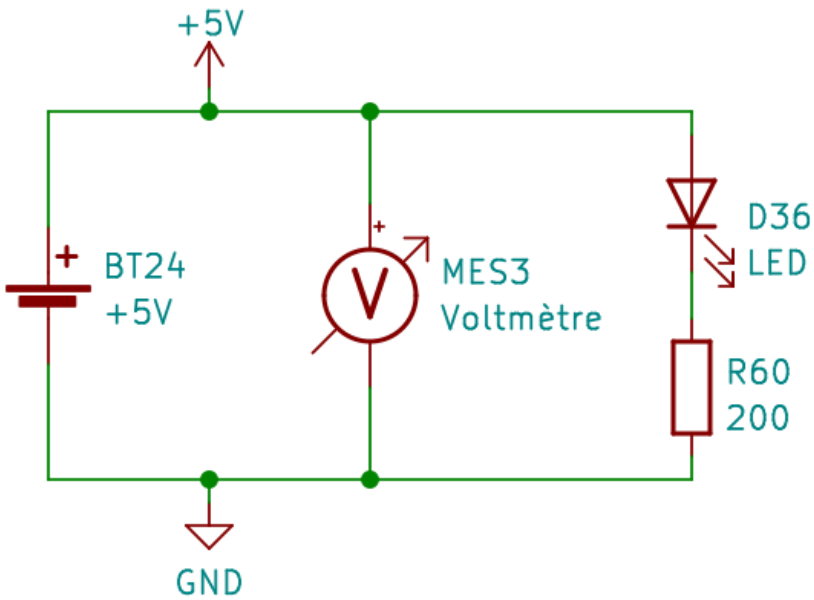


(présent dans la version courte)

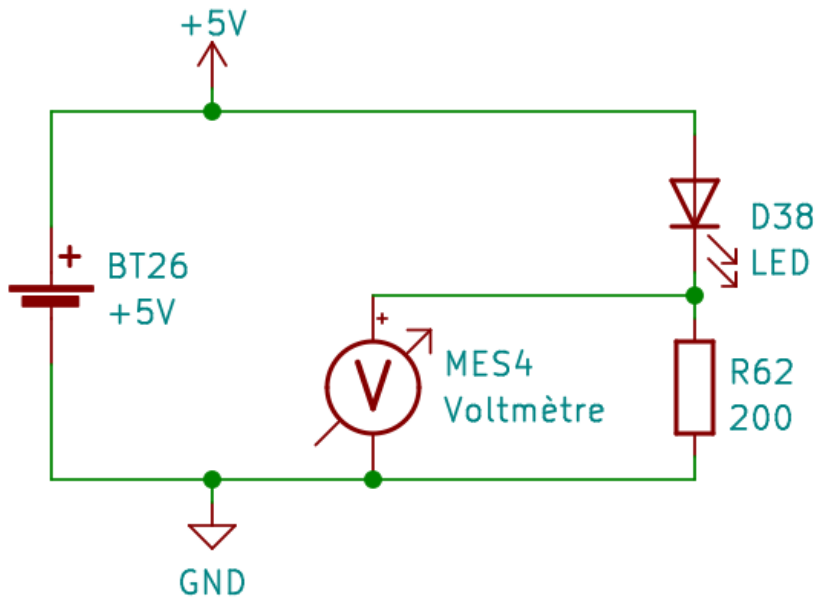
TD : Vérifier le bon fonctionnement de son circuit – 5/5

N'utilisez QUE le voltmètre ici

Vérifier la tension de l'alimentation :



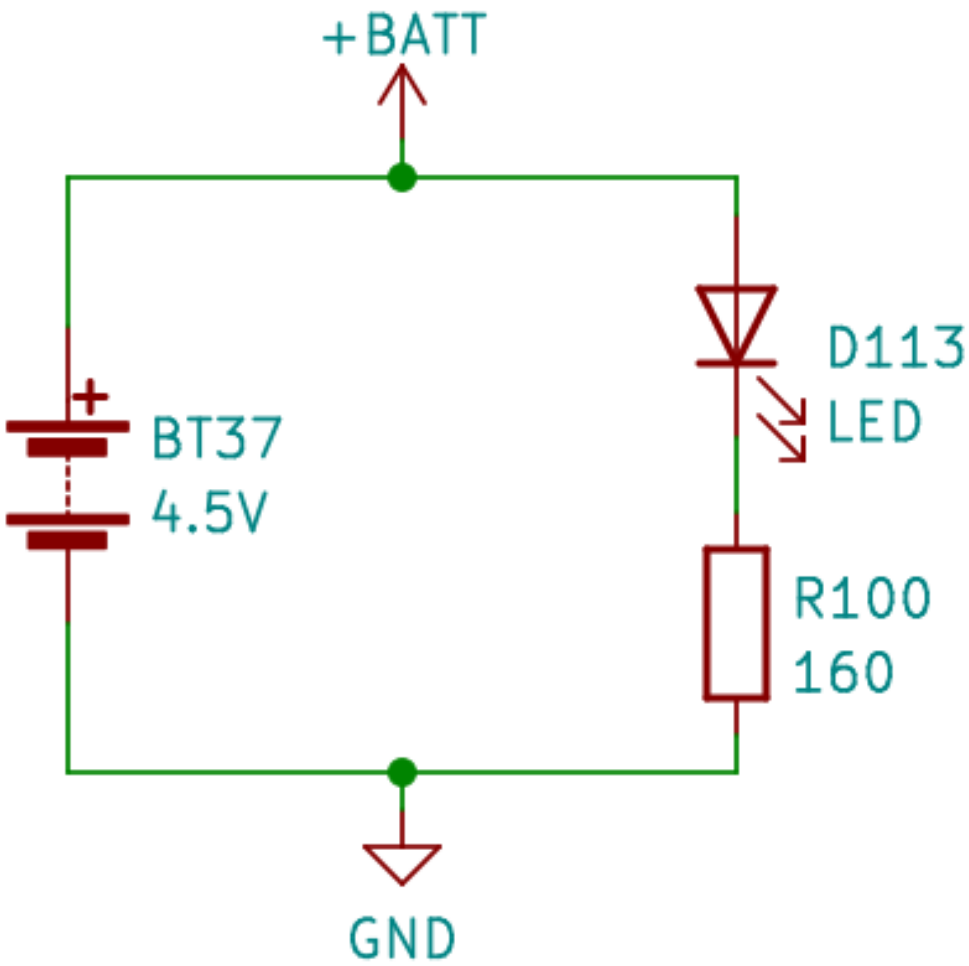
Vérifier la tension de la résistance :



(présent dans la version courte)

Alimenter votre circuit avec des piles AA 1.5V

Tension d'alimentation : entre $3V$ et $4.5V$.

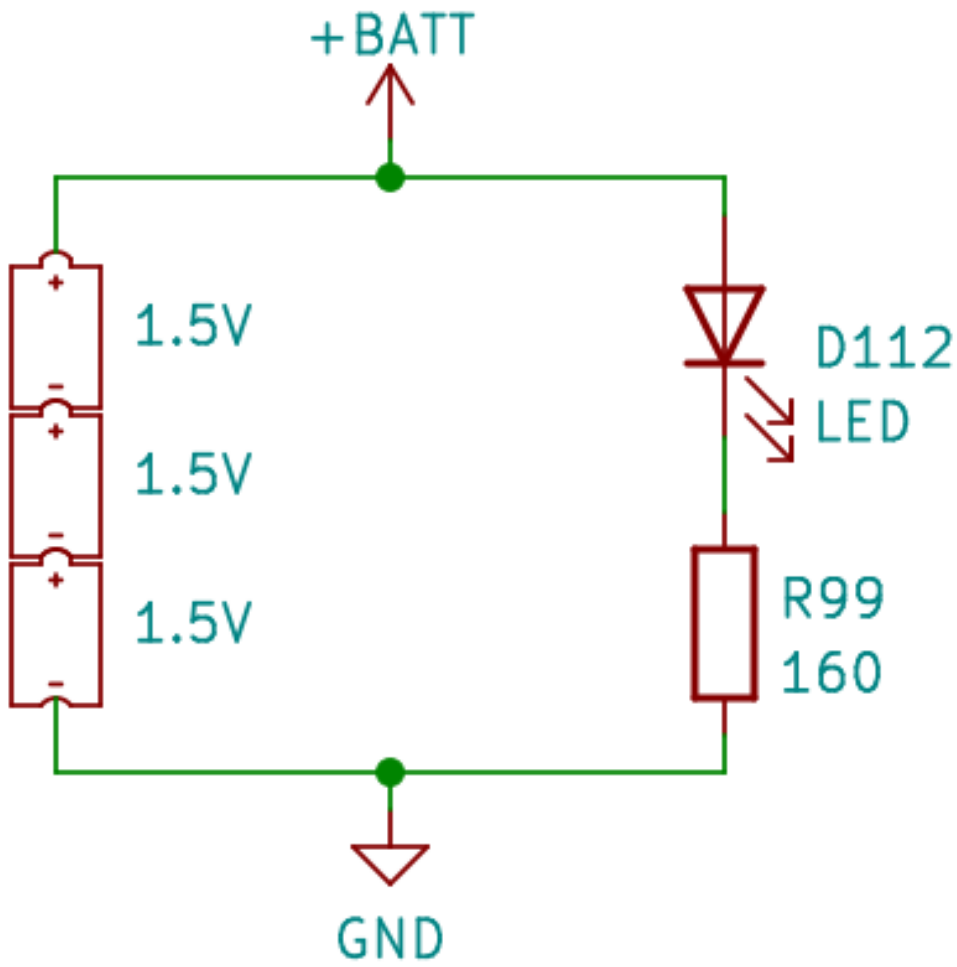


Pour connaître les plages de tension de fonctionnement des piles, vous pouvez consulter la table de la page 662.

(présent dans la version courte)

Alimenter votre circuit avec des piles AA 1.5V

Tension d'alimentation : entre $3V$ et $4.5V$.



Pour connaître les plages de tension de fonctionnement des piles, vous pouvez consulter la table de la page 662.

(présent dans la version courte)

Plan

(présent dans la version courte)

Documentation sur la sécurité

Voici quelques références sur la sécurité concernant les risques électriques :

- Le Site de prévention du risque de l'IRNS (Institut national de recherche et de sécurité pour la prévention des accidents du travail et des maladies professionnelles).
- La norme d'application obligatoire NF C15-100, Installations électriques à basse tension, Edition Afnor, 2005, consultable gratuitement.
- La norme d'application obligatoire NF EN 60529, Degrés de protection procurés par les enveloppes (code IP), Edition Afnor, 1992, consultable gratuitement.
- La norme NF EN 61140, Protection contre les chocs électriques - Aspects communs aux installations et aux matériels, Edition Afnor, 2016.
- La norme NF C18-510, Opérations sur les ouvrages et installations électriques et dans un environnement électrique - Prévention du risque électrique, Edition Afnor, 2012.
- La norme NF EN 61558-1, Sécurité des transformateurs, bobines d'inductance, blocs d'alimentation et des combinaisons de ces éléments - Partie 1 : exigences générales et essais, Edition Afnor, 2019.
- La norme IEC 60950-1:2005, Matériels de traitement de l'information - Sécurité - Partie 1 : exigences générales, Edition Afnor, 2005.
- Les normes NF EN 61558-2-*, pour traiter, entre autres, le cas des transformateurs de sécurité pour des alimentations linéaires et à découpage.

(sites consultés le 12 Septembre 2024)

Livres traitant de l'électricité et des installations électriques :

- Le grand livre de l'électricité, Thierry Gallauziaux, David Fedullo, 2021, Eyrolles.
- Guide pratique de la CEM, Alain Charoy, 3^{ième} édition, 2017, Dunod.

(présent dans la version courte)

Résistance du corps humain, courant et tension maximale

La traversée d'un courant $> 10mA$ peut provoquer la mort (contraction des muscles et fibrilation du coeur).

$$I_{\text{danger}} \geq 10mA$$

(présent dans la version courte)

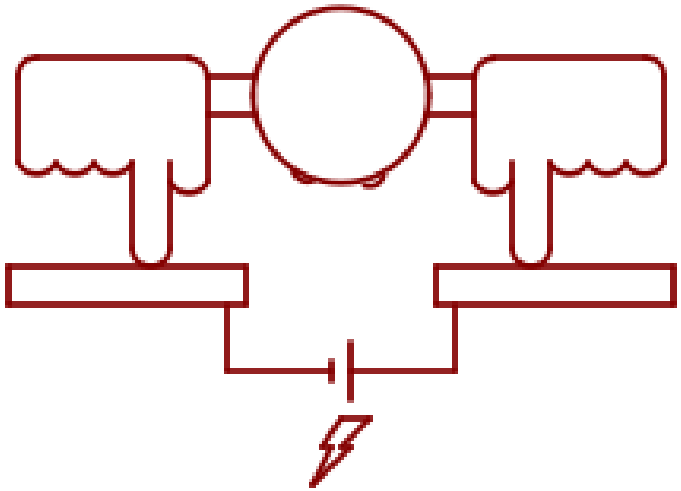
Résistance du corps humain, courant et tension maximale

La traversée d'un courant $> 10mA$ peut provoquer la mort (contraction des muscles et fibrilation du coeur).

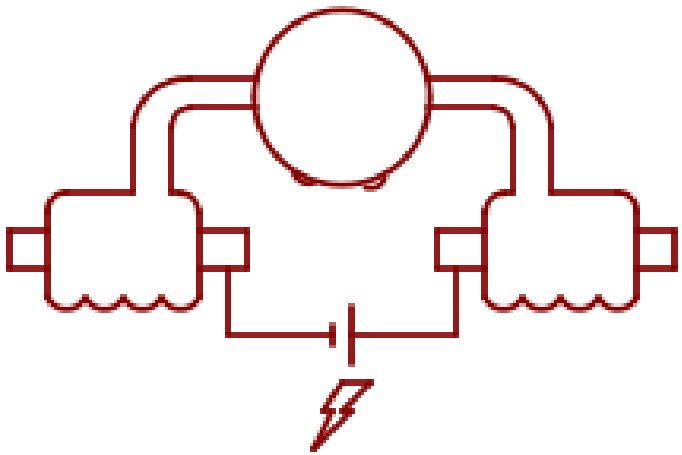
$$I_{\text{danger}} \geq 10mA$$

La résistance du corps humain varie selon la surface de contact et l'état des mains (sec ou humide).

$R_{\text{humain}} \in [50k\Omega, 2M\Omega]$



$R_{\text{humain}} \approx 5k\Omega$!



(présent dans la version courte)

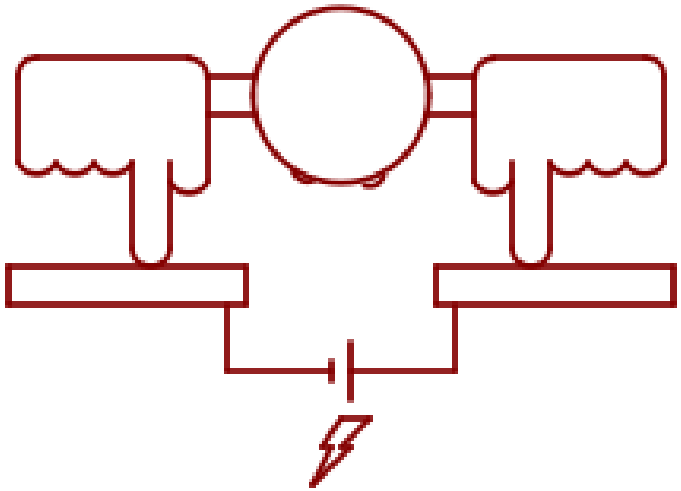
Résistance du corps humain, courant et tension maximale

La traversée d'un courant $> 10mA$ peut provoquer la mort (contraction des muscles et fibrilation du coeur).

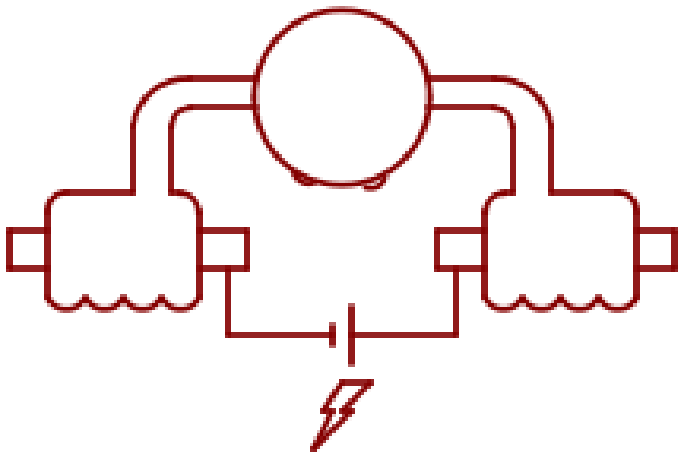
$$I_{\text{danger}} \geq 10mA$$

La résistance du corps humain varie selon la surface de contact et l'état des mains (sec ou humide).

$R_{\text{humain}} \in [50k\Omega, 2M\Omega]$



$R_{\text{humain}} \approx 5k\Omega$!



La tension de danger se déduit en prenant le pire cas :

$$U_{\text{danger}} = R_{humain} \times I_{\text{danger}} \geq 5k\Omega \times 10mA = 50V$$

(présent dans la version courte)

Tension maximale et isolant - les normes NF C15-100 et NF C18-510

TBT – très basse tension (ELV – Extra-low voltage) : tension limitée à 50V AC et 120V DC + isolation double ou renforcée avec toute les parties actives des autres installations.

TBTS – Très basse tension de sécurité (SELV – Safety extra low voltage) : TBT + isolation à la terre (circuit non relié à la terre)

TBTP – Très basse tension de protection (PELV – Protective extra low voltage) : TBT + une des bornes du secondaire est relié au conducteur de protection équipotentielle (en général la terre) ainsi que les parties conductrices exposées au contact physique humain.

AC : courant alternatif DC : courant continue

Nécessité de mettre une protection contre les contacts directs :

AC	$U \leq 12V$	$12V < U \leq 25V$	$25V < U \leq 50V$
DC	$U \leq 30V$	$30V < U \leq 60V$	$60 < U \leq 120V$
TBTS/SELV	pas nécessaire	pas nécessaire	nécessaire
TBTP/PELV	pas nécessaire	nécessaire ! ⚠	nécessaire

Si $U \leq 500V$, un isolant possède une résistance $> 50k\Omega$.

Isolants: les revêtements bois, moquettes, plastiques et linoléum.

Non isolant : métal, béton, carrelage, moquette avec quadrillage métallique relié à la terre.

(présent dans la version courte)

Contexte de travail

Dans tous le cours et pour toute l'option maker, nous ne travaillerons qu'avec des tensions continues d'une tension inférieure à 30V.

Nous utiliserons pour cela des blocs d'alimentation d'une tension inférieure à 24V, certifié SELV, issus du marché, aux normes françaises (et européennes), parfaitement isolés et parfaitement sécurisés.

Ainsi, il ne pourra pas y avoir de problèmes d'électrisation possible.

Cependant, il faut rester vigilant concernant d'autres dangers : les brûlures et les coupures !

Brûlure et coupure en Électronique

Même à faible tension un système peut provoquer des brûlures et coupures.

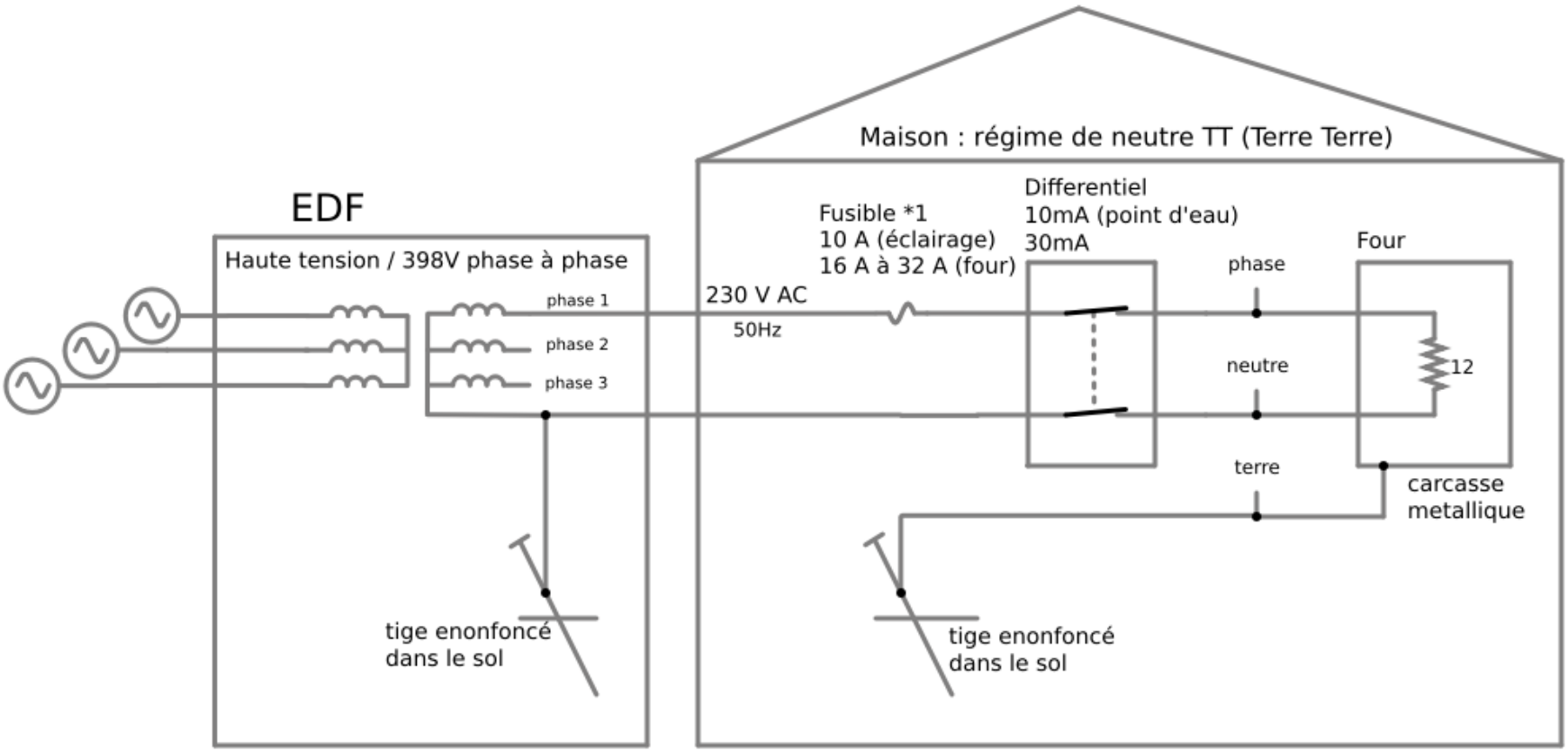
La puissance ne dépends pas que de la tension : $P = U \times I$.

$50W = 10A \times 5V$ est important (Un grille pain c'est 700W)

Même avec une faible puissance on peut réussir à se couper ou se brûler.

En mécanique il y a l'effet de levier : Faible puissance est compatible avec fort couple et forces importantes !
En thermodynamique il y a l'isolation : Faible puissance est compatible avec fortes températures !

Protection du secteur : fonctionnement normal

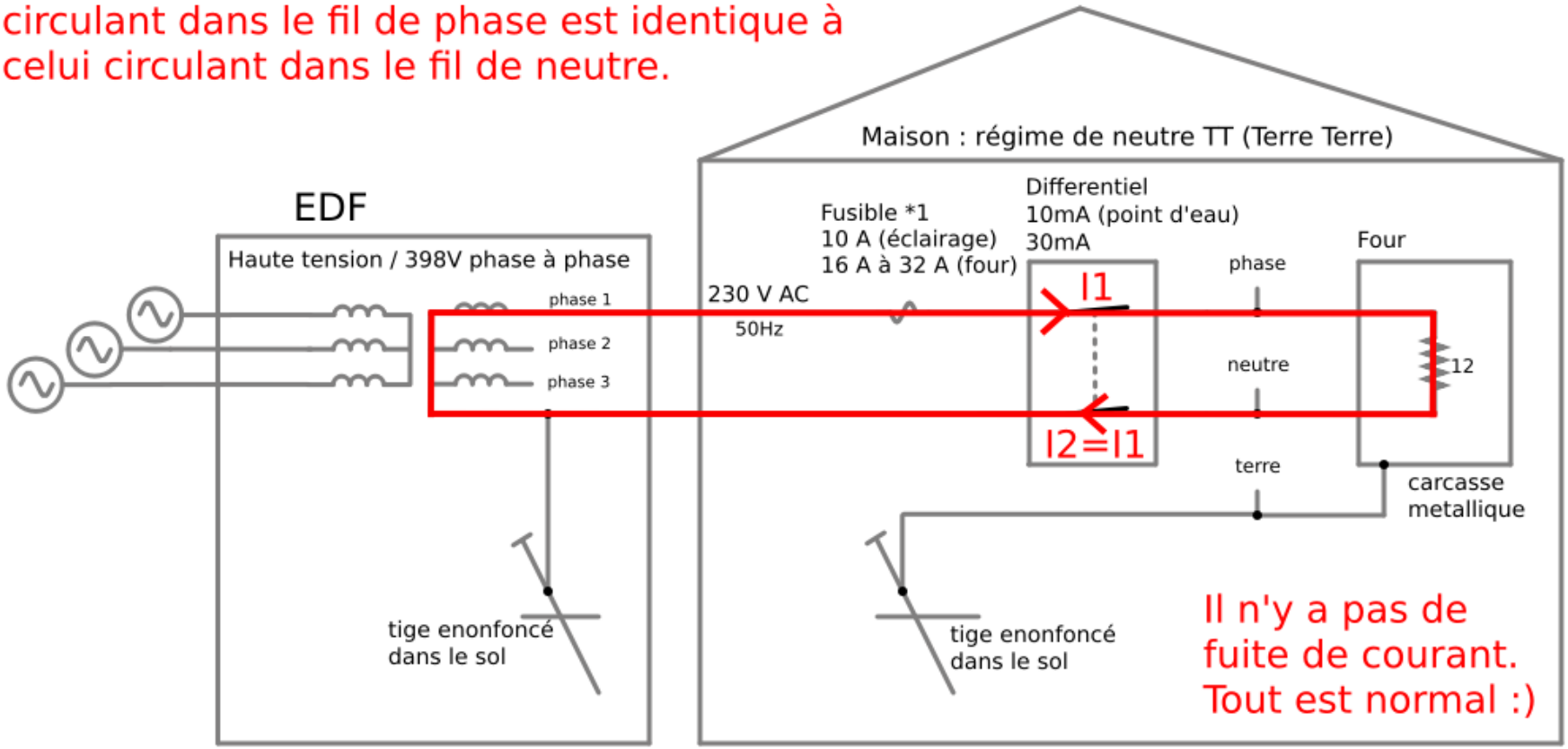


*1 : Attention: le calibre du fusible est choisie en fonction de la section des fils et la section des fils est dimensionner en fonction des appareils qu'ils alimentent.

(version longue)

Protection du secteur : fonctionnement normal

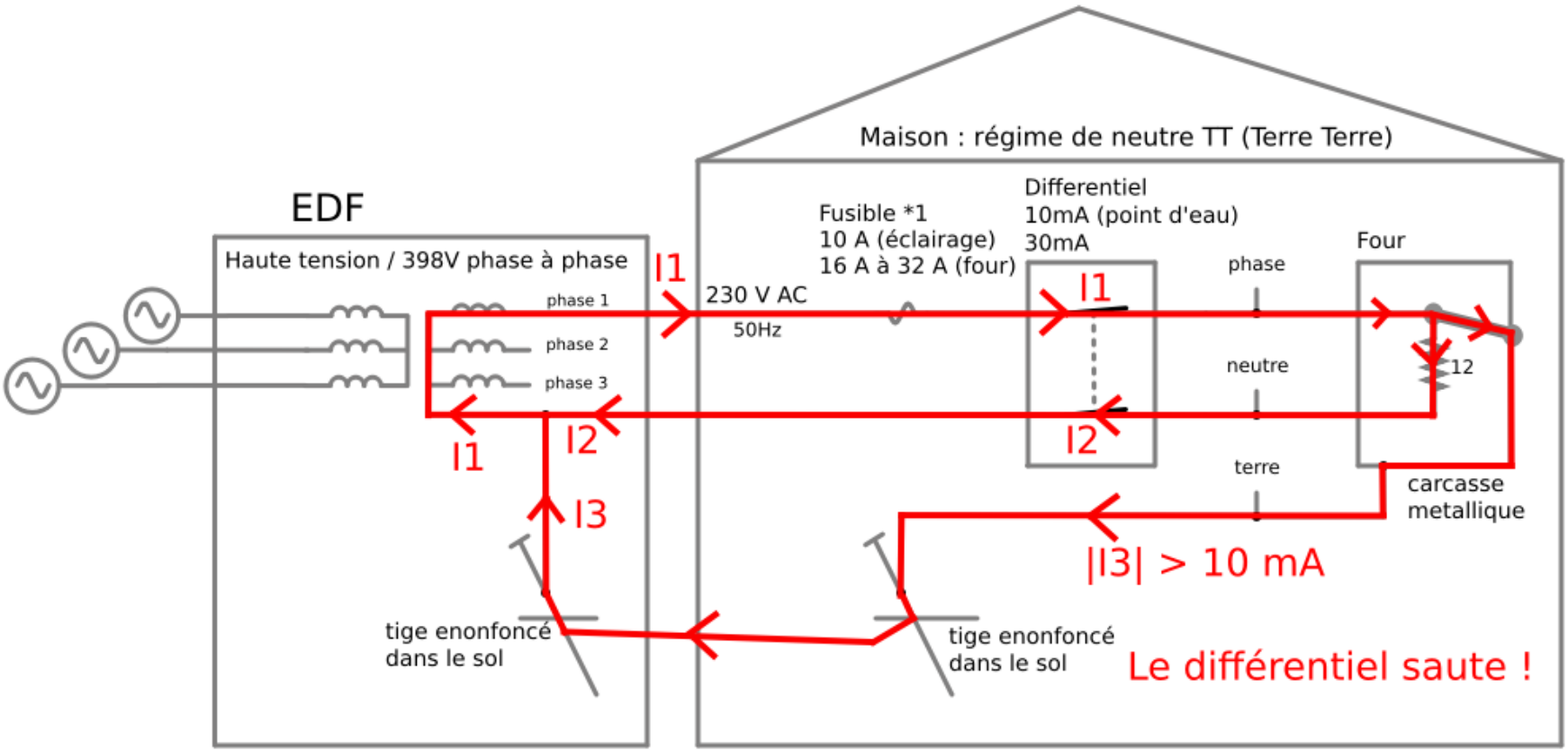
Le différentiel reste fermé car le courant circulant dans le fil de phase est identique à celui circulant dans le fil de neutre.



*1 : Attention: le calibre du fusible est choisie en fonction de la section des fils et la section des fils est dimensionner en fonction des appareils qu'ils alimentent.

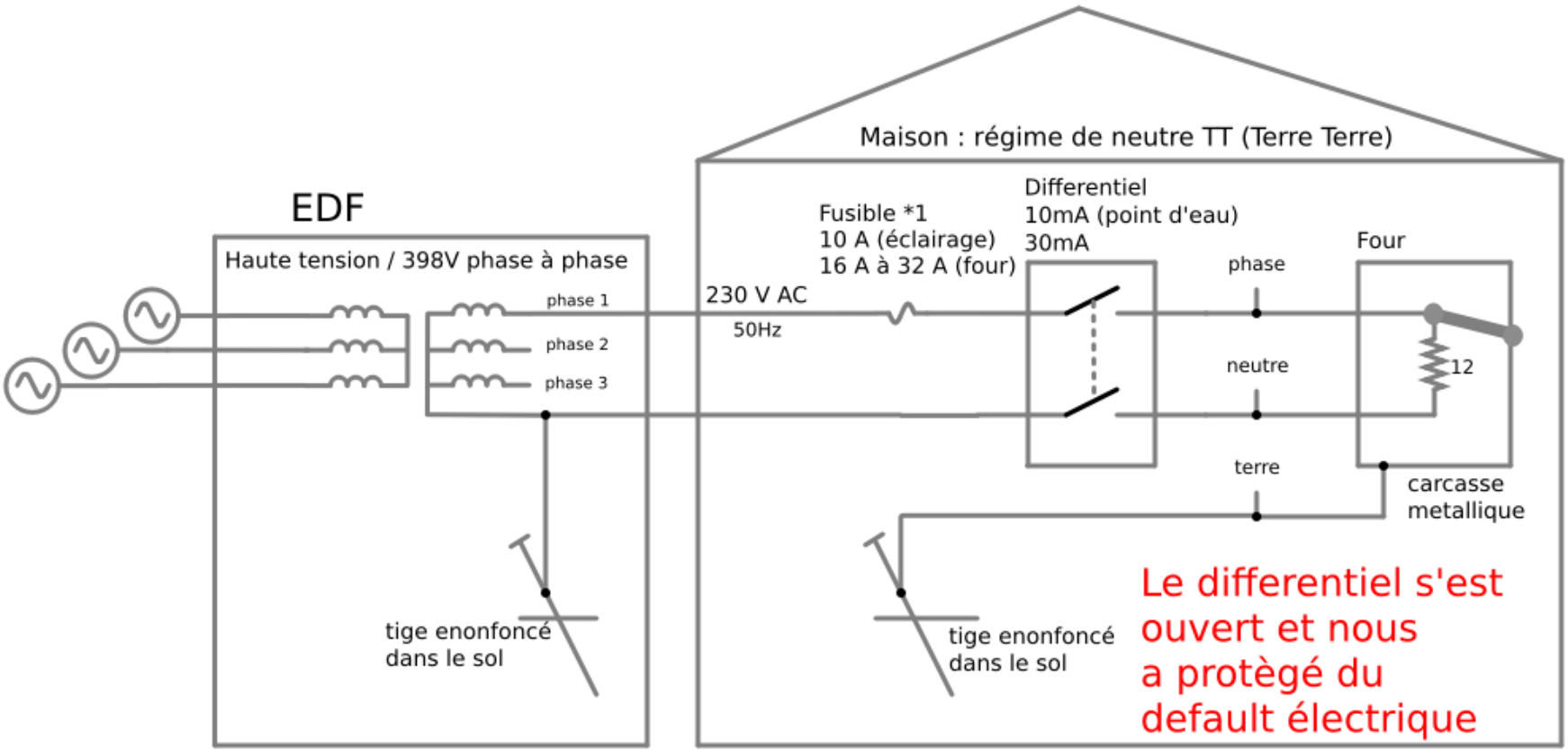
(version longue)

Le différentiel : prévention des défauts électriques et des électrisations



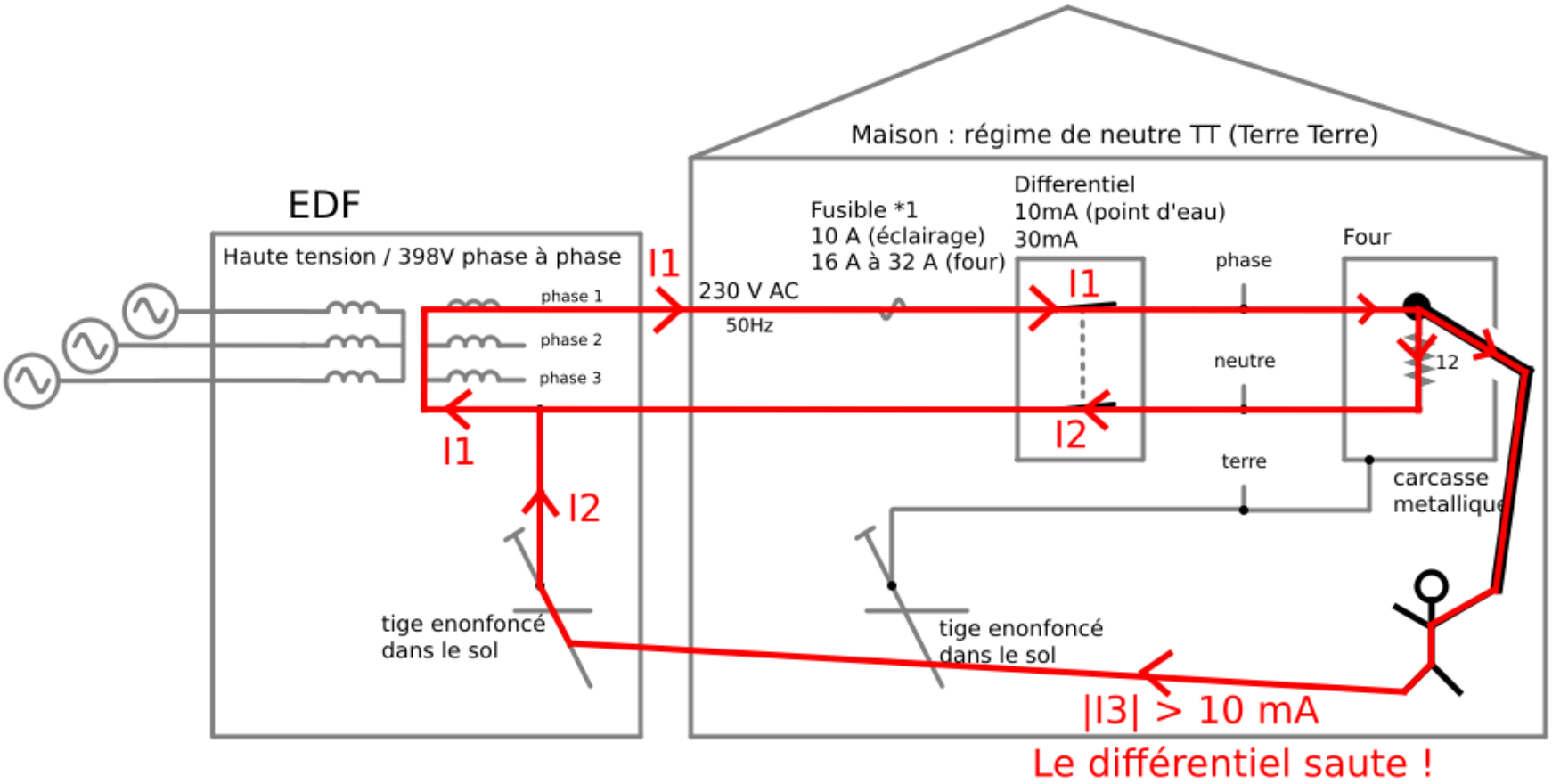
(version longue)

Le différentiel : prévention des défauts électriques et des électrisations



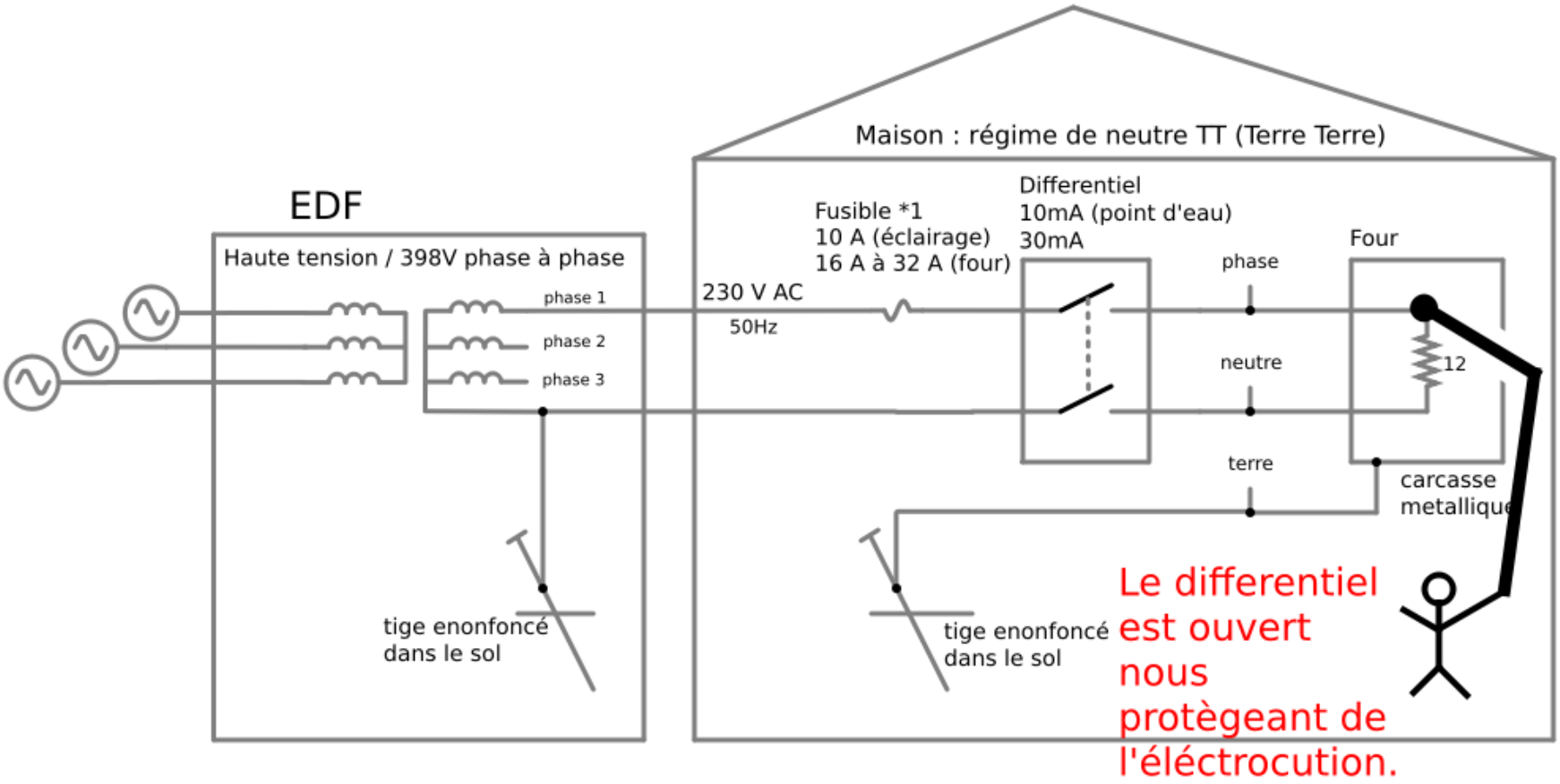
(version longue)

Le différentiel : prévention des électrocutions



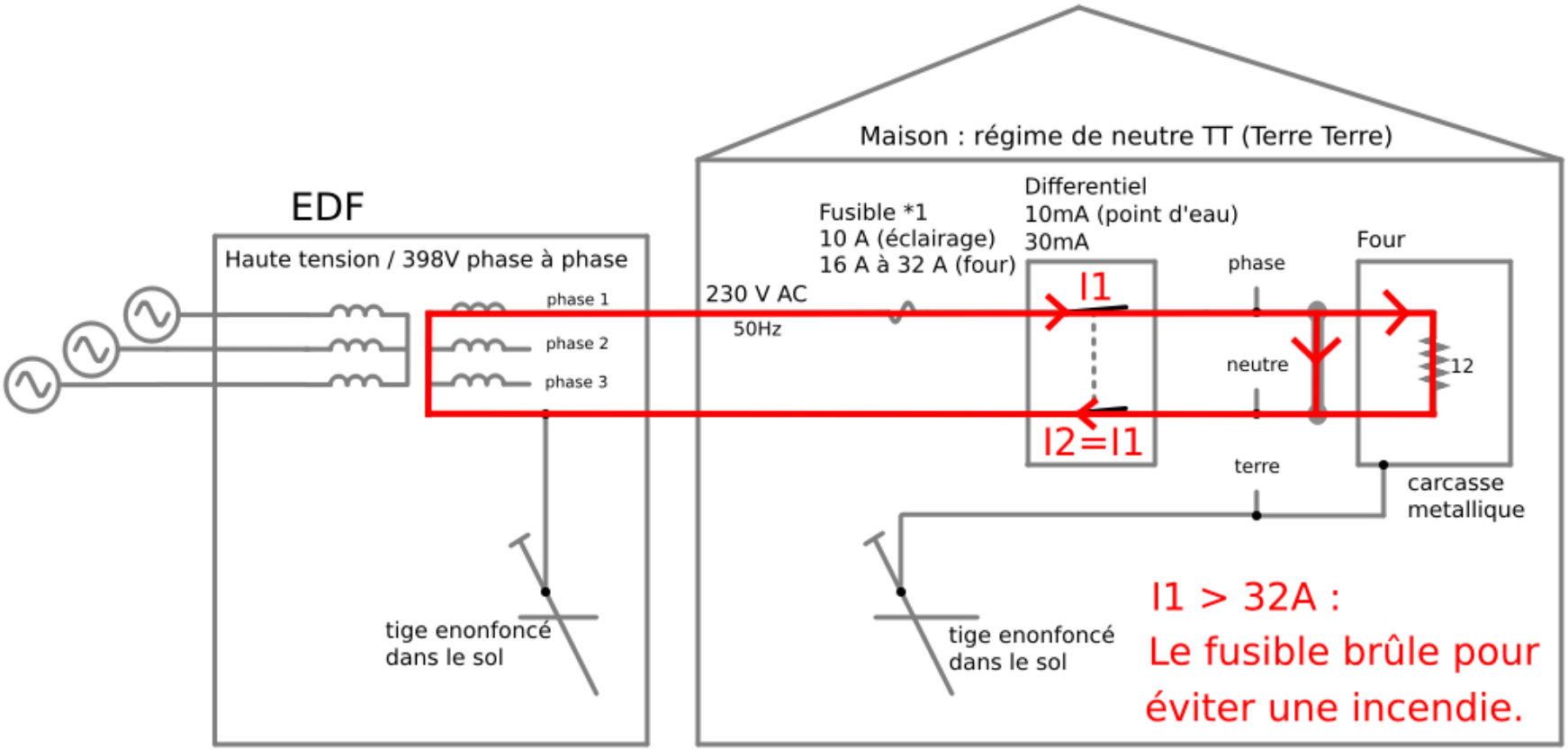
(version longue)

Le différentiel : prévention des électrocutions

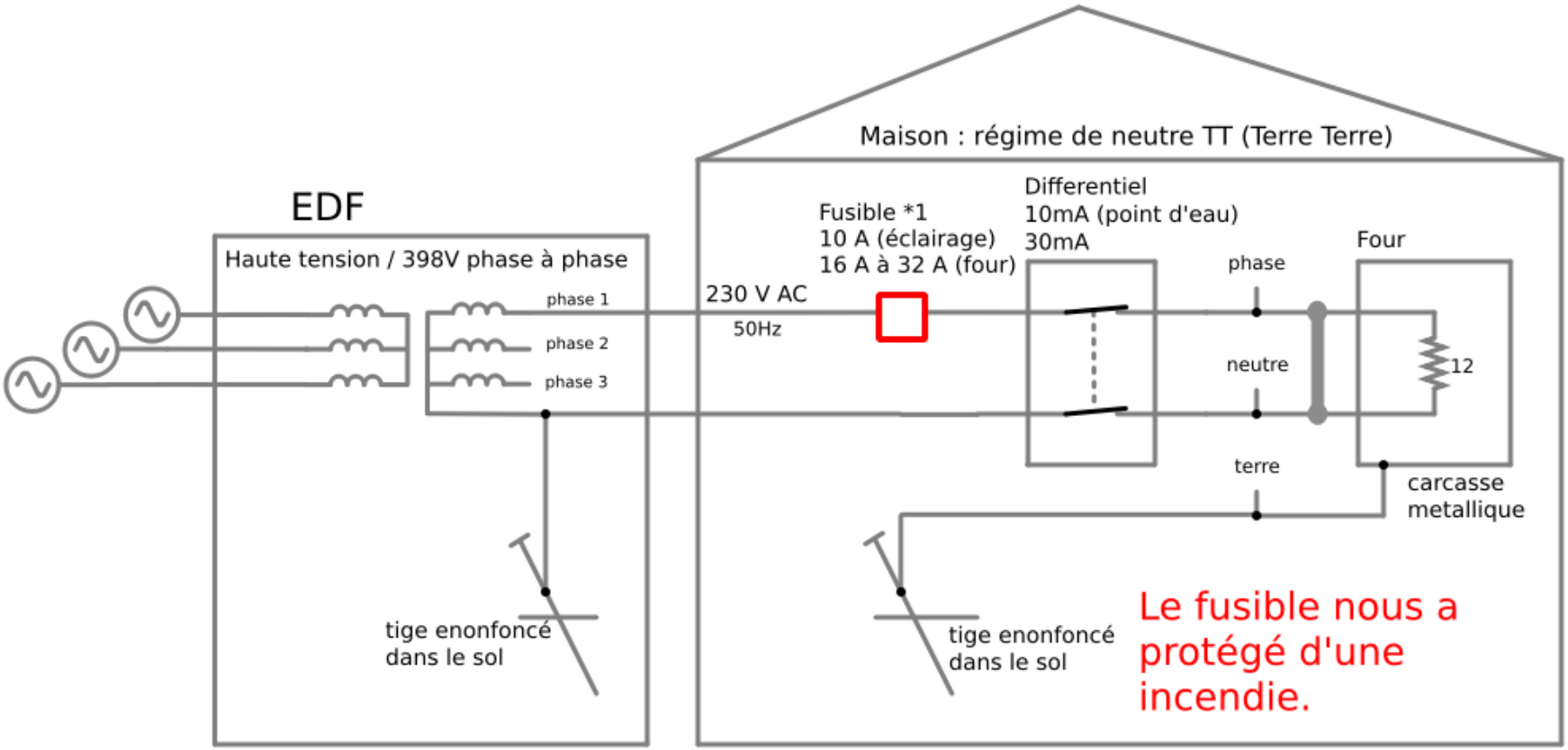


(version longue)

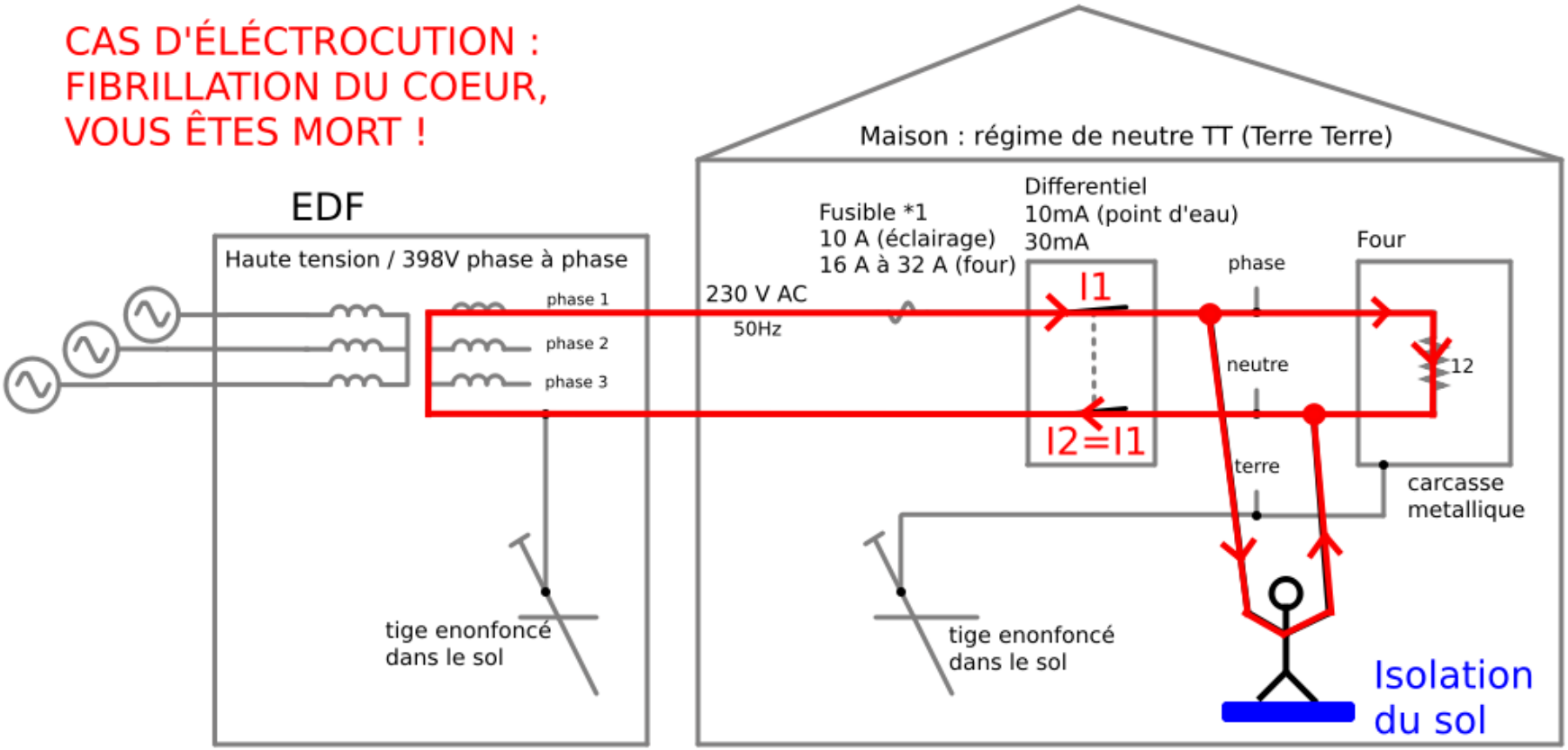
Le fusible : prévention des incendies et du matériel



Le fusible : prévention des incendies et du matériel



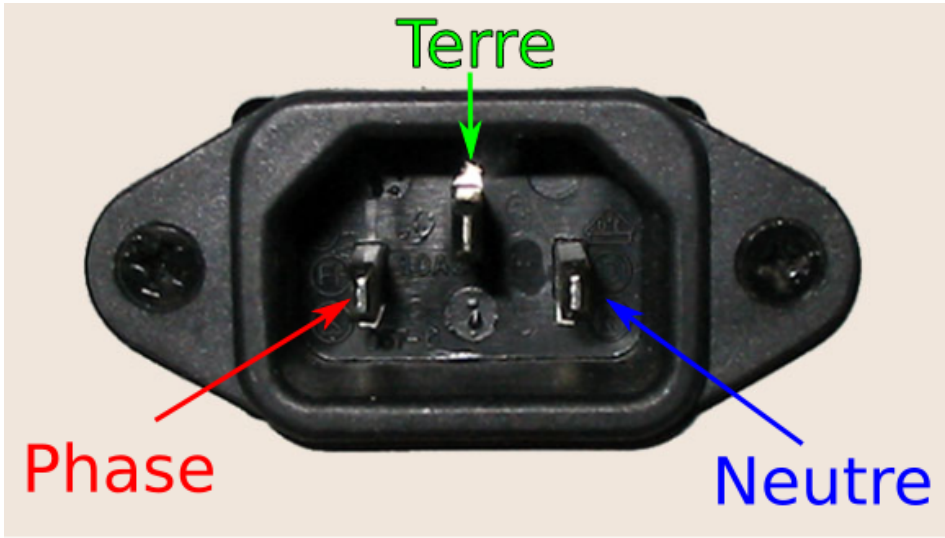
Électrocution, aucune protection ne peut vous sauver.



Quand on touche à la fois le neutre et la phase en étant isolé du sol.

La terre

Tout appareil avec une carlingue métallique doit être relié à la terre.



Prise mâle - IEC60320-C14

A la réception d'un appareil, il faut toujours vérifier, à l'aide d'un multimètre que la prise de terre est reliée à la carlingue métallique, notamment à ses vis métalliques. Attention, la présence de peinture isole. Il faut donc la gratter pour vérifier la connection à la terre de la carlingue.

(version longue)

Les multiprises

Si trop de courant passe dans une multiprise ou une rallonge, elle brûle, car la puissance dissipée est :

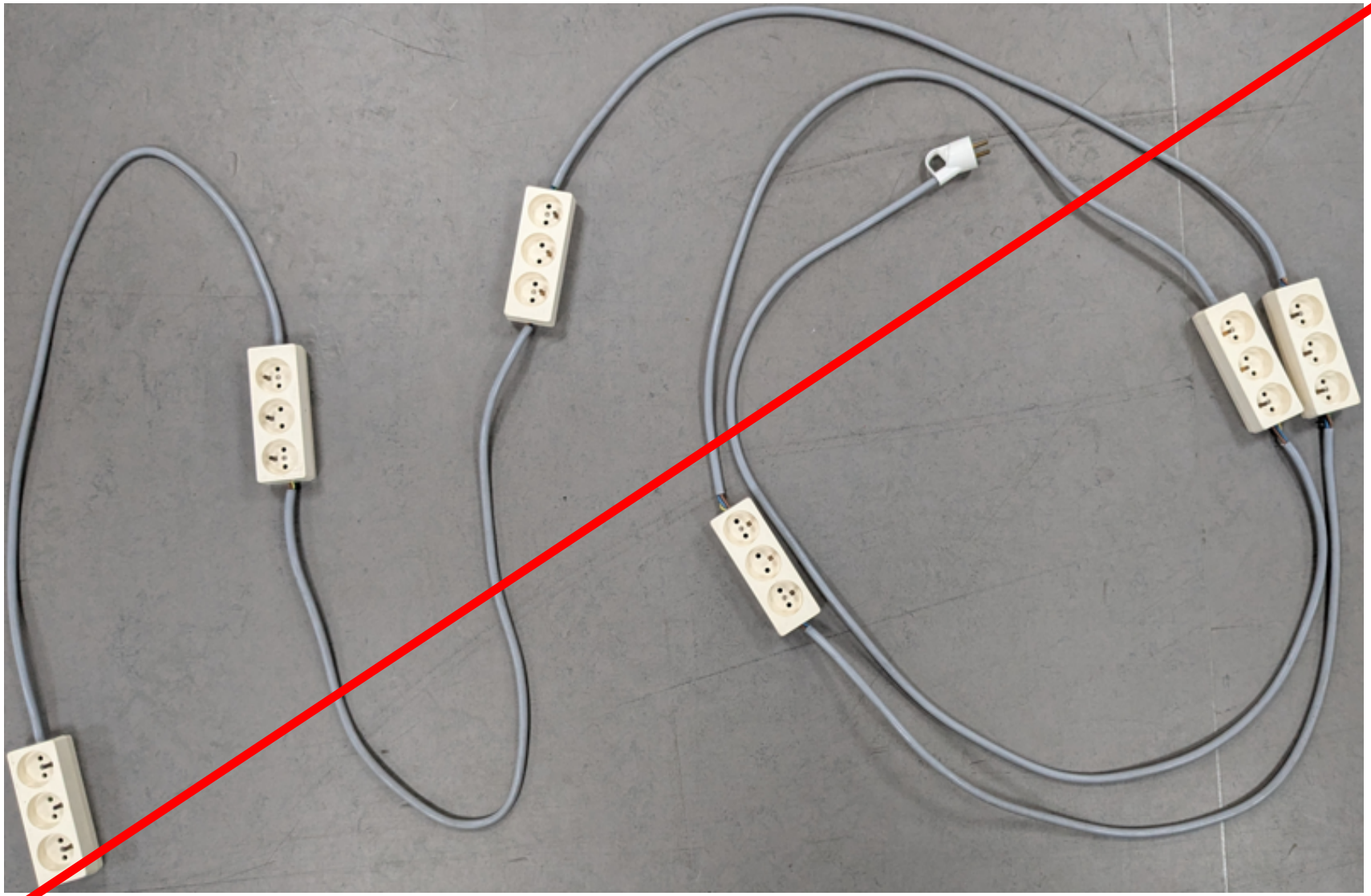
$$P_{\text{dissipé}} = U \times I = R \times I^2.$$

Une multiprise classique de 3500W peut faire passer, un courant total de 16A efficace pour 230V efficace.

Il ne faut pas brancher des appareils dont la puissance totale dépasse la puissance maximale supportée écrite sur la multiprise.

On ne branche pas les multiprises les unes derrières les autres car la première multiprise reçoit la somme des courants de toutes les multiprises qui lui sont branchées dessus.

Vu dans le cabinet des Horreurs. NE PAS REPRODUIRE !



(version longue)

Les rallonges

Si trop de courant passe dans une une rallonge, elle brûle :

$$P_{\text{dissipé}} = U \times I = \rho \times d \times I^2.$$

où ρ est la résistance par mètre et d la longueur de la rallonge.

Une rallonge doit toujours être déroulée pour dissiper la chaleur du courant qui circule dedans !

Plus une rallonge est grande, plus le câble doit avoir une grande section pour réduire sa résistance. Pour éviter les chutes de tensions on ne branche donc pas les rallonges en série.

Un rallonge est généralement prévue pour un seul appareil. On ne branche pas de multiprises sur une rallonge sans vérifier que la rallonge puisse transmettre la puissance maximale de la multiprise.

Concevoir un appareil sécurisé - les classes de protection

Pour installer et utiliser un appareil sans risque d'électrisation, il doit appartenir à l'une des classes de protection 1, 2 ou 3 définies dans les normes NF 61140, NF C15-100 et NF 61558-1, 60664-1 et 60364-4.

classe	caractéristiques	emploi	symbole
0	Isolation principale	Utilisation interdite	Pas de symbole
1	Les surfaces accessibles sont séparées des parties électriques par : – une isolation renforcée, ou – une isolation double, ou – une isolation principale + une isolation par écran (surface conductrice reliée à la terre). Masses reliées entre elles à la terre	Utilisation possible sur les lieux de travail pour les machines fixes	
2	Isolation Renforcée ou bien un double isolation. Masses non reliées à la terre	Utilisation possible sur les lieux de travail pour les machines non fixes	
3	Alimentation externe*1 TBTS ou TBTP Alimentation interne*1 TBTS (+règle suiv. ⇒piles,⋯) *1 à vérifier : l'interpretation alim. interne/externe Ne génère pas de tensions > 12 AC ou > 30V DC Masses non reliées à la terre L'isolation est non nécessaire	Obligatoire sur les appareils portatifs, non fixes en milieu confiné, humide ou mouillé	

Attention ! Le bois, le vernis et la peinture (bien qu'isolant) ne peuvent pas être utilisés pour réaliser une isolation renforcée ou une isolation simple d'une classe de protection.

(version longue)

Isolation renforcée, double, principale et secondaire - 1/4

Dans la norme NF EN C15-100 :

Un **obstacle** est un élément empêchant un contact direct fortuit mais ne s'opposant pas à un contact direct par une action délibérée. Il sert à protèger les personnes qualifiés ou avertis.

- en BasseTension, il empêche tout contact non intentionnel avec des parties actives dangereuse;
- il ne peut pas être utilisé pour protéger des utilisateurs ordinaires;
- son ouverture se fait avec une clé ou un outils, ou bien doit couper l'alimentation;
- il ne doit pas être utilisé dans le cadre d'une isolation supplémentaire.

Une **barrière** est une partie assurant la protection contre les contacts directs dans toute direction habituelle d'accès.

- son ouverture se fait avec une clé ou un outils, ou bien doit couper l'alimentation;
- en BasseTension, la barrière a un indice de protection IPXXB ou IP2X à l'exception des parties horizontales qui sont IPXXD ou IP4X.

Une **Enveloppe** est une enceinte assurant la protection des matériels contre certaines influences externes et dans toutes les directions. Elle assure la protection contre les contacts directs.

Une **Enveloppe de protection** est une enveloppe qui est métallique et reliée à la terre.

Dans la norme NF EN 60529 et 60950 :

Une barrière d'**Indice de portection IP2X ou IPXXB** empêche la pénétration de corps solide étrangers de diamètre ≥ 12.5 mm (la taille d'un doigt).

Une barrière d'**Indice de portection IP4X ou IPXXD** empêche la pénétration de corps solide étrangers de diamètre ≥ 1 mm (le diamètre d'un fil).

(version longue)

Isolation renforcée, isolation principale et secondaire - 2/4

Dans les normes NF EN 60664-1 et NF EN 60950-1:

La **distance d'isolement** est la plus petite distance dans l'air entre deux parties conductrices.

Les normes 60664-1 (tableaux F.1 et F.2) et 60950-1 (tableau 2H + 2K + 2L + 2M) donne des tables pour calculer cette distance :

- dans un "usage domestique pollué" (tension monophasé de 240V, catégorite de surtension III, tension assignée de tenue aux chocs 2500V, degrés de pollution 3 : le pire en milieu sec) cette distance est de 2.1 mm = max(1.5mm, 2 + 0.1mm).
- dans un "atelier pollué" (tension triphasé de 230/400V, catégorite de surtension III, tension assignée de tenue aux chocs 4000V, degrés de pollution 3 : le pire en milieu sec) cette distance est de 3 mm = max(3mm, 2 + 0.1mm).

Un **degré de pollution 3** correspond à la présence d'une pollution conductrice ou d'une pollution sèche, non conductrice, qui devient conductrice par suite de la condensation qui peut se produire.

Un **degré de pollution 4** correspond à une conductivité persistante qui apparaît due à la poussière conductrice, à la pluie ou à d'autres conditions humides.

Isolation renforcée, isolation principale et secondaire - 3/4

La **ligne de fuite** est la distance la plus courte, le long de la surface d'un isolant solide, entre deux parties conductrices.

Les normes 60664-1 (tableaux F.5) et 60950-1 (ableau 2N) donne des tables pour calculer cette distance :

- Dans un "usage domestique pollué", les lignes de fuite des groupes de matériaux sont :

pour une tension phase-neutre de 240V, et un degré de polution 3				
groupe	I	II	IIIa	IIIb
IRC (CTI)*1	≥ 600	$600 > IRC \geq 400$	$400 > IRC \geq 175$	$175 > IRC \geq 100$
ligne de fuite	3.2 mm	3.6 mm	4 mm	4 mm

*1 Indice de Résistance au Cheminement (Comparative Traking Index) du materiau.

- Dans un "atelier pollué", les lignes de fuite des groupes de matériaux sont (normes 60664, tableau F.5:

pour une tension phase-phase de 400V, et un degré de polution 3				
groupe	I	II	IIIa	IIIb
IRC (CTI)*1	≥ 600	$600 > IRC \geq 400$	$400 > IRC \geq 175$	$175 > IRC \geq 100$
ligne de fuite	5.0 mm	5.6 mm	6.3 mm	6.3 mm

*1 Indice de Résistance au Cheminement (Comparative Traking Index) du materiau.

Quelques valeurs approximative d'IRC (CTI) pour materiaux brut (sans colorant ou additif – typiquement le colorant noir peut être réalsé en carbone qui est conducteur) Vérifier les valeurs ! – Phenolic resin: 125; Polyimide et Kapton: 150; FR4 (PCB base material, glass fiber reinforced epoxy resin): ≥ 175/250; FR4 Type KF: 400; PE-LD, PE-HD (polyethylene), Polyester resin, PTFE (polytetrafluoroethylene), PMMA : 600, PBT: 500.

(version longue)

Isolation renforcée, isolation principale et secondaire - 4/3

Dans la norme NF EN C15-100 :

Isolation solide : matériau isolant solide, ou combinaison de matériaux isolants solides, placé entre deux parties conductrices ou entre une partie conductrice et une partie du corps

Isolation principale : Au choix, selon la nature de l'isolant,

- Une couche d'air d'épaisseur la distance d'isolement (cf. page 60 ⇝ 2.1 mm en contexte domestique) + un obstacle, une barrière ou une enveloppe de protection
- Une couche solide d'isolant d'épaisseur la ligne de fuite (cf. page 61 ⇝ 4 mm en contexte domestique). L'isolant doit empécher l'accès aux parties dangereuses.

Isolation supplémentaire : Au choix, selon la nature de l'isolant,

- Une couche d'air d'épaisseur la distance d'isolement (cf. page 60 ⇝ 2.1 mm en contexte domestique) + une barrière;
- Une couche solide d'épaisseur la ligne de fuite (cf. page 61 ⇝ 4 mm en contexte domestique). L'isolant doit empécher l'accès aux parties dangereuses.

Isolation double : Isolation principale + isolation supplémentaire

Isolation renforcée : Au choix, selon la nature de l'isolant,

- Une couche d'air d'épaisseur 2 × la distance d'isolement (cf. page 60 ⇝ 4.2 mm en contexte domestique) + une barrière;
- Une couche solide d'isolant d'épaisseur 2 × la ligne de fuite (cf. page 61 ⇝ 8 mm en contexte domestique) ou moins en cas d'isolant solide (cf. tables d'isolation). L'isolant doit empécher l'accès aux parties dangereuses.

(version longue)

Protection aux chocs mécaniques

La norme NF EN 62262 définit les indices de protections aux chocs IK:

IK	00	01	02	03	04	05	06	07	08	09	10
choc (joule)	non protégé	0.15	0.2	0.35	0.5	0.7	1	2	5	10	20
choc d'un objet de poids de (g)		200	200	200	200	200	500	500	1700	5000	5000
laché sur une hateur de (cm)		2.5	10	17.5	25	35	20	40	29.5	20	40

Le guide UTE C15-103 défini les protections aux chocs des objets à prévoir pour les installations publiques. En voici une approximation :

approximation (par majoration) du guide UTE C15-103

Usage	domestique	industriel	plein air et parking
IK	07 garage, dortoire, buanderie :7 chambre : 2	08	10

Exemple de boite métallique avec un indice de protection IK08 : La boite en aluminium de fibox de taille $600 \times 310 \times 110$ et d'épaisseur comprise entre 4 et 4.5 mm.

Alimentation TBTS, TBTP et transformateurs

Une Alimentation TBTS/SELV (≤ 50 V DC et ≤ 30 V AC) est au choix :

- Un bloc d'alimentation de classe 2 () contenant un transformateur de sécurité () ;
- Une batterie ou pile de tension $\leq 50V$;
- Un paneau solaire de tension $\leq 50V$.

Une Alimentation TBTP/PELV (≤ 50 V DC et ≤ 30 V AC) est :

- Un bloc d'alimentation de classe 1 () contenant un transformateur avec séparation de circuit () délivrant une tension ≤ 50 V DC et ≤ 30 V AC.

les transformateurs (NF EN 61558-1 et 61558-2-4/6/16)

nom	symbole	description
Un transformateur		composant électrique, présent dans les bloc d'alimentation secteur, qui convertit une tension AC (par ex. la tension du secteur 230V) au borne de son enroulement primaire en une autre tension AC (par ex. 24 V) délivré au borne de son enroulement secondaire.
Un transformateur avec séparation de circuit		un transformateur + isolation double ou renforcée entre le primaire et le secondaire.
Un transformateur de sécurité		transformateur de séparation de circuit + la tension de sortie (du secondaire) est ≤ 50 V DC et ≤ 30 V AC.

(version longue)

Alimentation Très Basse Tension de Sécurité (TBTS/SELV)

Alimentation très basse tension de sécurité certifié TBTS (SELV):

Alimentation de sécurité médicale:

Les alimentations de classe II ayant une tension $\leq 50V$ DC ou $\leq 30V$ AC avec la mention : "Medical safety approved (2xMOPP) accroding to EN60601-1/EN60601-1-11" sont des alimentations SELV.

Exemple de logos présents dans la documentation d'une alimentation de sécurité médicale :

IP21 VI ECO Medical 2xMOPP BF rated

c UR us — AAMI/ANSI ES60601-1, ES60601-1-11

TÜVRheinland CERTIFIED — EN 60601-1, IEC 60601-1, www.tuv.com ID 0000000000 — EN60601-1, EN60601-1-11

FC CE

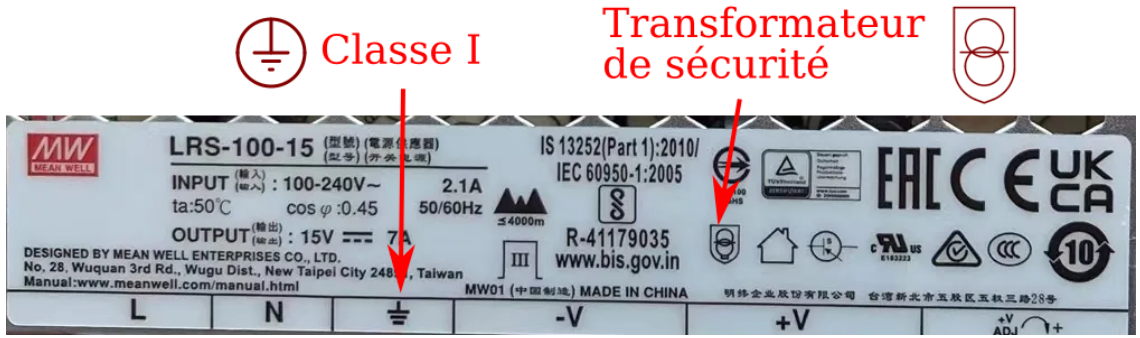
Le niveau 2xMOPP, resp. 2xMOOP, indique une double protection pour le patient, resp. l'opérateur. L'un ou bien l'autre est nécessaire à la qualification SELV.

Les niveaux 1xMOPP et 1xMOOP ne sont pas suffisants. Enfin, MOPP est plus protecteur que MOOP qui correspond au standard (non médical) EN60950.

Remarque : Les symboles UR à l'envers et UL à l'endroit sont des normes américaines de prévention contre les risques potentiels d'incendie, de chocs électriques et de risques mécaniques.

(version longue)

Alimentation Très Basse Tension de Protection (TBTP/PELV)



Remarque : Les symboles UR à l'envers et UL à l'endroit qui sont des normes américaines de prévention contre les risques potentiels d'incendie, de chocs électriques et de risques mécaniques.

Alimentation TBTS, TBTP et appareil de classe 3

Attention !

Un bloc d'alimentation de tension $\leq 50V$ DC et $\leq 30V$ AC de classe 2 () sans autre logo n'est pas une alimentation certifié TBTS/SELV (ni TBTP/PELV).

En effet, sans le symbole  , il faut considérer qu'il contient un simple transformateur (SANS séparation de circuit : ). Il ne doit donc pas être utilisé pour alimenter un appareil de classe 3.

Il ne doit pas alimenter non plus des appareils de classe 1 car ces derniers doivent impérativement être reliés à la terre.

Il ne peut donc qu'alimenter un circuit de classe 2.

Alimentation TBTS, TBTP et appareil de classe 3

Attention !

Un bloc d'alimentation de tension $\leq 50V$ DC et $\leq 30V$ AC de classe 2 () sans autre logo n'est pas une alimentation certifié TBTS/SELV (ni TBTP/PELV).

En effet, sans le symbole  , il faut considérer qu'il contient un simple transformateur (SANS séparation de circuit : ). Il ne doit donc pas être utilisé pour alimenter un appareil de classe 3.

Il ne doit pas alimenter non plus des appareils de classe 1 car ces derniers doivent impérativement être reliés à la terre.

Il ne peut donc qu'alimenter un circuit de classe 2.

Exemple d'appareils de classe 0, INTERDIT !

- La guirlande de noël du grand père dont les ampoules à fil sont alimentés par le secteur 230V via un fil avec une isolation simple.
- Une sirène basse tension DIY racordée au secteur et réalisée dans une boite en bois;
- Les appareils sans prise de terre avec une carlingue métallique (sauf cas particuliers)

Exemple d'appareils de classe 0, INTERDIT !

- La guirlande de noël du grand père dont les ampoules à fil sont alimentés par le secteur 230V via un fil avec une isolation simple.

  Remise au norme : jettez la guirelande et remplacez là par une guirelande à led 5V ou 12V alimentée par un bloc d'alimentation certifié SELV.

- Une sirène basse tension DIY racordée au secteur et réalisée dans une boite en bois;

- Les appareils sans prise de terre avec une carlingue métallique (sauf cas particuliers)

Exemple d'appareils de classe 0, INTERDIT !

- La guirlande de noël du grand père dont les ampoules à fil sont alimentés par le secteur 230V via un fil avec une isolation simple.

  Remise au norme : jettez la guirelande et remplacez là par une guirelande à led 5V ou 12V alimentée par un bloc d'alimentation certifié SELV.

- Une sirène basse tension DIY racordée au secteur et réalisée dans une boite en bois;

  Remise au norme 1 : Si l'appareil n'était pas relié à la terre remplacez la boite en bois par une boîte en plastique qualifiée isolation renforcée. Vous obtiendrez un appareil de classe 2.

- Les appareils sans prise de terre avec une carlingue métallique (sauf cas particuliers)

Exemple d'appareils de classe 0, INTERDIT !

- La guirlande de noël du grand père dont les ampoules à fil sont alimentés par le secteur 230V via un fil avec une isolation simple.

  Remise au norme : jettez la guirelande et remplacez là par une guirelande à led 5V ou 12V alimentée par un bloc d'alimentation certifié SELV.

- Une sirène basse tension DIY racordée au secteur et réalisée dans une boite en bois;

  Remise au norme 1 : Si l'appareil n'était pas relié à la terre remplacez la boite en bois par une boîte en plastique qualifiée isolation renforcée. Vous obtiendrez un appareil de classe 2.

  Remise au norme 2 : Si l'appareil était relié à la terre, remplacez la boite en bois par une boîte en métal, connectez la boîte à la terre, connectez les masses du circuit secondaire à la terre aussi, au plus proche du transformateur. Vous obtiendrez un appareil de classe 1.

- Les appareils sans prise de terre avec une carlingue métallique (sauf cas particuliers)

Exemple d'appareils de classe 0, INTERDIT !

- La guirlande de noël du grand père dont les ampoules à fil sont alimentés par le secteur 230V via un fil avec une isolation simple.

  Remise au norme : jettez la guirelande et remplacez là par une guirelande à led 5V ou 12V alimentée par un bloc d'alimentation certifié SELV.

- Une sirène basse tension DIY racordée au secteur et réalisée dans une boite en bois;

  Remise au norme 1 : Si l'appareil n'était pas relié à la terre remplacez la boite en bois par une boîte en plastique qualifiée isolation renforcée. Vous obtiendrez un appareil de classe 2.

  Remise au norme 2 : Si l'appareil était relié à la terre, remplacez la boite en bois par une boîte en métal, connectez la boîte à la terre, connectez les masses du circuit secondaire à la terre aussi, au plus proche du transformateur. Vous obtiendrez un appareil de classe 1.

- Les appareils sans prise de terre avec une carlingue métallique (sauf cas particuliers)

  Remise au norme : Remplacez le cable d'alimentation par un cordon possédant un cable de terre. Connectez le cable de terre à la carlingue ainsi qu'à la masse du circuit secondaire. Vous obtenez un appareil de classe 1.

(version longue)

Exemple d'appareils de classe 0, INTERDIT ! (suite)

- Une alimentation de laboratoire DIY racordée au secteur et réalisée dans une boite en bois;

A FAIRE : Ajouter la remarque suivante : La remise au norme 2 est valable que si l'appartement est au norme et que la mise à la terre est soigneuse. Sinon, c'est pire que tout.

Exemple d'appareils de classe 0, INTERDIT ! (suite)

- Une alimentation de laboratoire DIY racordée au secteur et réalisée dans une boite en bois;

  Comme il s'agit d'une alimentation de laboratoire, la basse tension est accessible à l'utilisateur. Il faut donc vérifier que l'alimentation est TBTS (SELV) ou TBTP (PELV).

  Remise au norme 1 : Si l'appareil n'était pas relié à la terre, vérifiez que la partie puissance de l'alimentation est TBTS (SELV), si non, remplacez-la par sa version TBTS. Remplacez la boite en bois par une boîte en plastique qualifiée isolation renforcée. Vous obtiendrez un appareil de classe 2.

A FAIRE : Ajouter la remarque suivante : La remise au norme 2 est valable que si l'appartement est au norme et que la mise à la terre est soigneuse. Sinon, c'est pire que tout.

Exemple d'appareils de classe 0, INTERDIT ! (suite)

- Une alimentation de laboratoire DIY racordée au secteur et réalisée dans une boite en bois;

  Comme il s'agit d'une alimentation de laboratoire, la basse tension est accessible à l'utilisateur. Il faut donc vérifier que l'alimentation est TBTS (SELV) ou TBTP (PELV).

  Remise au norme 1 : Si l'appareil n'était pas relié à la terre, vérifiez que la partie puissance de l'alimentation est TBTS (SELV), si non, remplacez-la par sa version TBTS. Remplacez la boite en bois par une boîte en plastique qualifiée isolation renforcée. Vous obtiendrez un appareil de classe 2.

  Remise au norme 2 : Si l'appareil était relié à la terre, vérifiez que la partie puissance de l'alimentation est TBTP (PELV), si non, remplacez-la par sa version TBTP. Remplacez ensuite la boite en bois par une boîte en métal, connectez la boîte à la terre, connectez les masses du circuit secondaire à la terre aussi, au plus proche du transformateur. Vous obtiendrez un appareil de classe 1.

A FAIRE : Ajouter la remarque suivante : La remise au norme 2 est valable que si l'appartement est au norme et que la mise à la terre est soigneuse. Sinon, c'est pire que tout.

Exemple d'appareils de classe 0, INTERDIT ! (suite)

- Un projet DIY contenant, dans une boite en bois, un circuit de classe 3 alimenté par un bloc d'alimentation de classe 2. Ce bloc est lui aussi dans la boite et est relié au secteur par le cordon prévu à cet effet. Le bloc fournit une tension suffisament faible ($<50V$ DC et $<30V$ AC) mais sans certification SELV.

Exemple d'appareils de classe 0, INTERDIT ! (suite)

- Un projet DIY contenant, dans une boite en bois, un circuit de classe 3 alimenté par un bloc d'alimentation de classe 2. Ce bloc est lui aussi dans la boite et est relié au secteur par le cordon prévu à cet effet. Le bloc fournit une tension suffisament faible ($<50V$ DC et $<30V$ AC) mais sans certification SELV.

  Remise au norme 1 : Remplacez le bloc d'alimentation à l'interieur de la boîte en bois par un bloc d'alimentation certifié SELV. Vous obtiendrez un appareil de classe 2.

Exemple d'appareils de classe 0, INTERDIT ! (suite)

- Un projet DIY contenant, dans une boite en bois, un circuit de classe 3 alimenté par un bloc d'alimentation de classe 2. Ce bloc est lui aussi dans la boite et est relié au secteur par le cordon prévu à cet effet. Le bloc fournit une tension suffisament faible ($<50V$ DC et $<30V$ AC) mais sans certification SELV.

  Remise au norme 1 : Remplacez le bloc d'alimentation à l'interieur de la boîte en bois par un bloc d'alimentation certifié SELV. Vous obtiendrez un appareil de classe 2.

  Remise au norme 2 : Remplacez la boîte en bois et tout conducteur accessible par une boite en plastique (au moins IP4X). Vous obtiendrez un appareil de classe 2.

Exemple d'appareils de classe 1

La présence des symboles suivant implique que l'appareil est de classe 1

 ou 

- Les appareils avec une carlingue métallique : Le frigo, le four, le lave-linge, les machines outils;
- Les radiateurs électriques avec une prise de terre;
- Les alimentations de laboratoire, les oscilloscopes;
- Les ordinateurs fixes;
- Vos projets DIY utilisant des tensions $\geq 50V$ DC ou $\geq 30V$ AC. Le plus simple est d'utiliser pour le projet une boîte métalique reliée à la terre. On visse, pour cela, sur la boite le fil de terre du secteur.

Terre (fonctionelle ou de protection), masse, chassis, équipotentiel

Nom	masse ou chassis	terre (raisons non spécifiées)	terre fonctionelle ou terre sans bruit	terre de protection	équipotentiel ou potentiel de référence
Symbole (*norme* IEC 60617–2)					

La masse ou le chassis est la partie conductrice d'un matériel, susceptible d'être touchée, et qui n'est pas normalement sous tension, mais peut le devenir lorsque l'isolation principale est défaillante.

La terre est la masse conductrice de la terre dont le potentiel électrique en chaque point est considéré comme égal à zéro.

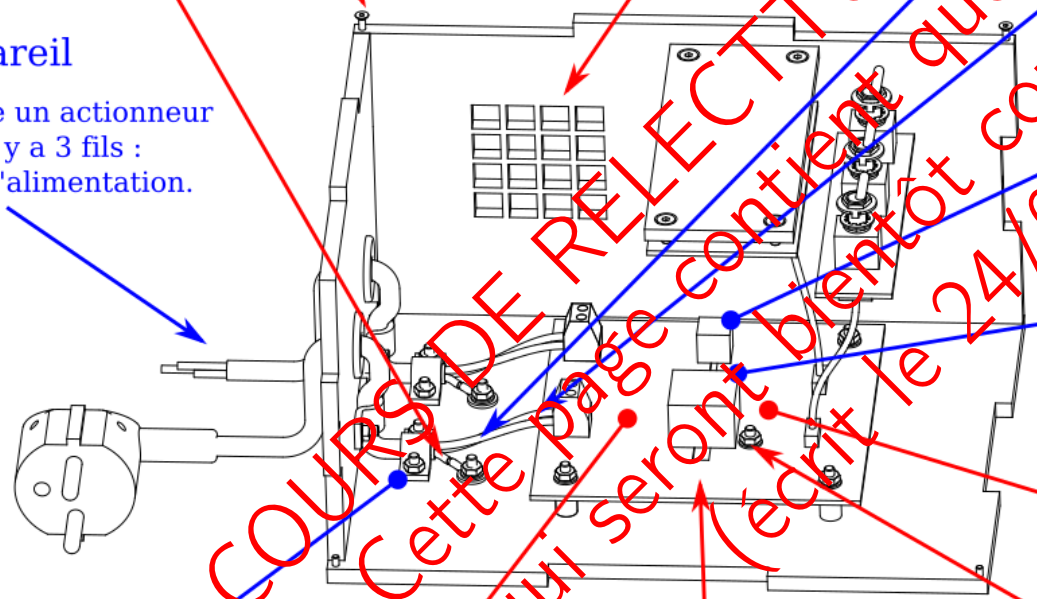
La terre fonctionelle ou terre sans bruit est une connection reliée à la terre uniquement pour réduire le bruit électromagnétique afin d'assurer une fonctionalité (comunication, métrologie). Le fil associé ne doit JAMAIS être jaune et verre (car il n'assure aucune protection). A FAIRE : FAIRE TOUT RENTRER DANS LA SLIDE

La terre de protection est une connection reliée à la terre pour des raisons uniquement de protection et de sécurité électrique.

Le potentiel de référence est le potentiel qui sert de référence dans un circuit isolé galvaniquement de la terre et des autres circuits. Ce potentiel est donc flottant vis à vis de la terre et des autres circuits. Il peut donc valoir n'importe quelle tension. Lorsque l'on étudie ce

(version longue)

Exemple de montage DIY de classe 1 en contexte "domestique pollué" – 1/4



Terre

(vert et jaune)
le fil de terre est vissée à la carlinge avec une cosse.

Vis de fixation

La Boîte ne doit pouvoir être ouverte qu'à l'aide d'une clef ou d'un tourne vis !

Grille d'aération

Indice de protection IP2X pour les paroies verticales : la taille des trous doivent être inférieure à 12.5 mm (la taille d'un doigt)

Phase (marron)

Neutre (bleu)

Sortie de l'appareil

l'appareil commande un actionneur alimenté en 230V. Il y a 3 fils : la terre + les 2 fils d'alimentation.

Relai

Il sépare l'interrupeteur qui est du côté primaire du circuit, de la commande qui est du côté du secondaire.

Transformateur

Prise mâle 230V type E norme CEE 7/7

Circuit secondaire (≤ 50V)

Cavalier

Il sert à empécher les fils d'être arrachés lors de la manipulation de l'appareil.

Circuit primaire (côté 230V)

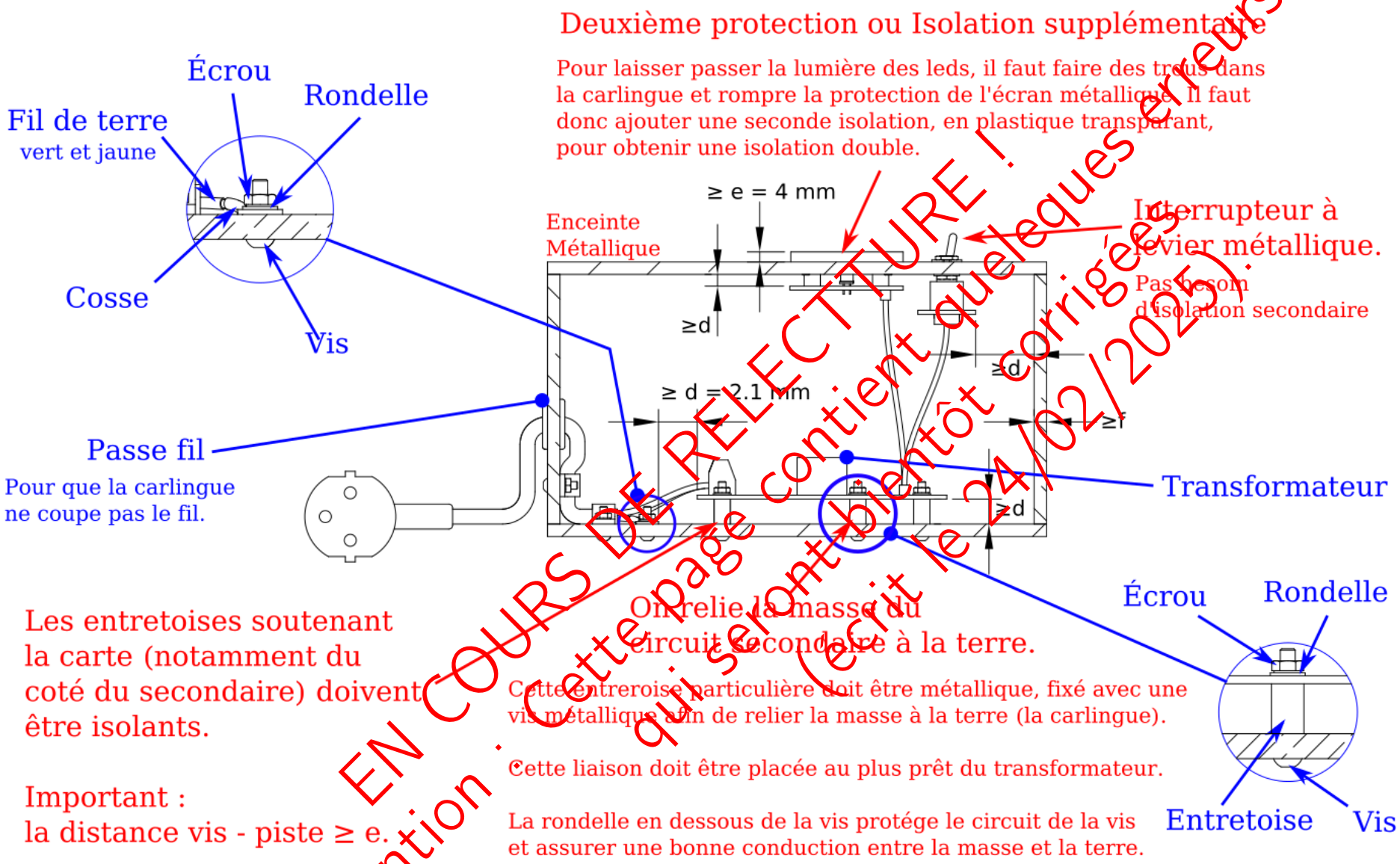
Le transformateur et le relai offrent une isolation galvanique entre le primaire et le secondaire

Mise à la terre du secondaire (Obligatoire !)

Attention : EN COURS DE RELECTURE ! Cette page contient quelques erreurs. qui seront bientôt corrigées. (écrit le 24/02/2025).

(version longue)

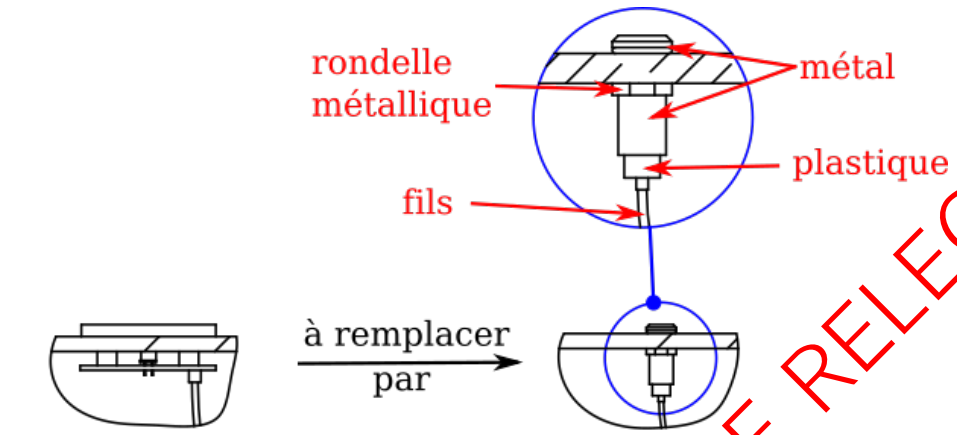
Exemple de montage DIY de classe 1 en contexte "domestique pollué" – 2/4



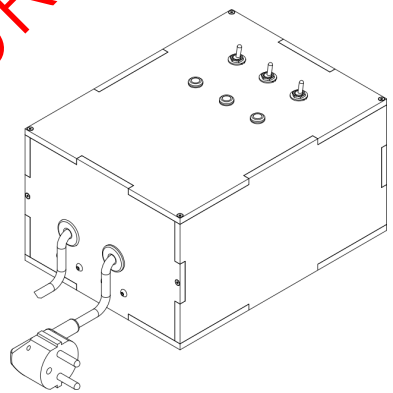
Norme 60664-1 et 60950-1 : distance dans l'air $d \geq 2.1mm$ (tension efficace = 240 V et choc éléctrique ≤ 2500 V); ligne de fuite $e \geq 4mm$ (tension efficace = 240 V et groupe de materiaux *IIIb*). La valeur de f varie selon les boîtes. Par exemple, fibox propose une boite de taille $600X310X110$ où $f \in [4,4.5]$ mm et l'indice de protection aux chocs est IK8 (Guide UTE C15-103 : à la maison ⇝ IK7; site industriel ⇝ IK8; parking et lieu plein air ⇝ IK10). (version longue)

Exemple de montage DIY de classe 1 en contexte "domestique pollué" – 3/4

Il est bien plus sécuritaire de remplacer les leds et leurs isolations suplémentaires par des voyants led industriels compatbiles avec la classe 1 !

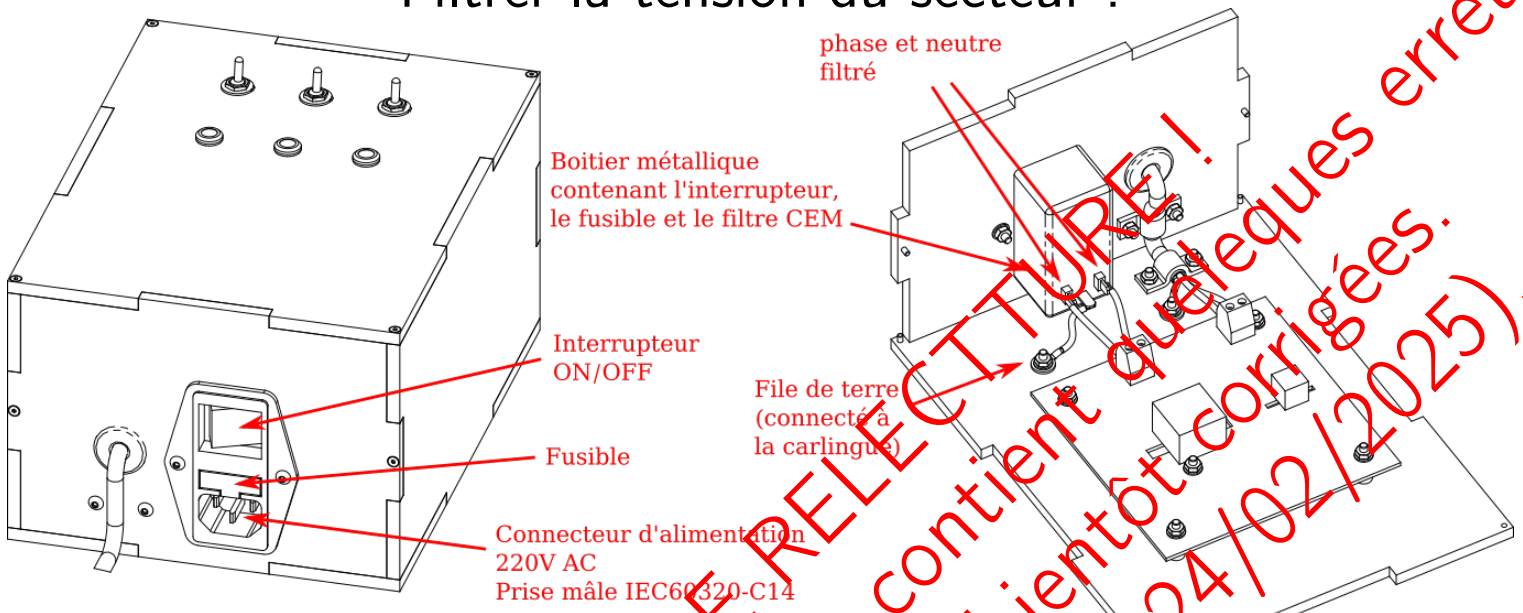


Voyant led industriel
ou
Indicateur de panneau



Exemple de montage DIY de classe 1 en contexte "domestique pollué" – 4/4

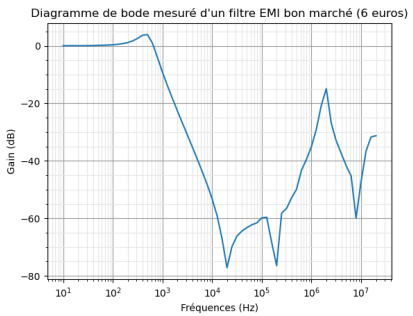
Filtrer la tension du secteur :

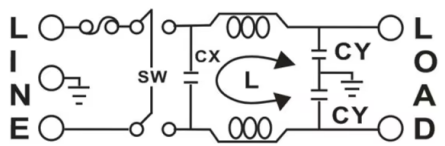


Il ne s'agit pas d'une mesure de sécurité, cette mesure est optionnelle.

Filtrer l'alimentation permet de protéger le circuit du bruit elctromagnétique exterieur ou de ne pas polluer le secteur.

Le plus simple est d'acheter un filtre EMI (ElectroMagnetic Interference) tout prêt contenant aussi le connecteur 230V, un fusible et un interrupteur ON/OFF le tout compatible au design d'une classe 1.





$C_X = 100nF, \quad L = 2 \times 3.3mH, \quad C_Y = 3.3nF$

Pour plus d'information sur ce filtre, consulter le livre "Guide pratique de la CEM" de Alain Charoy (2017, Dunod).

(version longue)

Exemple d'appareils de classe 2

- Les blocs d'alimentations secteur (certifiés ou non certifiés SELV);
- Les ordinateurs portables (classe 2 avec terre fonctionnelle  : la masse est reliée à la terre pour des raisons uniquement fonctionelles, pour réduire le bruit);
- Les chargeurs de téléphone;
- Les perceuses électriques avec ou sans fils;
- Certaines lampes et radiateurs électriques sans prise de terre;
- Les sèches-cheveux;
- Les petits appareils électroménagés (batteur, robot, ...).

Exemple d'appareils de classe 3

- Matériel de piscine;
- Appliques imergées, lampes et interrupteur de salle de bains;
- Vos projets DIY de tension $\leq 50V$ DC et $\leq 30V$ AC alimentés par une batterie, le tout enpaqueté dans une boîte en bois ou en plastique pour protéger la batterie des chocs;
- Vos projets DIY de tension $\leq 50V$ DC et $\leq 30V$ AC alimenté via une prise Jack (sur laquel il sera possible de connecter un bloc d'alimentation certifié SELV). Le circuit pourra être indiférement à l'air libre ou protégé par une boîte en bois ou en plastique avec la prise Jack en facade.

Quelle classe utiliser dans vos projets DIY ? A vérifier/relire

Lorsque l'on utilise des tensions $\leq 30V$ DC ou $\leq 12V$ AC, le plus simple est de créer un appareil de classe III alimenté à l'aide d'une alimentation externe (via une prise Jack), puis d'alimenter l'appareil à l'aide d'un bloc d'alimentation secteur très basse tension de sécurité (SELV).

Pour des tensions supérieures, quand on cherche à rendre public l'objet et/ou le vendre, on préfère souvent créer un objet de classe I car la classe II est bien plus compliquée à qualifier. En effet, il faut prouver que l'utilisateur ne peut pas, en toute circonstance, rentrer en contact avec une tension interne de l'appareil. Pour la classe I, il suffit "d'entourer l'appareil d'une enveloppe métallique reliée à la terre".

Maintenant, du point de vue de la sécurité, la discussion diffère.

La sécurité de la classe II repose sur l'isolation de l'objet entre ses parties actives et l'utilisateur. Ainsi, bien conçu, un objet de classe II est très sécuritaire pusique il n'y a aucun moyen de rentrer en contact avec les parties actives dangereuses de l'appareil.

Dans un matériel de classe I, la sécurité repose sur le disjoncteur différentiel de l'installation électrique de votre bâtiment. Ainsi, un appareil de classe I, relié à un réseau électrique qui n'est pas au norme, devient un objet extrèmenent dangereux puisque le moindre defaut vous expose, avec certitude, à cause de la cralingue métallique, au courant du secteur.

Ainsi, il vaut mieux un matériel de classe II mal conçu qu'un matériel de classe I mal conçu ou relié à un réseau électrique défaillant (pas de différentiel ou pas de prise de terre par exemple).

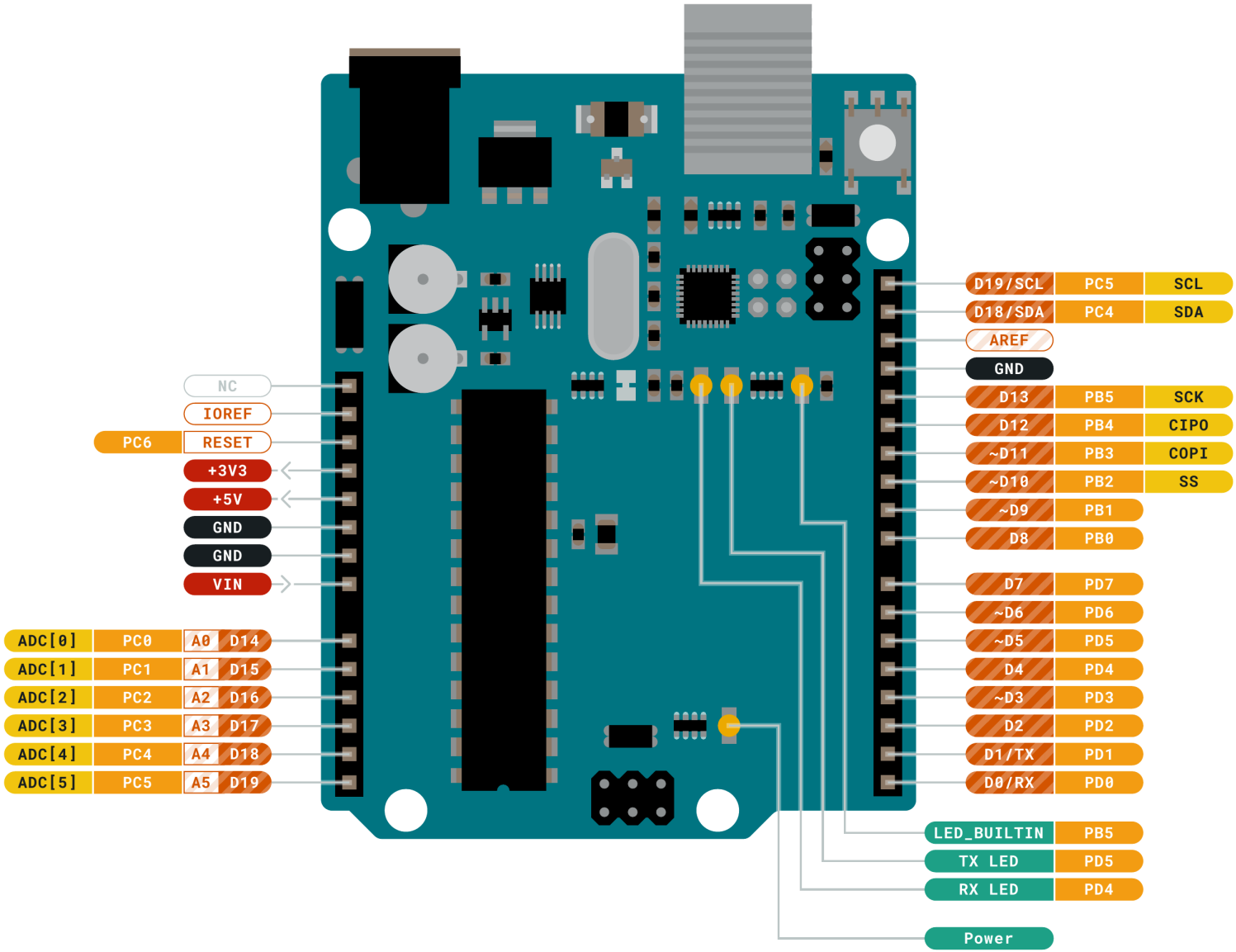
Dans un contexte DIY où la sécurité des réseaux électriques et des objets peuvent être douteuse, à défaut de refuser d'utiliser le réseau, pour utiliser un appareil de classe I, il faut vérifier les prises terres du secteur, vérifier que le disjoncteur differentiel fonctionne et vérifier la connection des carlingues à la terre.

(version longue)

Plan

(présent dans la version courte)

Présentation de la carte Arduino Uno R3



(présent dans la version courte)

Présentation de la carte Arduino Uno R3

Arduino Uno est OpenHardware

Prix : 28 euros (officiel), 6 euros (non officiel).

Microcontrôleur : Atmega328P

Types gérés par la carte :

- Entier 8 bits : natifs ($[-127,127]$ ou $[0,255]$).
- Entier 16 bits : natifs ($[-32767,32767]$ ou $[0,65535]$).
- Entier 32 et 64 bits : émulés !
- Réel 32 et 64 bits : émulés !

Fréquence du microcontrôleur : 16 Mhz.

(présent dans la version courte)

Table des types pour l'Arduino Uno R3

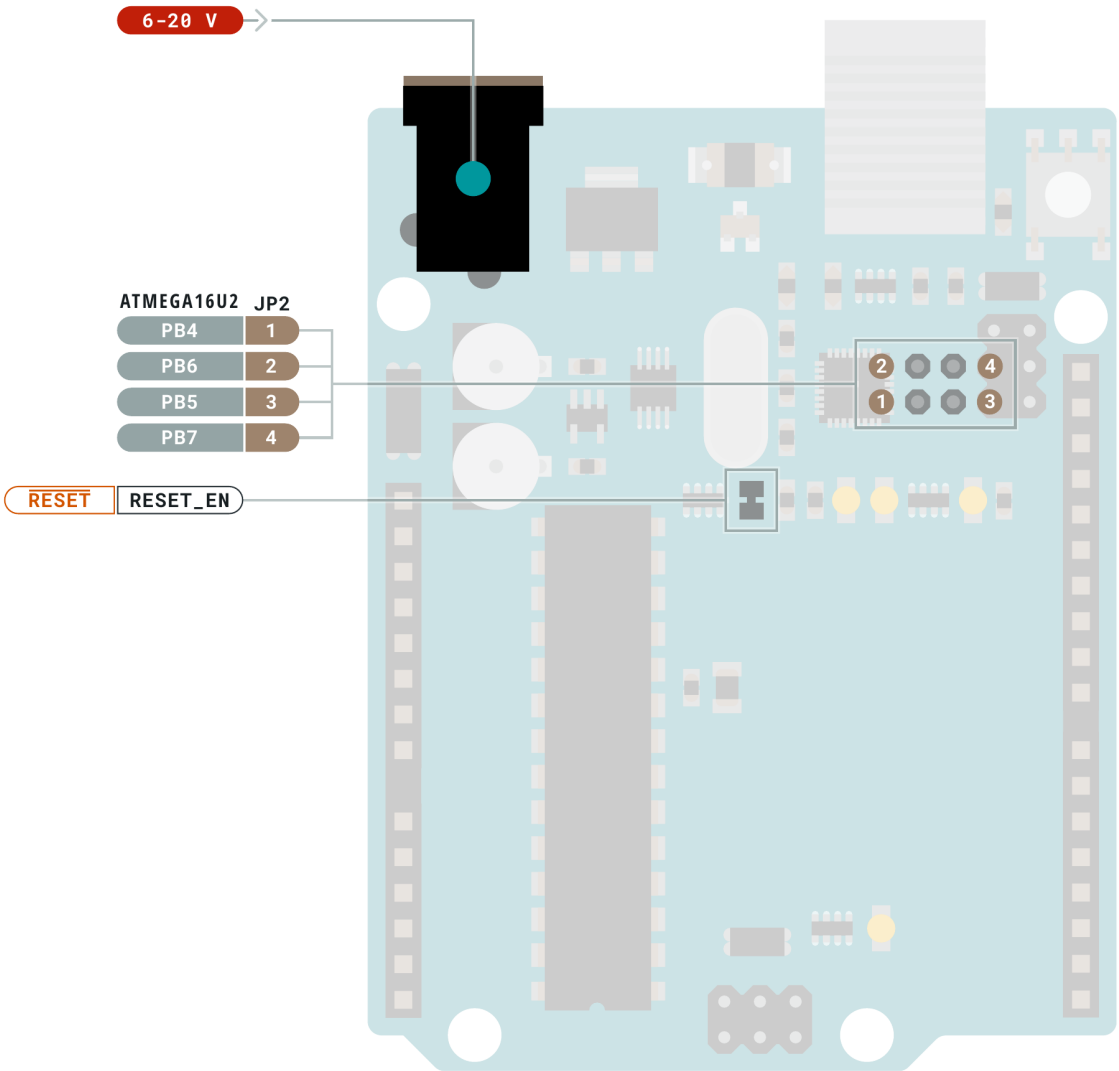
Contrairement aux esp32 et aux ordinateurs, les types int et short sont identiques !

type	natif	entier				réel		signé	intervale
		nombre de bits							
		8	16	32	64	32	64		
char int8_t	✓	✓						✓	[−127, 127]
unsigned char uint8_t	✓	✓							[0, 255]
int = short int16_t	✓		✓					✓	[−32767, 32767]
unsigned int uint16_t	✓		✓						[0, 65535]
long int int32_t				✓				✓	[−2147483647, 2147483647]
unsigned long int uint32_t				✓					[0, 4294967295]
long long int int64_t					✓			✓	$[-2^{63}+1, 2^{63}-1]$
unsigned long long int uint64_t					✓				$[0, 2^{64}-1]$
float						✓		✓	
double							✓	✓	

Les types char, int, long int et long long int et leurs versions non signées, dépendent de l'architecture. Il vaut mieux utiliser leurs versions explicites : int8_t, int16_t, int32_t, ...

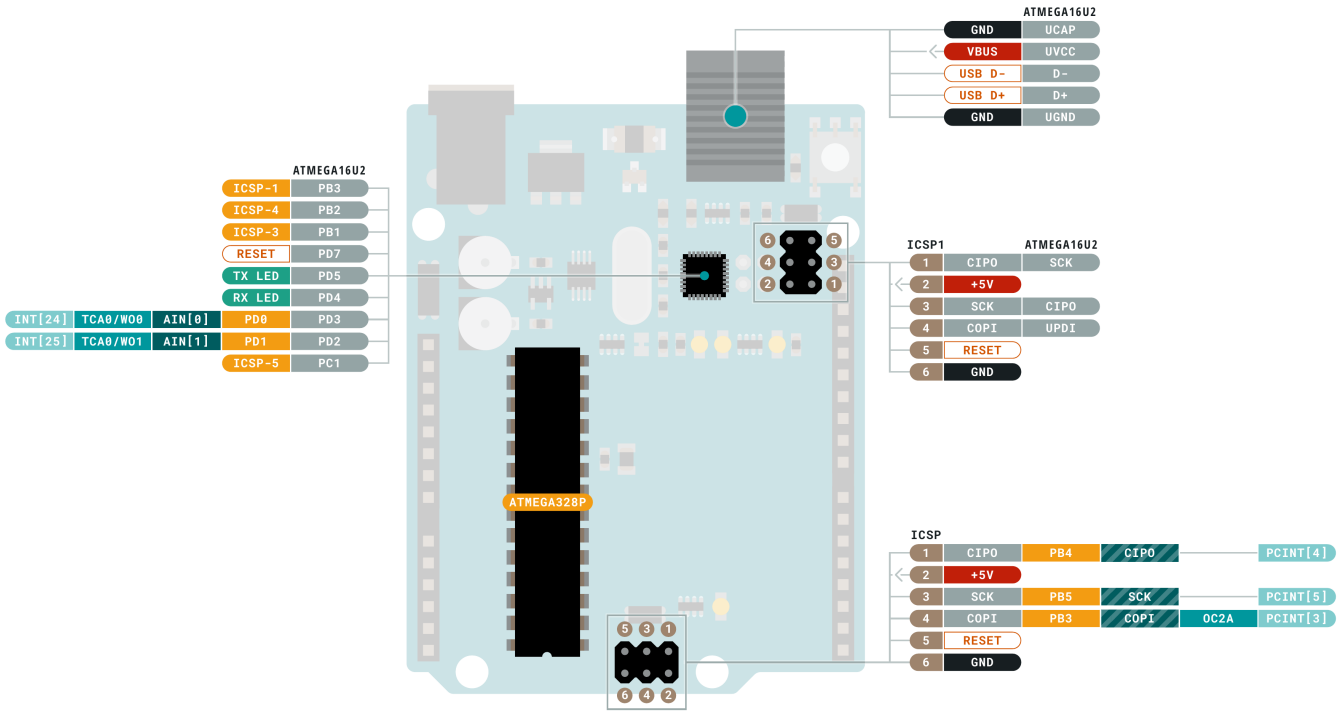
(version longue)

Présentation de la carte Arduino Uno R3 - puissance



(version longue)

Présentation de la carte Arduino Uno R3 - complet 2

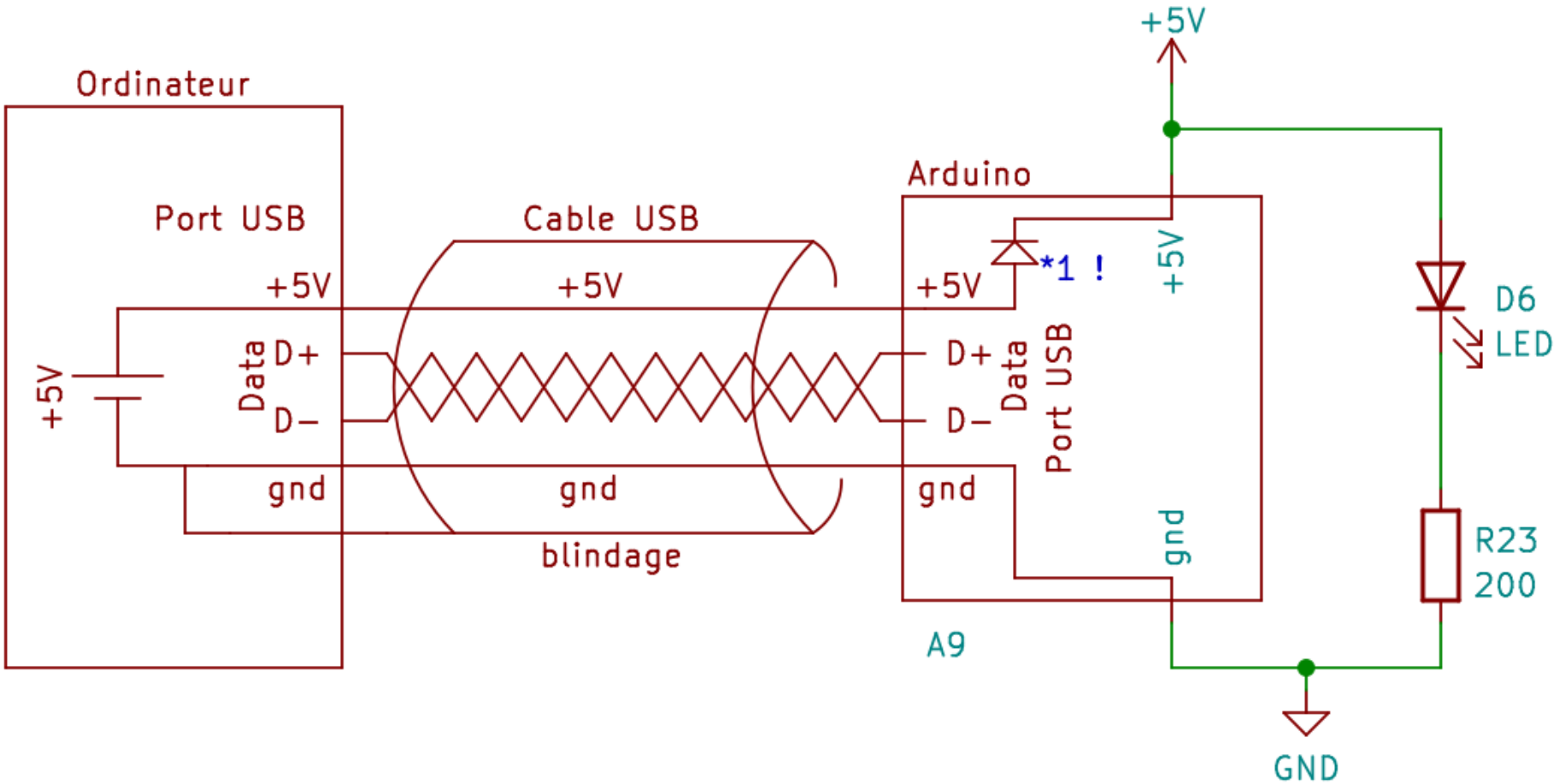


(version longue)

Plan

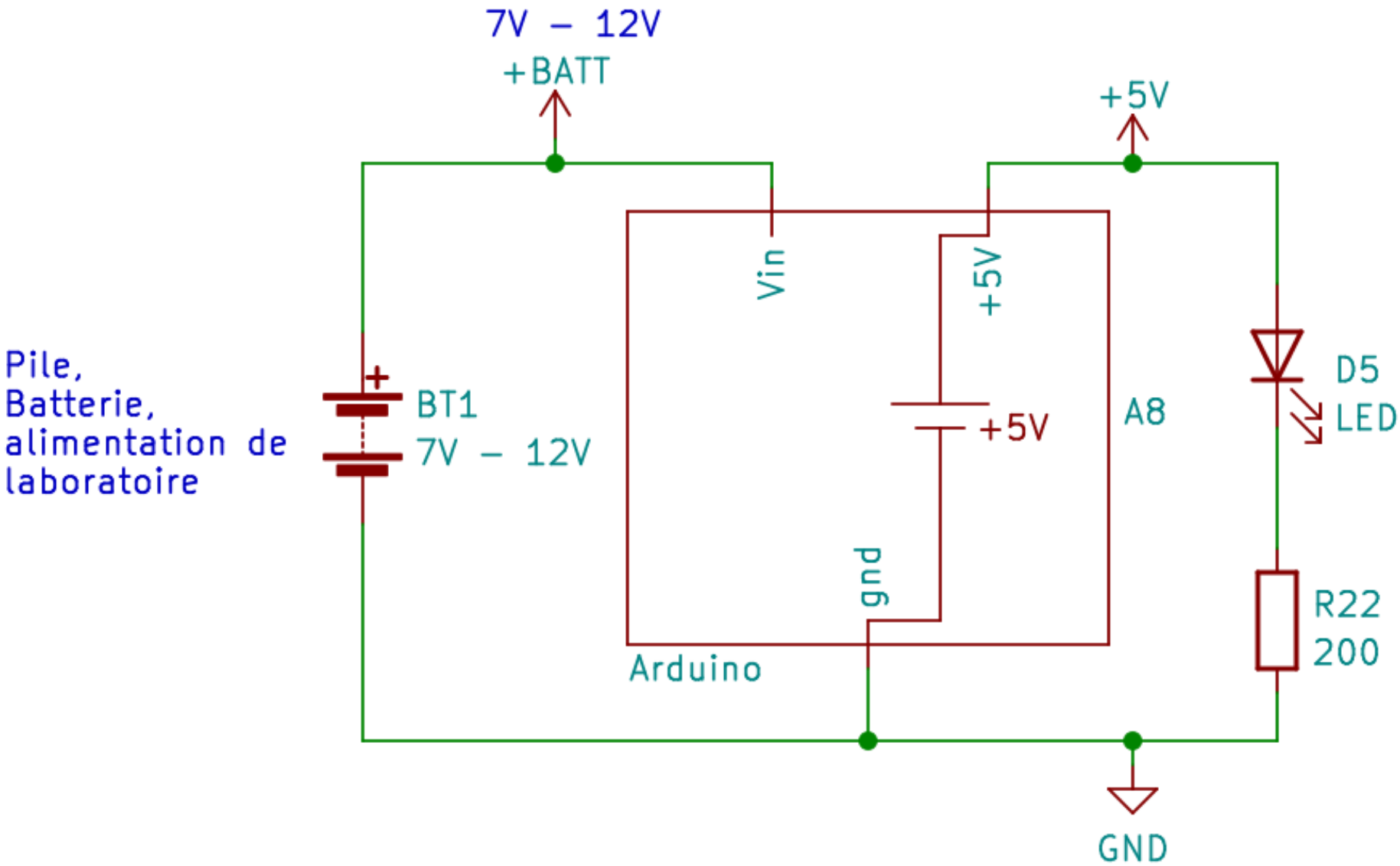
(présent dans la version courte)

Alimenter votre circuit avec Arduino via USB



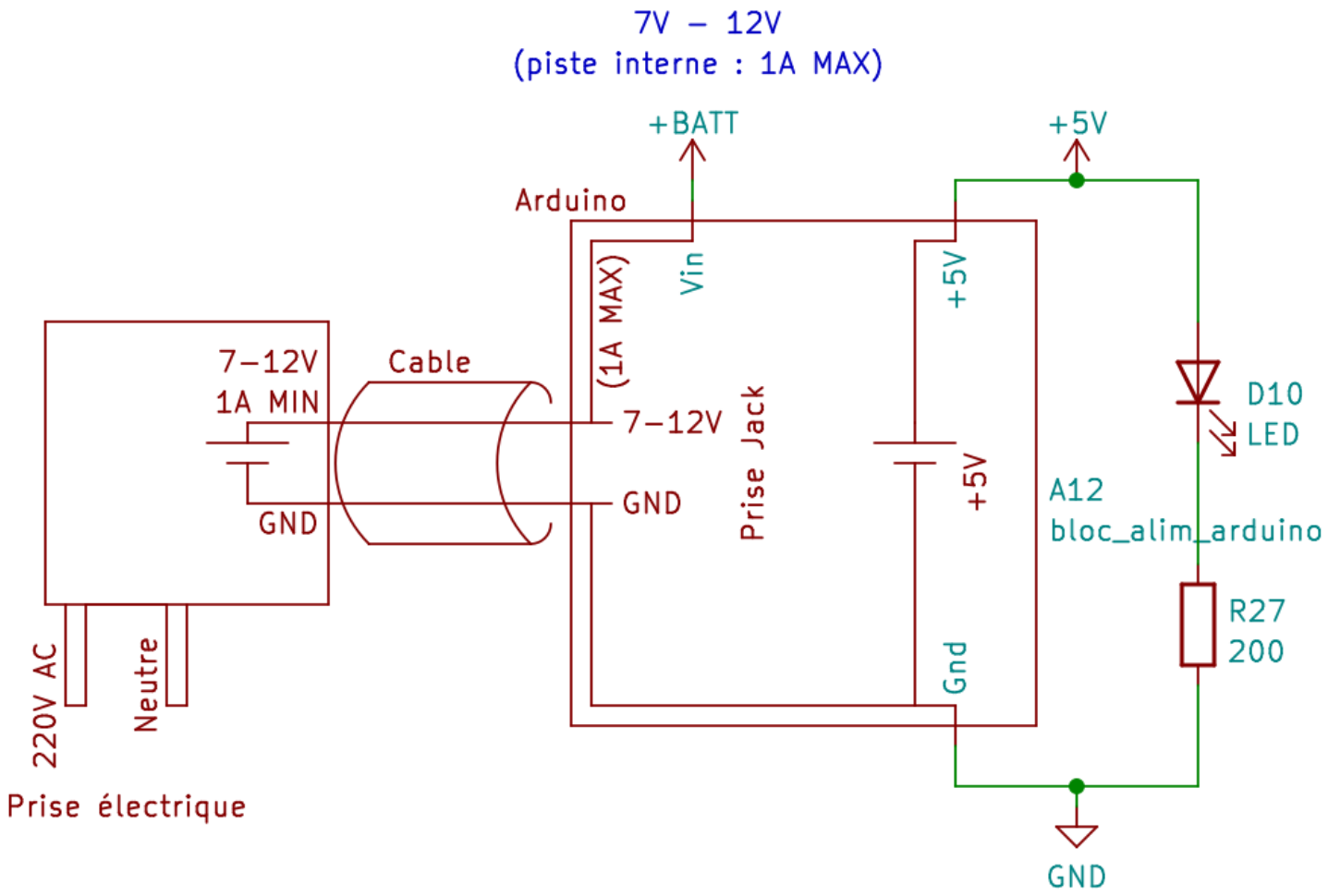
(présent dans la version courte)

Alimenter votre circuit avec Arduino via une batterie



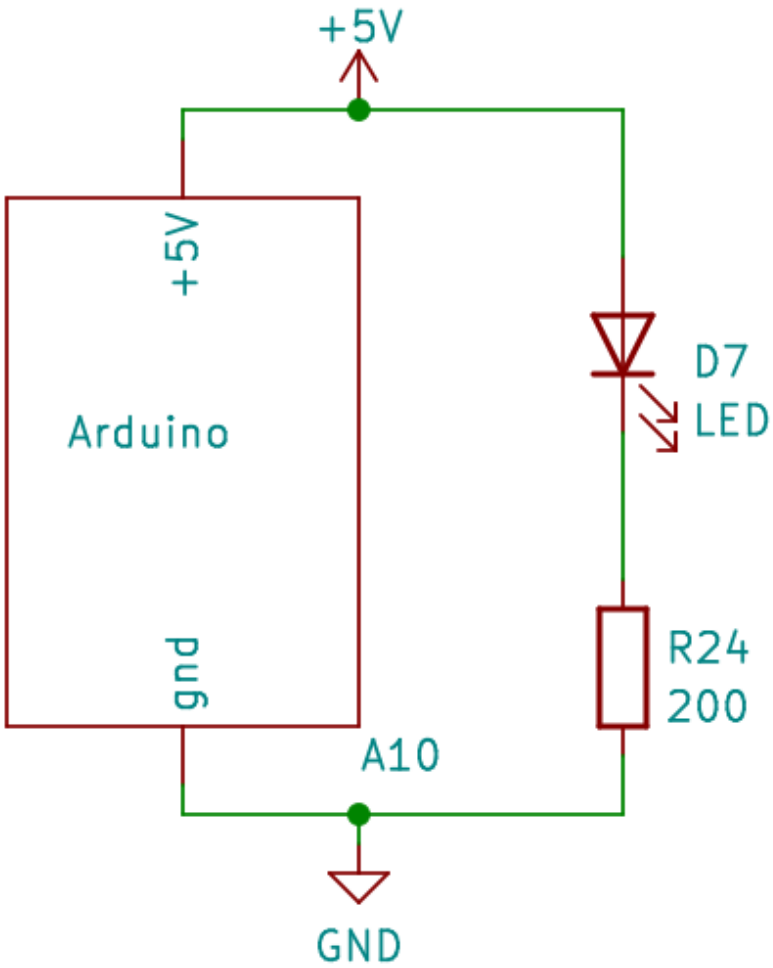
(présent dans la version courte)

Alimenter votre circuit avec Arduino via un bloc d'alimentation secteur



(présent dans la version courte)

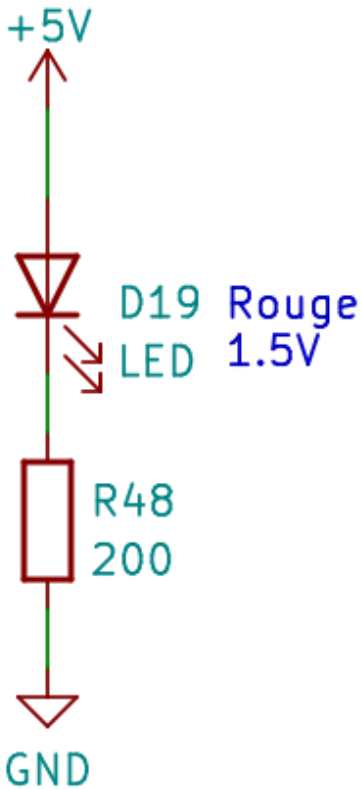
Convention de cours 1



Dans le reste du cours nous ne représenterons pas l'alimentation de l'arduino.

Nous supposons que l'arduino est alimenté par batterie, USB ou bloc d'alimentation secteur.

Convention de cours 2



Quand l'arduino n'est utilisé que pour alimenter le circuit, nous ne représenterons pas l'arduino !

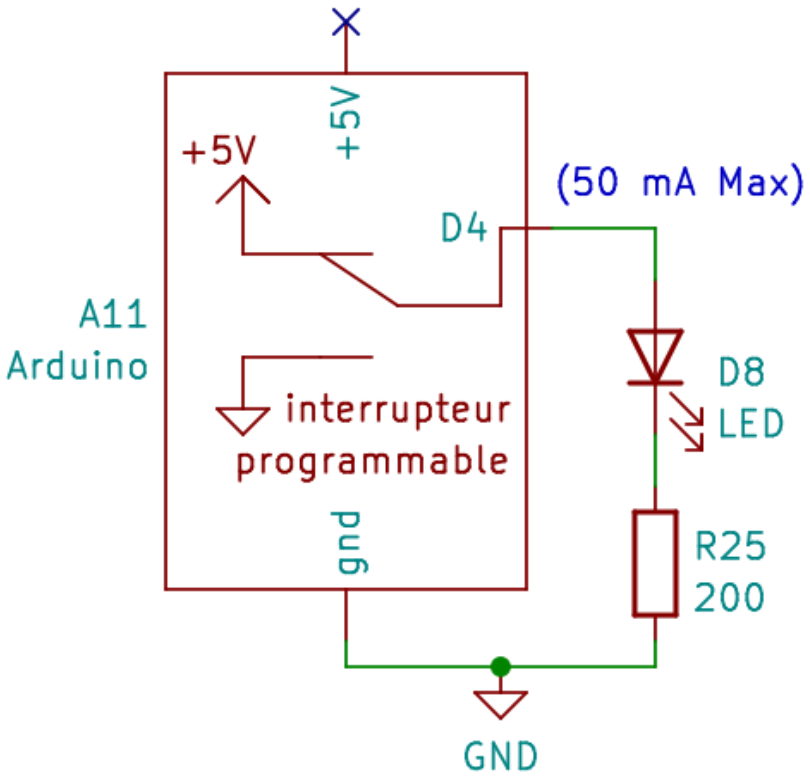
Nous supposons que le circuit est alimenté par pile, arduino, bloc d'alimentation secteur, etc ...

(présent dans la version courte)

Plan

(présent dans la version courte)

TD : Arduino - Commander une LED - Sortie ON/OFF



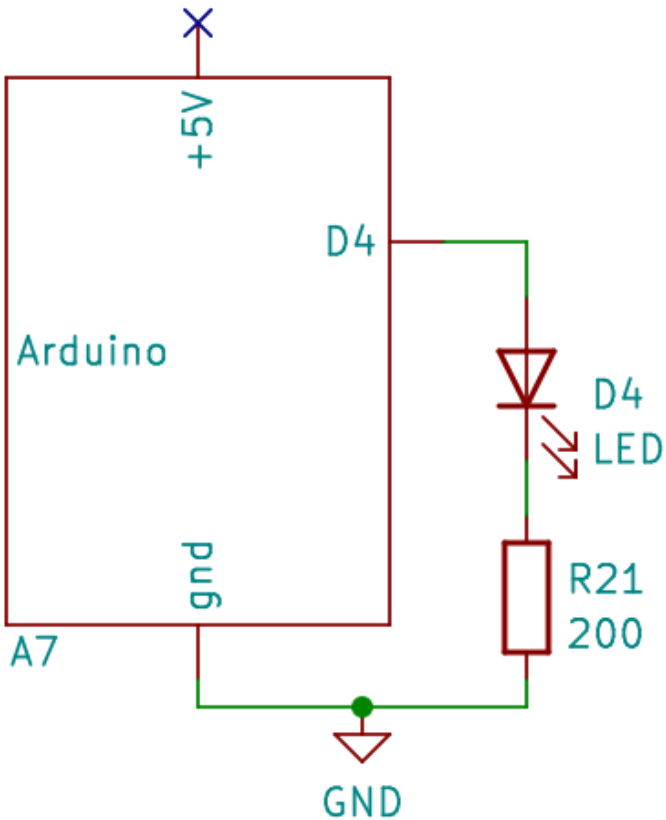
Ce programme fait clignoter une led toutes les secondes.

```
const int LED_PIN = 4;

void setup() {
  // Set led pin to output
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  // turn LED on
  digitalWrite(LED_PIN, HIGH);
  // wait 1 second
  delay(1000);

  // turn LED off
  digitalWrite(LED_PIN, LOW);
  // wait 1 second
  delay(1000);
}
```

code/digital_write.cpp

(présent dans la version courte)

TD : Arduino - Commander une LED - Sortie ON/OFF



Ce programme fait clignoter une led toutes les secondes.

```
const int LED_PIN = 4;

void setup() {
  // Set led pin to output
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  // turn LED on
  digitalWrite(LED_PIN, HIGH);
  // wait 1 second
  delay(1000);

  // turn LED off
  digitalWrite(LED_PIN, LOW);
  // wait 1 second
  delay(1000);
}
```

code/digital_write.cpp

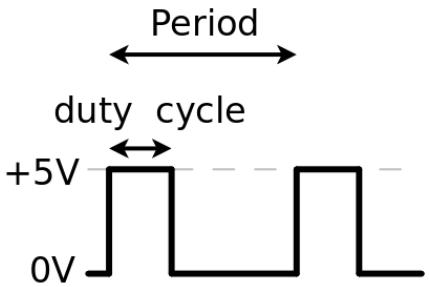
(présent dans la version courte)

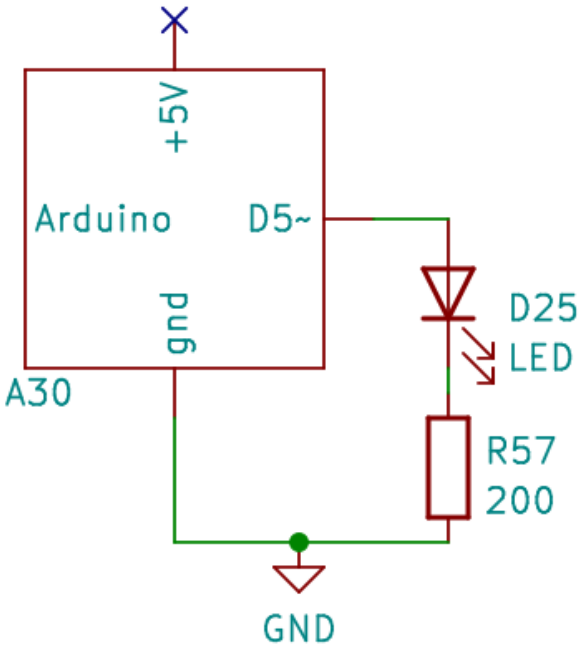
TD : Arduino - Varier la luminosité - Sortie PWM

Les PWMs sont des signaux carrés qui simulent un signal analogique : un signal de 3V est équivalant à une PWM à 5V de rapport cyclique 3/5 (cf. page 430) .

Les broches permettant de faire des PWMs sont symbolisées par ~

Par défaut, les pwms de l'arduino ont une fréquence de 500 Hz ou 1 KHz.






```
const int pwm_pin = 5;
int duty_cycle = 0;

void setup() { }

void loop() {
  for(int i=0; i<=100; i++){
    duty_cycle = map(
      i, 0, 100, 0, 255
    ); // i % of duty cycle
    analogWrite(
      pwm_pin, duty_cycle
    );
    delay(100);
  }
}
```

code/pwm.cpp

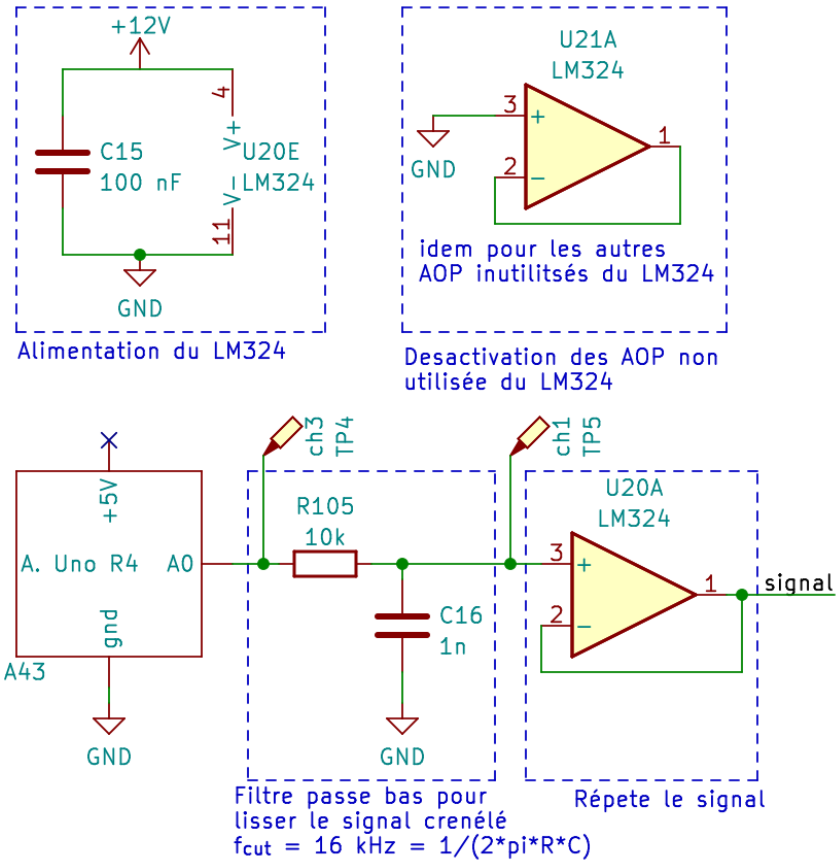
En boucle, la lumière s'éteint puis s'allume graduellement sur 10 s.

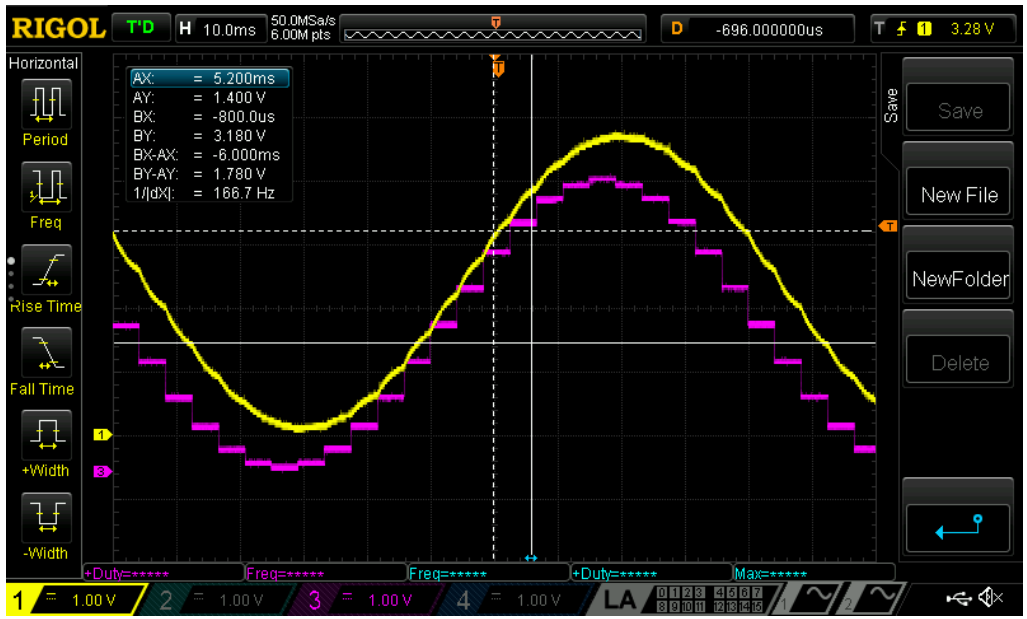
(présent dans la version courte)

Plan

(version longue)

Pour Ard. UNO R4 - créer une signal analogique - DAC





La résistance R105 et le condensateur C16 doivent être choisis en fonction du signal à restituer, de la période d'échantillonnage choisi et de la distorsion que l'on souhaite obtenir vis à vis de la forme du signal choisi.

(version longue)

Programmer un signal analogique - DAC

```
#include <analogWave.h>

analogWave wave(DAC); // Broche DAC = broche A0
int frequence = 10; // en hertz

void setup(){
  wave.sine(frequence);
  // wave.saw(frequence);
  // wave.square(frequence);
}

void loop() {
  delay(1000);
  frequence = 440;
  wave.freq(frequence);

  delay(1000);
  frequence = 1000;
  wave.freq(frequence);
}
```

code/dac.cpp

(version longue)

Plan

(présent dans la version courte)

Comunication de l'Arduino vers l'ordinateur

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

void loop() {
  Serial.print("Bonjour !");
  delay(1000);
}
```

code/serial_com_from_arduino.cpp

Le programme suivant permet à la carte d'intialiser une comunication série via l'USB et d'envoyer "Bonjour !" toute les secondes à l'ordinateur.

Pour lire les messages envoyés, vous devez ouvrir le "moniteur série" dans l'onglet "tool" d'arduino IDE. Vous devez aussi choisir la bonne vitesse de comunication qui a été configuré par la ligne Serial.begin(...). Dans notre cas, il s'agit de 9600 bits par seconde.

Les choix de vitesses de transferts possibles sont : 9600, 19200, 115200.

(présent dans la version courte)

Dialogue entre Arduino et ordinateur

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

void loop(){
  while( Serial.available() ){
    char incoming_Byte = Serial.read();
    Serial.print(incoming_Byte);
  }
}
```

code/serial_com_arduino_copy.cpp

Danc ce programme, l'arduino renvoie à l'ordinateur tous les octets que'il a reçu.

Dans le moniteur série de l'IDE Arduino, vous pouvez écrire un message, qui sera reçu et renvoyé par l'arduino.

(présent dans la version courte)

Dialogue entre Arduino et ordinateur - lecture d'une ligne

Dans ce programme, l'arduino renvoie la ligne que l'ordinateur a envoyé et détermine s'il s'agit de la commande "set MODE VELOCITY" ou MODE est un entier et VELOCITY un réel.

```
const int MAX_LENGTH = 20; char incoming_Byte = '\r'; String text("");

bool is_end_of_line(char c){ return c == '\r' or c == '\n'; }
bool read_line(){
  if(is_end_of_line(incoming_Byte) or text.length() == MAX_LENGTH){
    text = ""; incoming_Byte = 'a';
  }
  while(Serial.available() and text.length() < MAX_LENGTH){
    incoming_Byte = Serial.read();
    if( is_end_of_line(incoming_Byte) ){ break; }
    text += incoming_Byte;
  }
  return is_end_of_line(incoming_Byte) or text.length() == MAX_LENGTH;
}
void execute_commands(){
  while( read_line() ){
    Serial.print( "Received line : "); Serial.println(text);
    if(text.startsWith("velocity ")){
      text.remove(0, text.indexOf(' ')+1);
      int mode = text.toInt();
      text.remove(0, text.indexOf(' ')+1);
      float velocity = text.toFloat();
      Serial.print("Setting mode to "); Serial.print(mode);
      Serial.print(" and velocity to "); Serial.println(velocity);
    }
    text.remove(0);
  }
}

void setup() { Serial.begin(9600); }
void loop(){ execute_commands(); }
```

code/serial_com_arduino_read_a_line.cpp

(présent dans la version courte)

Se connecter sur un port série en ligne de commande

Pour vous connecter au port série de votre carte, vous pouvez utiliser l'outil cu de Linux:

```
cu -s 9600 /dev/ttyUSB0
```

L'option -s 9600 est la vitesse de comunication (baudrate) définie dans le code source (rappel : Serial.begin(9600)).

Si vous tapez du texte, il est envoyé à l'arduino via le port série.

Pour quitter le programme cu, il faut taper le caractére tilde : ~ suivi du caractère . ce qui done : "~.".

Vous pouvez aussi utiliser l'outil screen ;

```
screen /dev/ttyUSB0 9600
```

Si vous tapez du texte, il est envoyé à l'arduino via le port série.

Pour quitter screen, vous devez taper : Ctrl+a et ensuite k.

(version longue)

Se connecter sur un port série avec un programme Python

Programmez le microcontroleur avec le programme présenté à la page 111.
Ce programme envoie "Hello" et attend la réponse du microcontrôlleur.

code/connect_to_serial_port.py

```
import serial
import time

baudrate = 9600
port = "/dev/ttyUSB0"
ser = serial.Serial(port, baudrate)

data_to_send = "Hello\n"
ser.write(data_to_send.encode("ascii"))
print("Sent data :")
print(data_to_send)

# Be carfull : This code works only if arduino send the end of line
# character, by using for example Serial.println("") or Serial.print("\n")
received_data = ser.readline().decode("ascii")
print(f"Microcontoler answer :")
print(received_data)
```

La bibliothèque envoie les données dans le port série quand elle possède suffisament de caractères ou quand "\n" est reçu.

Utilisez ser.flush() après ser.write(...) pour forcer l'envoie quand vous envoyez des données en flux tendu sans utiliser le caractère de fin de ligne.

(version longue)

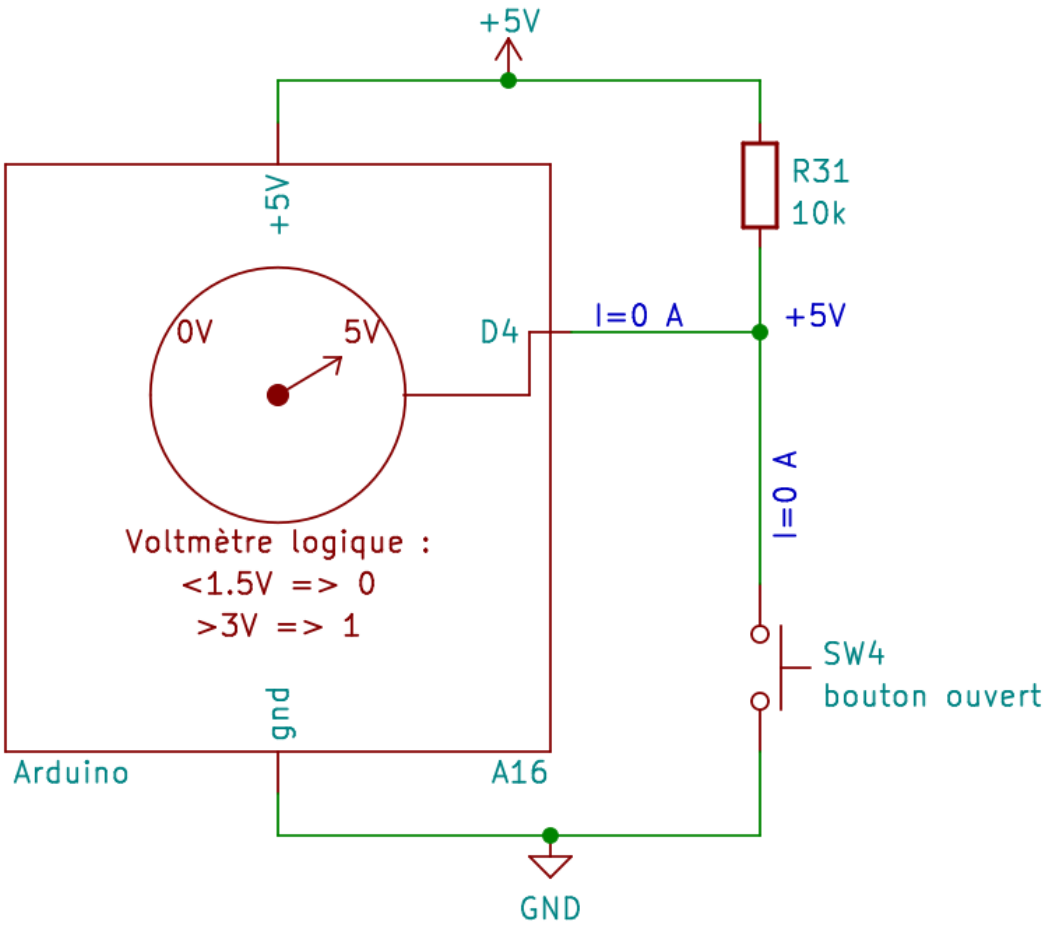
Plan

(présent dans la version courte)

Les entrées logiques - Le montage avec une résistance de pull-up

Bouton fermé = 0 logique

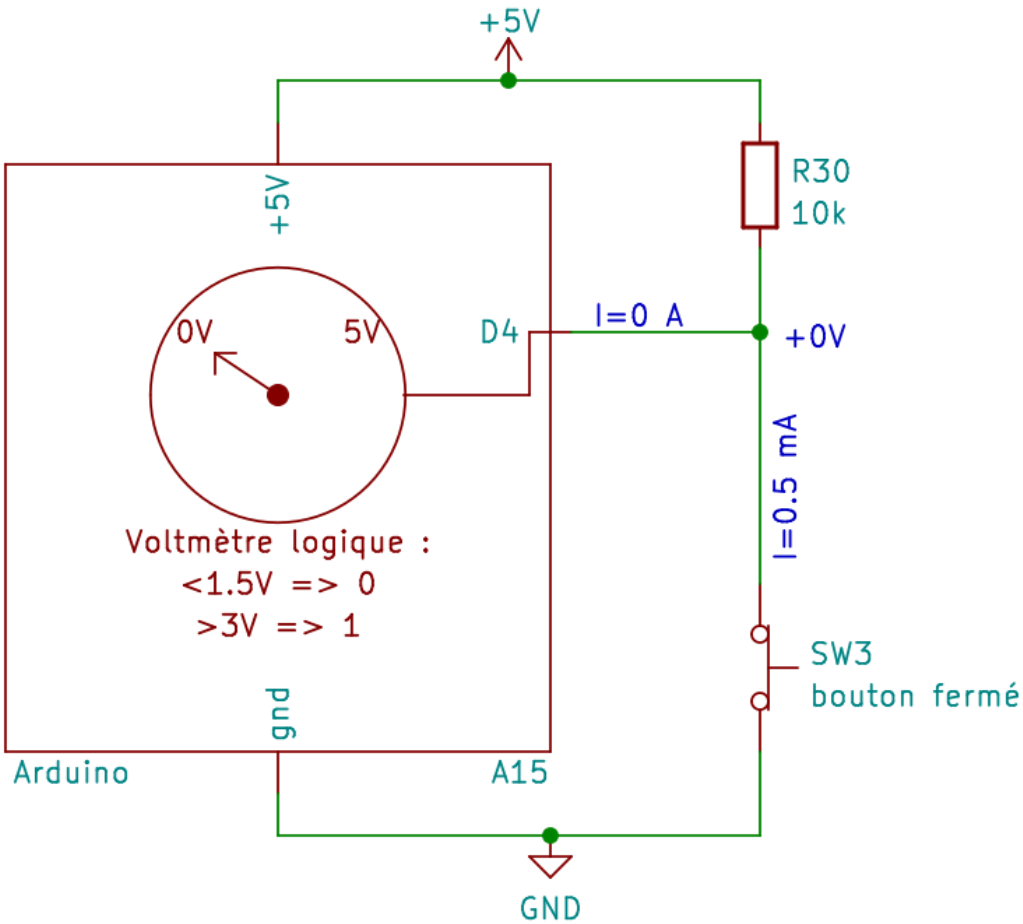
Bouton ouvert = 1 logique



Les entrées logiques - Le montage avec une résistance de pull-up

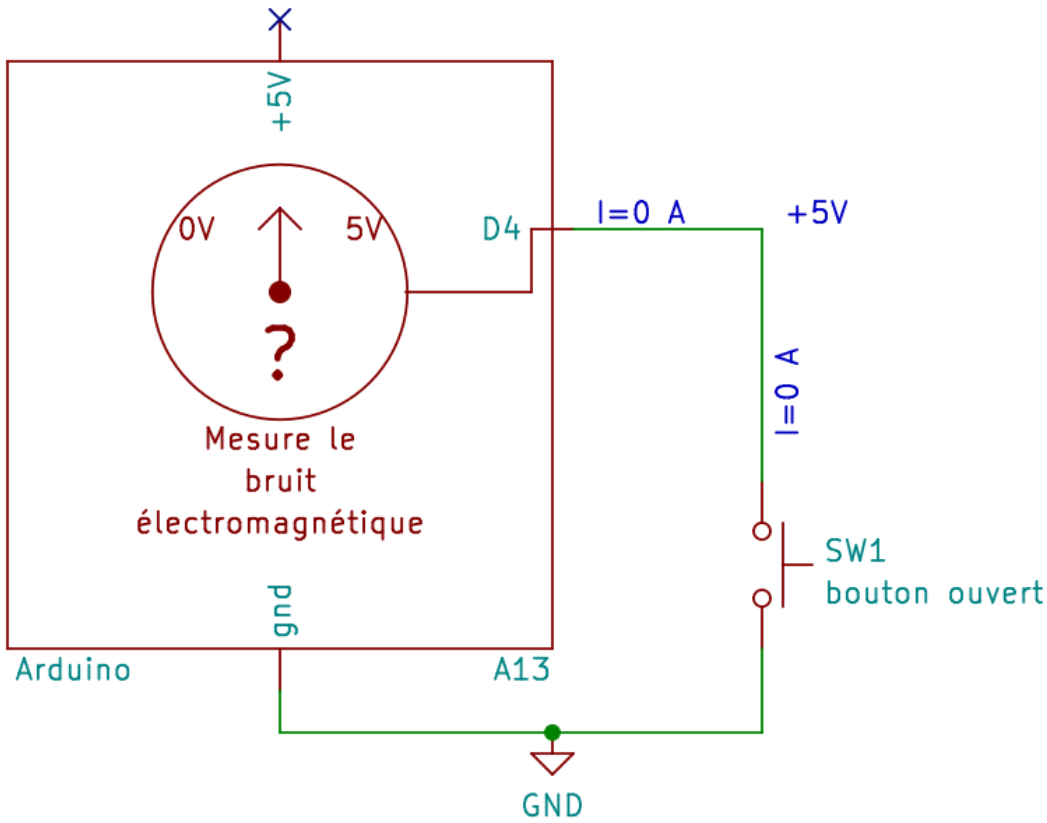
Bouton fermé = 0 logique

Bouton ouvert = 1 logique



(présent dans la version courte)

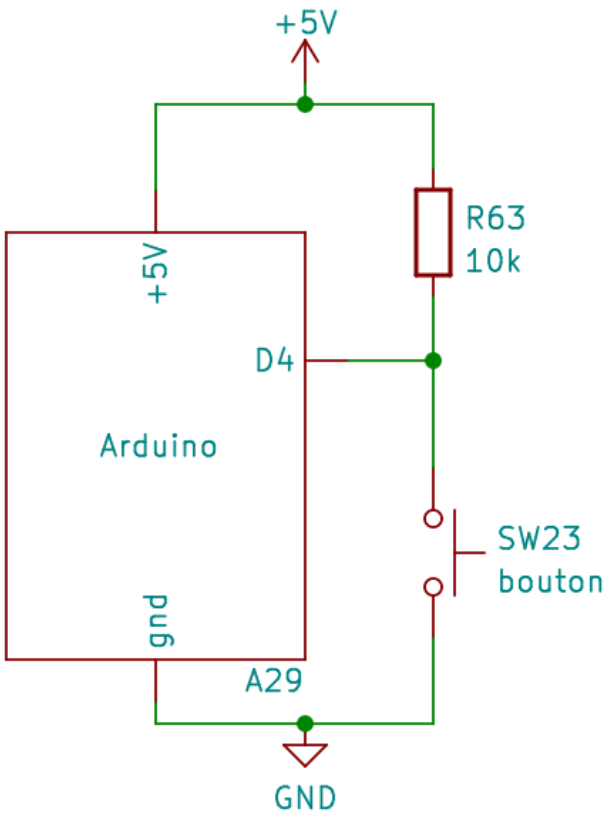
Les entrées logiques - Oublier la résistance de pull-up



(présent dans la version courte)

TD : Les entrées logiques - Programmer avec l'arduino

Toute les secondes on affiche l'état du bouton via le port série.



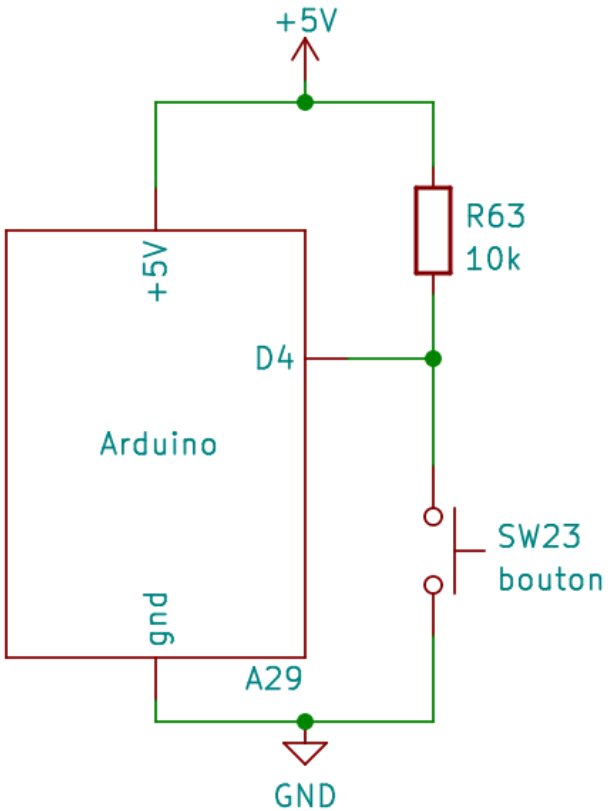

```cpp
const int button_pin = 4;
int button_state;

void setup() {
  Serial.begin(9600);
  pinMode(button_pin, INPUT);
}

void loop() {
  button_state = digitalRead(button_pin);

  if (button_state == HIGH) {
    Serial.println("OPEN");
  } else {
    Serial.println("CLOSED");
  }
  delay(1000);
}
```

code/digital_read.cpp

(présent dans la version courte)

TD : Les fronts montants - Programmer avec l'arduino

Sur un front montant/descendant on affiche "bouton relâche/appuyé".

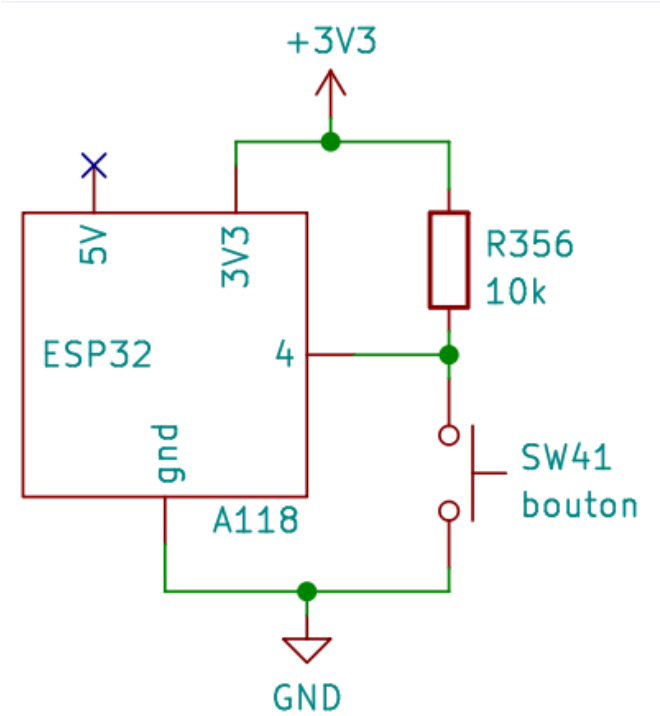


code/digital_read_front_montant.cpp

```
const int button_pin = 4;
int button_state;
int old_state;

void setup() { Serial.begin(9600);
  pinMode(button_pin, INPUT); }

void loop() {
  old_state = button_state;
  button_state = digitalRead(button_pin);

  // Front montant
  if (old_state < button_state){
    Serial.println("BOUTON RELACHE");
  }

  // Front descendant
  if(old_state > button_state){
    Serial.println("BOUTON APPUYE");
  }
}
```

(présent dans la version courte)

Mettre une résitance de pull-up avec les ESP32



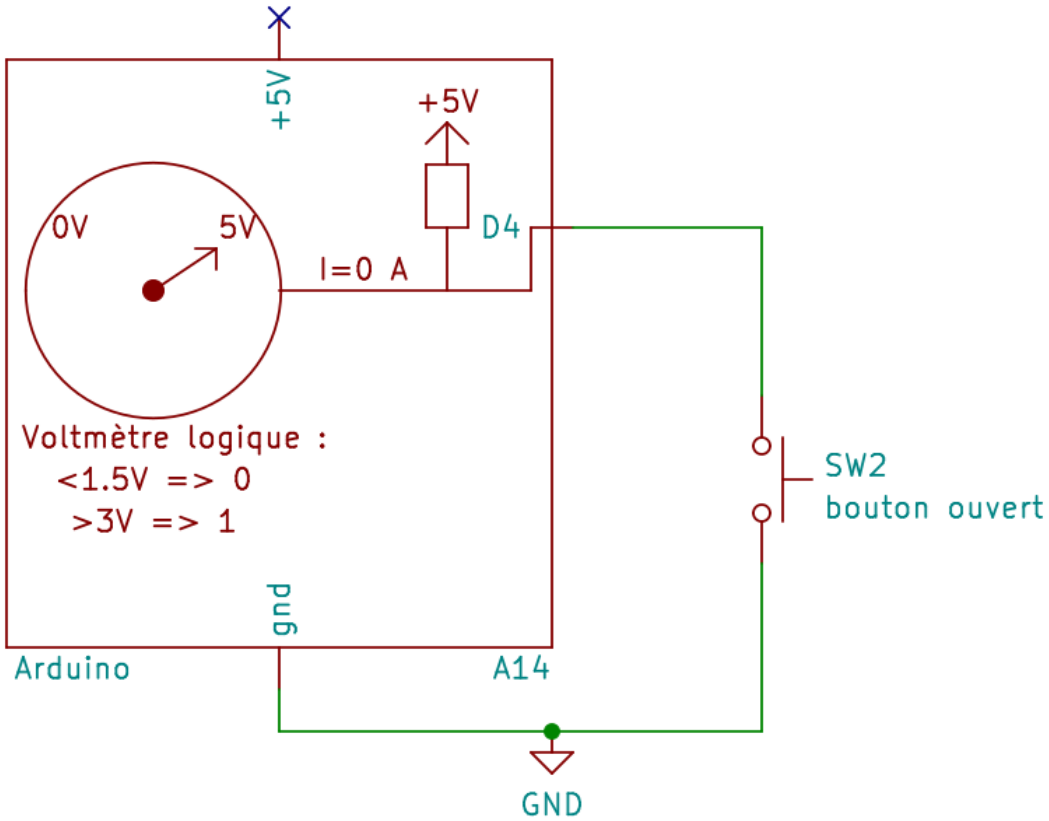
Attention !

Pour les esp32, il faut connecter la résistance de pull-up au 3V3.

Appliquer une tension > 3.3V aux broches de l'ESP32 peut détruire la carte !

Comme vous allez le voir dans les slides suivantes, le plus simple est de ne pas mettre de résistance et d'utiliser à la place résistance de pull-up interne de l'ESP32.

Les entrées logiques - utiliser une résistance pull-up interne



On peut demander à l'arduino d'ajouter une résistance de pull up interne.

La fonction de setup devient :

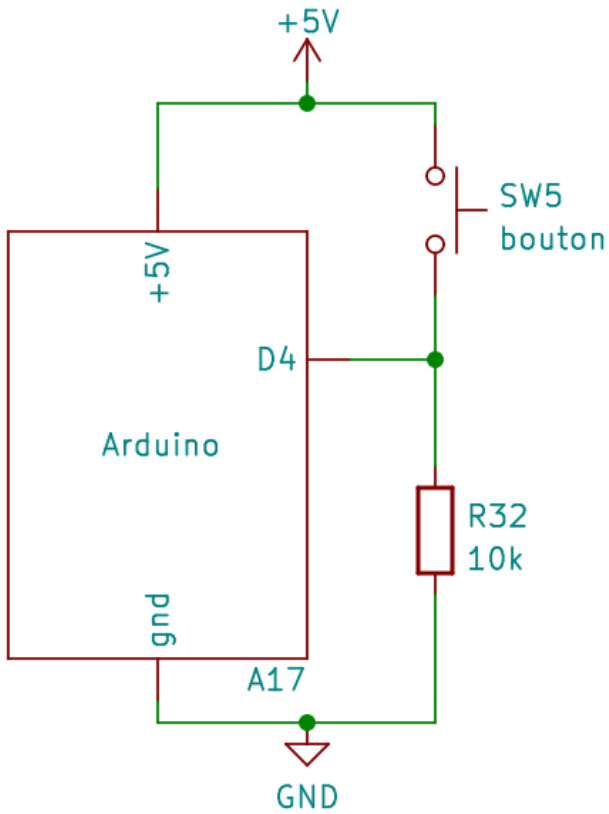
```
void setup() {
  Serial.begin(9600);
  pinMode(button_pin, INPUT_PULLUP);
}
```

(présent dans la version courte)

Les entrées logiques - Utiliser une résistance pull-down

Les signaux sont inversés par rapport au montage avec résistance pull-up.

Bouton fermé = 1 logique Bouton ouvert = 0 logique



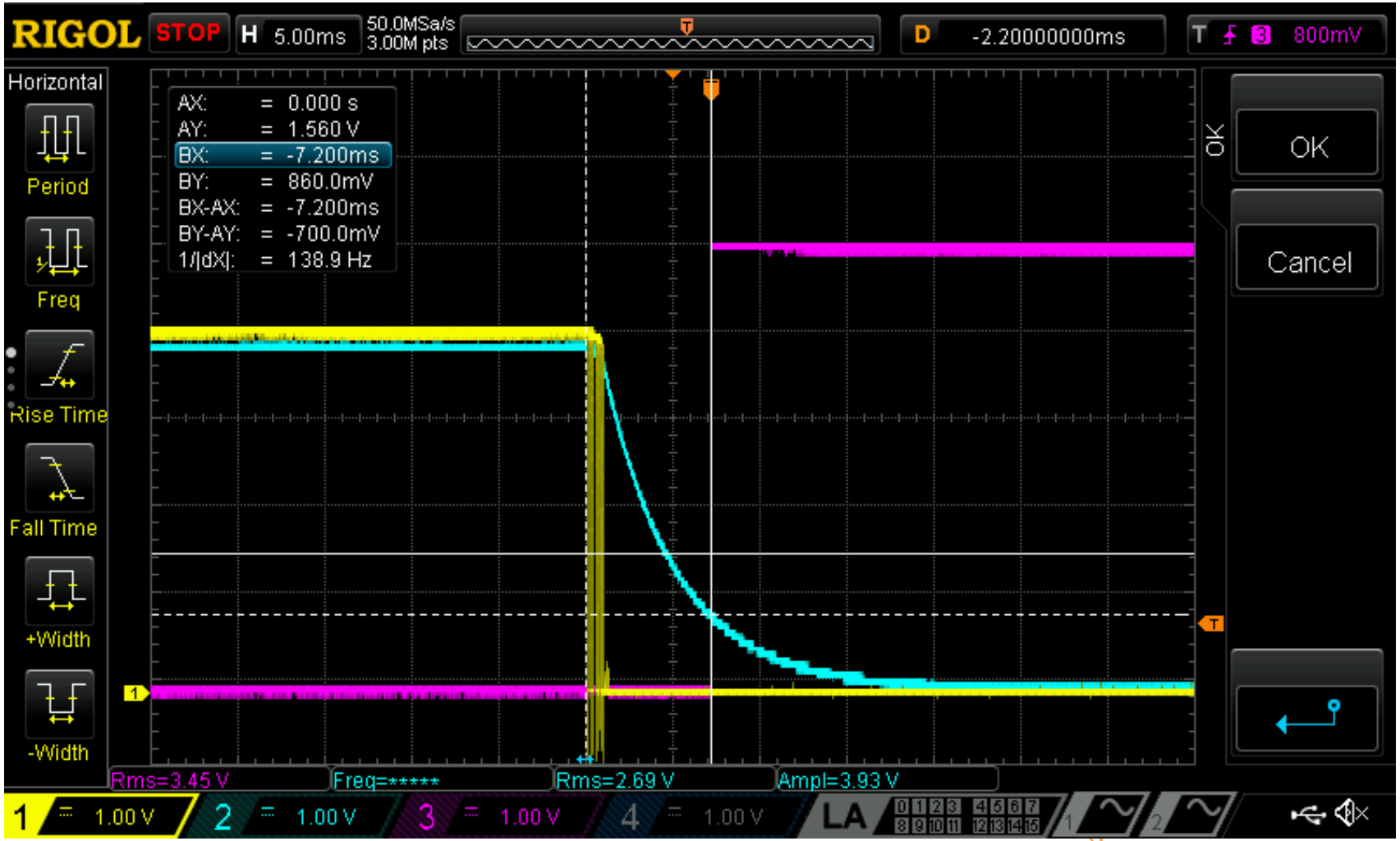
Des interrupteurs et des rebonds

Canal 1 (Jaune) : tension au borne du bouton

Canal 2 (Bleu) : tension filtrée électroniquement

Canal 3 (Violet): état déduit du bouton

Durée totale de tous les rebonds : 1 ms.



courte)

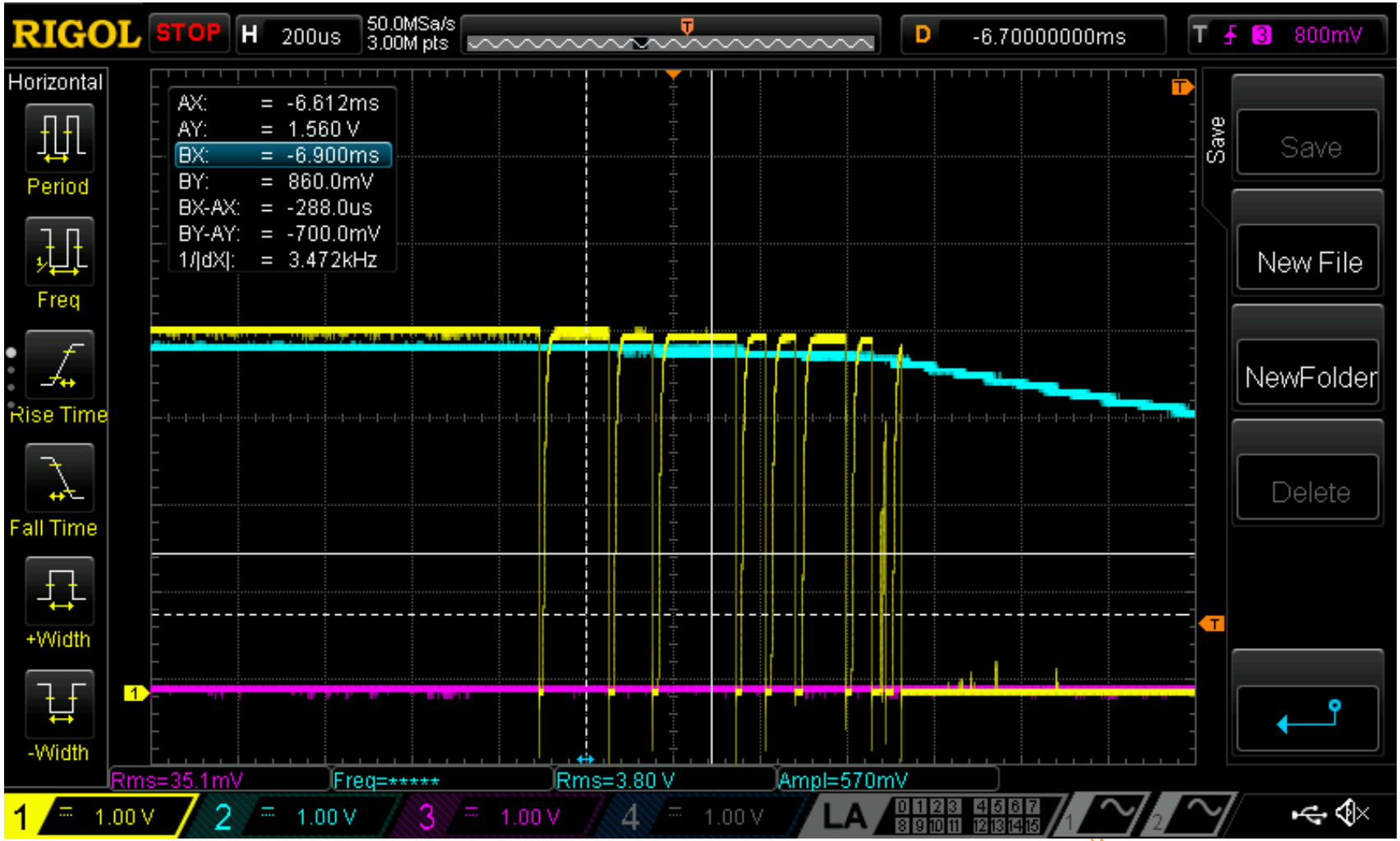
Des interrupteurs et des rebonds

Canal 1 (Jaune) : tension au borne du bouton

Canal 2 (Bleu) : tension filtrée électroniquement

Canal 3 (Violet): état déduit du bouton

Durée totale de tous les rebonds : 1 ms.

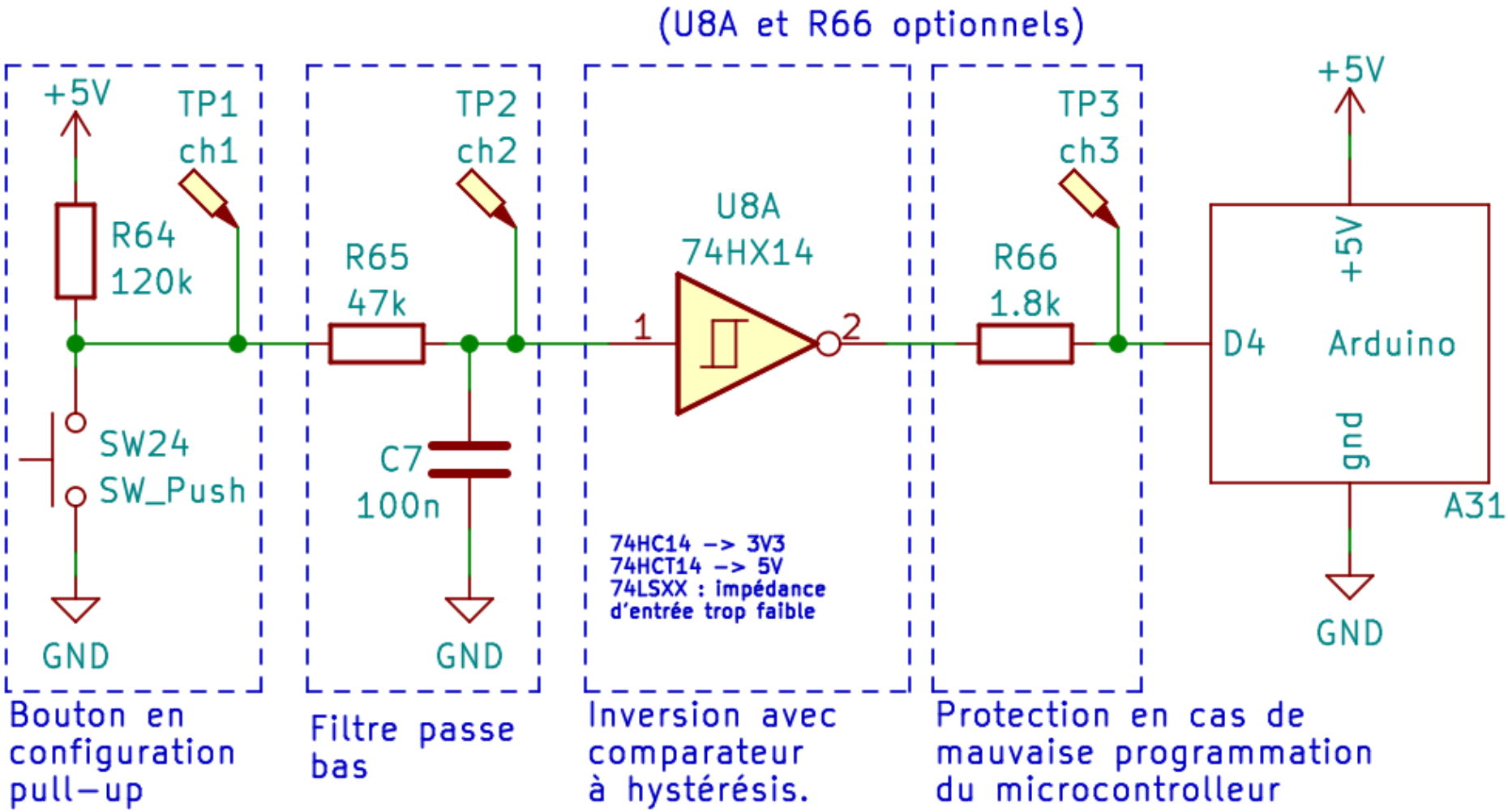


courte)

Gérer les rebonds - en électronique

Les canaux 1, 2 et 3 de l'oscilloscope des affichages précédent :

ch1 : jaune ch2 : bleu ch3 : violet



(version longue)

Gérer les rebonds - en programmation

Il faut confirmer une valeur lue avant de l'utiliser:

- on échantillonne toutes les 5 ms ($>2\times$ la période totale des rebonds)
- Sot U_k la tension lue, on calcule l'état du bouton ainsi :

$$V_k = \text{état bouton} = \begin{cases} U_k & \text{si } U_k = U_{k-1}; \\ U_{k-2} & \text{sinon.} \end{cases}$$

On calcule ensuite, les fronts montants et descendants:

$$\underset{\text{montant}}{\text{front}} = \begin{cases} 1 & \text{si } V_{k-1} < V_k; \\ 0 & \text{sinon.} \end{cases} \qquad \underset{\text{descendant}}{\text{front}} = \begin{cases} 1 & \text{si } V_{k-1} > V_k; \\ 0 & \text{sinon.} \end{cases}$$

C'est fort pénible, il vaut mieux ...

(présent dans la version courte)

Gérer les rebonds - Utiliser une bibliothèque

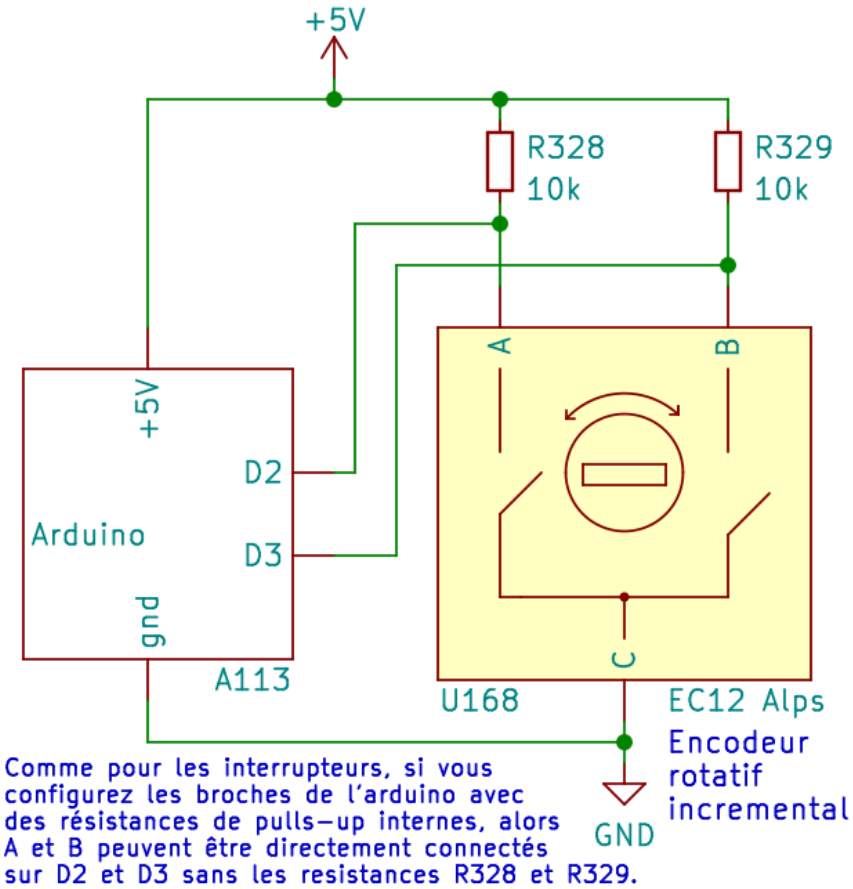
Utiliser la bibliothèque : debounced_button.hpp,

```
#include "debounced_button.hpp"

const int DEBOUNCE_DELAY = 10; // In milli-seconds
const int BUTTON_PIN = 4;         // ADAPTER A VOTRE CIRCUIT !
const bool INTERNAL_PULLUP = false; // ADAPTER A VOTRE CIRCUIT !
const bool INVERSE_STATE = true; // ADAPTER A VOTRE CIRCUIT !

static Debounced_button button(
  BUTTON_PIN, INTERNAL_PULLUP, INVERSE_STATE, DEBOUNCE_DELAY);

void setup() {
  button.setup();
}

void loop() {  // NE PLUS UTILISER DE DELAIS (la fonction delay()) !
  button.update();

  if(button.rising_edge){ /*Mettre ici votre code*/ }
  if(button.falling_edge){ /*Mettre ici votre code*/ }

  if(button.state){ /*Mettre ici votre code*/ }else{ /*Mettre ici votre code*/ }
}
```

code/digital_read_rebond.cpp

(présent dans la version courte)

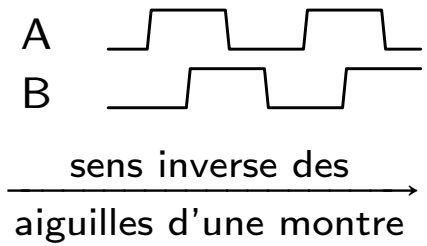
Les encodeurs rotatifs incrémentaux



Un encodeur rotatif incrémental est constitué de deux interrupteurs mécaniques qui s'actionnent avec un déphasage de 45 degré.

Ils s'utilisent donc en montage pull-up comme le montre le schéma de gauche (le montage pull-down est aussi possible).

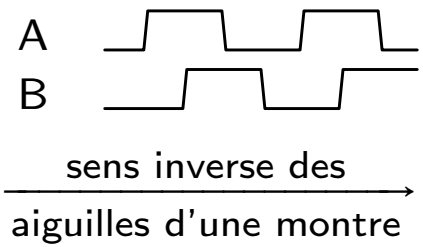
Lorsque l'on fait tourner l'encodeur du montage dans le sens inverse des aiguilles d'une montre, on obtient le chronogramme suivant pour les broches A et B de l'encodeur:



Le code formé par les broches A et B est un code de gray sur 2 bits: les broches A et B ne changent jamais d'état en même temps et peuvent prendre toutes les combinaisons de valeurs possibles (00, 01, 10 et 11).

(version longue)

Programmer pour lire un encodeur rotatif incrémental



Pour déterminer la rotation de l'encodeur, on utilise la broche A comme horologe et la broche B comme donnée ;

1. on fait une lecture de l'état B uniquement sur les fronts montants et descendants de A.
2. Si $A \neq B$ alors l'encodeur s'est déplacé d'un cran dans le sens direct (sens inverse des aiguilles d'une montre) sinon il s'est déplacé dans le sens indirect.

Enfin, on utilise le code du fichier debounced_button.hpp pour calculer les fronts montants et descendants et retirer les rebonds. Le document de spécification de l'encodeur préconise de mettre un valeur de DEBOUNCE_DELAY de 2 ms, mais de ne pas depasser 3 ms.

code/encodeur_rotatif_incremental_EC12.cpp

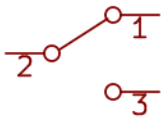
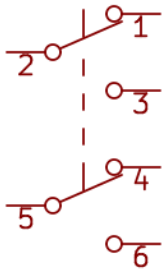
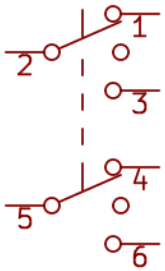
```
#include "debounced_button.hpp"

const int DEBOUNCE_DELAY = 2; // In milli-seconds
const int A_PIN = 2, B_PIN = 3;
const bool INTERNAL_PULLUP = false, INVERSE_STATE = false;

static Debounced_button encoder_clock(
  A_PIN, INTERNAL_PULLUP, INVERSE_STATE, DEBOUNCE_DELAY);
static Debounced_button encoder_data(
  B_PIN, INTERNAL_PULLUP, INVERSE_STATE, DEBOUNCE_DELAY);

int encoder_position = 0;
void setup(){
  Serial.begin(9600);
  encoder_clock.setup();
  encoder_data.setup();          // On rapelle qu'il ne faut
}                                // pas utiliser la fonction
                                 // delay() pour des delais
void loop(){                     // >= DEBOUNCE_DELAY/2
  encoder_clock.update();        // avec debounced_button.hpp
  encoder_data.update();
  if(
    encoder_clock.rising_edge or encoder_clock.falling_edge
  ){
    if(encoder_clock.state != encoder_data.state){
      encoder_position += 1; // Sens direct
    }else{
      encoder_position -= 1; // Sens indirect
    }
    Serial.println(encoder_position);
  }
}
```

(version longue)

Plan

(présent dans la version courte)

Différents types d'interrupteur - 1/2

Bouton poussoir		Commutateurs à bascule ou à levier			
interrupteur monostable SPST OFF-MOM SPST OFF-(ON)	interupteur bistable SPST ON-OFF	SPDT ON-ON	SPDT ON-OFF-ON	permutateur DPDT ON-ON	DPDT ON-OFF-ON
					
	 				

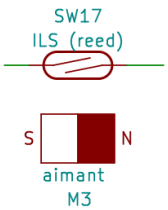
S = Single P = Pole D =Dual/Double T = Throw

ON = connecté à une broche OFF = non connecté MOM (ON) = Momentané

(présent dans la version courte)

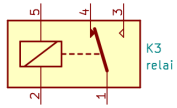
Différents types d'interrupteur - 2/2

Interrupteur magnétique : l'interrupteur se ferme prêt d'une source de champs magnétique.



Interrupteur à lame souple

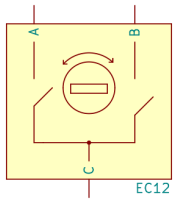
Interrupteur électrique : l'interrupteur se ferme quand un courant passe dans une bobine.



Relai

Consulter les pages 280, 281 et 282 pour plus de détail.

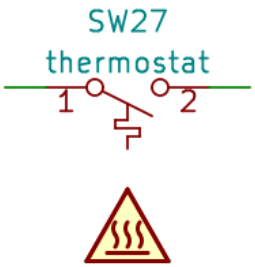
Encodeurs rotatifs : capteur constitué de deux interrupteurs permettant de mesurer la rotation d'un bouton.



Enocodeur EC12

Consulter la page 128 pour plus de détail.

Interrupteur thermique : l'interrupteur s'ouvre prêt d'une source de température.



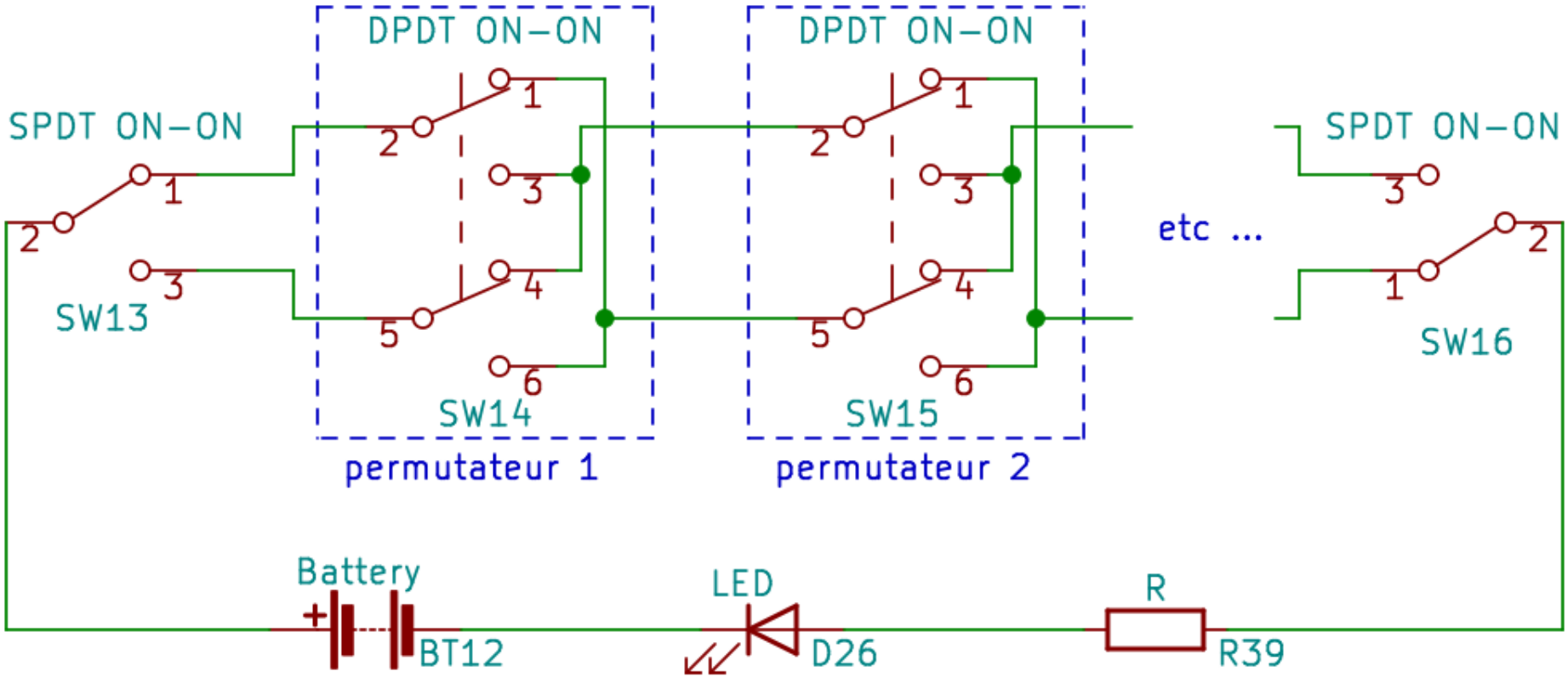
Fusible thermique

Thermostat bilames

Thermostat capillaire à bulbe

(présent dans la version courte)

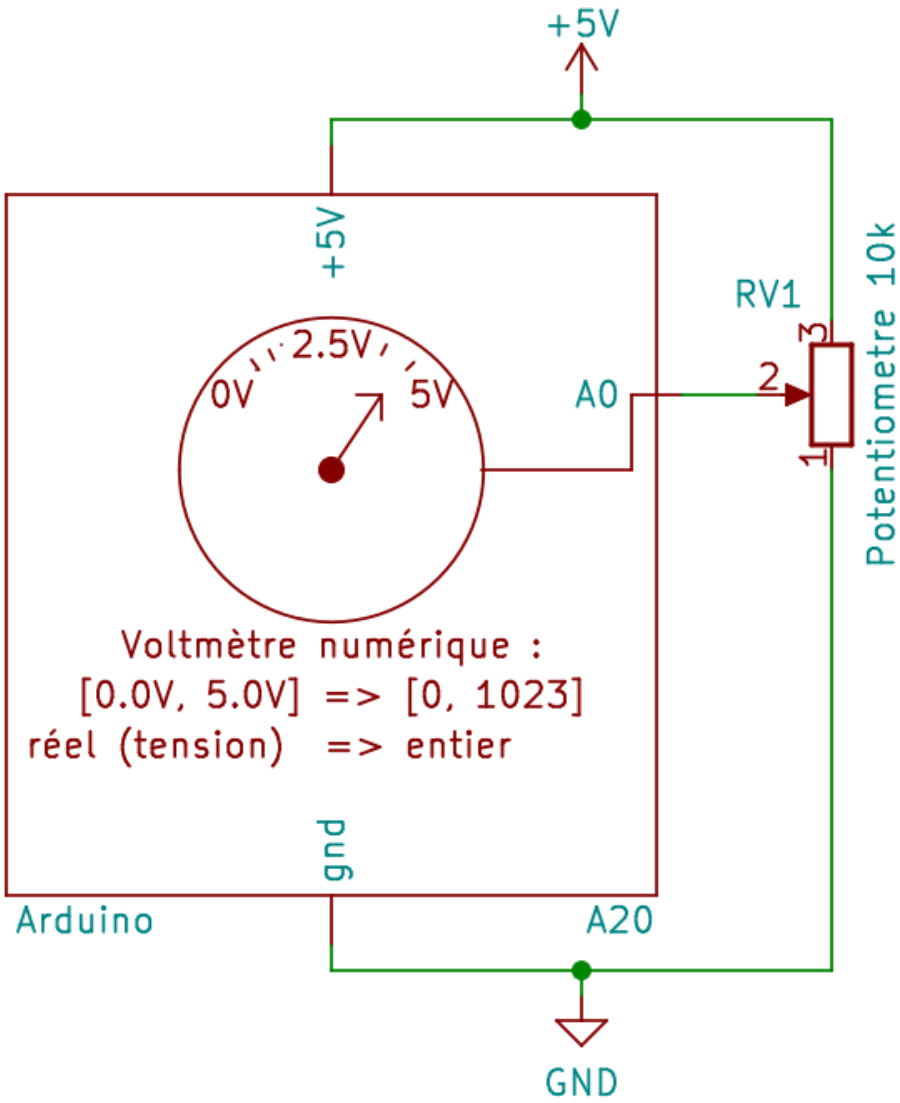
Le va et vient où comment allumer et éteindre la lumière avec plusieurs interrupteurs (permutateurs)



Plan

(présent dans la version courte)

TD - Arduino - Les entrées analogique - ADC



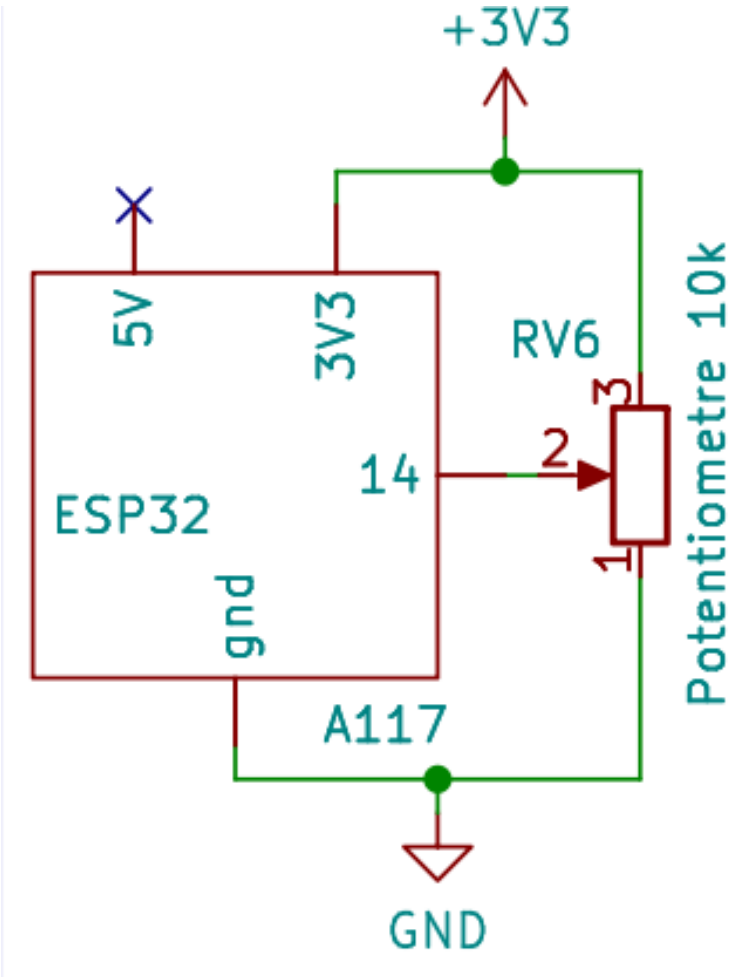

```ino
const int sensor_pin = A0;
int value;
float voltage;

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

void loop() {
  value = analogRead(sensor_pin);
  voltage = (5.0*value)/1023;

  Serial.print("voltage : ");
  Serial.println(voltage);

  delay(1000);
}
```

code/analog_read.cpp

L'ADC est optimisé pour des impédances de sortie de 10 $k\Omega$ ou moins.

(présent dans la version courte)

TD - ESP32 - Les entrées analogique - ADC




```cpp
const int sensor_pin = 14;
int value;
float voltage;

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

void loop() {
  value = analogRead(sensor_pin);
  voltage = (3.3*value)/4095;

  Serial.print("voltage : ");
  Serial.println(voltage);

  delay(1000);
}
```

code/analog_read_esp32.cpp

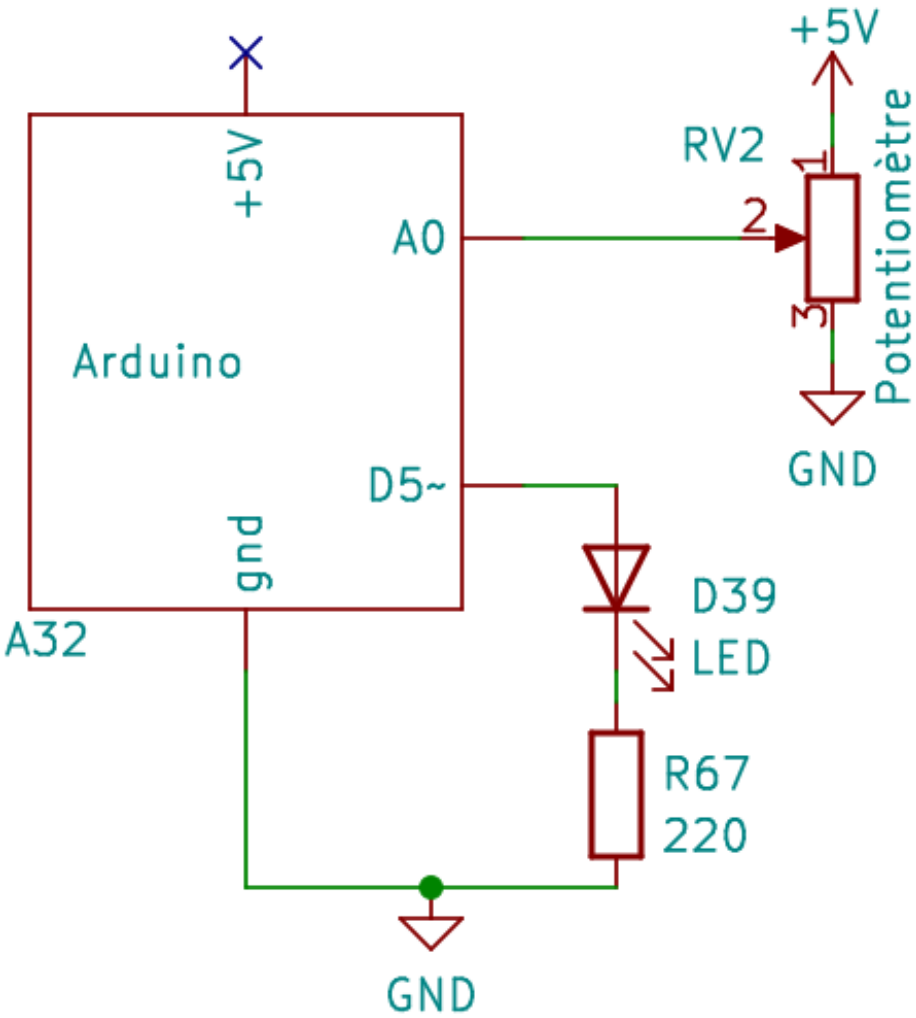
Attention, les broches des ESP32 ne peuvent pas recevoir plus de 3V3 !

La résolution des ADC est 12 bit. Les valeurs sont comprises ente 0 et 4095 inclus.

L'ADC est optimisé pour des impédances de sortie de 10 $k\Omega$ ou moins.

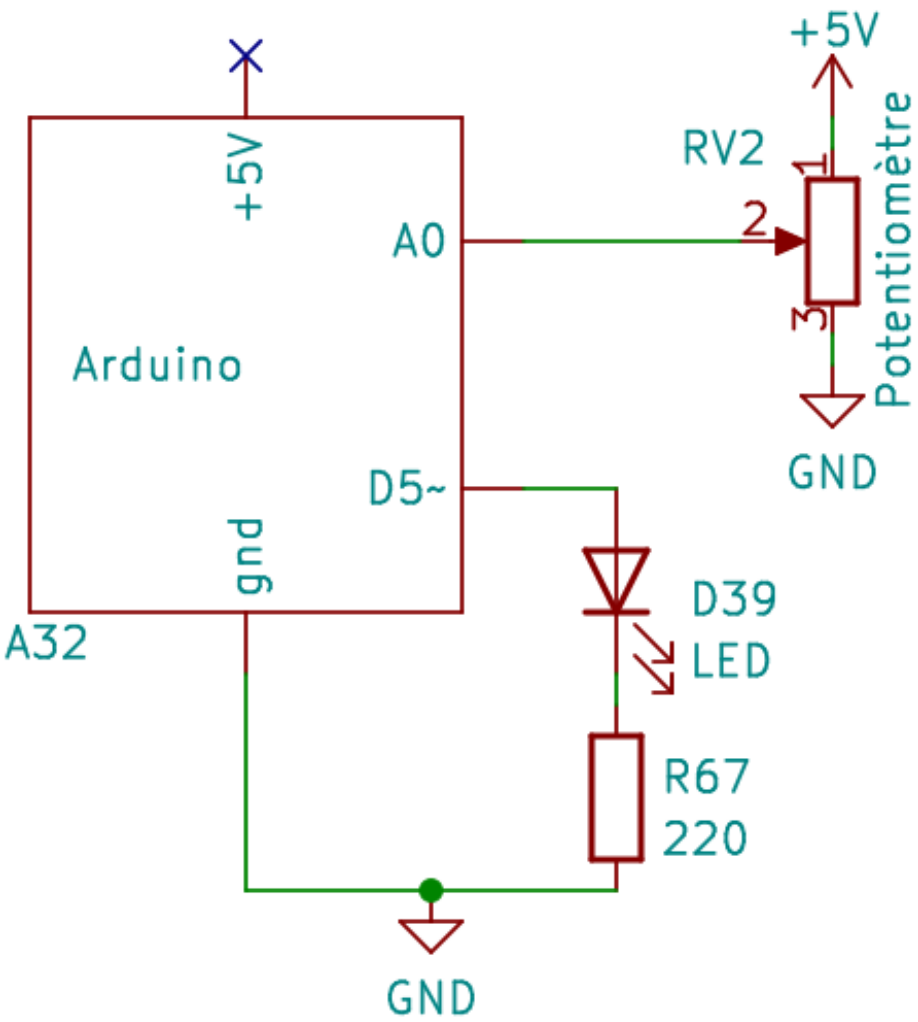
(version longue)

TD pratique !



(présent dans la version courte)

TD pratique !

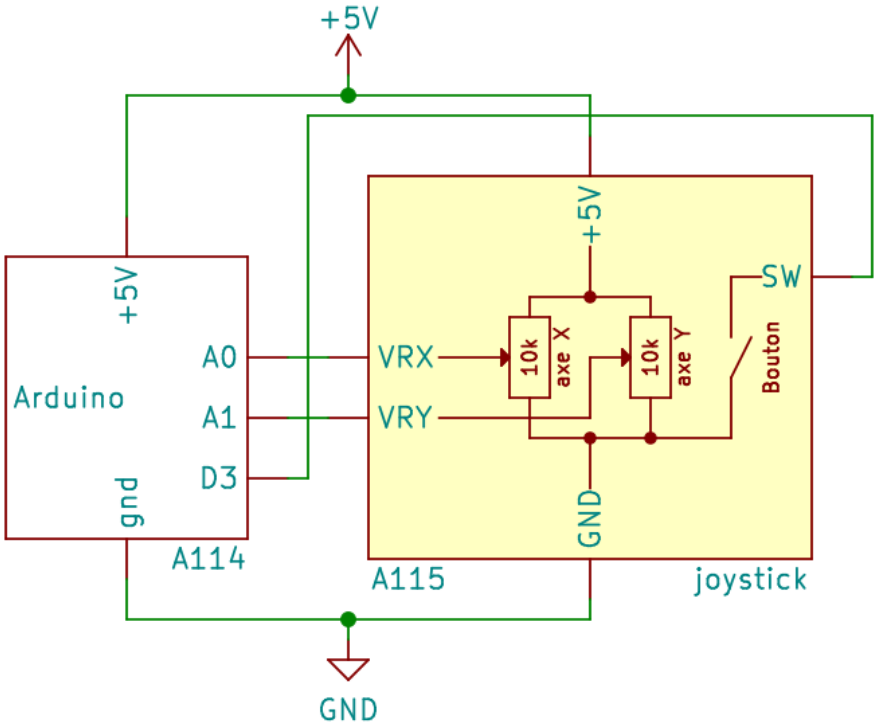



```
const int sensor_pin = A0;
int value;

const int led_pin = 5;
int led_pwm;

unsigned int time, last_time;

void setup() {
  last_time = millis();
}

void loop() {
  time = millis();
  if( time - last_time > 30 ){
    value = analogRead(sensor_pin);
    led_pwm = map(
      value, 0, 1023, 0, 255
    );

    analogWrite(led_pin, led_pwm);
    last_time = time;
  }
}
```

code/fade_in_led.cpp

(présent dans la version courte)

Utiliser un jostick du module arduino



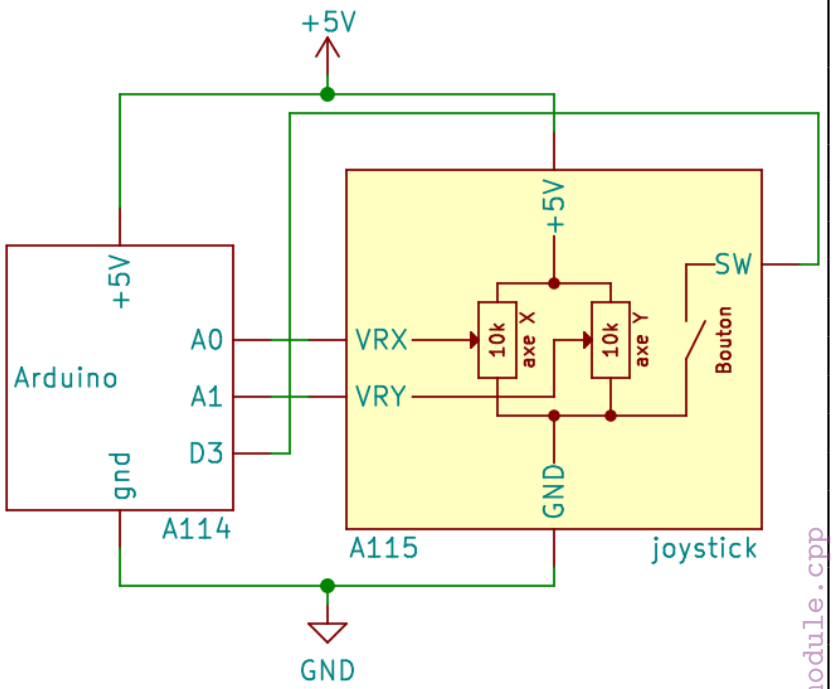
Le module joystick du kit arduino contient 2 potentiomètres de 10k et 1 bouton.

Quand on deplace le manche du joystick sur l'axe X (resp. Y), on change la valeur du potentimètre relié à VRX (resp. VRY). Quand on clique sur le manche du joystick on appuye sur le bouton qui est reliré à SW.

On connecte donc VRX et VRY sur deux entrées analogiques (A0 et A1) et SW sur une entrée digitale (D3).

Programme pour lire les positions d'un joystick

Ce programme utilise la bibliothèque : code/debounced_button.hpp présent dans le fichier code.zip.

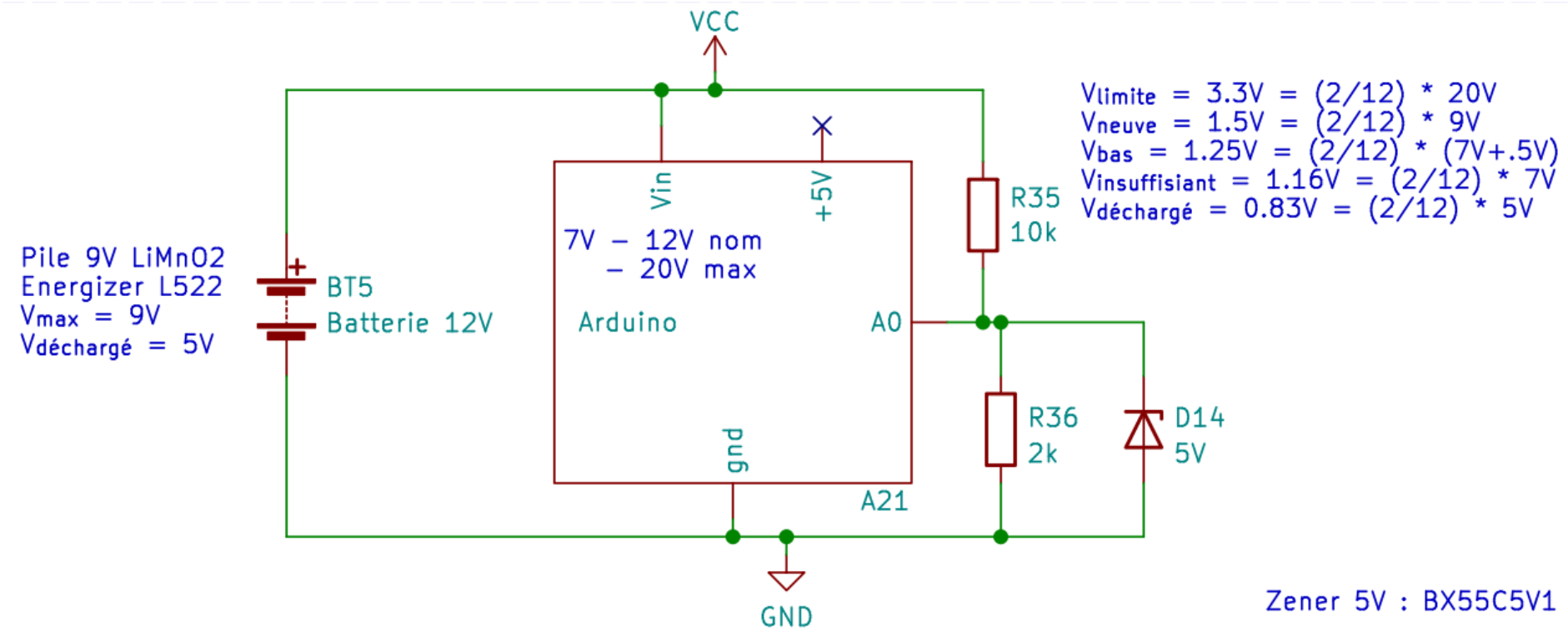


code/read_joystick_module.cpp

```
#include "debounced_button.hpp"

const int DEBOUNCE_DELAY = 10; // In ms
const int SW_PIN = 3;
const bool INTERNAL_PULLUP = true, INVERSE_STATE = false;
Debounced_button joystick_button(
  SW_PIN, INTERNAL_PULLUP, INVERSE_STATE, DEBOUNCE_DELAY
);
const int VRX_PIN = A0, VRY_PIN = A1;

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

unsigned long int old_time=0, current_time=0;
void loop() {
  joystick_button.update();
  if(joystick_button.rising_edge){
    Serial.println("Josytick Click");
  }
  current_time = millis();
  if(current_time - old_time > 100){
    int value_x = analogRead(VRX_PIN);
    Serial.print("X : "); Serial.print(value_x);
    Serial.print("/1023, ");
    int value_y = analogRead(VRY_PIN);
    Serial.print("Y : "); Serial.print(value_y);
    Serial.println("/1023");
    old_time = current_time;
  }
}
```

(version longue)

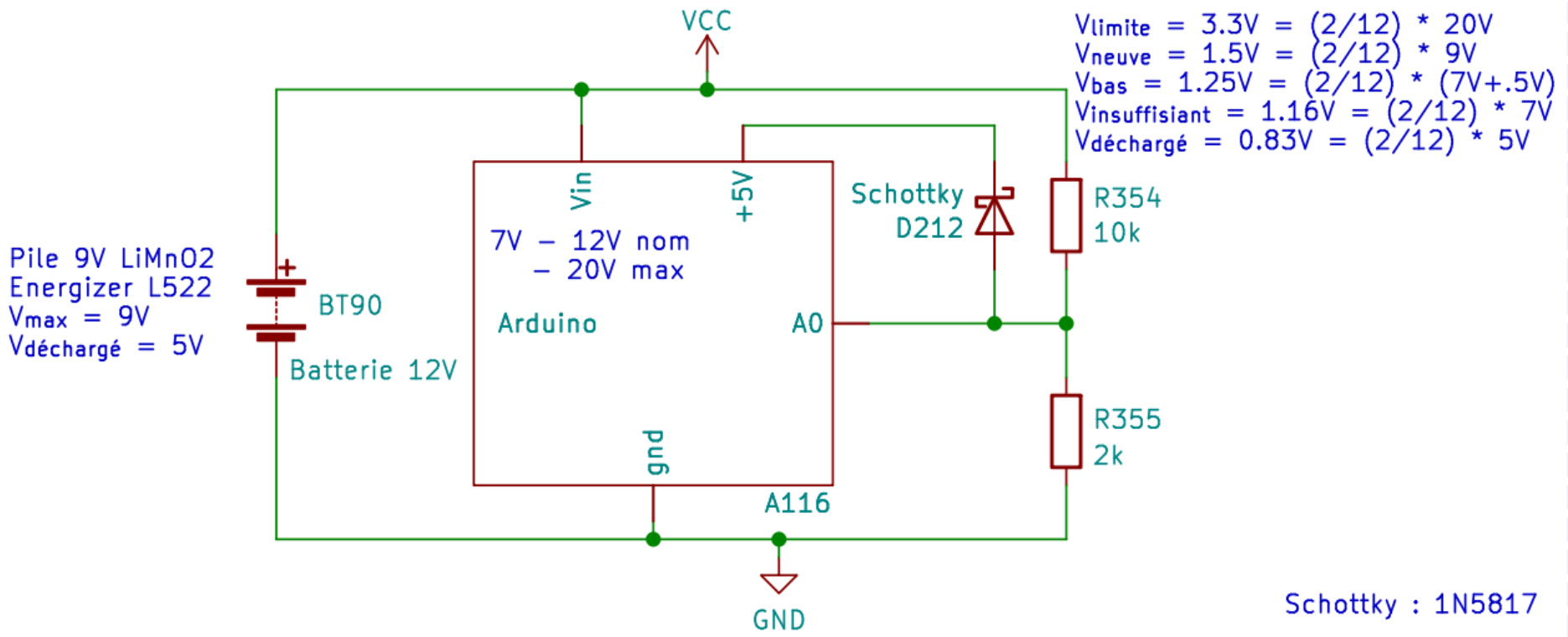
Les entrées analogique - suivre la décharge de la batterie - 1/2



La tension au borne de A0 ne peut pas dépasser la tension d'avalanche de 5V de la diode zener D14. La diode zener permet de protéger l'entrée analogique de tous pics de tension provenant du rail d'alimentation.

(version longue)

Les entrées analogique - suivre la décharge de la batterie - 2/2



La tension au borne de A0 ne peut pas dépasser la tension de $3.3V + 0.2V = 3.5V$ car la diode schotky devient passante en cas de surtension sur le rail d'alimentation.

(version longue)

Plan

(version longue)

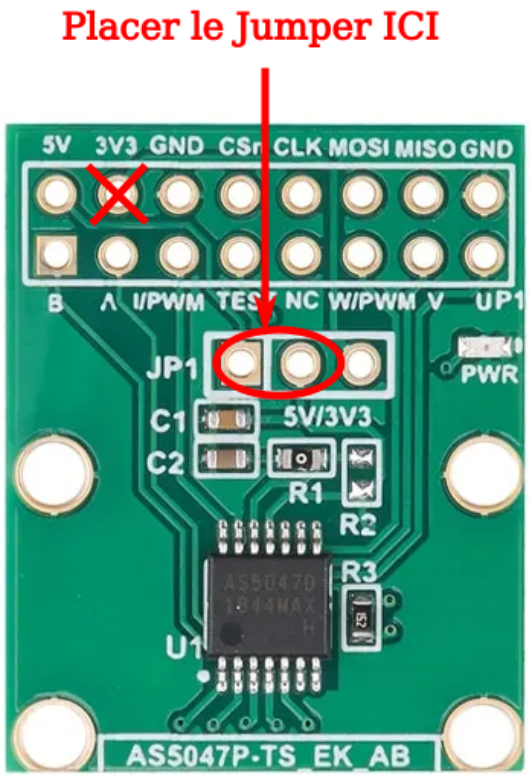
Plan

- Les capteurs de position, de vitesse et d'inclinaison
- Les capteurs de luminosité
- Les capteurs infrarouges
- Les capteurs de pression mécanique
- Les capteurs de pression pneumatique (gaz, souffle)
- Les capteurs de température

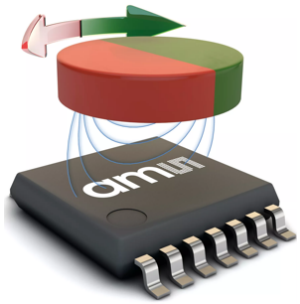
Le module contenant le capteur à effet Hall AS5047D

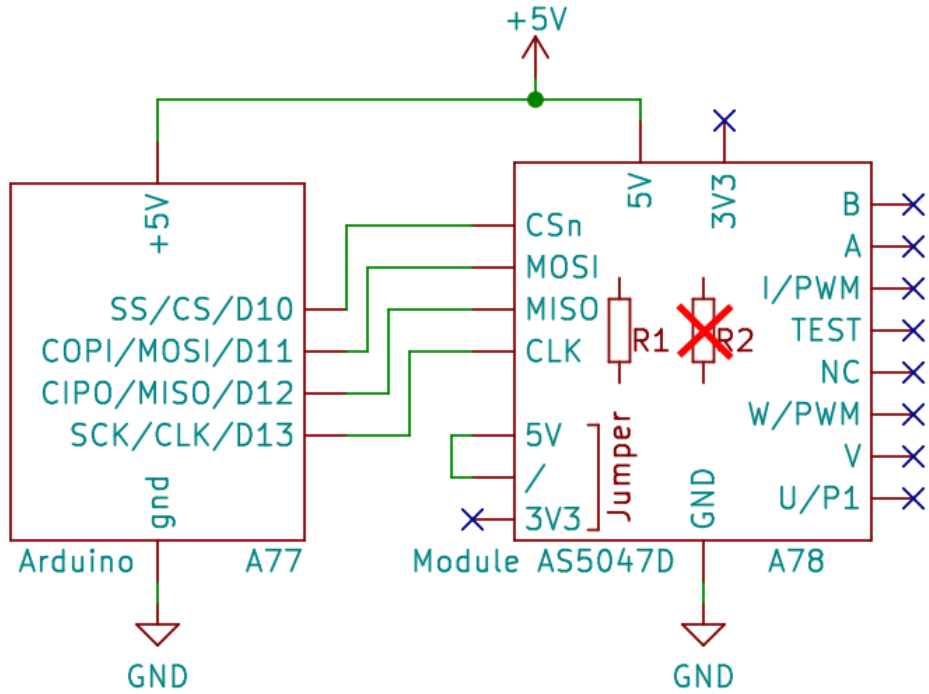
Ce capteur permet de mesurer la position angulaire d'un aimant. Pour mesurer la rotation de l'axe d'un moteur, on colle un aimant au bout de l'axe et on place le capteur AS5047D à une distance comprise entre 0.5 mm et 2.5mm.

En mode 5V



Carte en mode 5V





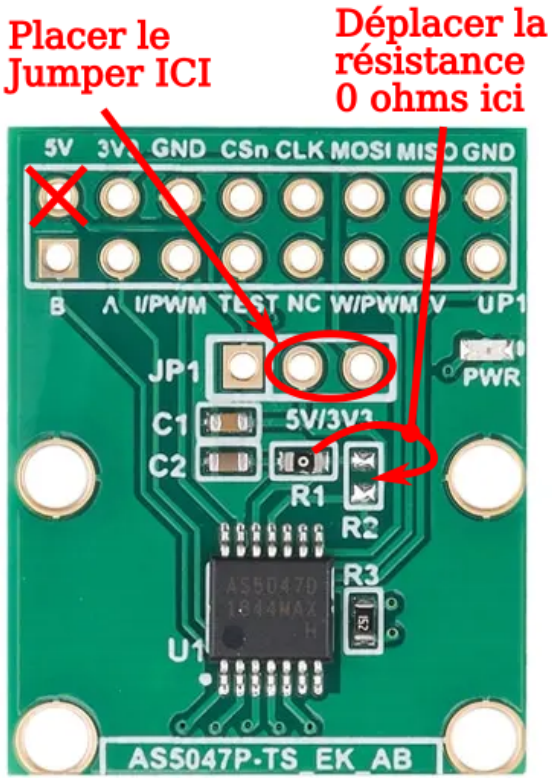
L'aimant doit être sous forme cylindrique, de type N35H, de diamètre 8 mm et d'épaisseur 3 mm. De plus la polarité N-S doit être horizontal.

(version longue)

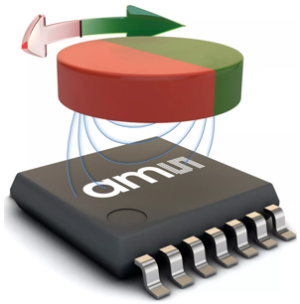
Le module contenant le capteur à effet Hall AS5047D

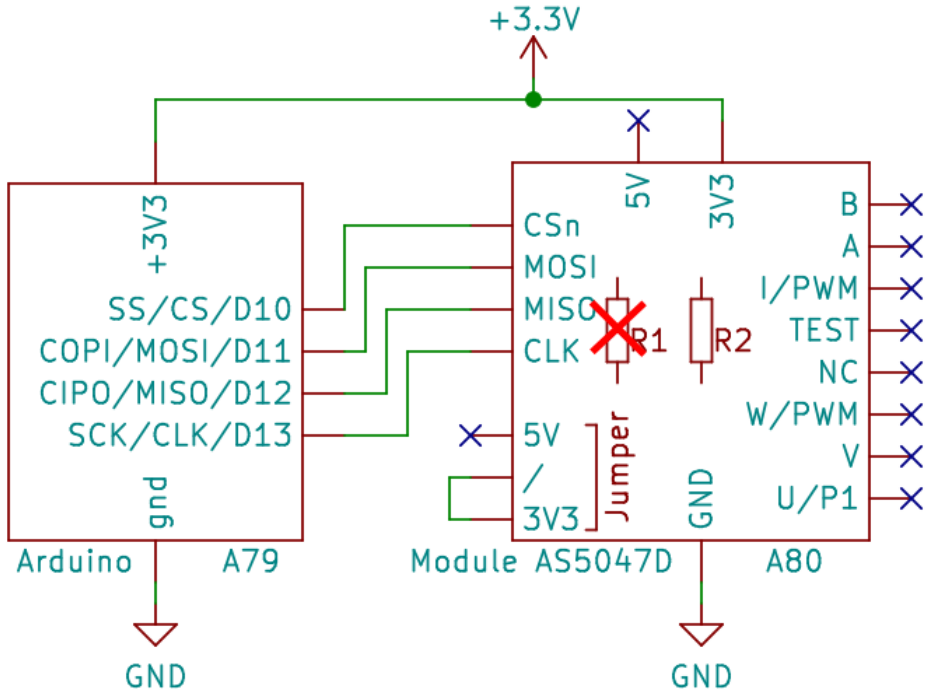
Ce capteur permet de mesurer la position angulaire d'un aimant. Pour mesurer la rotation de l'axe d'un moteur, on colle un aimant au bout de l'axe et on place le capteur AS5047D à une distance comprise entre 0.5 mm et 2.5mm.

En mode 3.3V



Carte en mode 3V3





L'aimant doit être sous forme cylindrique, de type N35H, de diamètre 8 mm et d'épaisseur 3 mm. De plus la polarité N-S doit être horizontal.

(version longue)

Programmer le capteur à effet Hall AS5047D 1/2

Il faut installer la bibliothèque simplefoc/SimpleFOCDrivers qui contient une bibliothèque pour gérer le capteur AS5047D..

```
#include <Wire.h>
#include <SPI.h>
#include <encoders/as5047/MagneticSensorAS5047.h>

#define SENSOR_CS 5 // some digital pin that you're using as the nCS pin
#define FAST_MODE true
MagneticSensorAS5047 sensor(SENSOR_CS, FAST_MODE, AS5047SPISettings);

void make_diagnostic(){
    AS5047Diagnostics diagnostics = sensor.readDiagnostics();
    if(diagnostics.magh) Serial.println("Magnetic field strength is too high !");
    if(diagnostics.magl) Serial.println("Magnetic field strength is too low !");
    if(!diagnostics.lf) Serial.println("Offset compensation is not completed !");
    if(!diagnostics.cof) Serial.println("CORDIC overflow ! Mesured angle is not reliable.");
}

void setup() {
    sensor.init(&SPI); // Default SPI

    delay(1000);

    sensor.update();
    make_diagnostic();
}
```

code/capteurs/capteur_hall/read_as5047.cpp

(version longue)

Programmer le capteur à effet Hall AS5047D 2/2

```
void loop(){
    sensor.update();

    float full_rotation_angle = sensor.getAngle(); // rad
    float velocity = sensor.getVelocity(); // rad/s, always call getAngle() before.
    float angle = sensor.getCurrentAngle(); // in [-pi, pi] rad
    uint16_t raw_angle = sensor.readRawAngle(); // 14 bit value
    float magnetic_field_strength = sensor.readMagnitude();

    // check for errors
    if (sensor.isErrorFlag()) {
        AS5047Error error = sensor.clearErrorFlag();
        if (error.parityError) {
            // also error.framingError, error.commandInvalid, etc...
            Serial.println("The new value is Corrupted.");
        }
    }

    Serial.print("full rotation angle : "); Serial.println(full_rotation_angle);
    Serial.print("angle : "); Serial.println(angle);
    Serial.print("raw angle : "); Serial.println(raw_angle);
    Serial.print("velocity : "); Serial.println(velocity);
    Serial.print("magnetic field : "); Serial.println(magnetic_field_strength);
}
```

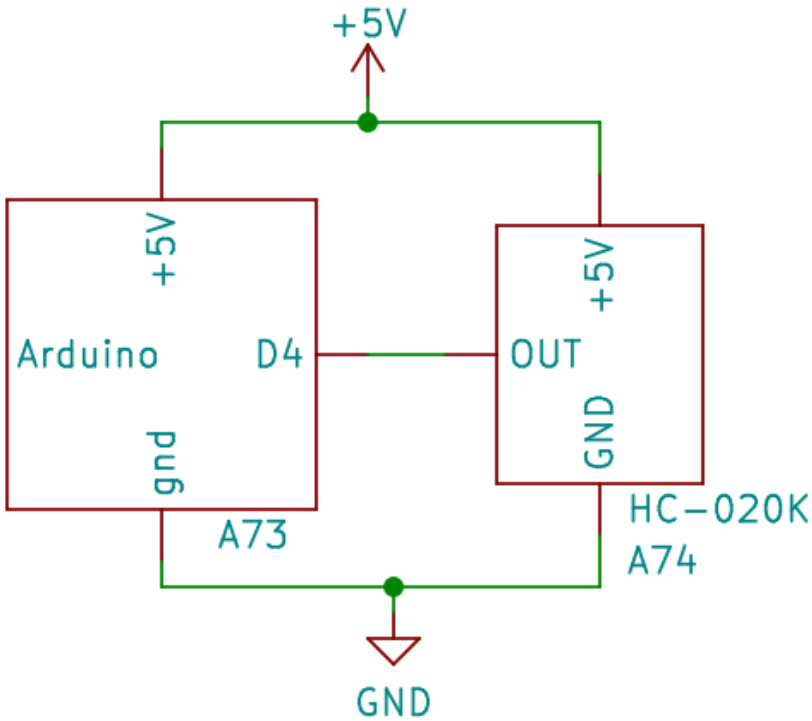
code/capteurs/capteur_hall/read_as5047.cpp

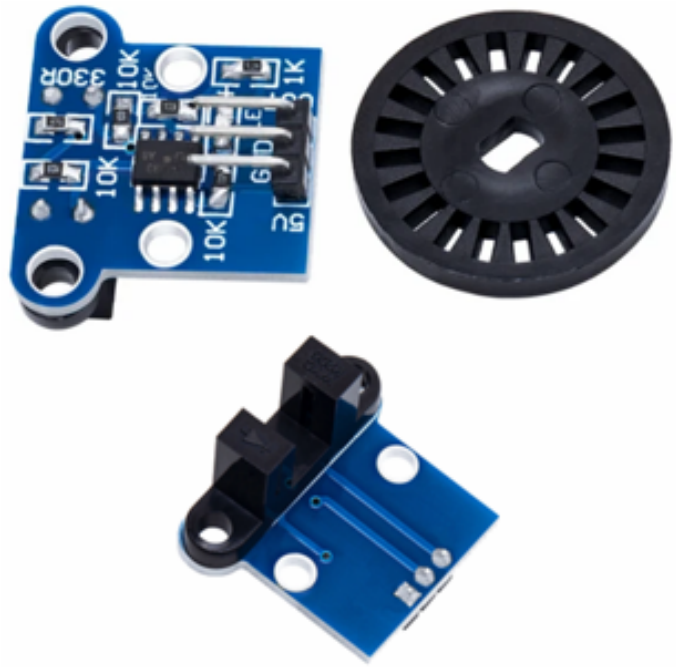
Le retard de la mesure raw_angle issue du capteur est de 100 μs.

Les capteurs à ultrasons

A FAIRE :

La barrière optique du module HC-020k - 1/3

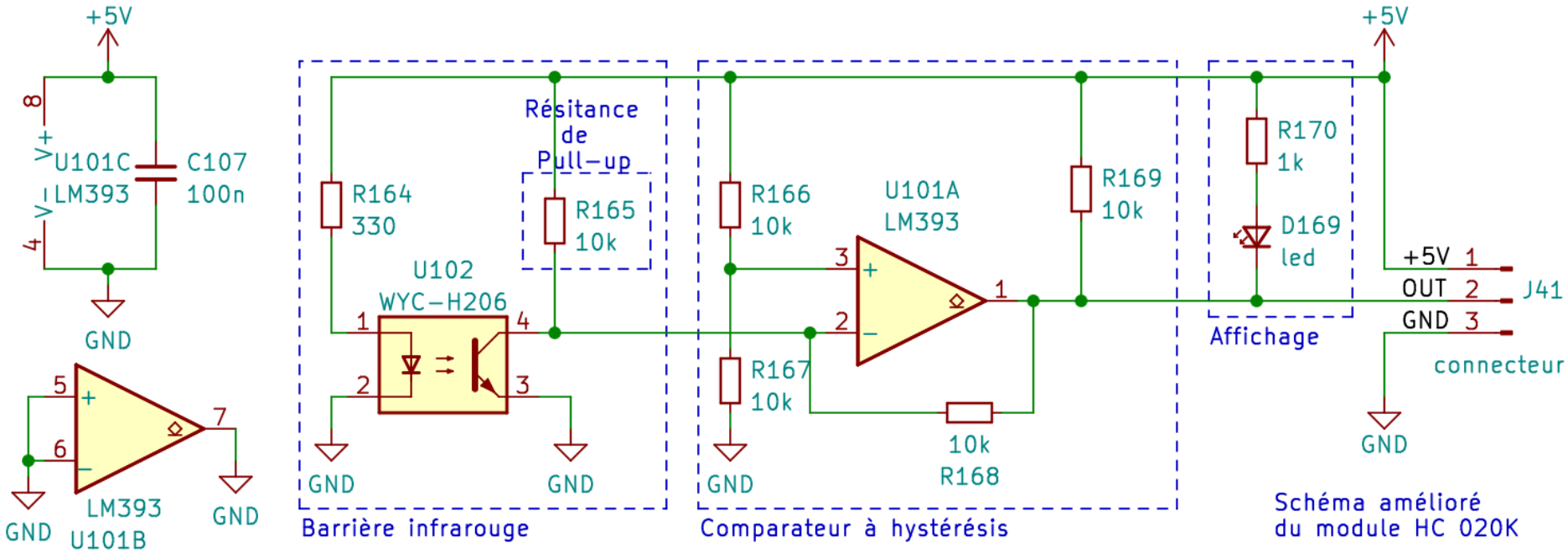




A chaque fois qu'un trou de la roue encodeuse passe devant la barrière infra rouge, la sortie du module passe de 0V à 5V.

Comme la roue contrent 20 trous, ce composant peut donc mesurer des rotations de 1/20 de tours.

Le schéma amélioré du module HC-020k - 2/3



Danc ce schéma, la résistance R170 a été ajouté pour améliorer le circuit afin d'obtenir un hystérésis et une comutation claire. Le condensateur C107 de découplage a aussi été ajouté au borne du comparateur afin de stabiliser son alimentation.

Mesurer position et vitesse avec le module HC-020k - 3/3

A FAIRE : Proposer une version plus robuste !

```
unsigned int counter = 0; // Holds the count of pulses detected

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

  Timer1.initialize(1000000);         // Set timer to trigger every 1 second
  attachInterrupt(0, Do_Count, RISING); // Create interrupt handler to count pulses
                                      // on rising edge of Int0
  Timer1.attachInterrupt( Timer_Isr ); // Define Interrupt Service Routing (ISR)
}

void loop(){ }

void Do_Count(){
  counter++; // increase +1 the counter value
}

void Timer_Isr(){
  Timer1.detachInterrupt();       // Disable the timer
  Serial.print("Count: "); Serial.print (counter);
  float rotation = (counter / 20.0); // divide by number of holes in Disc
  Serial.print(" Motor Speed: ");
  Serial.print(rotation); Serial.print(" RPS ");    // Rotations Per Second
  Serial.print(rotation * 60.0); Serial.println(" RPM"); // Rotations Per Minute
  counter = 0;                      // Reset counter to zero
  Timer1.attachInterrupt( Timer_Isr ); // Re-enable the timer
}
```

code/capteurs/barriere_optique/vitesse_position_barriere_optique.cpp

(version longue)

Les encodeurs incrémentaux

A FAIRE :

Plan

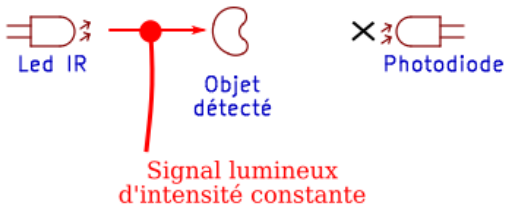
Les photorésistances

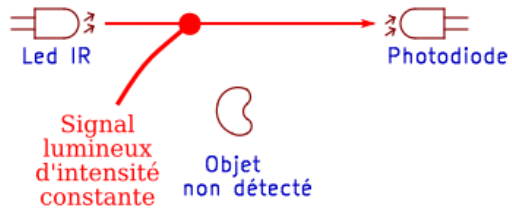
A FAIRE :

Plan

Le principe des barrières infrarouges

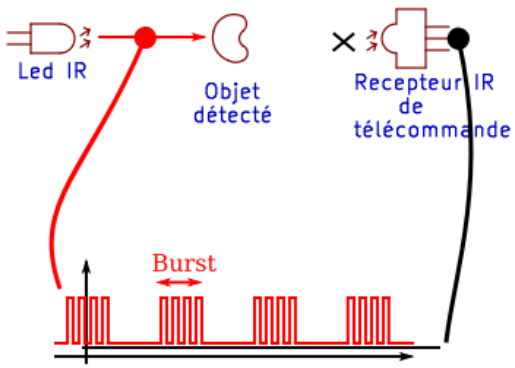
Une barrière infra rouge est constituée d'une led infrarouge (led IR) emmétant un signal lumineux qui est détecté par une photodiode. Une photodiode est une led qui génére un faible courant électrique ($\leq$10 à 100 μA) en sens inverse lorsqu'elle recoit un signal lumineux. En mesure le courant de la photdiode, on peut detecter le passage d'un objet entre la led emétrice et la led réceptrice.

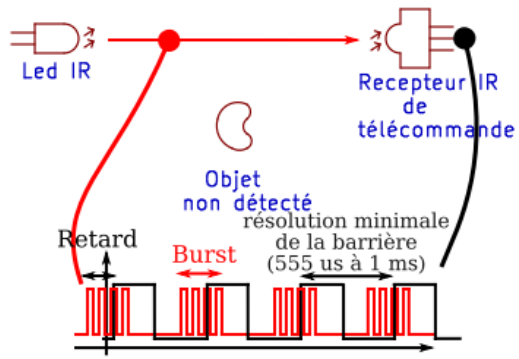




Pour réaliser ces barrières, vous pouvez constuire les montages de la page 173 à la page 176.
La résolution des mesures dépendent du montage et des diodes émétrice et réceptrices choisies.

Les photodiodes sont sensibles à la lumière du jour et sont vites saturées en plein jour. Pour éviter cela, on utilise des récepteurs IR pour télécommande qui contiennent une photodiode et un circuit pour démoduler un signal lumineux constitués de "burst" qui sert à discerner le signal envoyé de la lumière du soleil.





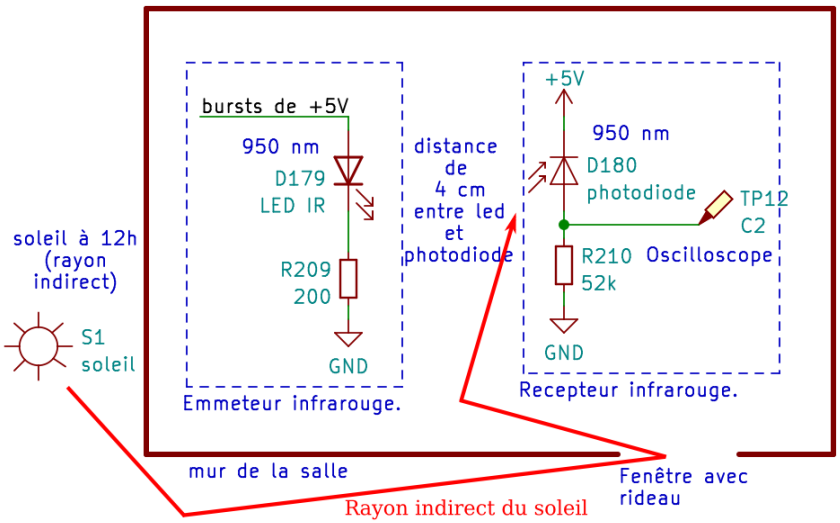
Pour réaliser ces barrières, vous pouvez utiliser les modules ky-032 et ky-022.

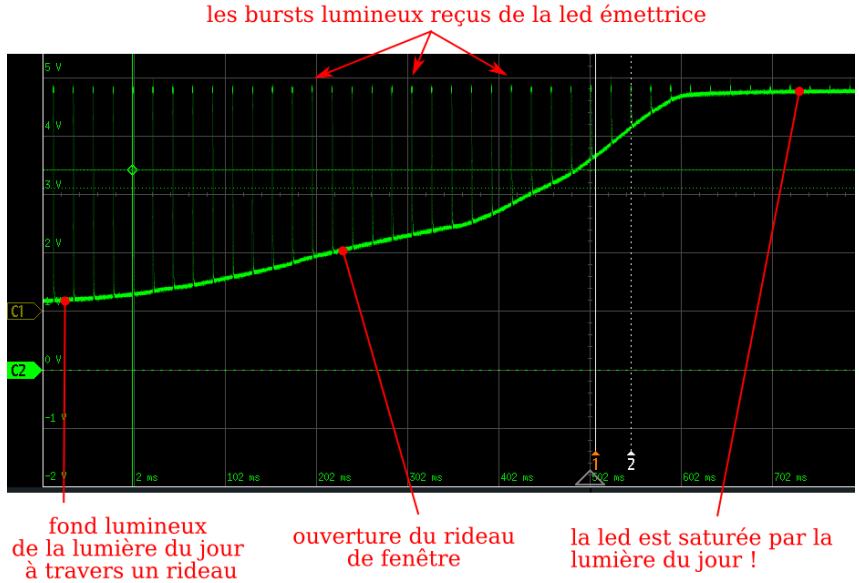
(version longue)

Influence du soleil sur les photodiodes

Les photodiodes sont sensibles à la lumière du jour qui contient un rayonnement infrarouge.

L'expérience ci-dessous montre la saturation d'un capteur infrarouge à l'ouverture d'un rideau d'une fenêtre. Le canal 2 de l'oscilloscope de la figure de gauche est montré à droite.





Les modules ky-032 et ky-022 réussisent à recevoir les signaux car, malgrès la lumière ambiante, ils possèdent, en plus de la photodiode, un circuit interne pour detecter les bursts envoyés par la led émétrice.

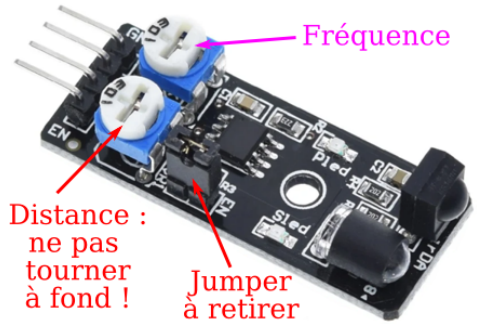
En l'abscence de ce circuit (cf. circuits page 141 à page 144), il faut protéger la photodiode des sources lumineuses parasites (soleil, lampes fluorescentes, ampoules tungstène, etc...).

Une solution consiste à déplacer les sources lumineuses en dehors du champs de vision de la photodiode et de protéger la photodiode avec un tube.

Vous pouvez aussi envoyer des pulses lumineux de 1A au lieu de 20 mA en remplacant le circuit de la led éméttrice par le montage de la page 145. Faites attention à ne pas saturer le recepteur !

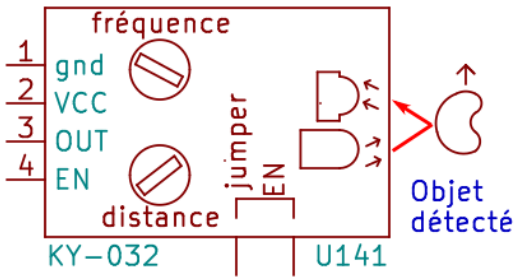
(version longue)

La Barrière infrarouge avec le module KY-032 pour Arduino



Le module KY-032 contient à la fois une led IR émettrice, mais aussi un récepteur IR de télécommande. Il peut s'utiliser au choix comme :

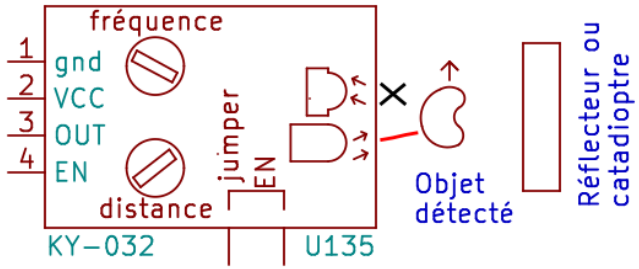
Détecteur d'obstacle de proximité :



L'objet doit être très proche du capteur !

Cette méthode permet de détecter des objets sur de courtes distances.

Détecteur de présence :

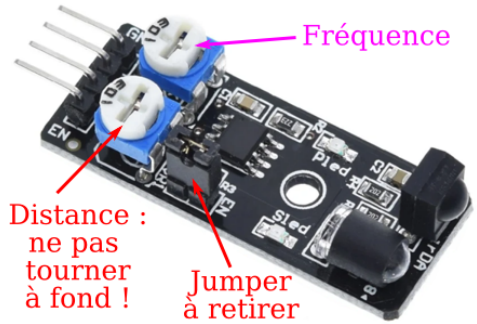


Cette méthode permet de détecter des objets sur de longues distances.

L'objet ne doit pas réfléchir la lumière infrarouge ou doit être suffisament éloigné de la source infrarouge.

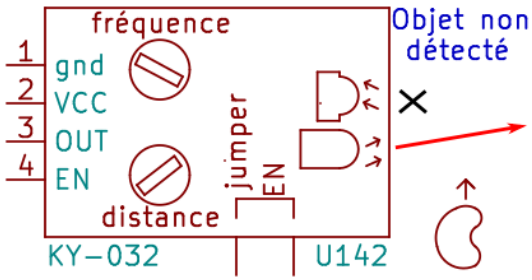
(version longue)

La Barrière infrarouge avec le module KY-032 pour Arduino



Le module KY-032 contient à la fois une led IR émettrice, mais aussi un récepteur IR de télécommande. Il peut s'utiliser au choix comme :

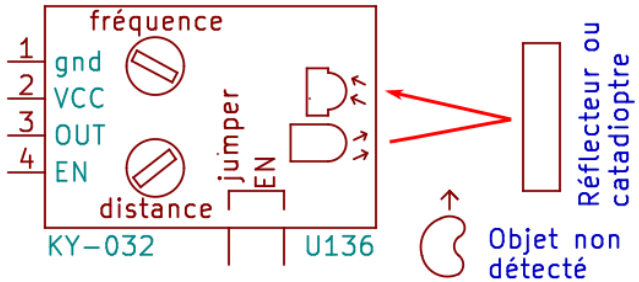
Détecteur d'obstacle de proximité :



L'objet doit être très proche du capteur !

Cette méthode permet de détecter des objets sur de courtes distances.

Détecteur de présence :



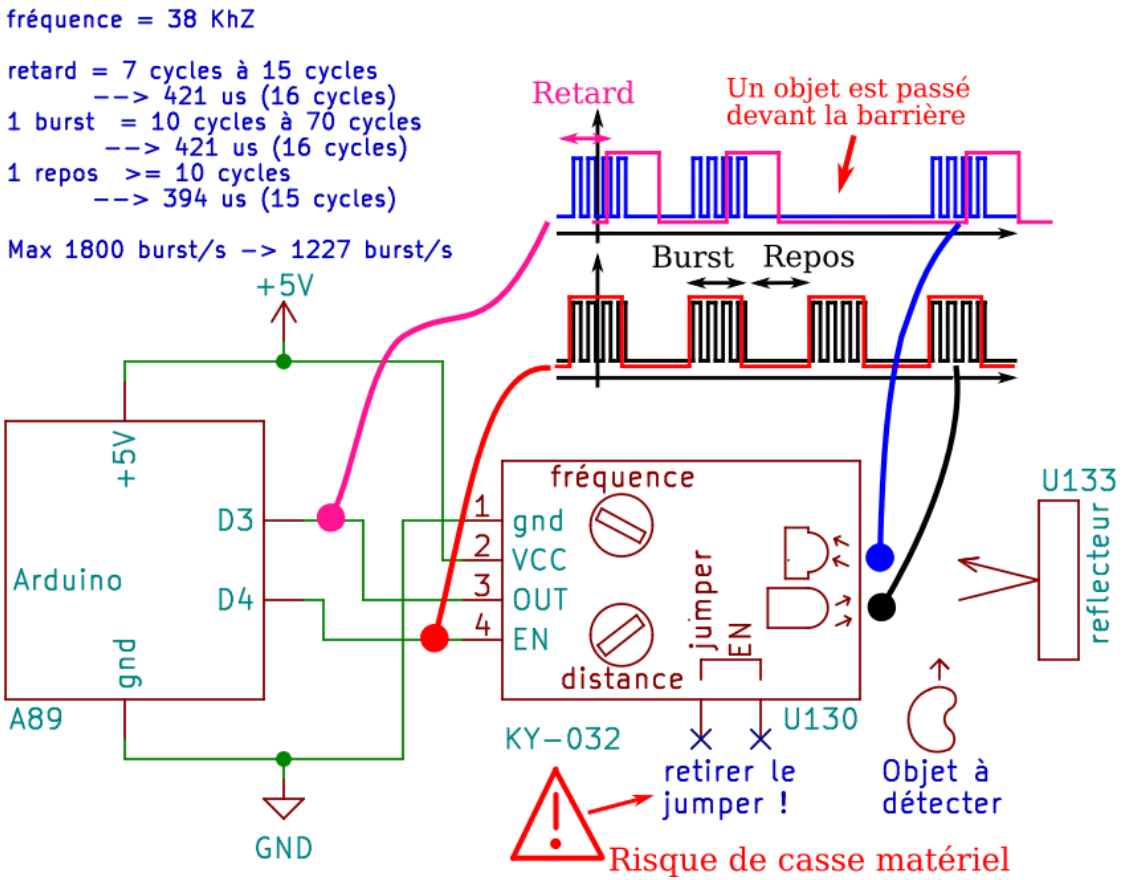
Cette méthode permet de détecter des objets sur de longues distances.

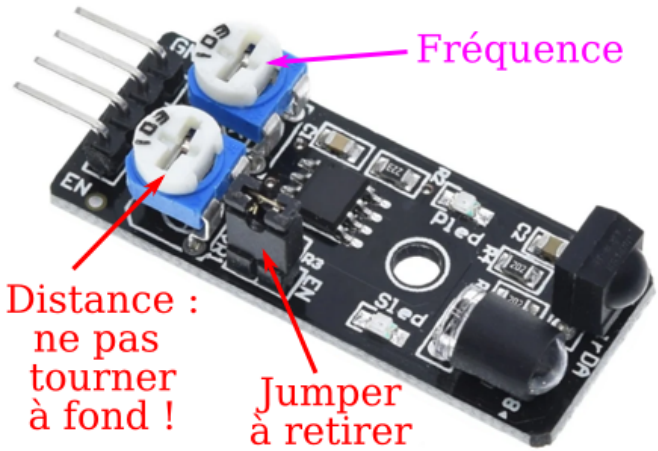
L'objet ne doit pas réfléchir la lumière infrarouge ou doit être suffisament éloigné de la source infrarouge.

(version longue)

La Barrière infrarouge avec le module KY-032 pour Arduino

Voici comment utiliser le module ky-032 avec un arduino : Avec D4 on provoque un burst à interval régulier. Pendant le burst, on mersure D3 pour détecter un objet.





Dans le module, le potentiomètre du haut sert à régler la fréquence du signal émis par le module. Laisser le réglage constructeur ou régler-là à 38 kHz avec l'aide d'un oscilloscope en connectant la sonde à l'un des pieds de la led IR émmétrice.

Le potentiomètre du bas sert à diminuer ou augmenter la puissance de la lumière émise. Régler-la avec précaution en fonction de votre usage (risque de casse, cf. slide suivante).

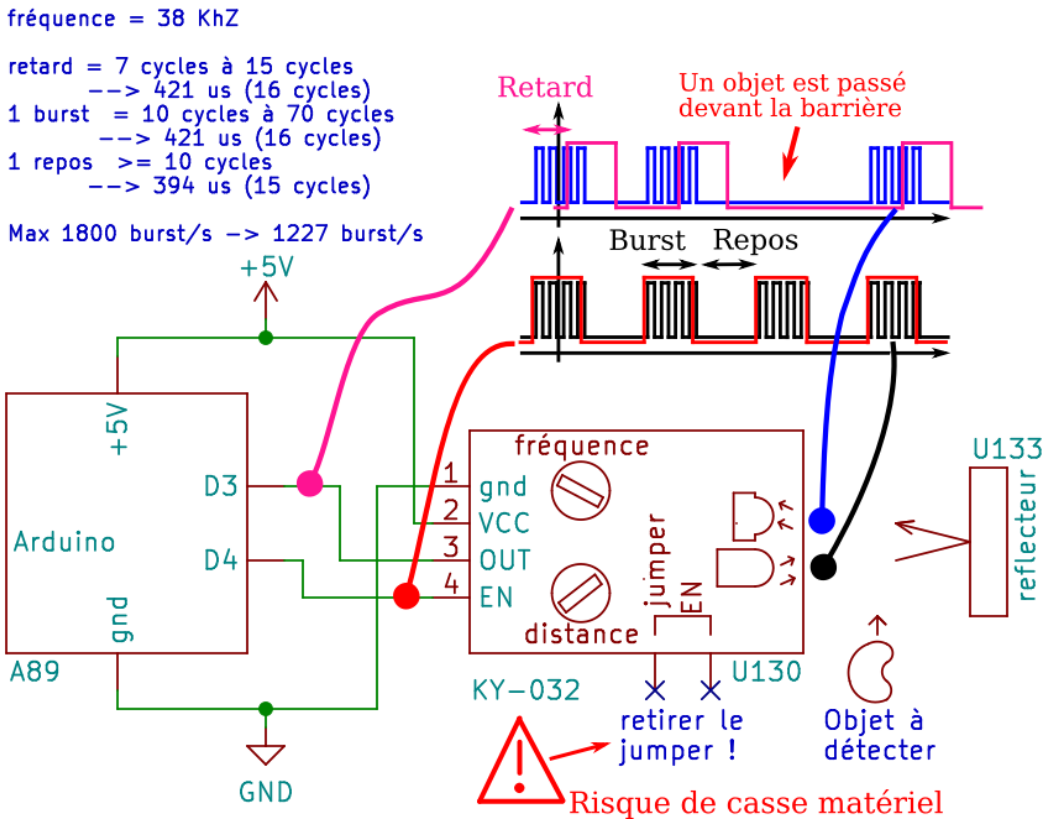
(version longue)

Le programme controlant le module KY-032 sous Arduino

ATTENTION ! RISQUE DE CASSE MATERIEL !

Il faut retirer le jumper pour ne pas détruire l'Arduino !

Ne jamais tourner le potentiometre "distance" à fond, sous peine de brûler le potentiomètre ou la led IR (cf. le schema du KY-032 page 163). Si vous brûlez le potentiomètre, remplacez le par une résistance de 220 Ω pour un module alimenté en 5 V et une résistance 100 Ω pour la version 3.3 V.

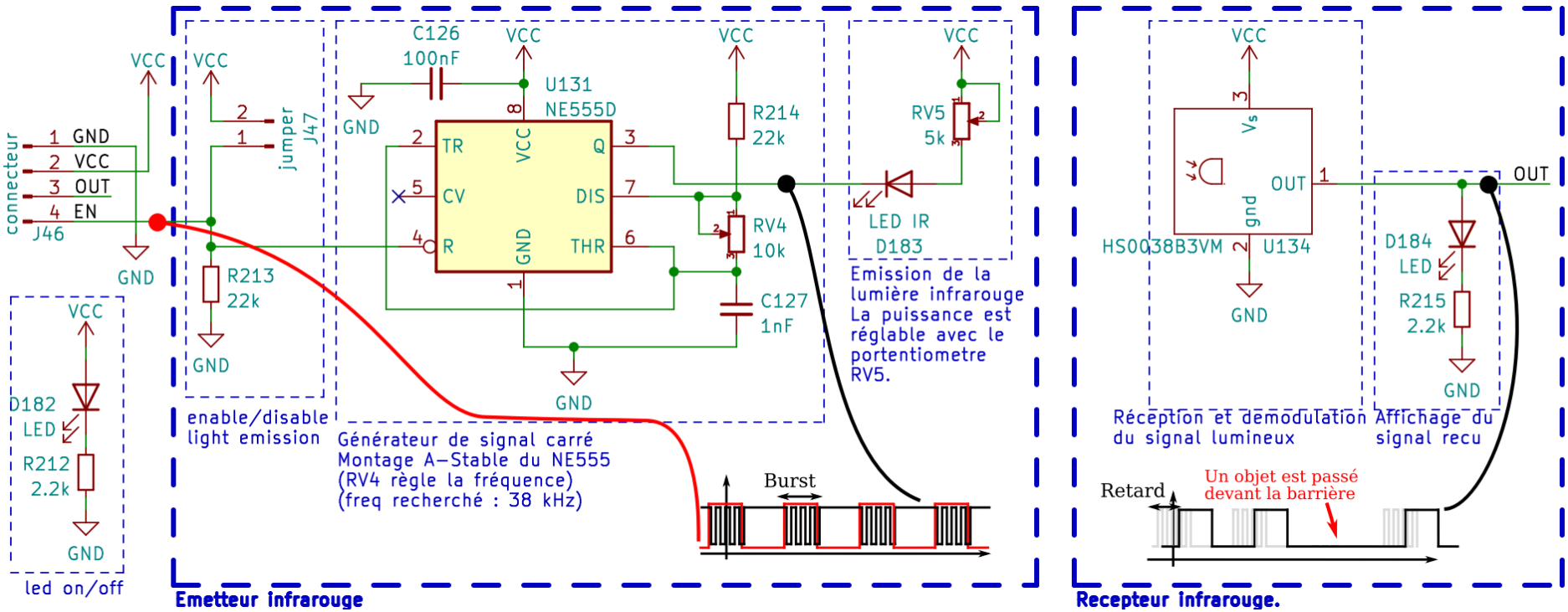


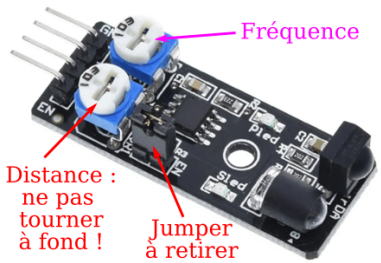
code/capteurs/barriere_optique/barriere_ir_ky036.cpp

```
const int EN_pin = 4, OUT_pin = 3;
const int output_delay_time = 505; // us
   // 505 us = 421 us + 20% of margin
const int rest_time = 394; // us

void setup(){
  Serial.begin(9600);
  pinMode(EN_pin, OUTPUT);
  digitalWrite(EN_pin, LOW);
  pinMode(OUT_pin, INPUT);
}

int detect_an_object(){
  // Start a burst
  digitalWrite(EN_pin, HIGH);
  // Wait some time for demodulation
  delayMicroseconds(output_delay_time);
  // Make a measure
  int state = digitalRead(OUT_pin);
  // Stop the burst
  digitalWrite(EN_pin, LOW);
  // Wait the rest time
  delayMicroseconds(rest_time);
  return state;
}

void loop(){
  int object_presence = detect_an_object();
  Serial.print("Detected object : ");
  Serial.println(object_presence);
  // delay(500);
}
```

(version longue)

Le schéma électronique du module KY-032 pour Arduino



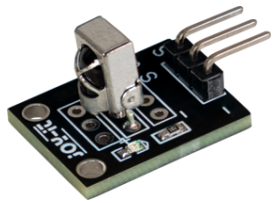


(version longue)

Barrière infrarouge avec le module KY-022 pour Arduino

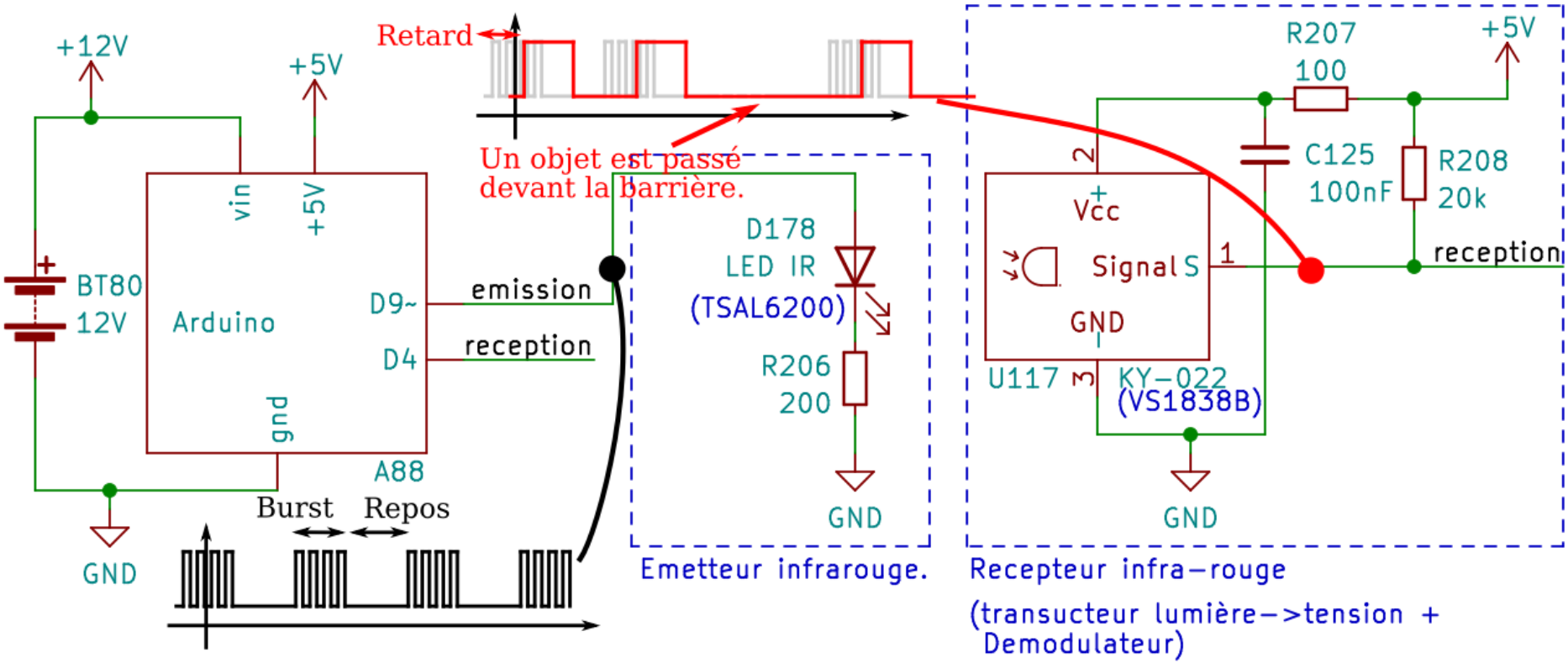
Ce module est une photodiode contenant un démodulateur qui détecte des bursts de signaux lumineux de formes carrés dont la porteuse est à 38 kHz.

C'est une version simplifiée du module KY-032 qui est réduit au seul composant VS1838B. C'est une version bon marché et de très mauvaise qualité des TSOP34838 et HS0038B3VM (cf. page 163). Il fonctionne de la même manière sauf que le temps de repos est très long (28 ms au lieu de 394 μs) !



Module KY-022
(*VS1838B*)

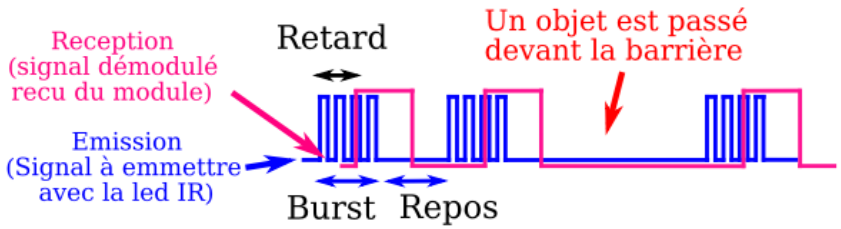
Led IR
(TSAL6200)



Barrière infrarouge avec le module KY-022 pour Arduino

Ce programme utilise la bibliothèque TimerOne de Paul Stoffrege qui fonctionne que pour certaines cartes arduino.

Voici les chronogrammes des signaux d'émission et de réception :



Caractéristiques du chronograme :

- fréquence de la pwm : 38 kHz
  ⟶ la période vaut : $\frac{1}{38\text{kHz}} \approx 26\ \mu s$.
- rapport cyclique de la pwm : ≤ 50 %
  ⟶ On choisit : 4 %.
- durée du burst : entre 500 et 700 μs
  ⟶ on l'arrêtera après la mesure.
- durée du repos : la datasheet dit n'imp.
  ⟶ 28 *ms* fonctionne
- retard attendu : non spécifié, on mesure généralement : 270 μs qui peut aller jusqu'à 740 μs.
  ⟶ 600 μs fonctionne.

```
#include <TimerOne.h>

const int light_pin = 9, module_pin = 4;
const int module_delay_time = 600; // us
const int rest_time_in_ms = 28; // ms
const int period = 26; // us, = 1/(38 kHz)
const float duty_cycle = 5; // percentage, <= 5%

void setup(){
  Timer1.initialize(period); Timer1.pwm(light_pin, 0);
  pinMode(module_pin, INPUT);
  Serial.begin(9600);
}

int detect_an_object(){
  // Start the burst
  Timer1.pwm(light_pin, (duty_cycle / 100) * 1023);
  // Wait the end of demodulation process
  delayMicroseconds(module_delay_time);
  int state = digitalRead(module_pin); // Make a measure
  Timer1.pwm(light_pin, 0); // Stop the burst
  delay(rest_time_in_ms); // Wait the rest time
  return state;
}

void loop(){
  int object_presence = detect_an_object();
  Serial.print("Detected object : ");
  Serial.println(object_presence);
}
```

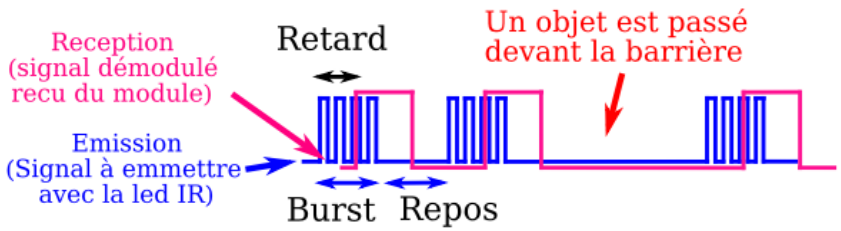
code/capteurs/barriere_optique/barriere_ir_ky022_pour_arduino.cpp

(version longue)

Barrière infrarouge avec le module KY-022 pour ESP32

Attention : Comme il n'y a pas forcément de broche 9 dans les ESP32, la broche de la lumière a été changée en 19 !

Voici les chronogrammes des signaux d'émissions et de réception :



Caractéristiques du chronograme :

- fréquence de la pwm : 38 kHz
  ⟶ la période vaut : $\frac{1}{38\text{kHz}} \approx 26\ \mu s$.
- rapport cyclique de la pwm : ≤ 50 %
  ⟶ On choisit : 4 %.
- durée du burst : entre 500 et 700 μs
  ⟶ on l'arrêtera après la mesure.
- durée du repos : la datasheet dit n'imp.
  ⟶ 28 *ms* fonctionne
- retard attendu : non spécifié, on mesure généralement : 270 μs qui peut aller jusqu'à 740 μs.
  ⟶ 600 μs fonctionne

code/capteurs/barriere_optique/barriere_ir_ky022_pour_esp32.cpp

```
const int PWM_RESOLUTION = 8;
const int MAX_DUTY = (int)(pow(2, PWM_RESOLUTION)-1);
const int light_pin = 19, module_pin = 4;
const int module_delay_time = 600; // us, 421 us + 20%
const int rest_time_in_ms = 28; // ms
const int frequency = 38000; // Hz
const float duty_cycle = 4; // percentage, < 10%

void setup(){
  ledcAttach(light_pin, frequency,
      PWM_RESOLUTION);
  ledcWrite(light_pin, 0);
  pinMode(module_pin, INPUT);
  Serial.begin(115200);
}

int detect_an_object(){
  // Start the burst
  ledcWrite(light_pin, (duty_cycle/100)*MAX_DUTY);
  // Wait the end of demodulation process
  delayMicroseconds(module_delay_time);
  int state = digitalRead(module_pin); // Make the measure
  // Stop the burst
  ledcWrite(light_pin, 0); // Start the burst
  delay(rest_time_in_ms); // Wait the rest time
  return state;
}

void loop(){
  int object_presence = detect_an_object();
  Serial.print("Detected object : ");
  Serial.println(object_presence);
}
```

(version longue)

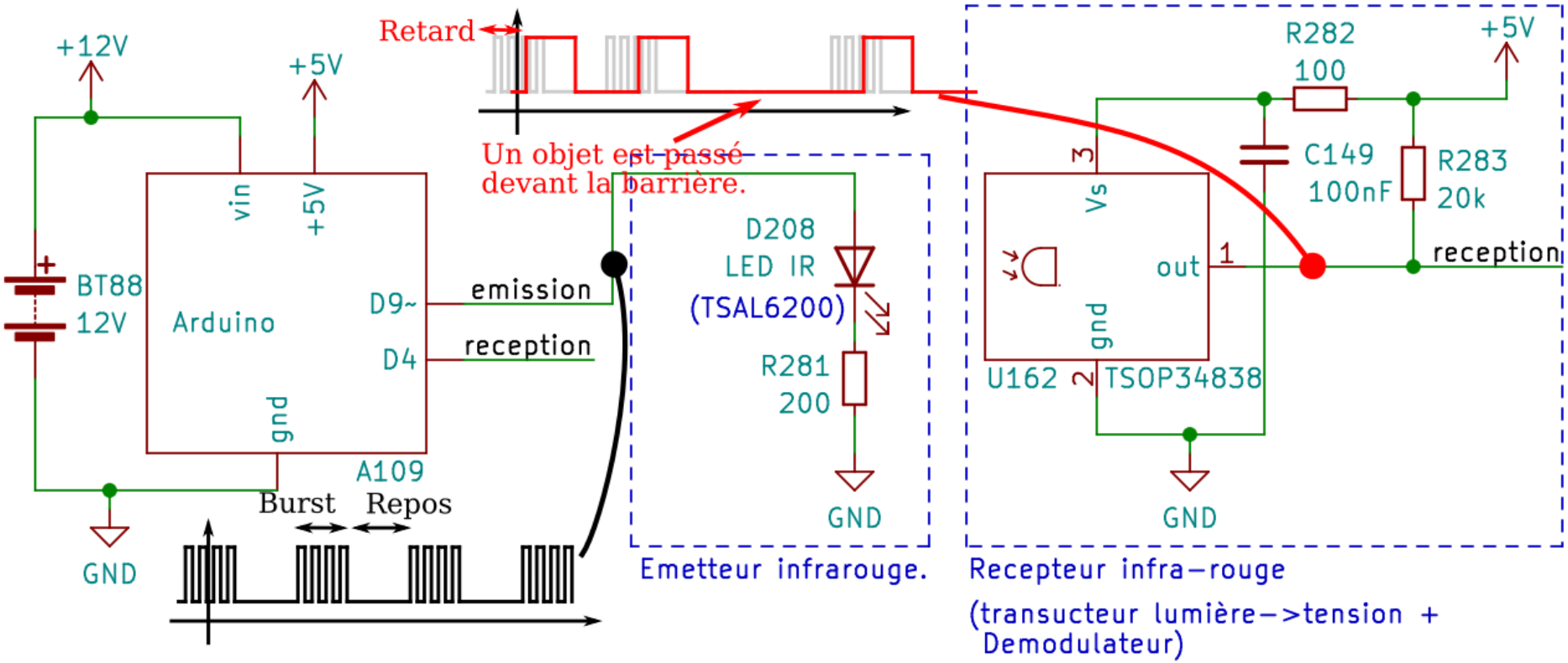
Barrière IR avec TSOP34838 ou HS0038B3VM pour Arduino

Le TSOP34838 (ou le HS0038B3VM) est une photodiode contenant un démodulateur qui détecte des bursts de signaux lumineux de formes carrés dont la porteuse est à 38 kHz.

C'est juste la partie reception du module du module KY-032. Il fonctionne de la même manière sauf que la conception de l'emission est à notre charge.

TSOP34838

Led IR (TSAL6200)

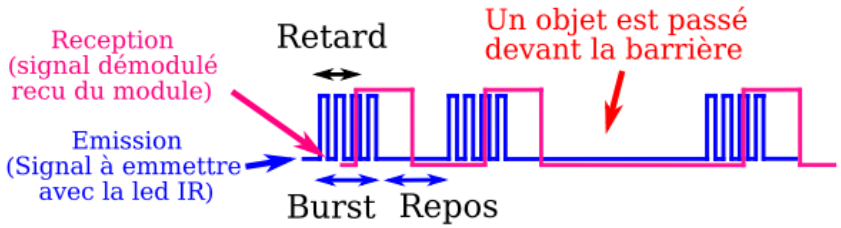


(version longue)

Barrière IR avec TSOP34838 ou HS0038B3VM pour Arduino

Ce programme utilise la bibliothèque TimerOne de Paul Stoffrege qui fonctionne que pour certaines cartes arduino.

Voici les chronogrammes des signaux d'émission et de réception :



Caractéristiques du chronograme :

- fréquence de la pwm : 38 kHz
  ⟶ la période vaut : $\frac{1}{38\text{kHz}} \approx 26\ \mu s$.
- rapport cyclique de la pwm : ≤ 50 %
  ⟶ On choisit : 4 %.
- durée du burst : 10 à 70 cycles pwm
  ⟶ on l'arrêtera après la mesure.
- durée du repos : ≥ 14 cycles pwm
  ⟶ 15 cycles : 394 μs
- retard attendu : 7 à 15 cycles
  ⟶ 15 cycles + 20 % : 505 μ s.

code/capteurs/barriere_optique/barriere_ir_tsop34838_pour_arduino.cpp

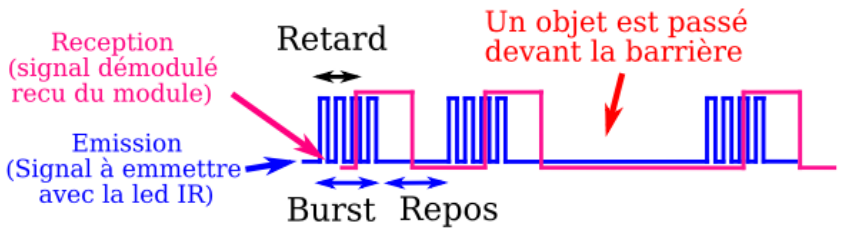
```
#include <TimerOne.h>

const int light_pin = 9, module_pin = 4;
const int module_delay_time = 505; // us
    // 505 us = 421 us + 20% of margin
const int rest_time = 394; // us
const int period = 26; // us, = 1/(38 kHz)
const float duty_cycle = 5; // percentage, <= 5%

void setup(){
  Timer1.initialize(period); Timer1.pwm(light_pin, 0);
  pinMode(module_pin, INPUT);
  Serial.begin(9600);
}

int detect_an_object(){
  // Start the burst
  Timer1.pwm(light_pin, (duty_cycle / 100) * 1023);
  // Wait the end of demodulation process
  delayMicroseconds(module_delay_time);
  int state = digitalRead(module_pin); // Make a measure
  Timer1.pwm(light_pin, 0); // Stop the burst
  delayMicroseconds(rest_time); // Wait the rest time
  return state;
}

void loop(){
  int object_presence = detect_an_object();
  Serial.print("Detected object : ");
  Serial.println(object_presence);
}
```

(version longue)

Barrière IR avec TSOP34838 ou HS0038B3VM pour ESP32

Attention : Comme il n'y a pas forcément de broche 9 dans les ESP32, la broche de la lumière a été changée en 19 !

Voici les chronogrammes des signaux d'émissions et de réception :



Caractéristiques du chronograme :

- fréquence de la pwm : 38 kHz
  ⟶ la période vaut : $\frac{1}{38\text{kHz}} \approx 26\ \mu s$.
- rapport cyclique de la pwm : ≤ 50 %
  ⟶ On choisit : 4 %.
- durée du burst : 10 à 70 cycles pwm
  ⟶ on l'arrêtera après la mesure.
- durée du repos : ≥ 14 cycles pwm
  ⟶ 15 cycles : 394 μs
- retard attendu : 7 à 15 cycles
  ⟶ 15 cycles + 20 % : 505 μ s.

code/capteurs/barriere_optique/barriere_ir_tsop34838_pour_esp32.cpp

```
const int PWM_RESOLUTION = 8;
const int MAX_DUTY = (int)(pow(2, PWM_RESOLUTION)-1);
const int light_pin = 19, module_pin = 4;
const int module_delay_time = 505; // us, 421 us + 20%
const int rest_time = 394; // us
const int frequency = 38000; // Hz
const float duty_cycle = 4; // percentage, < 10%

void setup(){
  ledcAttach(light_pin, frequency,
      PWM_RESOLUTION);
  ledcWrite(light_pin, 0);
  pinMode(module_pin, INPUT);
  Serial.begin(115200);
}

int detect_an_object(){
  // Start the burst
  ledcWrite(light_pin, (duty_cycle/100)*MAX_DUTY);
  // Wait the end of demodulation process
  delayMicroseconds(module_delay_time);
  int state = digitalRead(module_pin); // Make the measure
  // Stop the burst
  ledcWrite(light_pin, 0); // Start the burst
  delayMicroseconds(rest_time); // Wait the rest time
  return state;
}

void loop(){
  int object_presence = detect_an_object();
  Serial.print("Detected object : ");
  Serial.println(object_presence);
}
```

(version longue)

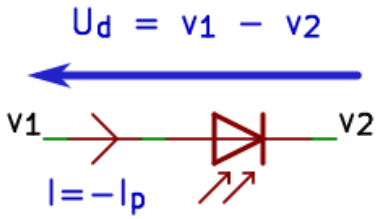
Télécommande IR et module KY-022 pour Arduino

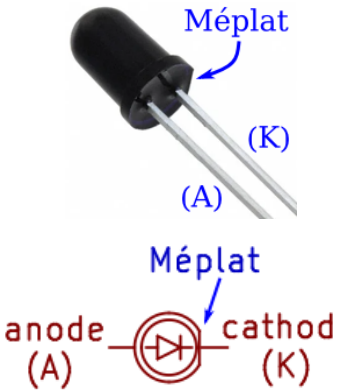
A FAIRE : En attendant, voici quelques liens utiles :
RECEPTEUR IR avec le module ky-022

Télécommande IR avec le module ky-005 (une led IR)

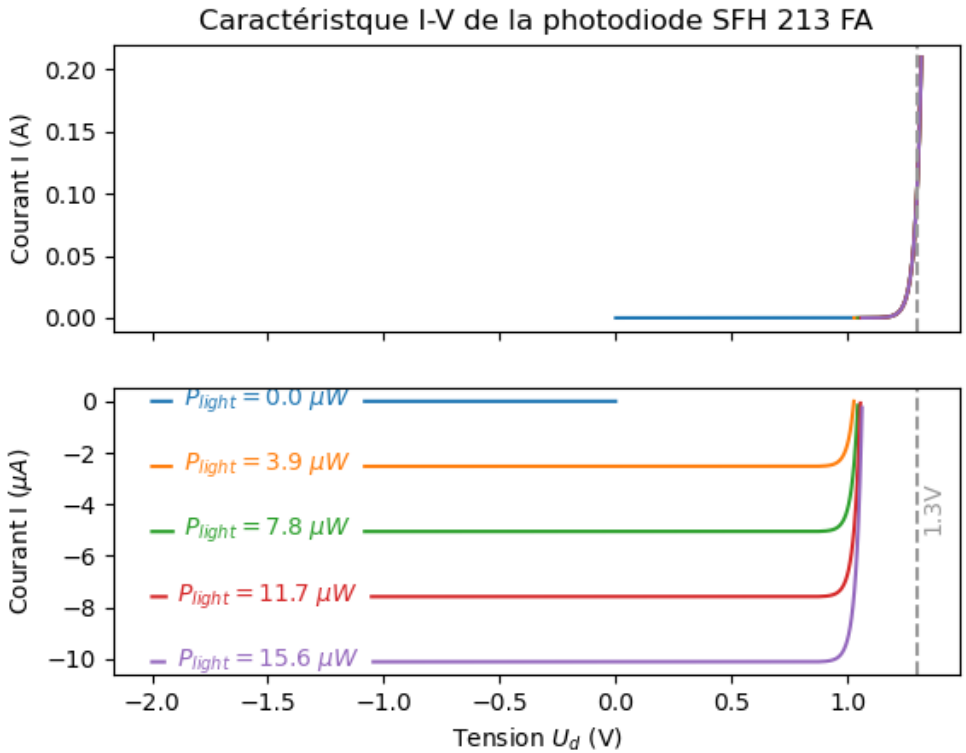
Format de donnée.

Modèle simplifiée des photodiodes





vue de dessus



Lorsque l'on polarise une photodiode en dessous de $0.7V$, elle se comporte comme un générateur de très faible courant ($\leq 100\ \mu A$) commandé linéairement par la puissance lumineuse reçue.

Lorsque l'on éclaire une photodiode, un photocourant $I_p \leq 100\ \mu A$ s'additione au courant inverse de la jonction de la diode. Ce courant I_p (A) est égal à :

$$I_p = S_\lambda \times P_{light}$$

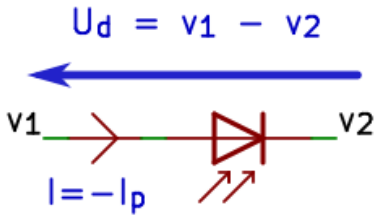
où S_λ est la réacitvité (A/W) et P_{light} à la puissance (W) lumineuse reçue.

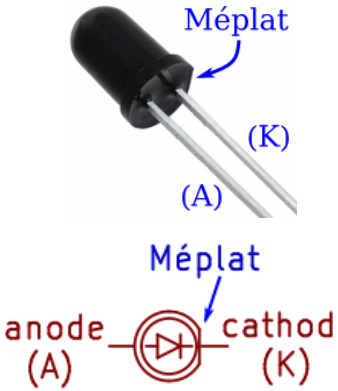
Le modèle simplifié de la photodiode est alors :

$$\left(\quad I \approx -S_\lambda \times P_{light} \quad \text{et} \quad U_d \leq 1.3V \quad \right) \qquad \text{ou bien} \qquad \left(\quad I > 0 \quad \text{et} \quad U_d \approx 1.3V \quad \right)$$

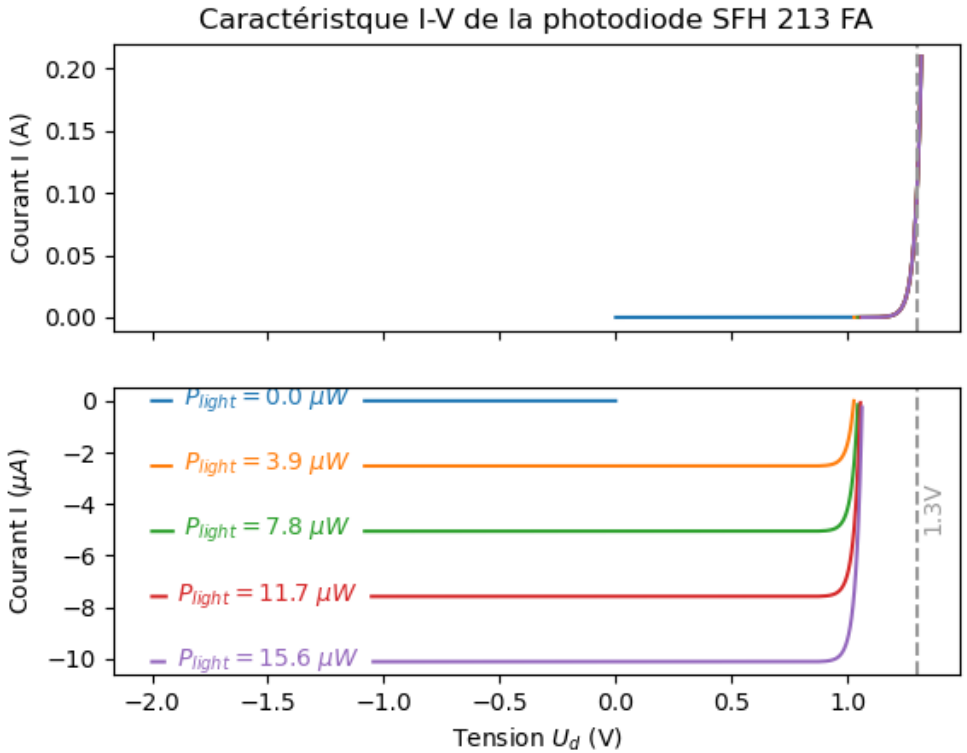
(version longue)

Usage des photodiodes

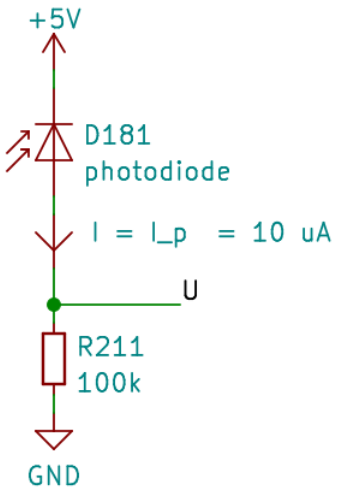




vue de dessus



Lorsque l'on polarise une photodiode en dessous de $0.7V$, elle se comporte comme un générateur de très faible courant ($\leq 100\ \mu A$) commandé linéairement par la puissance lumineuse reçue.



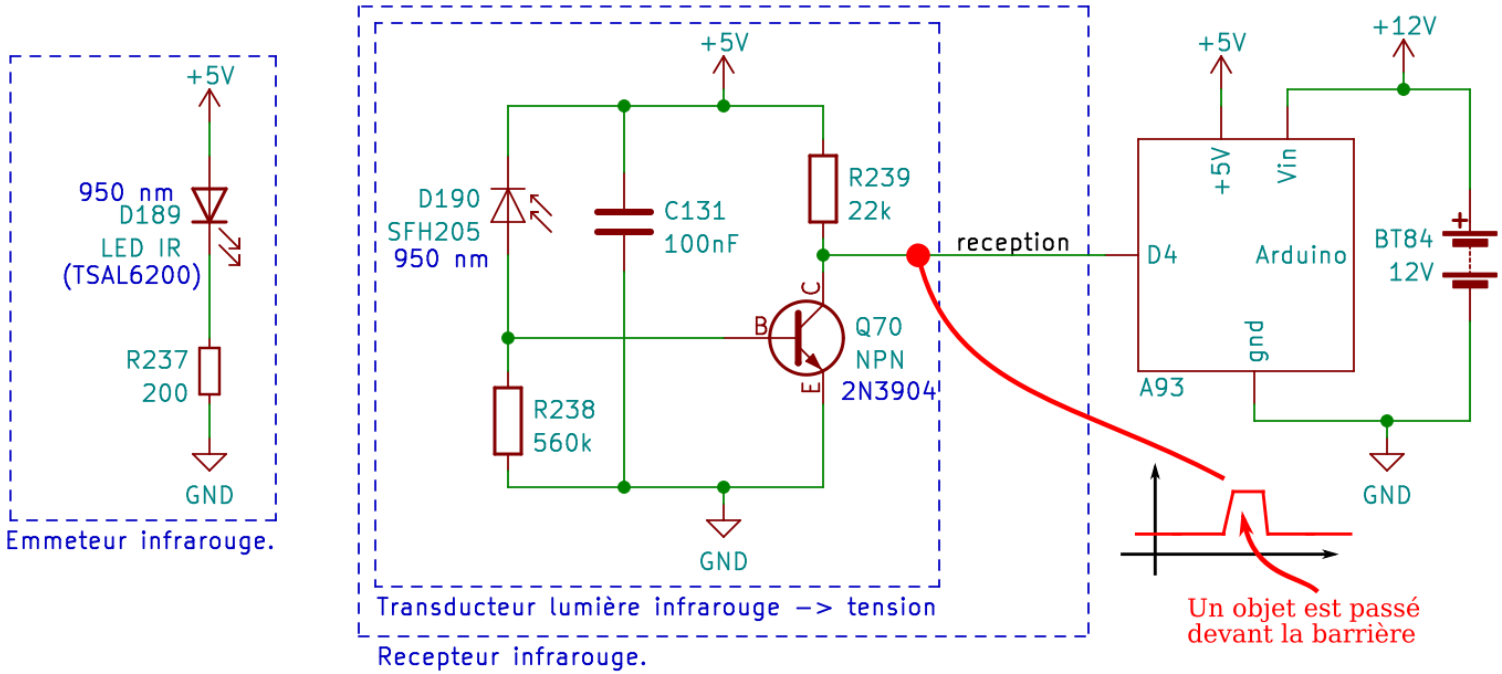
Les photodiodes s'utilisent "à l"envers". Pour un courant $I = 10\ \mu$ A on a

$$\begin{aligned} U &\approx R \cdot I = R \cdot S_\lambda \cdot P_{light} \\ &\approx 100k \times 10\mu A = 1.0V \end{aligned}$$

Comme les courants sont très faibles, il est obligatoire d'utiliser des montages plus complexes comme ceux allant de la pages 173 aux la page 176.

(version longue)

Barrière infrarouge simple avec un transistror NPN 1/2



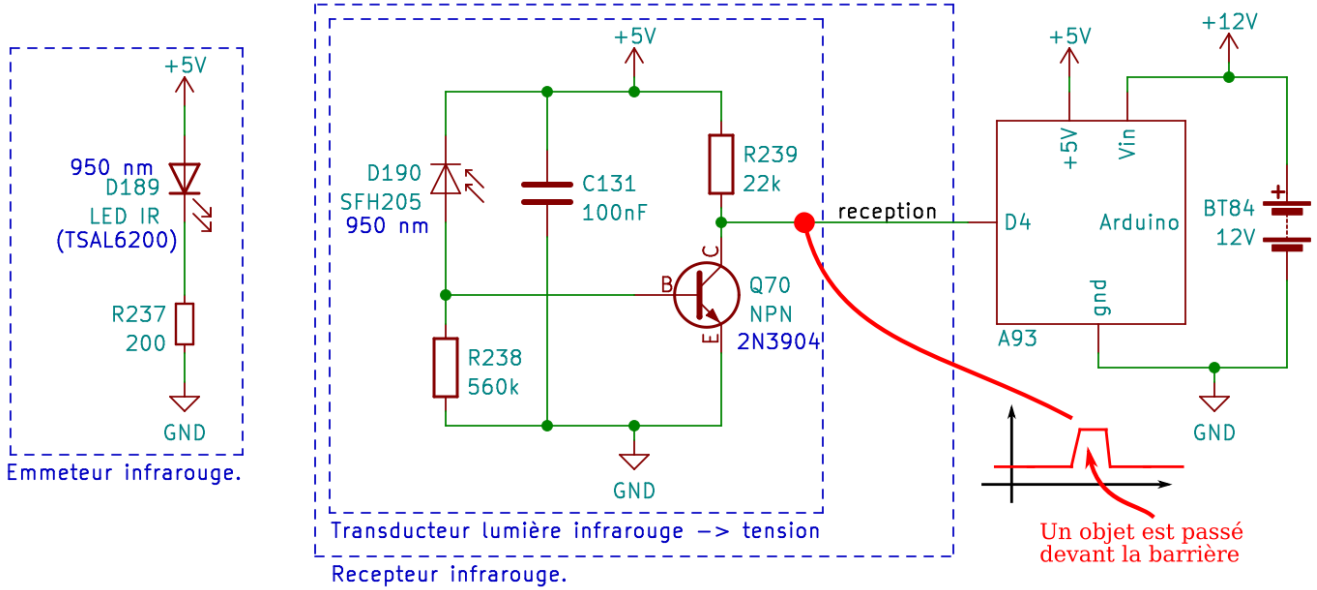
Pour faire fonctionner cette barrière il faut que :

- les longueurs d'onde des leds soient accordées
- les angles solides d'émission et réception soient les plus petits possibles, ce qui implique que les diodes soient précisement alignées.
- il n'y ai pas d'autres sources lumineuses parasites comme la lumière du jour (soleil)!
- la lumière de la source ne se reflete pas sur des parois parasités.

```
const int barrier_ir_pin = 4;

void setup(){
  Serial.begin(9600);
  pinMode(barrier_ir_pin, INPUT);
}

void loop(){
  int presence = digitalRead(barrier_ir_pin);
  Serial.print("Barriere IR state : ");
  Serial.println(presence);
  delay(1000);
}
```

(version longue)

Barrière infrarouge simple avec un transistror NPN 2/2

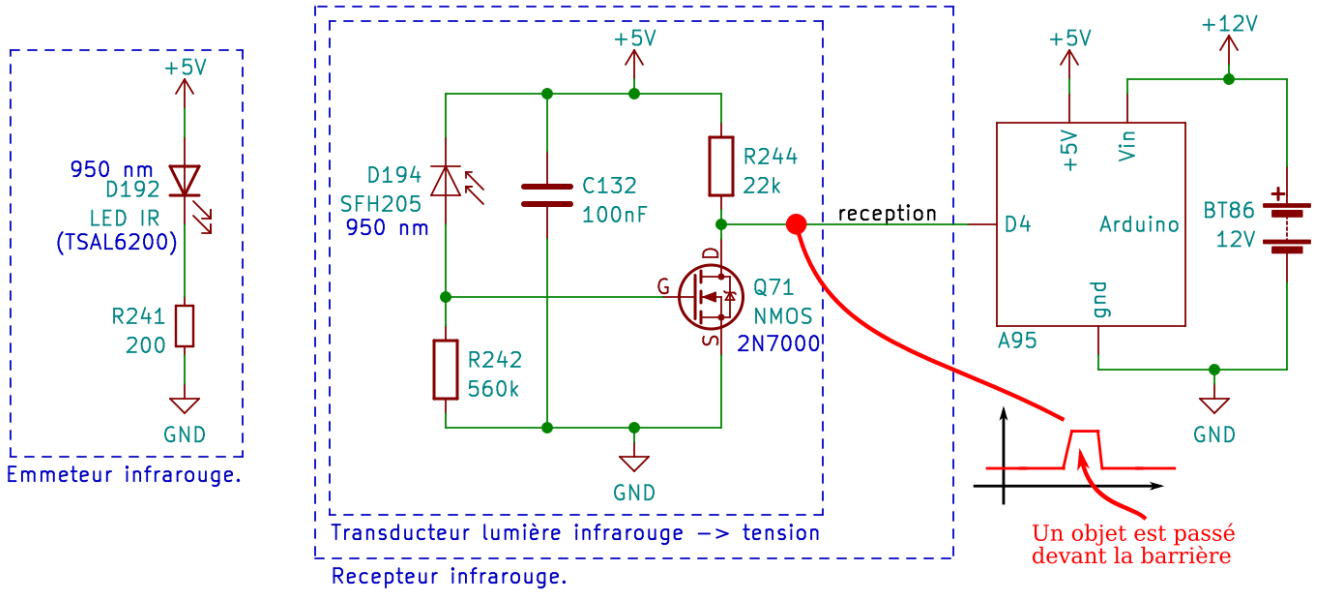


Explication du schéma :

Lorsque la led réceptrice D190 recoit de la lumière, elle emet un courant de $10\mu A$. Si la resistance recoit ce courant, la tension à ses bornes seraient de $220k\Omega \times 10\mu A \geq 0.6V$, ce n'est pas possible car le transistor Q70 se polarise et limite la tension entre B et E à 0.6V. Le courant passant dans la résistance R238 vaut donc $\approx 0.6V/560k\Omega = 1\mu A$. Ainsi, le courant qui circule dans la base du transistor est $10-1=9\mu A$. Comme le transistor 2N3904 amplifie ce courant d'un facteur minimal de 30, le courant du collecteur, s'il n'est pas saturé, est alors supérieur à $30 \times 9\ \mu A = 270\ \mu A$. Or, la résistance R239 est choisie pour provoquer la saturation du transistor et obtenir une une chute de tension de $5 - V_{CE,saturation} = 5 - .3 = 4.7V$, à savoir $270\mu A \times 22k\Omega \geq 4.7V$. La tension de la broche "reception" de l'arduino vaut alors $V_{CE,saturation} = 0.3$ V.

Si la led D190 ne reçoit pas de lumière, la tension $V_{BE} = 0$ pour le transistor. Il est alors ouvert et la broche "reception" de l'arduino est tiré à +5V par la résistance de pull-up R239 (version longue)

Barrière infrarouge simple avec un Mosfet canal N

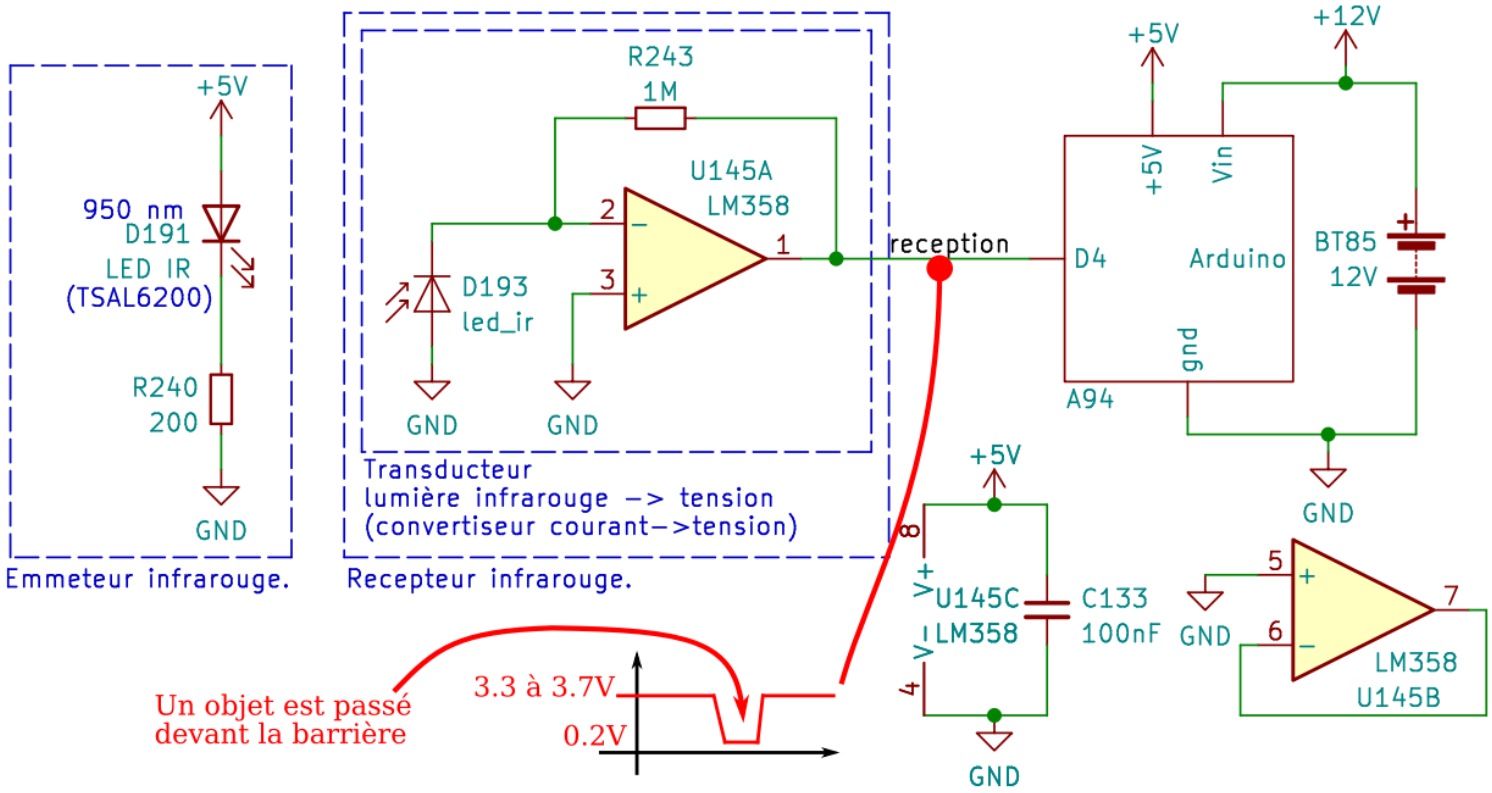


Le même montage fonctionne avec un MOSFET Canal N.

Lorsque la led réceptrice D194 recoit de la lumière, elle emet un courant de $10\mu A$. Comme la résistance R242 recoit ce courant, et que la photodiode D194 ne peut pas inverser sa polarité, la tension au borne de R244 vaut $min(5V, 560k\Omega \times 10\mu A) = 5V$ qui est aussi la tension V_{GS} entre la source et la porte du transitor Q71. Le transistor Q71 se ferme et la broche "reception" est à 0V.

Si la led D194 ne reçoit pas de lumière, la tension du $V_{GS} = 0V$ pour le transistor. Le transistor est alors ouvert et la broche "reception" de l'arduino est à +5V grâce à la résistance de pull-up R244.

Barrière infrarouge avec un Amplificateur Opérationel



Selon la led IR et la photodiode, utilisez une résistance de $100k\Omega$, $1M\Omega$ ou $10M\Omega$ pour R243.

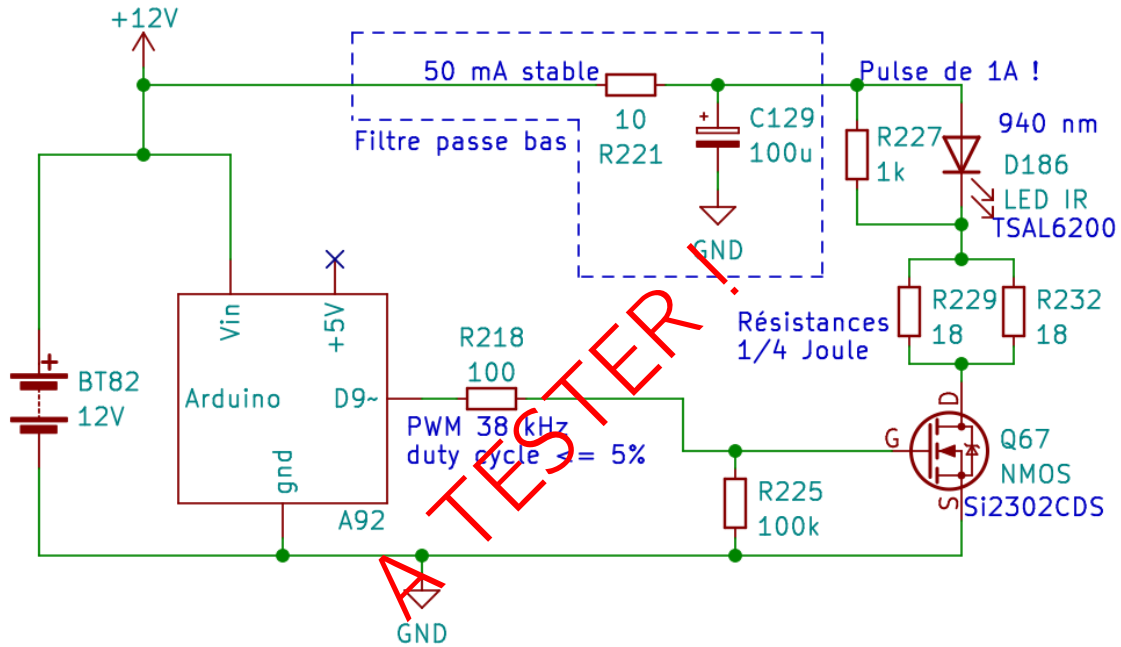
Si la resistance est suffisament grande, le signal "reception" est saturée à 0 ou 3V3 et est digital. Dans le cas contraire, la tension est proportionnelle à la puissance lumineuse reçue par la photdiode ce qui peut aussi s'avérer utile.

L'étude des AOP dépasse le cadre de ce cours. Pour plus d'informations sur le fonctionnement de ce circuit, vous pouvez consulter les ressources suivantes :
– The Art of Electronics; Paul Horowitz et Winfield Hill; 3th Edition, 2015; Cambridge Press; section 4.3.1-C page 233, section 4.6.3 page 265 et section 12.6.1 page 841.
– The Art of Electronics: The X-Chapters; Paul Horowitz et Winfield Hill; 2020; Cambridge Press; le chapitre 4x.3 page 283.

(version longue)

Booster les pulses de la led émmetrice

Pour faire une barrière infrarouge sur une longue distance, vous pouvez booster les pulses de la led émetrices de 20 mA à 1 A en utilisant le schéma suivant pour la led IR :



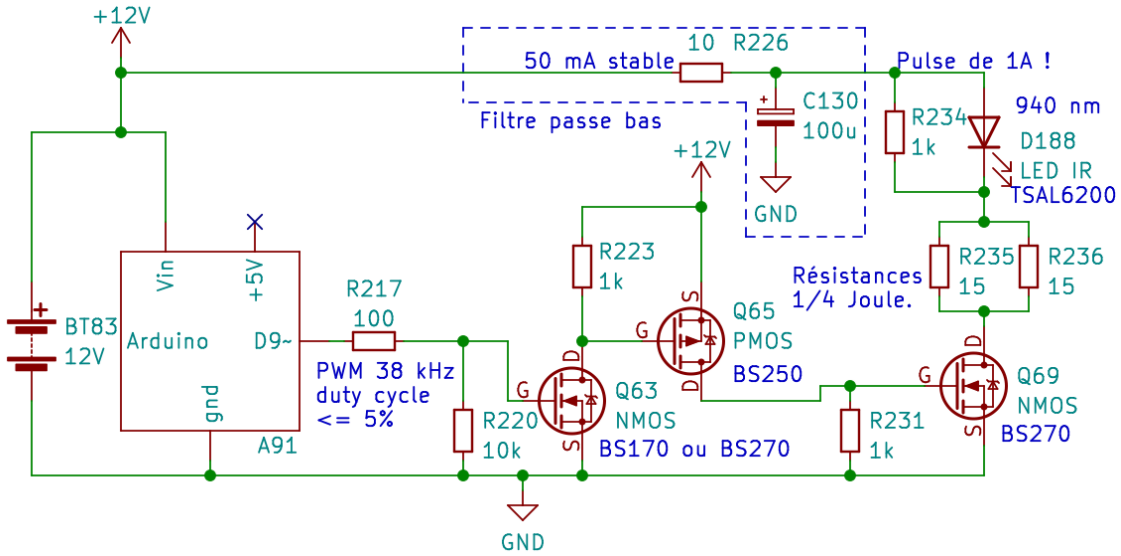
ATTENTION : Risque de casse matériel !

Si vous ne voulez pas brûler la diode D186, les résistances R229 et R232 ou le transistor Q67, il faut quel la broche 9 soit, au démarrage, à 0V et que la pwm ne dépasse jamais 5% pour des fréquences supérieures à 10 kHz (donc 38 kHz dans notre cas).

Simulation du circuit avec Falstad

Booster les pulses avec des mosfets "classiques"

Le mosfet du schéma précédent possède une résistance $R_{ds,on}$ très faible avec un tension de grille seuil très faible. Ce n'est pas le cas des mosfets plus classiques comme le BS170 et BS270 où il faut au moins 6V sur la grille pour réduire suffisament la résistance R_{ds} qui est de 3 ohms. Voici le schéma qu'il faut utiliser si vous avez ces mosfets:



ATTENTION : Risque de casse matériel !

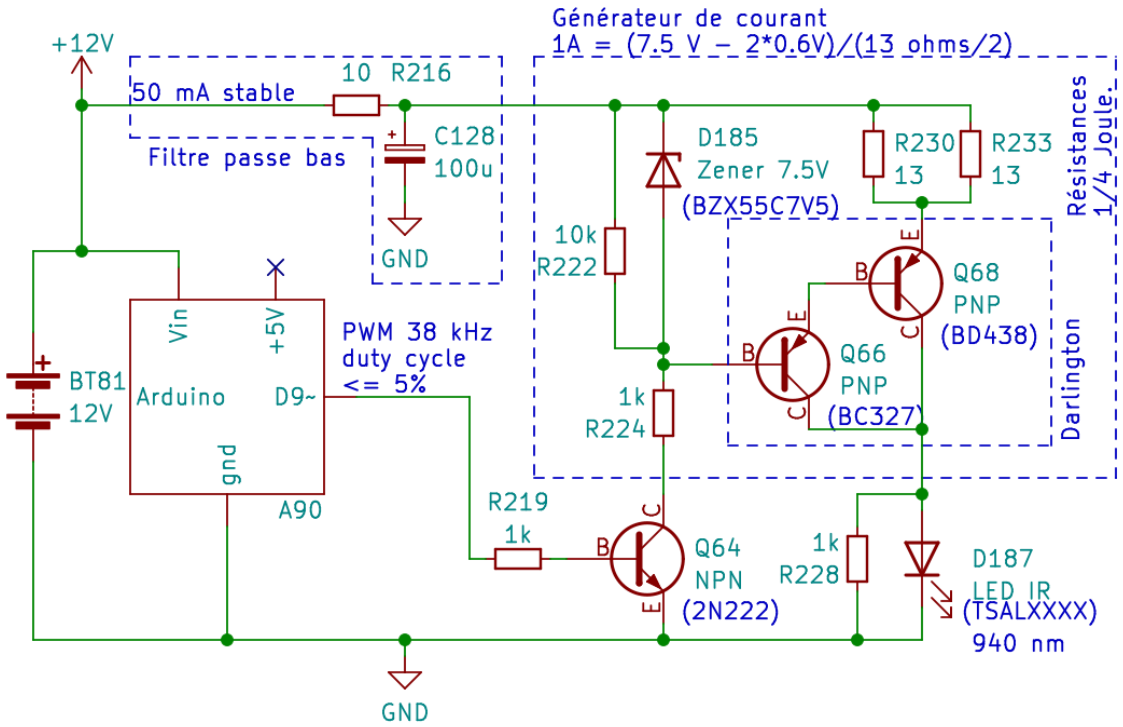
La valeur des résistance R223 et R231 doivent être de 1 $k\Omega$ car elle servent à décharger les capacités parasites des mosfets. Avec des valeurs plus grande, la pwm obtenu dépasse excessivement 5% ce qui risque de dégrader leds, résistances et transistors.

Si vous ne voulez pas brûler la diode D188, les résistances R235 et R236 ou le transistor Q69, il faut quel la broche 9 soit, au démarrage, à 0V et que la pwm ne dépasse jamais 5% pour des fréquence supérieures à 10 kHz (donc 38 kHz dans notre cas).

Simulation du circuit et sa version avec les capacités parasites des transistors et des diodes. (version longue)

Booster les pulses avec des transistors PNP et NPN

Ce schéma est basé sur un générateur de courant 1 A réalisé à l'aide de la diode zener 7.3V qui impose une tension de $7.3V - 2 \times 0.6V$ en parallèle aux resistances de 13 ohms. La résistance R260 est une résistance de pull-up qui sert à vider les capacités parasités et fermer rapidement le transistor darlington. La résistance R264 sert à décharger la capacité parasite de la diode.



ATTENTION : Risque de casse matériel !

Si vous ne voulez pas brûler la diode D187, les résistances R230 et R233 ou le transistor Q68, il faut quel la broche 9 soit, au démarrage, à 0V et que la pwm ne dépasse jamais 5% pour des fréquence supérieures à 10 kHz (donc 38 kHz dans notre cas).

Simulation du circuit et sa version avec les capacités parasites des transistors et des diodes. (version longue)

Choix de quelques associations de led IR et de photodiodes

Voici quelques exemples d'associations entre une photodiode et une led IR compatibles :

photodiode	diode
BPV22F BPV23F SFH 213 FA SFH 203 FA SFH 205 FA SFH 203 PFA	TSALXXXX SFH 454X
BPV22NF BPV23NF	TSFFXXXX TSHFXXXX

Voici quelques associations entre un émetteur IR avec demodulation et une led IR compatibles :

émetteur IR avec démodulation	diode
TSOPXXXXX HS0038B3VM VS1838B	TSALXXXX SFH 454X

Remplacez les X par les numéros qui correspondent à vos besoins. Consultez les tables des pages suivantes pour selectionner les diodes et photodiodes qui vous arrangent.

Consultez le catalogue de Vishay pour plus de choix (consulté le 29/09/2024).

Photodiodes Infrarouges

Tension directe : 1.3V

nom	marque *1	dimensions (mm)	angle (deg.)	longueur d'onde (nm)		photo-courant (μA)	temps de monté/ descente (ns)	aire sensitive (mm^2)	capacité (pF)	réact-ance (A/W)
SFH 229 FA	a	3	±15	900	740···1100	11	6000	0.31	12	
SFH 213 FA	a	5	±10	900	750···1100	42	5	1	11	0.65
SFH 203 FA	a	5	±20	900	750···1100	25	5	1	11	0.62
SFH 205 FA	a	5	±60	900	740···1100	56	20	7.02	72	0.62
SFH 235 FA	a	3	±65	900	740···1120	22	20	7.02	72	0.65
SFH 203 PFA	a	5	±75	900	750···1100	3	5	1	11	0.62
BPV10NF	v	5	±20	940	780···1050	60	80		11	0.55
BPV22NF	v	4.5x5x6	±60	940	790···1050	85	100	7.5	70	0.6
BPV23NF	v	4.5x5x6	±60	940	790···1050	65	70	4.4	48	0.6
BPW82	v	5x4x6.8	±65	950	790···1050	38	100	7.5	70	
BPW83	v	5x3x6.4	±65	950	790···1050	38	100	7.5	70	
BPV09NF	v	5	±22	940	780···1050	55	80/60		11	
BPV22F	v	4.5x5x6	±60	950	870···1050	80	100	7.5	70	0.6
BPV23F	v	4.5x5x6	±60	950	870···1050	60	70	4.4	48	0.6
BPW41N	v	5x4x6.8	±65	950	870···1050	38	100	7.5	70	
SFH 229	a	3	±15	860	380···1100	14	1000	0.31	12	
BPV10	v	5	±20	920	380···1100	65	80/60		11	0.55
SFH 213	a	5	±10	850	400···1100	125	5	1	11	0.65
SFH 203	a	5	±20	850	400···1100	80	5	1	11	0.62
SFH 203P	a	5	±75	850	400···1100	9.3	5	1	11	0.62
SFH 206 K	a	5	±60	920	420···1120	80	20	1	72	0.62
BPW46	v	5x3x6.4	±65	900	430···1100	47	100	7.5	70	
BPW24R	v	4.7	±12	940	610···1040	55	80/60	0.88	11	

*1: "a" pour ams-OSRAM, "v" pour Vishay.

(version longue)

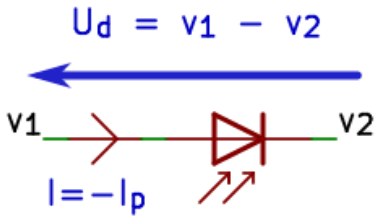
Led Infrarouge

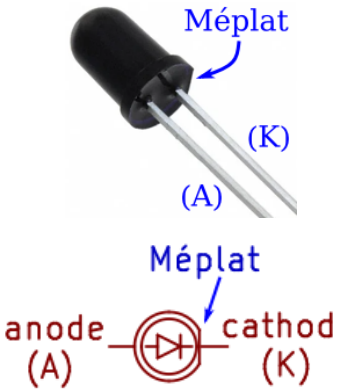
nom	marque *1	diamètre (mm)	longueur d'onde (nm)	angle (deg)	Intensité angulaire pour 1A (mW/sr)	temps de montée/ descente (ns)	tension directe (V)	tension directe à 1A (V)	courant direct (mA)	courant pulsé (A)	puissance (mW)
SFH 4550	a	5	860	±3	8500	12	1.5-1.7	2.4-2.9	100	1	180
SFH 4554	a	5	860	±10	2250	12	1.7-1.9	3.6-4.5	100	1	200
SFH 4555	a	5	860	±5	4400	12	1.5-1.7	2.4-2.9	100	1	180
SFH 4556	a	5	860	±20	1150	12	1.5-1.7	2.4-2.9	100	1	180
SFH 4557	a	5	860	±30	450	12	1.5-1.7	2.4-2.9	100	1	180
TSFF5210*2	v	5	870	±10	1800	15	1.5-1.8	2.3-3	100	1	180
TSFF5410*2	v	5	870	±22	700	15	1.5-1.8	2.3-3	100	1	180
TSFF6210*2	v	5	870	±10	1800	15	1.5-1.8	2.3-3	100	1	180
TSFF6410*2	v	5	870	±22	700	15	1.5-1.8	2.3-3	100	1	180
TSHF5210	v	5	890	±8	2700	10	1.5-1.7	3	100	1	170
TSHF5410	v	5	890	±27	528	10	1.5-1.7	3	100	1	170
TSHF6210	v	5	890	±8	2700	10	1.5-1.7	3	100	1	170
TSHF6410	v	5	890	±27	528	10	1.5-1.7	3	100	1	170
TSAL6100	v	5	940	±10	1450	15	1.35-1.6	2.2-3	100	1	160
TSAL6200	v	5	940	±17	600	15	1.35-1.6	2.2-3	100	1	160
TSAL6400	v	5	940	±25	420	15	1.35-1.6	2.2-3	100	1	160
TSAL4400	v	3	940	±25	290	15	1.35-1.6	2.2-3	100	1	160
SFH 4544	a	5	950	±10	2300	12	1.6-1.8	3.6-4.5	100	1	200
SFH 4545	a	5	950	±5	4200	12	1.5-1.7	2.3-2.9	100	1	180
SFH 4546	a	5	950	±20	1000	12	1.5-1.7	2.3-2.9	100	1	180
SFH 4547	a	5	950	±30	375	12	1.5-1.7	2.3-2.9	100	1	180

*1: "a" pour ams-OSRAM, "v" pour Vishay; *2: ce composant n'est plus produit.

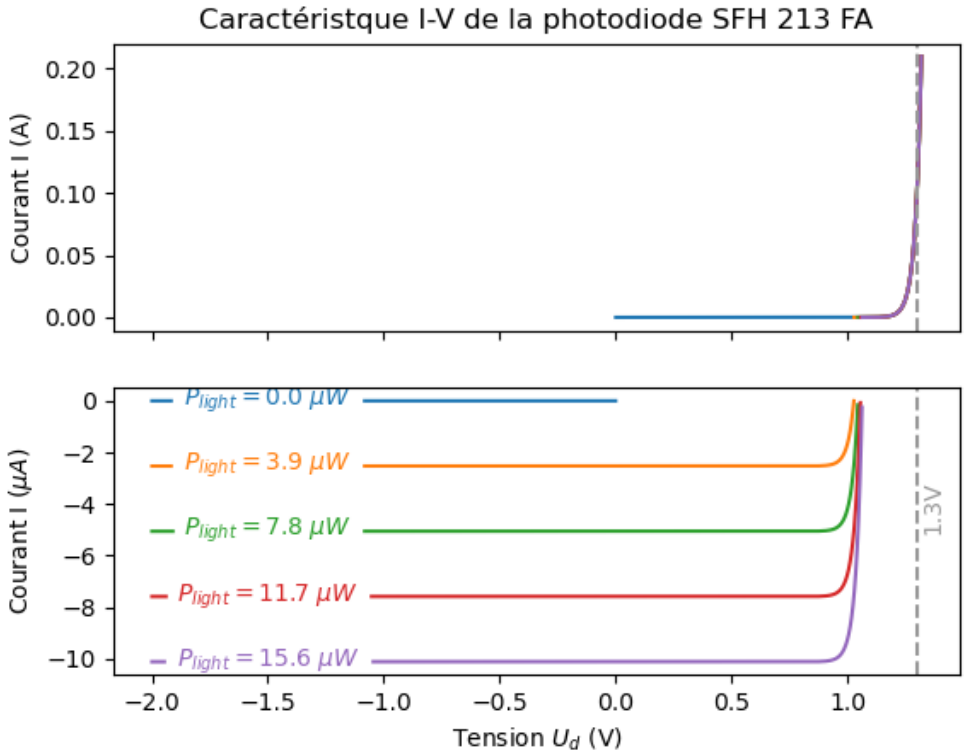
(version longue)

Modèle complet des photodiodes





vue de dessus



Lorsque l'on polarise une photodiode en dessous de $0.7V$, elle se comporte comme un générateur de très faible courant ($\leq 100\ \mu A$) commandé linéairement par la puissance lumineuse reçue.

Le modèle complet de la photodiode est en fait :

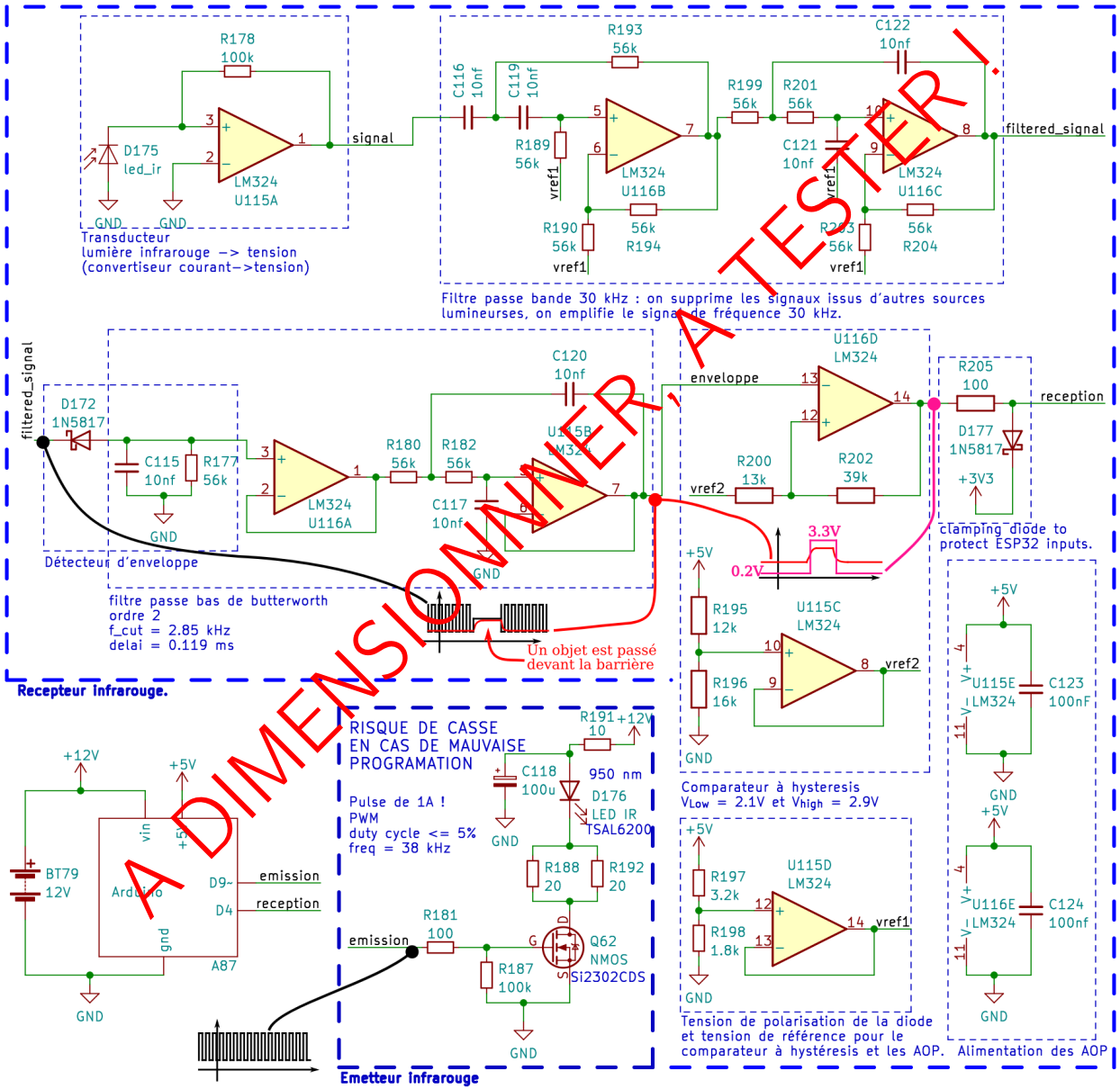
$$I = I_{sat} \cdot \left(e^{\frac{q \cdot U_d}{k_B \cdot T}} - 1 \right) - I_p \qquad \text{avec} \qquad I_p = S_\lambda \times P_{light}$$

où S_λ est la réacitvité (A/W) et P_{light} à la puissance (W) lumineuse reçue., $q = -1.60 \times 10^{-19}$ C est la charge d'un électron, U_d la tension (V) au borne de la photodiode, I_{sat} est le courant de saturation (A), T la température (K) de la jonction et $k_B = 1.38 \times 10^{-23} J/K$ est la constante de Boltzman.

(version longue)

Barrière infrarouge avec détection de bursts à 200 kHz.

Cette barrière infrarouge n'est pas perturbée par les signaux des télécomandes.



(version longue)

Plan

- Les capteurs de position, de vitesse et d'inclinaison
- Les capteurs de luminosité
- Les capteurs infrarouges
- Les capteurs de pression mécanique
- Les capteurs de pression pneumatique (gaz, souffle)
- Les capteurs de température

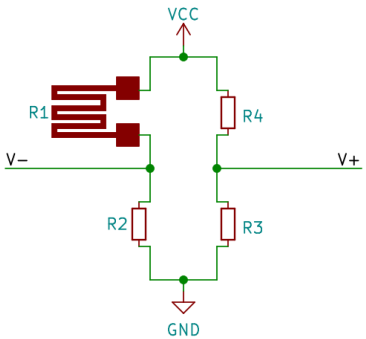
Les jauges résistives de déformation (jauges de contraite)

Jauge résistive de déformation

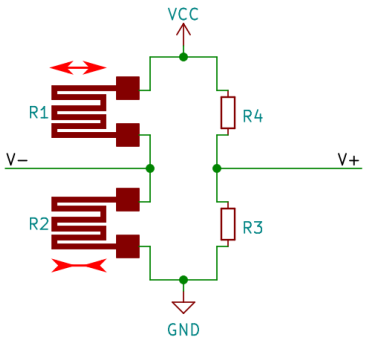
Les jauges sont des résistances variables faites à partir de fines piste flexibles. Leur résistance varie linéairement en fonction de la déformation des pistes. Une fois la jauge collée sur la surface d'une pièce, elle mesure la déformation de la pièce. Selon la position de la jauge et la géométrie de la surface de la pièce, il est possible de mesurer des étirements, des contraction, des torsions, etc...

La résistance de la jauge vaut $R_{jauge} = \left(1 + \alpha \frac{\Delta}{L}\right) R$ où R est la résistance de la jauge au repos (Ω), Δ la longueur d'élongation de la jauge (m), L sa ongueur au repos (m) et α un coefficient (sans unité).

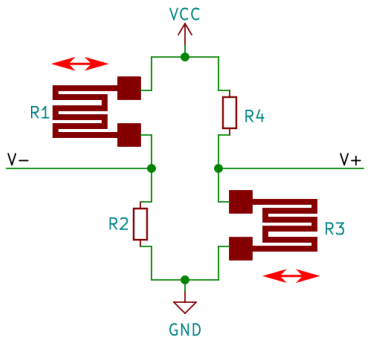
Les jauges s'utilisent dans des ponts de Whestone. Voici différentes configurations classiques :



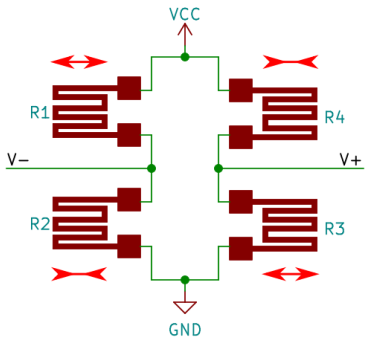
Quart de pont



Demi pont



Diagonale



Pont complet

↔ : Jauge placée sur une face en contraction/élongation

⋈ : Jauge placée sur la face opposé en élongation/contraction

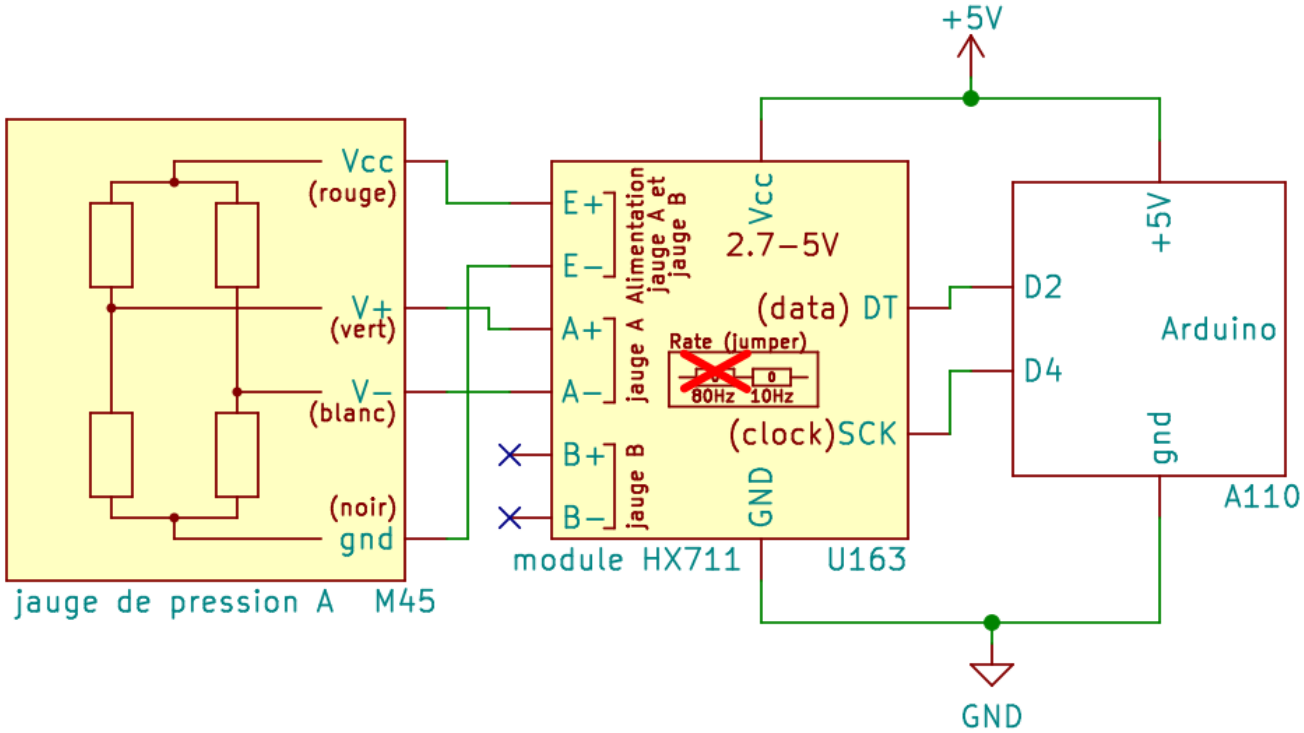
Pour mesurer la déformation de la pièce on mesure $V_+ - V_-$. Par exemple, pour le pont complet on obtient :

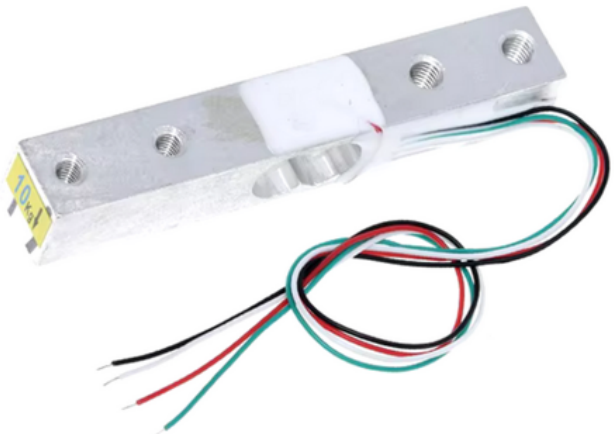
$$V_+ - V_- = V_{cc}\left(\frac{R_2}{R_1+R_2} - \frac{R3}{R_3+R_4}\right) = V_{cc}\left(\frac{R(1+\alpha\frac{\Delta}{L})}{R(1+\alpha\frac{\Delta}{L})+R(1-\alpha\frac{\Delta}{L})} - \frac{R(1-\alpha\frac{\Delta}{L})}{R(1-\alpha\frac{\Delta}{L})+R(1+\alpha\frac{\Delta}{L})}\right) = \alpha\frac{\Delta}{L}V_{cc}$$

(version longue)

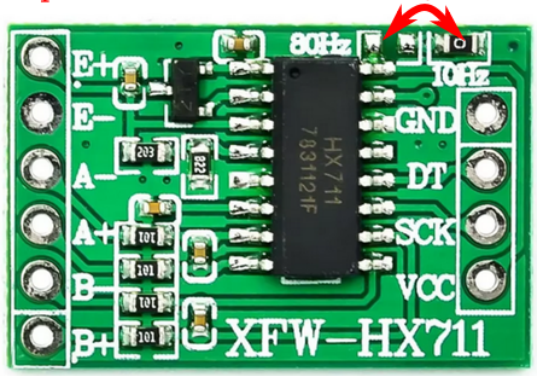
Les jauges de contrainte avec le module HX711

On place un pont de whestone consituté de jauges de contrainte sur une barre. Lorsqu'elle subie une torsion, les resistances du pont changent et la tension $V_+ - V-$ est proportionelle à la pression qui s'exerce sur la barre. Le module HX711 sert à amplifier et mesurer cette tension.





Deplacer la resistance 0 Ohms pour échantillonner à 80/10 Hz.



ersion longue)

Lire sporadiquement une valeure avec le module HX711

Voici un progamme qui lit toutes les secondes la différence de tension d'un pont de Whestone.

Dès que le composant HX711 a fini de transférer une donnée, il réalise une nouvelle mesure et la stock en attendant de l'envoyer. Ainsi, si l'on fait des mesures sporadiquement, il faut toujours faire 2 mesures successves, l'une pour jeter la mesure périmée, et la suivante pour avoir une mesure récente.

Dans la bibliothèqe Adafruit HX71 que l'on utilise, la fonction readChannelBlocking fait toujours ces deux mesures.

Ce programme utilise la bibliothèque Adafruit HX711

code/capteurs/pression/hx711_mesure_sporadique.cpp

```
#include "Adafruit_HX711.h"

#define GAIN_A CHAN_A_GAIN_128 // CHAN_A_GAIN_64 is also possible.
#define GAIN_B CHAN_B_GAIN_32 // No other value are avalaible for chan. B
const uint8_t HX711_DATA_PIN = 2; // Can use any pins!
const uint8_t HX711_CLOCK_PIN = 4; // Can use any pins!
Adafruit_HX711 hx711(HX711_DATA_PIN, HX711_CLOCK_PIN);

void tare_channels(){
  for (uint8_t i=0; i<3; i++) {
    hx711.tareA(hx711.readChannelRaw(GAIN_A));
    hx711.tareA(hx711.readChannelRaw(GAIN_A));
    // hx711.tareB(hx711.readChannelRaw(GAIN_B));
    // hx711.tareB(hx711.readChannelRaw(GAIN_B));
  }
}
void setup() {
  Serial.begin(9600);
  hx711.begin();
  tare_channels();
}

void loop() {
  int32_t value_a = hx711.readChannelBlocking(GAIN_A);
  Serial.print("Channel A : ");
  Serial.println(value_a);
  // int32_t value_b = hx711.readChannelBlocking(GAIN_A);
  // Serial.print("Channel B : ");
  // Serial.println(value_b);
  delay(1000);
}
```

(version longue)

Programme utilisant le module HX711 2/2

Voici un progamme qui lit en flux tendu la différence de tension d'un pont de Whestone.

Pour lire des valeurs en flux tendu, il faut utiliser la fonction : reachChannel. Cette fonction ne demande qu'une seule valeur. Il faut donc exécuter cette fonction dès que possible sinon, la donnée envoyée par le composant devient périmée (cf. slide précédante).

En flus tendu, la fréquence de rafraichissement est d'environ 10 Hz (plutôt 10.74 Hz).

Vous pouvez l'augmenter à 80 Hz en déplacant une résistance. Ceci est expliqué dans les slides de la page 155 et de la page 158

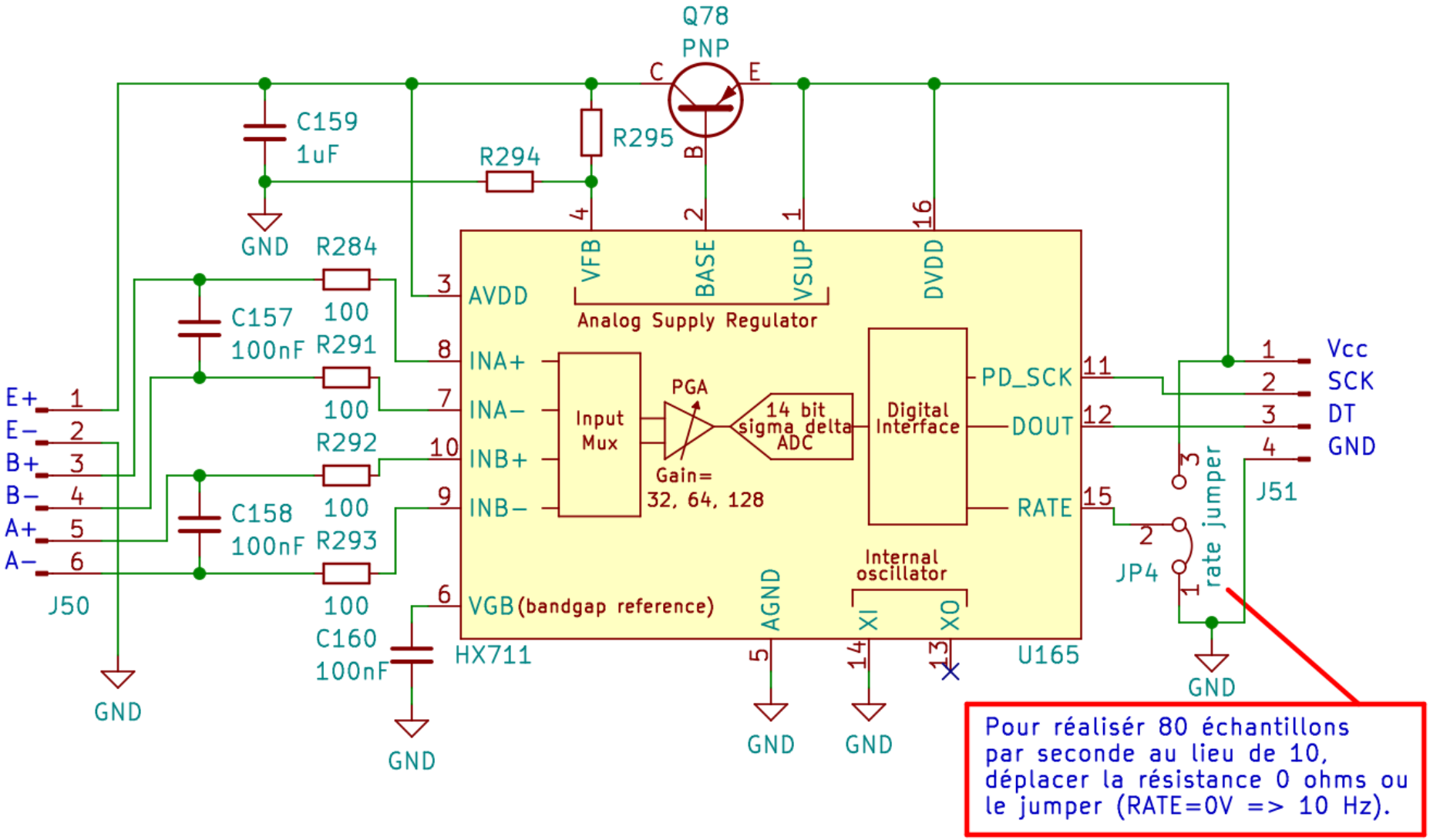
Ce programme utilise la bibliothèque Adafruit HX711

code/capteurs/pression/hx711_mesure_en_flux_tendu.cpp

```
#include "Adafruit_HX711.h"

#define GAIN_A CHAN_A_GAIN_128 // CHAN_A_GAIN_64 is also possible.
#define GAIN_B CHAN_B_GAIN_32 // No other value are avalaible for chan. B
const uint8_t HX711_DATA_PIN = 2; // Can use any pins!
const uint8_t HX711_CLOCK_PIN = 4; // Can use any pins!
Adafruit_HX711 hx711(HX711_DATA_PIN, HX711_CLOCK_PIN);

void tare_channels(){
  for (uint8_t i=0; i<3; i++) {
    hx711.tareA(hx711.readChannelRaw(GAIN_A));
    hx711.tareA(hx711.readChannelRaw(GAIN_A));
    // hx711.tareB(hx711.readChannelRaw(GAIN_B));
    // hx711.tareB(hx711.readChannelRaw(GAIN_B));
  }
}
void setup() {
  Serial.begin(9600);
  hx711.begin();
  tare_channels();
}

void loop() {
  if( ! hx711.isBusy() ){
    int32_t value_a = hx711.readChannel(GAIN_A); // (blocking function)
    Serial.print("Channel A : ");
    Serial.println(value_a);
    // int32_t value_b = hx711.readChannel(GAIN_B);
    // Serial.print("Channel B : ");
    // Serial.println(value_b);
  }
}
```

(version longue)

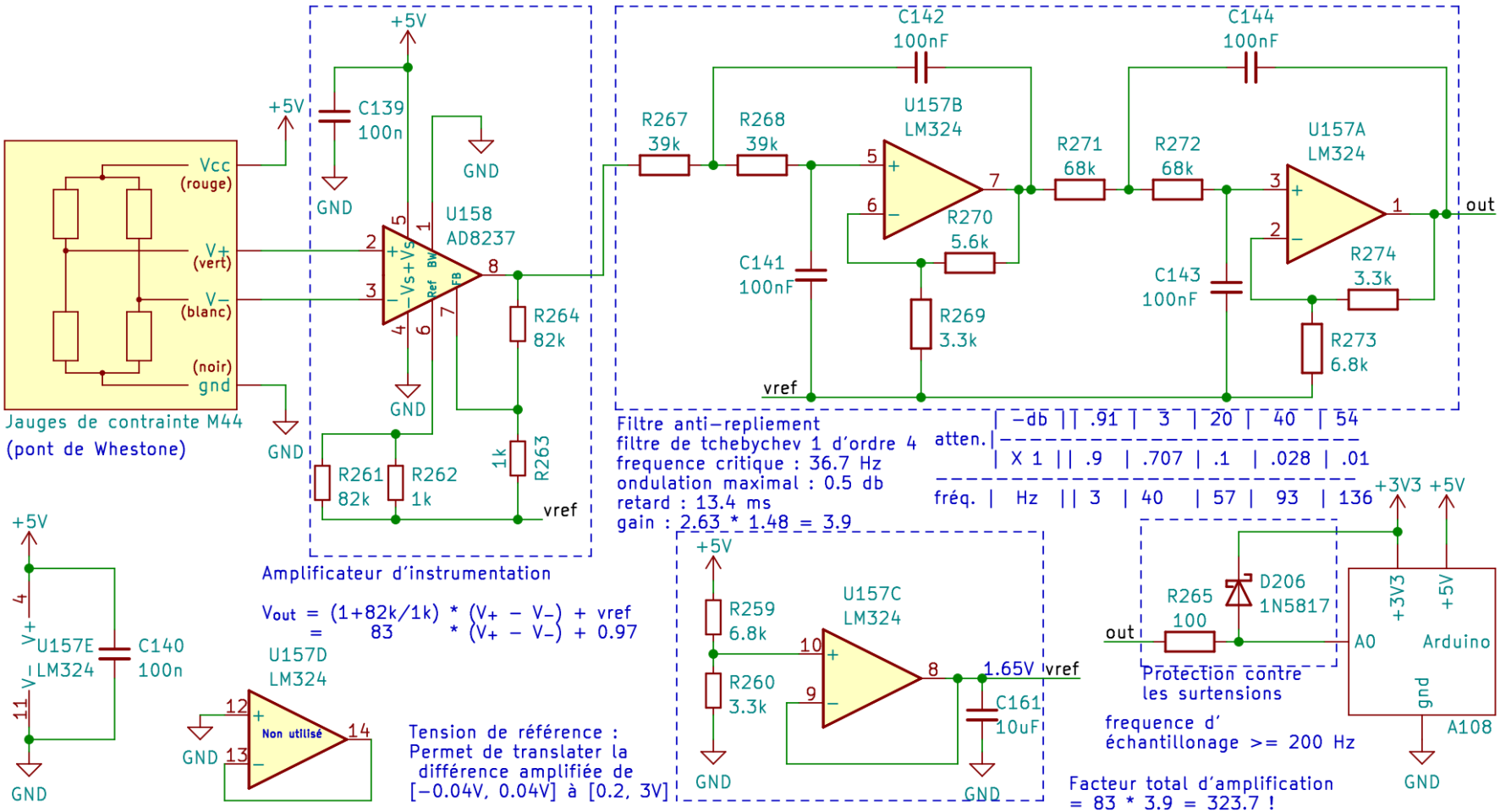
Schéma du module HX711



(version longue)

Les jauges avec un amplificateur d'instrumentation

Si vous devez réaliser des mesures avec des fréquence plus grandes, comme par exemple 200 Hz, vous devez amplifier la différence de tension au borne du pont de Wheatstone à l'aide d'une amplificateur d'instrumenation, puis filter le signal (cf. la section sur les filtres, page 13).



(version longue)

Programmer l'arduino pour les jauges utilisant un amplificateur d'instrumentation.

Pour mesure la pression mécanique, il suffit de réaliser une lecture analogique sur la broche A0 de l'arduino.

Il faudra prévoir une étape de calibration, pour connaître la tension de repos de votre jauge de contrainte. Vous pouvez aussi connecter la tension V_{ref} sur la borche A1, et mesurer A1 pour connaître la tension où $V_+ = V_-$.

Vous pouvez consulter la page 106 pour savoir comment réaliser naïvement une mesure analogique sur la broche A0.

Dans notre cas, il est nécessaire de filtrer digitalement le signal. Consultez la page 196 pour synthétiser un filtre numérique, et la page 200 pour l'implémenter.

Le fichier *code/capteur/pression/lecture_amplificateur_instumentation_esp32.cpp* montre un exemple d'implémentation d'un filtre numrique pour échanitilloner à 300 Hz, filtrer le signal filtre digital avec un filtre de chebychev 2, d'ordre 4, ayant une fréquence de coupure à -3dB de 32 Hz.

Plan

- Les capteurs de position, de vitesse et d'inclinaison
- Les capteurs de luminosité
- Les capteurs infrarouges
- Les capteurs de pression mécanique
- Les capteurs de pression pneumatique (gaz, souffle)
- Les capteurs de température

Présentation des capteurs de souffle

A FAIRE : Ecrire des slides pour expliquer les différents capteurs de pressions.

Le site my.avnet.com donne le fonctionnement des capteurs de pression.

Il y a 3 types de capteurs de pression :

- les gauges;
- les capteurs différentiels;
- les capteurs absolus.

Pour les capteurs de soufle, il faut trouver des capteurs capables de détecter des pression allant de 0 à 10 KPa. Par exemple, la série MP3V5010 propose des gauges ou des capteur différentiel pour détecter le souffle d'un humain.

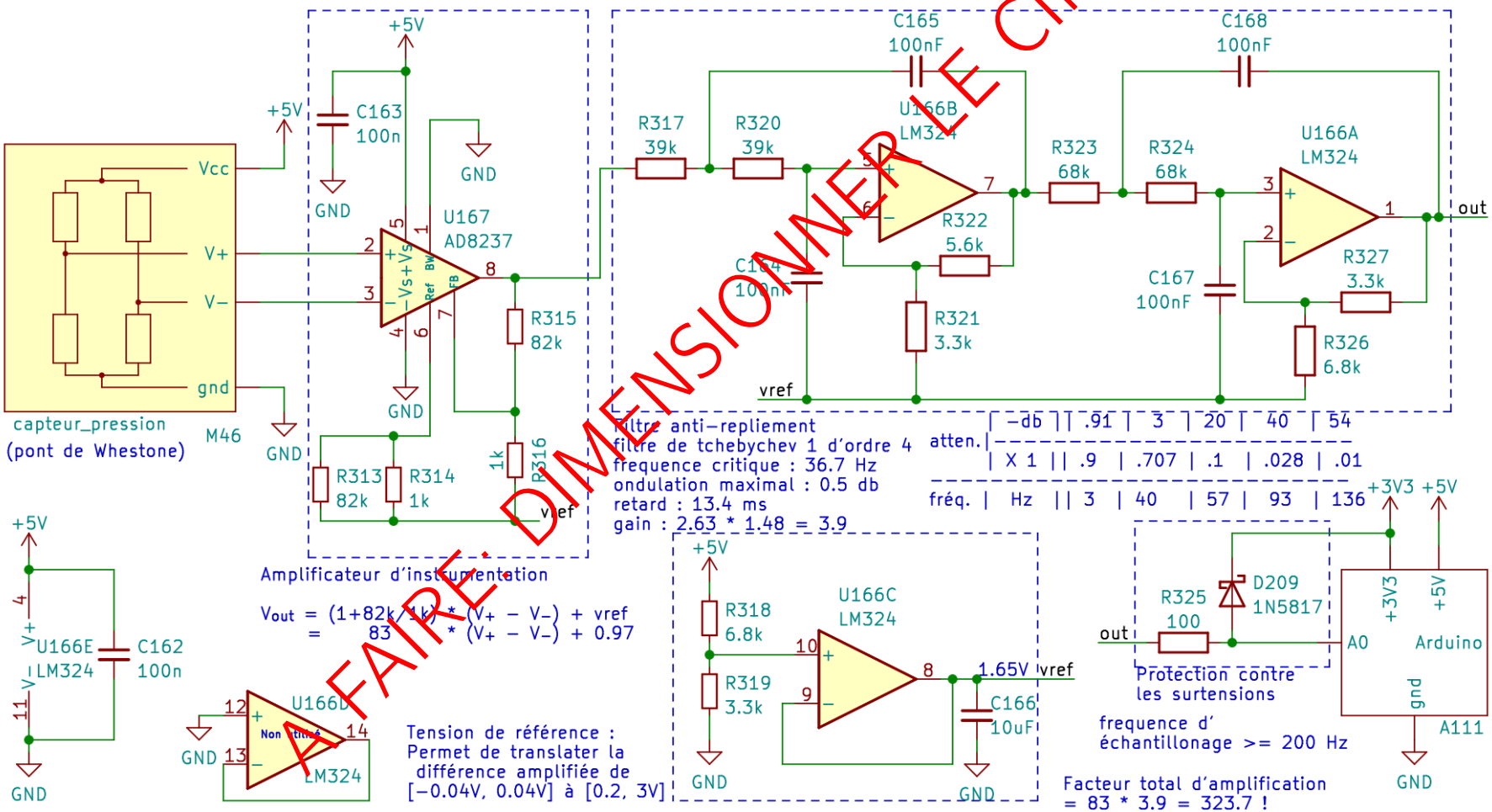
A FAIRE : Faire un tableau d'ordre de grandeur des pressions. A FAIRE : Faire un tableau de differentes références de composants.

Les capteurs de pressions avec circuit d'amplification intégré

A FAIRE : Traiter l'exemple de la série MP3V5010

Les capteurs de pressions sans circuit d'amplification

Quand le capteur de souffle est réduit à un pont de Wheatstone sans circuit d'amplification, il faut amplifier la différence de tension du pont à l'aide d'un amplificateur d'instrumenation, puis filter le résultat. Voici un exemple de schéma :



(version longue)

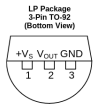
programmer l'arduino pour le capteur de souffle (0-10 kpa)

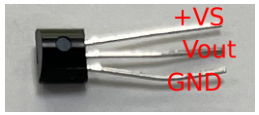
A FAIRE :

Plan

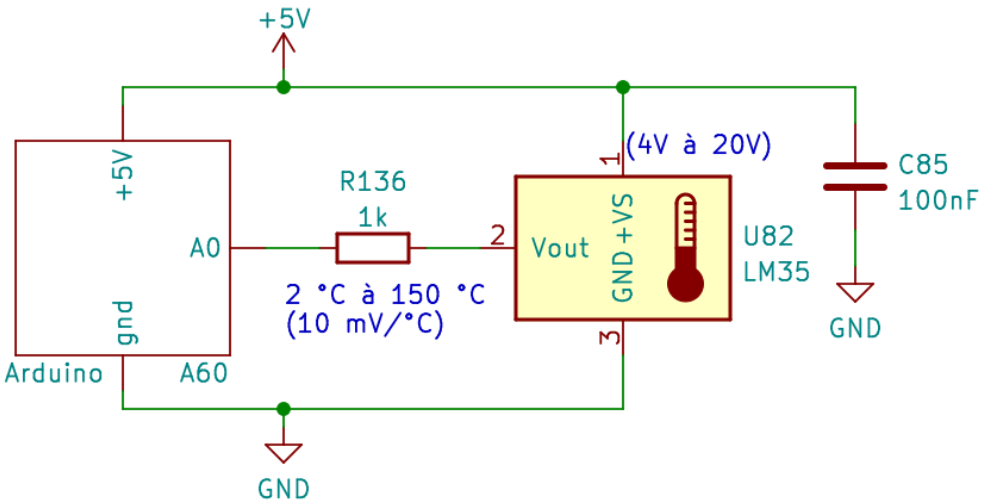
Le capteur de température LM35

La tension de sortie du LM35 est égale à $10mV/^oC \times T$ où est T est la température ambiante.





LM35



Avec ce montage, on ne peut mesurer que des températures entre 2^oC et 150^oC. Pour mesurer des température entre -55^oC et 150^oC consultez le document technique du composant.

Le condensateur C85 est un condensateur de découplage et sert de stockage d'énergie (un "chateau d'eau" pour les électrons) pour répondre à la demande en énergie du composant LM35. Le condensateur doit être placé au plus près de la patte +VS du LM35.

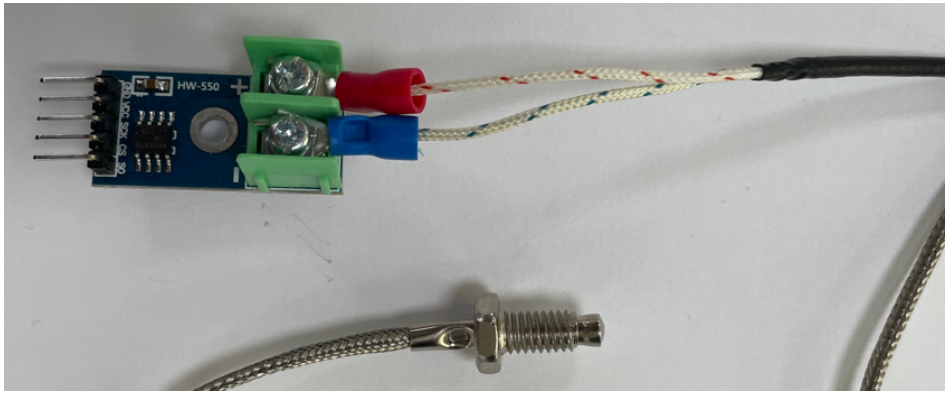
On utilise l'entrée analogique A0 pour mesure la tension de sortie du LM35.

Programme pour le capteur LM35

```
const float tension_max_adc = 5; // Volt
const float gain_lm35 = 100; // degres / Volt - gain du LM35
const float valeur_maximale_adc = 1023; // La resolution de l'ADC est de 10 bits.
const float gain_total = tension_max_adc * gain_lm35 / valeur_maximale_adc;
const int temp_ref = 0; // degres celsus
const int temperature_sensor_pin = A0;

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

static float moyenne_glissante = 25;

void loop(){
  int valeur = analogRead(temperature_sensor_pin);
  float temperature = valeur * gain_total + temp_ref;

  moyenne_glissante = .9 * moyenne_glissante + 0.1 * temperature;

  Serial.println("---");
  Serial.print("Temp. : ");
  Serial.println(temperature);
  Serial.print("Moyenne glissante : ");
  Serial.println(moyenne_glissante);
  delay(500);
}
```

code/capteurs/temperature/capteur_temperature_LM35.cpp

Présentation du module MAX6675 et de sa sonde de température K



Module MAX6675 et sa sonde thermocouple de type K

Sonde thermocouple de type K

Schéma pour le module MAX6675

Le module MAX6675 prêt à l'emploi se commande par un protocole série SPI.

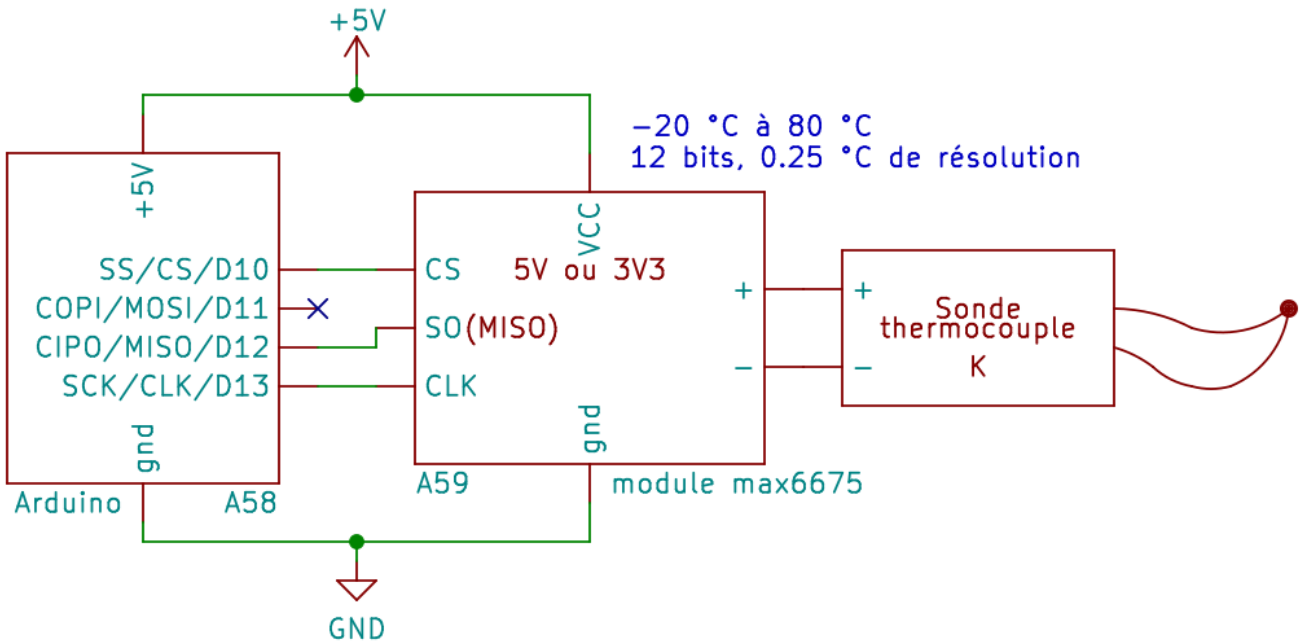
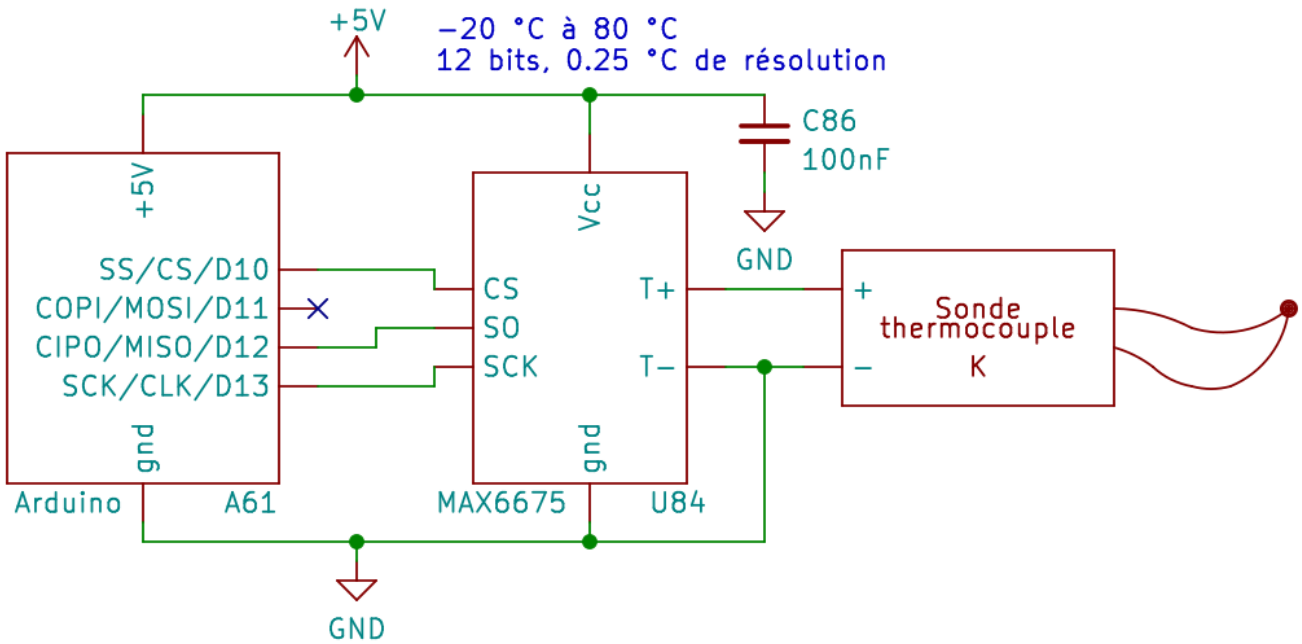


Schéma pour le composant MAX6675

Si vous utilisez directement le composant MAX6675, vous devez le monter ainsi (consulter son document technique):



Le condensateur C86 est un condensateur de découplage, il sert à mettre de l'énergie à disposition du composant MAX6675 il sert aussi à filtrer le bruit pour réduire le bruit de mesure.

Programme pour le module MAX6675

On utilise la bibliothèque "adafruit/MAX6675 library". (la bibliothèque "zhenek-kreker/MAX6675 with hardware SPI" ne fonctionne pas)

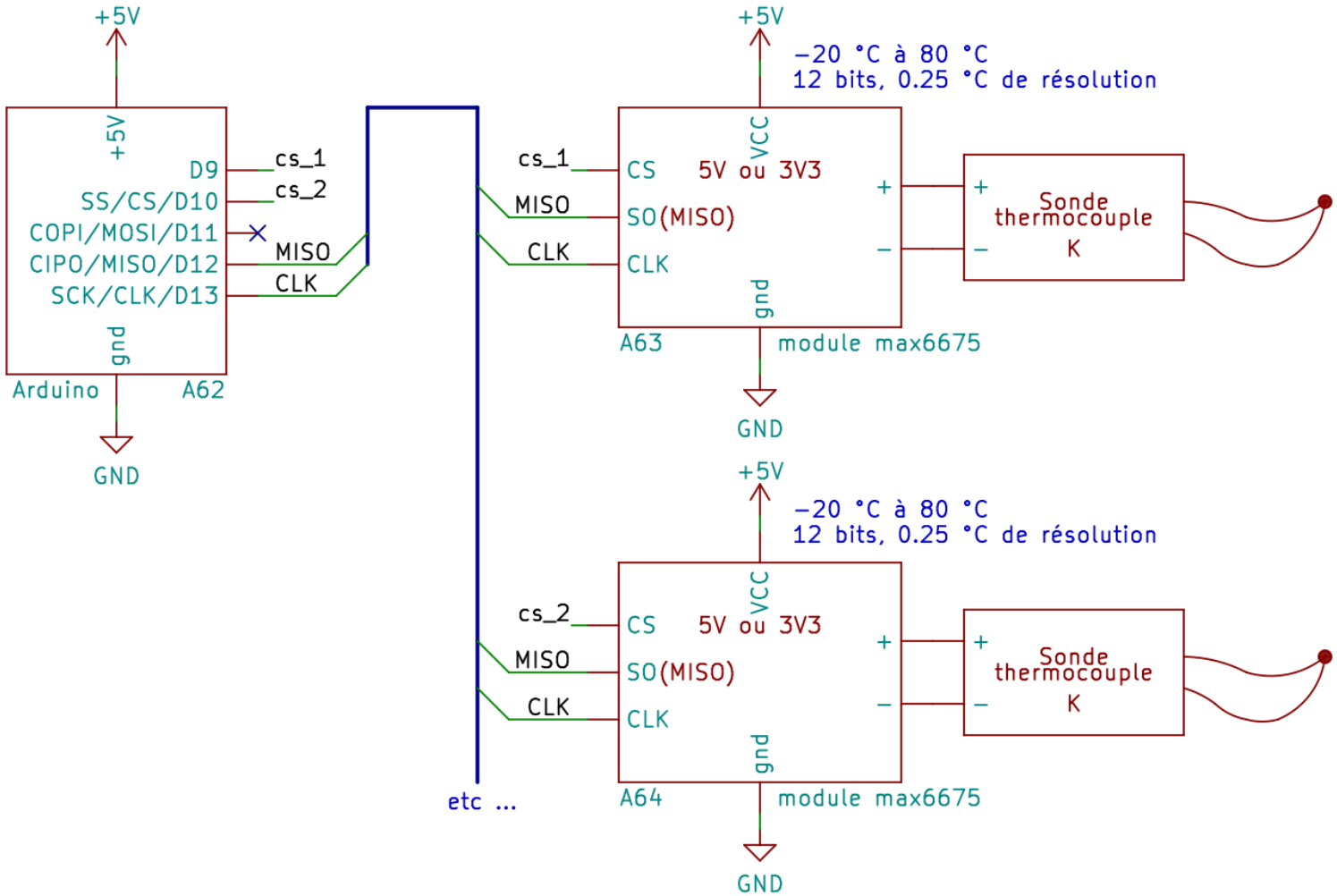
code/capteurs/temperature/capteur_temperature_max6675.cpp

```
#include <max6675.h>

int therm_CLK = 13;
int therm_MISO = 12;

int therm_CS = 10;
MAX6675 thermocouple(therm_CLK, therm_CS, therm_MISO);

void setup() {
  Serial.begin(9600);
  delay(500); // On Attends que le composant MAX6675 se stabilise.
}

void loop() {
   Serial.print("C = ");
   Serial.println(thermocouple.readCelsius());
   Serial.print("F = ");
   Serial.println(thermocouple.readFahrenheit());

   // Entre chaque mesure il faut attendre au moins 250 ms !
   delay(1000);
}
```

(version longue)

Schéma pour le module MAX6675

Grâce à la connection série SPI on peut connecteur plusieurs capteurs de température.



On relie les 3 broches SPI des compsants sur le même port SPI de la carte. La ligne bleue symbolise un bus qui contient 3 fils séparés, un fil qui relie toutes les broches MISO ensenble, un autre qui relie tous les broches MOSI ensemble et un dernier pour CLK. Ne reliez donc surtout pas MISO, MOSI et SCK ensemble !

(version longue)

Programme pour plusieurs modules MAX6675

On utilise la bibliothèque "adafruit/MAX6675 library". (la bibliothèque "zhenek-kreker/MAX6675 with hardware SPI" ne fonctionne pas)

```
#include <max6675.h>

int therm_CLK = 13;
int therm_MISO = 12;
int therm_CS_1 = 10;
MAX6675 thermocouple_1(therm_CLK, therm_CS_1, therm_MISO);
int therm_CS_2 = 9;
MAX6675 thermocouple_2(therm_CLK, therm_CS_2, therm_MISO);

void setup() {
  Serial.begin(9600);
  delay(500); // On attends que les composants MAX6675 se stabilisent.
}

void loop() {
   Serial.print("Thermo 1 : C = ");
   Serial.println(thermocouple_1.readCelsius());
   Serial.print("F = ");
   Serial.println(thermocouple_1.readFahrenheit());

   Serial.print("Thermo 2 : C = ");
   Serial.println(thermocouple_2.readCelsius());
   Serial.print("F = ");
   Serial.println(thermocouple_2.readFahrenheit());

   // Entre chaque mesure il faut attendre au moins 250 ms !
   delay(1000);
}
```

code/capteurs/temperature/capteur_temperature_plusieurs_max6675.cpp

(version longue)

Plan

Avertissement

Cette section est nettement plus difficile que les autres sections. Elle sort du cadre d'un cours dit : "Premier pas en électronique" et manque cruellement d'explications.

Cette section est ici pour aider tout ceux qui ont besoin de réduire le bruit de leurs mesures.

Cette section fera probablement l'objet d'un cours complet et plus détaillé à venir ... :)

Plan

Références sur les filtres

Cette section présente des méthodes pour concevoir les filtres analogiques et les filtres digtitaux. Leur fonctionnement théorique est rapidement présentée sans donner de justifications. Pour mieux comprendre comment fonctionne ces filtres et pour avoir une justification mathémathique rigoureuse des différents résultats utilisés, vous devez consultez les références suivantes.

Pour les filtres analogiques :

1. Électronique, Fondements et applications, J.-P. Pérez, C. Lagoute, J.-Y. Fourniols et S. Bouhours, 2ieme Édition, 2012, Dunod – Chapitre 10 : les filtres actifs.
2. The Art of Electronics, Paul Horowitz et Winfield Hill, 3th Edition, 2015, Cambridge Press – Chapitre 6 : Filters

Pour les filtres digitaux :

3. Understand digital signal processing, Richard G. Lyons, 3th Edition, 2011, Prentice Hall – Tout le livre. C'est un excellent livre à lire impérativement !

Pour les échantillonneurs :

4. Électronique, Fondements et applications, J.-P. Pérez, C. Lagoute, J.-Y. Fourniols et S. Bouhours, 2ieme Édition, 2012, Dunod – Chapitre 19 : Conversions analogique-numérique.
5. The Art of Electronics, Paul Horowitz et Winfield Hill, 3th Edition, 2015, Cambridge Press – Chapitre 13 : Digital meets Analog.
6. How to optimize the ADC accuracy in the STM32 MCUs, Application note AN2834 rev. 9, August 2023.

Pour des preuves détaillées sur les distributions et transformées de Fourrier :

7. Distribution theory and transform analysis, An introduction to generalized functions, with applications, JA.H. Zemanian, 1965, McGraw-Hill.

(version longue)

Plan

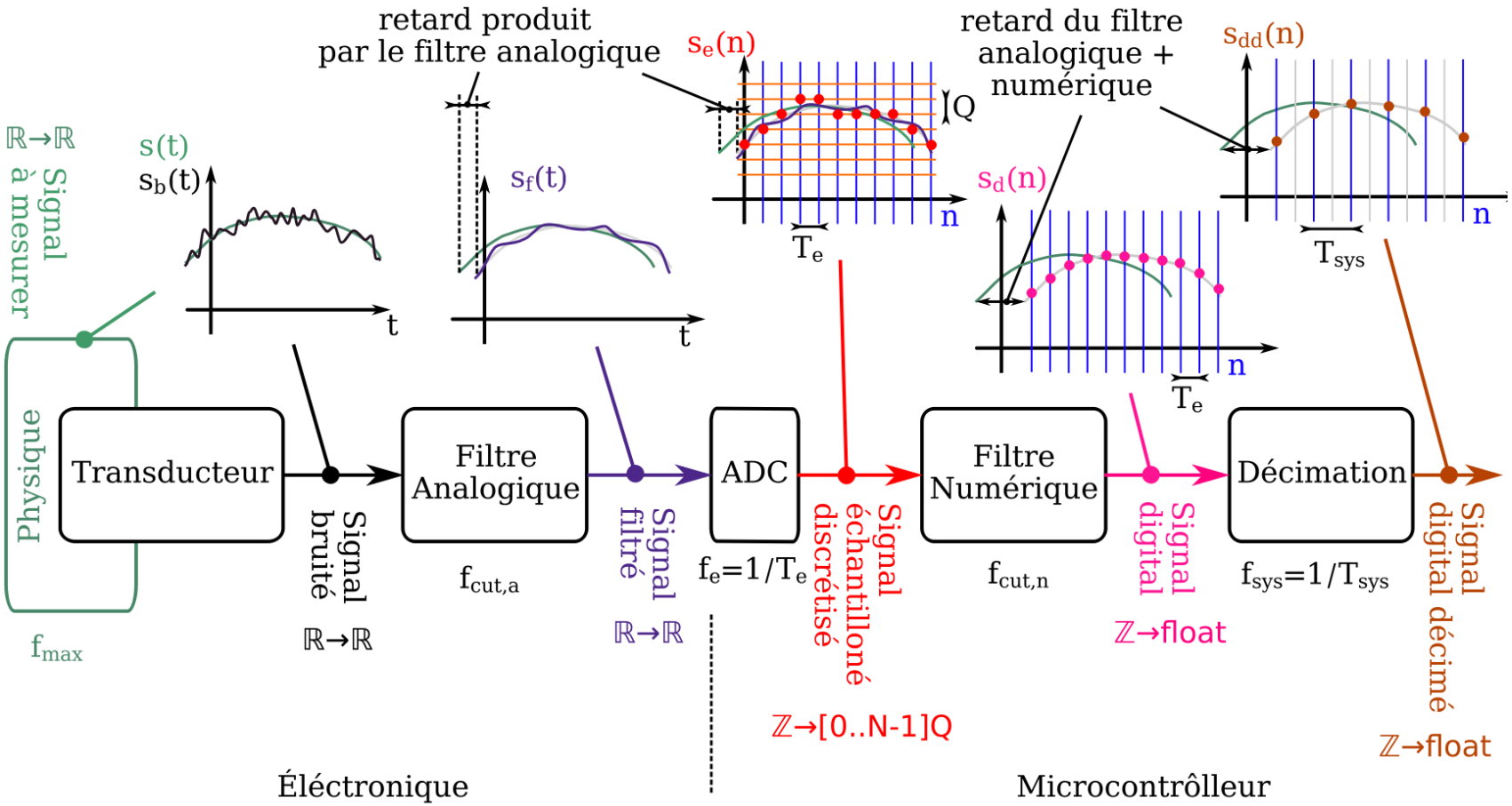
Principe

A FAIRE : Shanon, décomposition fréquentiel, filtre fréquentiel

A FAIRE : Parler de l'instabilité des filtres numériques si les valeurs sont >1. Il faut donc normaliser les valeurs. Expliquer que comme les filtres de cette section sont des filtres passe bas de gains 1, on peut aussi translater le signal si nécéssaire pour le ramener à une valeur entre -1 et 1 et ne pas oublier de rajouter ensuite la valeur translatée au moment de son utilisation.

A FAIRE : Regénérer tous les codes et vérifier la coéhence et le bon fonctionnement de tous les codes de cette section.

Chaîne de filtrage 1/7

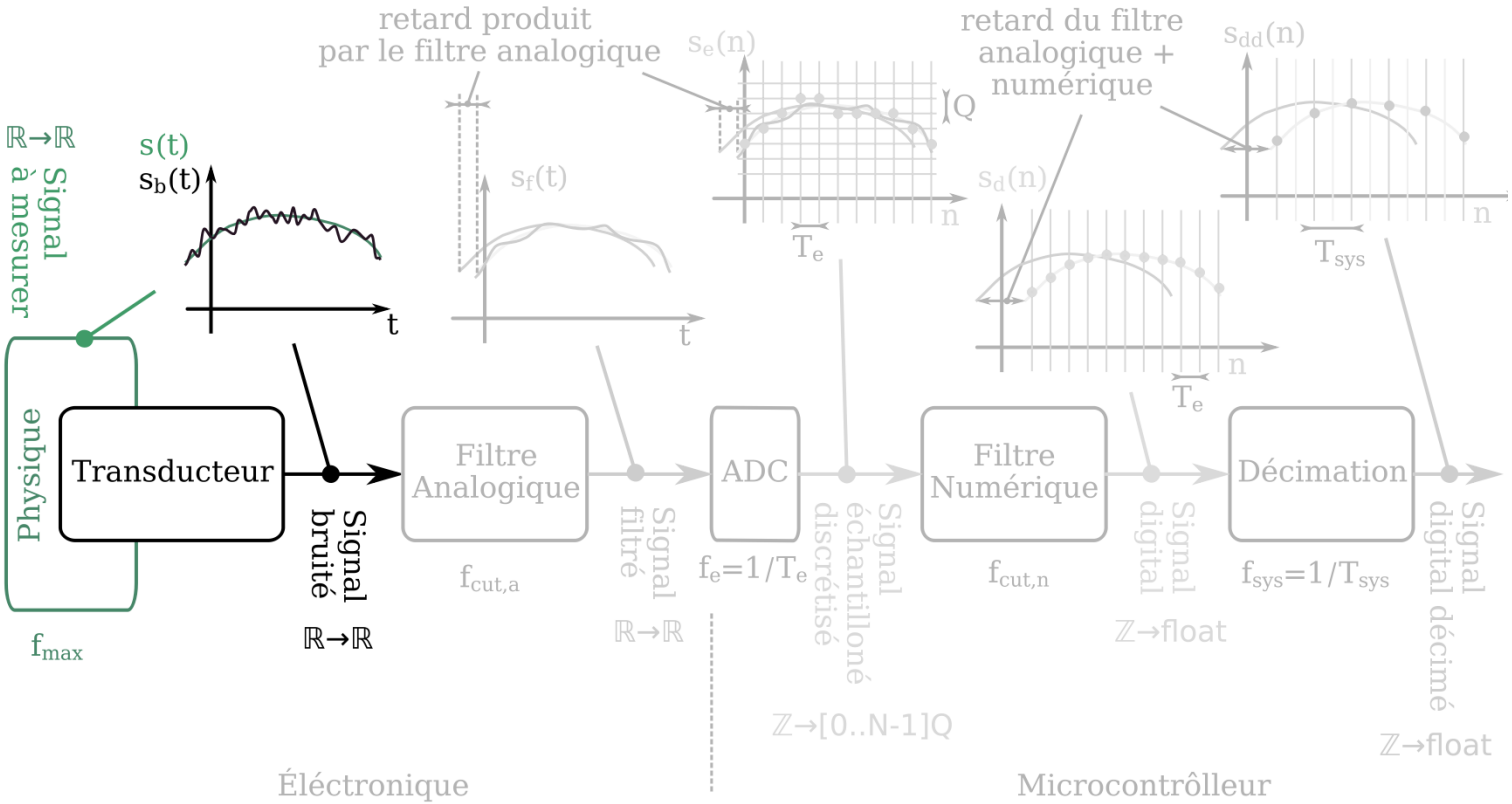


$S(t)$: signal à mesurer
$S_b(t)$: signal bruité récupéré
$S_f(t)$: signal filtré par le filtre analogique
$S_e(n)$: signal échantillonné par l'ADC
$S_d(n)$: signal digital
$S_{dd}(n)$: signal digital décimé
Q : quanta

N : résolution de l'ADC
f_{max} : fréquence max du contenu fréquentiel de $s(t)$.
$f_{cut,a}$: fréquence de coupure du filtre analogique,
f_e/T_e: fréquence/période d'échantillonnage
$f_{cut,n}$: fréquence de coupure du filtre numérique,
f_{sys}/T_{sys}: fréquence/période du système

(version longue)

Chaîne de filtrage - le transducteur 2/7



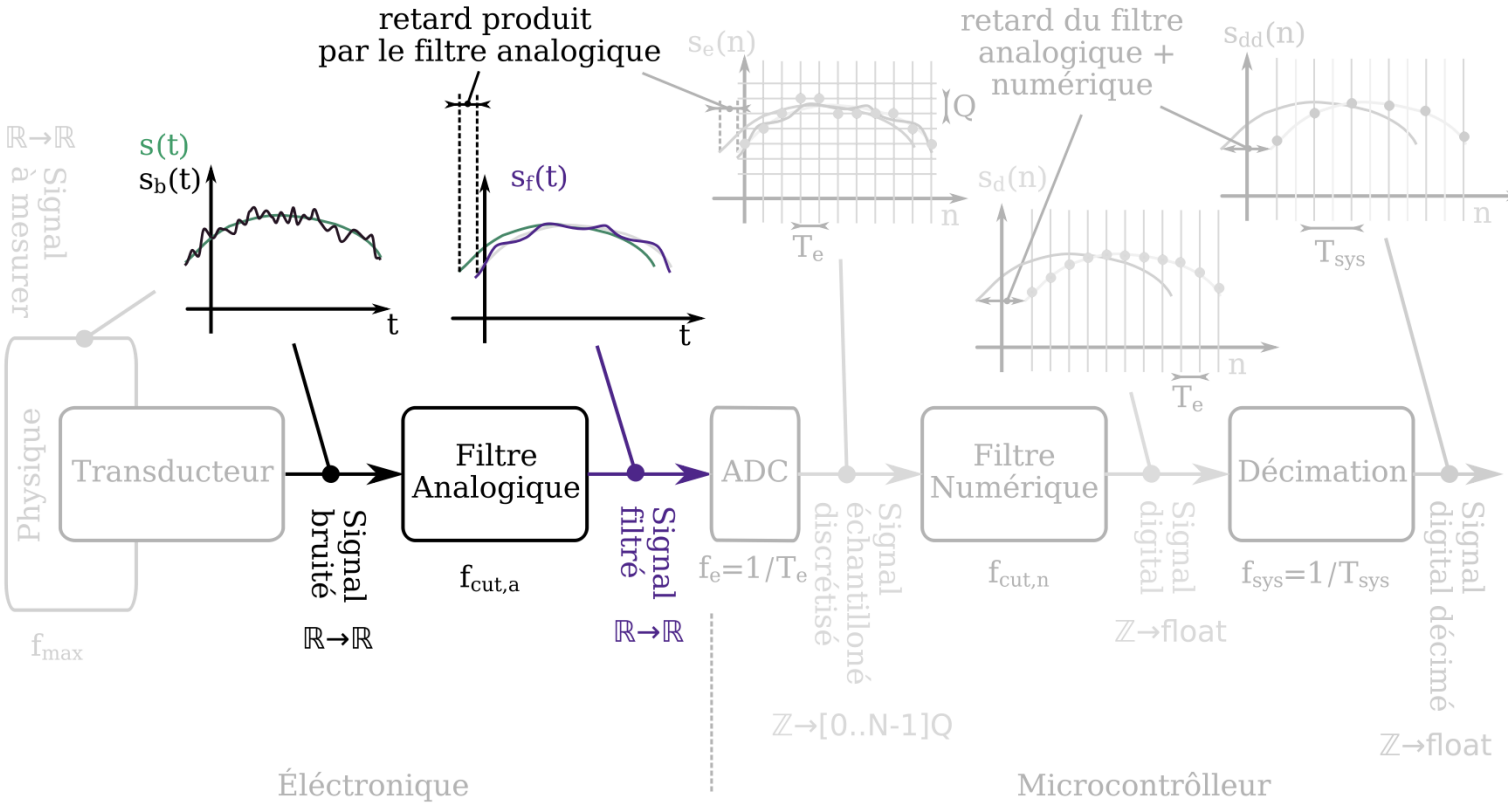
Le transducteur convertit un signal physique $s(t)$ à mesurer en un signal éléctrique bruité $s_b(t)$.

On suppose que le contenu fréquenciel du signal est borné par f_{max}, c'est à dire, que l'énergie correspondant aux fréquences plus grandes que f_{max} du signal est négligeable vis à vis de son énegie total.

Le signal bruité $s_b(t)$ contient les fréquences de $s(t)$ auxquelles s'ajoutes celles du bruit contenant des fréquences plus grandes que f_{max}.

(version longue)

Chaîne de filtrage - le filtre analogique 3/7



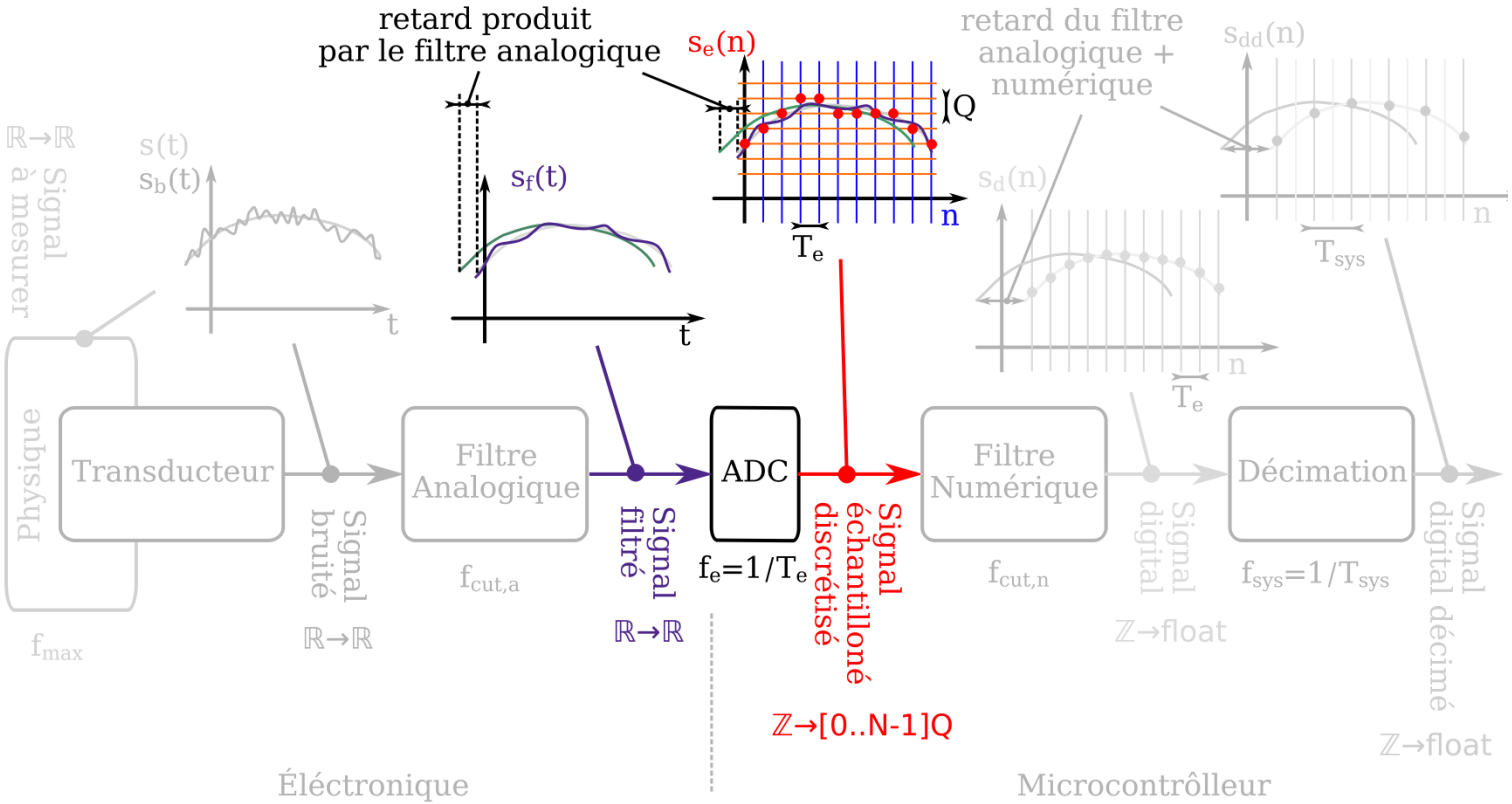
Lors de l'échantillonnage, le bruit de contenu fréquentiel $> f_e/2$ se replie est s'ajoute au contenu fréquenciel du signal entre $[0, f_e]$, ce qui détruit le signal.

Le but du filtre analogique n'est pas de retirer tout le bruit amené par le transducteur. Le but de ce filtre, apppellé aussi filtre anti-repliement, est de supprimer le bruit ayant une fréquénce supérieure à $f_e/2$, pour que l'on puisse reconstituer SANS AUCUNE PERTE D'INFORMATION, par traitement numérique, le signal $s(t)$ à partir du signal échantillonné.

Le théorème de shanon explique qu'il faut que $f_{cut,a} < f_e/2$. Dans la pratique, on essaye d'écarter $f_{cut,a}$ de $f_e/2$ pouir réduire le repliement du bruit. Par contre, plus $f_{cut,a}$ est petir, plus le retard du filtre analogique est grand. De même, on verra que plus f_e est grand plus le bruit de

(version longue)

Chaîne de filtrage - l'échantillonneur (ADC) 4/7



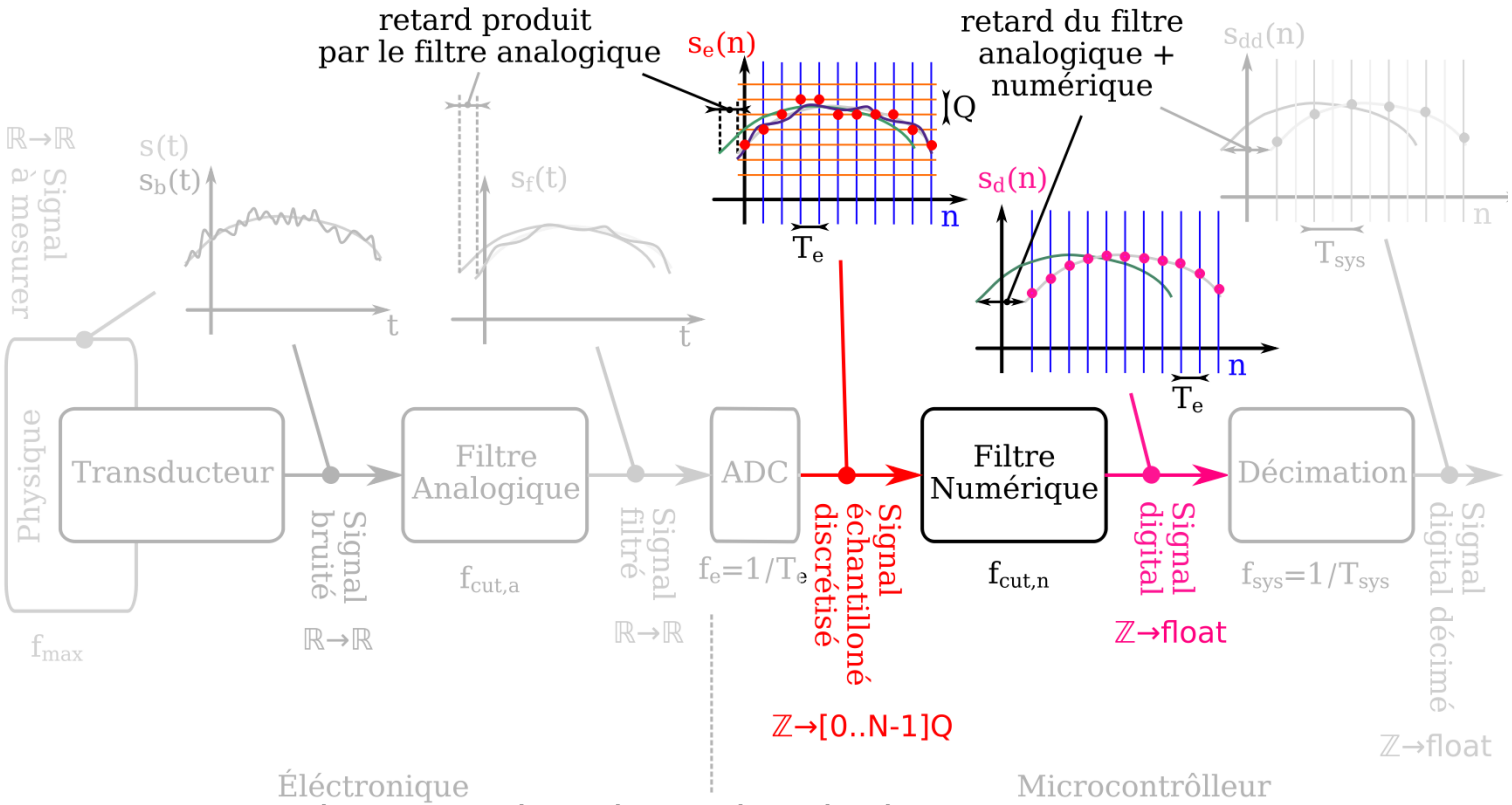
L'ADC échantillonne, à la fréquence $f_e = \frac{1}{T_e}$ le signal numérique pour produire un signal échantillonné. Les valeurs obtenues sont des entiers entre 0 et $N-1$ où N est la résolution de l'ADC. Pour l'Arduino UNO R3, $N = 1024$.

Nous utiliserons la fonction analogRead() d'Arduino. Mais sachez qu'il est possible de configurer finement cette étape pour accélérer les vitesses de lectures et réduire les offsets de mesures.

La note d'application suivante détaille ce point là pour les StmF32 :
How to optimize the ADC accuracy in the STM32 MCUs, Application note AN2834 rev. 9, August 2023.

(version longue)

Chaîne de filtrage - le filtre numérique 5/7



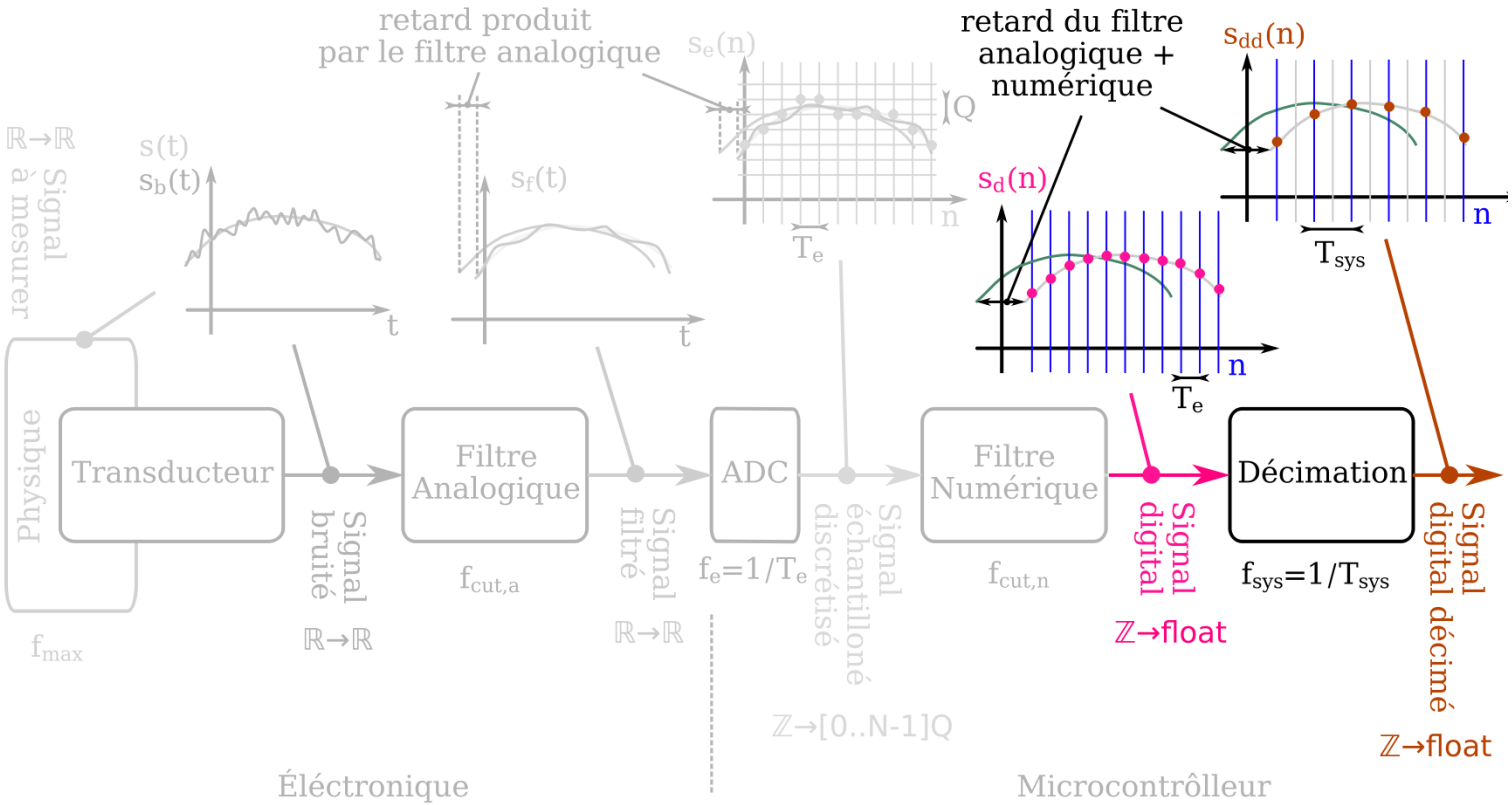
Le filtre numérique joue plusieurs rôles. Il joue le rôle de :

- filtre anti-repliement vis à vis de l'étape de décimation qui suit : $f_{cut,n} < f_{sys}/2$;
- filtre pour retirer le bruit de discrétisation amené par l'échantillonneur (ADC) qui transforme une tension en un valeur avec seulement N valeurs espacées d'un quanta Q;
- filtre pour retirer le bruit amené par le transducteur qui n'a pas été retiré par le filtre analogique.

La fréquence de coupure de ce filtre doit donc vérifier la relation : $f_{max} < f_{cut,d} < f_{sys}/2$.
Plus $F_{cut,d}$ est proche de f_{max} plus on restitue au mieux le signal d'origine $S(t)$.
Dans le cadre d'un calcul en temps réel, cela s'accompagne d'une augmentation du retard entre le signal réel et le signal filtré obtenu.

(version longue)

Chaîne de filtrage - la décimation 6/7



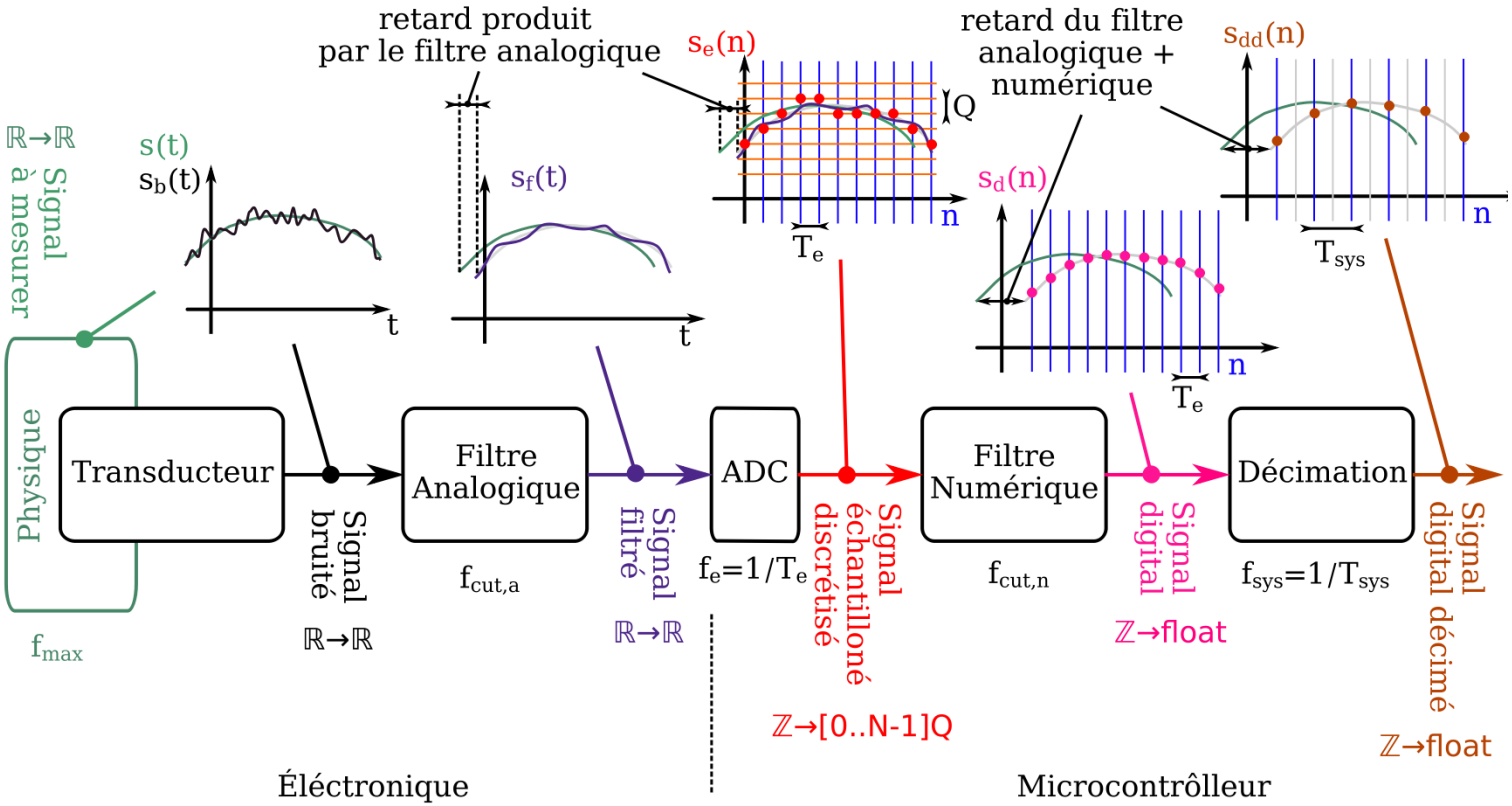
La capacité de calcul et de comunication du microcontrôleur est limitée. La fréquence de mise à jour de tous les services du microconrolleur f_{sys} doit rester la plus basse possible afin de ne pas surcharger le microcontrôleur en calcul.

A contrario, plus la fréquence d'échantillonnage f_e est grande, plus on réduit le bruit de quantification grace au filtre numérique et plus on peut efficacement concevoir un filtre analogique qui supprime le bruit au dela de $f_e/2$.

Ainsi, la décimation concilie les deux problématiques. Elle supprime une valeur sur $M = F_e/F_{sys}$ permettant d'obtenir un signal échantillonné à la fréquence du système f_{sys}.

(version longue)

Chaîne de filtrage - les pièges 7/7



Attention, la décimation doit toujours être précédée d'un filtre numérique qui joue le rôle de filtre anti-repliement afin de supprimer le bruit de fréquence supérieure à $f_{sys}/2$.

Si vous ne faite pas cela, le signal peut être détruit sans qu'il soit possible de récupérer l'information initiale.

De même, il faut toujours filtrer analogiquement un signal avant de procéder à son échantillonnage. Si cela n'est pas fait, la présence d'un bruit en haute fréquence peut détruire le signal originel.

(version longue)

Plan

Exemple : Acquisition d'un signal issu d'un joystick 1/2

Contrainte du contexte :

- la fréquence maximale d'un mouvement de joystick que peut faire un homme varie entre 8Hz et 16Hz. Le contenu fréquentiel du signal à acquérir est entre 0 et 20 Hz :

$$f_{max} = 20 \text{ Hz}$$

- La position du joystick doit être transmise à l'ordinateur à un temps inférieur au temps minimal de perception de l'homme : 13 ms. (On dervrait perende une marge, mais il est diffcile de concilier bon filtrage et faible retard).

$$\text{retard analogique} + \text{retard digital} < 13 \text{ ms}$$

- Le microcontroleur est un arduino Uno R4, qui fait des calculs 32 bit natif. Il est largement capable d'échantillonnée et filtrer à 2 kHz les valeurs des 2 axes du joystick (ce qui n'est pas le cas de l'arduino Uno R3).

$$\text{type des réels natifs : float} \qquad f_{sample} \leq 2 \text{ kHz}$$

- Dans les systèmes d'exploitation, la communication des souris et gamepads se fait à 125 Hz.

$$f_{sys} = 125 \text{ Hz}$$

Contrainte de filtrage :

- Le filtre analogique est un filtre anti-repliement vis à vis de l'échantillonneur :

$$f_{max} < f_{cut,a} < f_{sample}/2$$

- Le filtre analogique est un filtre anti-repliement vis à vis du décimateur :

$$f_{max} < f_{cut,d} < f_{sys}/2$$

(version longue)

Exemple : Acquisition d'un signal issu d'un joystick 2/2

Taux de réduction du bruit à choisir :
Les taux de réduction du bruit dépendent du contexte. Un choix judicieux se fait en étudiant le contenu fréquentiel du signal à l'aide d'un analyseur de spectre. En évaluant, toujours vis à vis du contexte, la variance du bruit maximal tolérée en fin de filtrage et en étudiant la propagation du bruit à chaque étage, on peut déduire les facteurs de réductions à choisir. Ce travail est un peu complexe. Nous proposons une méthode plus ampirique en choisissant les facteurs d'atténuations suivants. Vous les ajusterez ensuite au vu des résultats expérimentaux que vous obtiendrez :

- Le taux de réduction du bruit du filtre analogique à $f_{cut,a}$ doit être d'au moins de $1/20$:

$$\text{attenuation analogique à } f_{cut,a} \geq -20 \times \log_{10}(1/20)\ \text{dB} = 26\ \text{dB}$$

- Le taux de réduction du bruit du filtre numérique à $f_{cut,d}$ doit être d'au moins de $1/20$:

$$\text{attenuation digital à } f_{cut,d} \geq -20 \times \log_{10}(1/20)\ \text{dB} = 26\ \text{dB}$$

Nous allons maintenant choisir et synthétiser des filtres analogiques en faisant varier leurs paramètres de façon à assurer les contraintes données ci-dessus.

Dans les pages qui suivent, les ordres des filtres, le choix des filtres, le choix de la fréquence d'échantillonnage et les choix des facteurs d'atténuations ont été trouvée expérimentalement afin d'assurer toutes les contraintes ennoncées précédement.

Plan

La synthèse des filtres analogiques 1/2

Téléchargez la bibliothèqe code/filtre/filter_tools.py du fichier code.zip et exécutez les lignes suivantes pour synthétiser un filtre analogique de Butterworth qui réduit au moins d'un facteur 20 tout bruit de contenu fréquentiel supérieur à f_sample/2.

code/filtre/synthesis_of_analogic_and_digital_filters.py

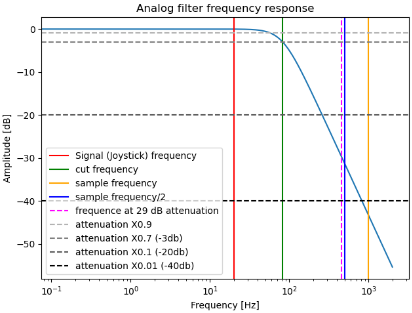
```
import filter_tools
import numpy as np

f_max = 20 # Hz
f_sys = 125 # Hz
f_sample = 2000 # Hz
# f_sample = 1000 # Hz
filter_tools.check(f_max < f_sys < f_sample)
########################

analog_order = 2
wanted_analog_attenuation_at_fsample_over_2 = filter_tools.scaleToDb(1/20) # dB4
analog_filter = filter_tools.design_analog_filter(
    analog_order, wanted_analog_attenuation_at_fsample_over_2, f_sample, f_max,
    # resistors=filter_tools.E24, capacitors=filter_tools.C6,
    resistors=filter_tools.E24_minus_12, capacitors=filter_tools.C1,
    filter_name = 'butterworth',
    #filter_name = 'chebychev1', wanted_low_attenuation = .5 # dB
)

f_cut_a = analog_filter['f_cut_high_attenuation']
filter_tools.check(f_max < analog_filter['f_cut_low_attenuation'] < f_sample)
filter_tools.check(f_max < f_cut_a < f_sample)
```

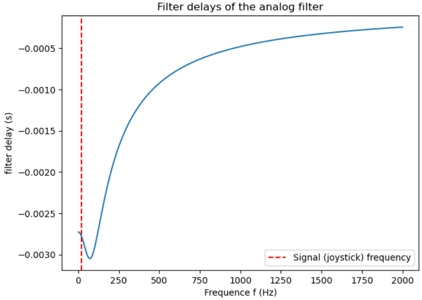
Ce programme dessine aussi la réponse fréquentiel du filtre (le diagrame de Bode), le retard du filtre en fonction du contenu fréquentiel, le retard maximal que l'on peut observer pour un signal de contenu fréquentiel borné par f_max mais aussi l'implémentation électronique du filtre.

(version longue)

La synthèse des filtres analogiques 2/2

On obtient les résultats suivants:





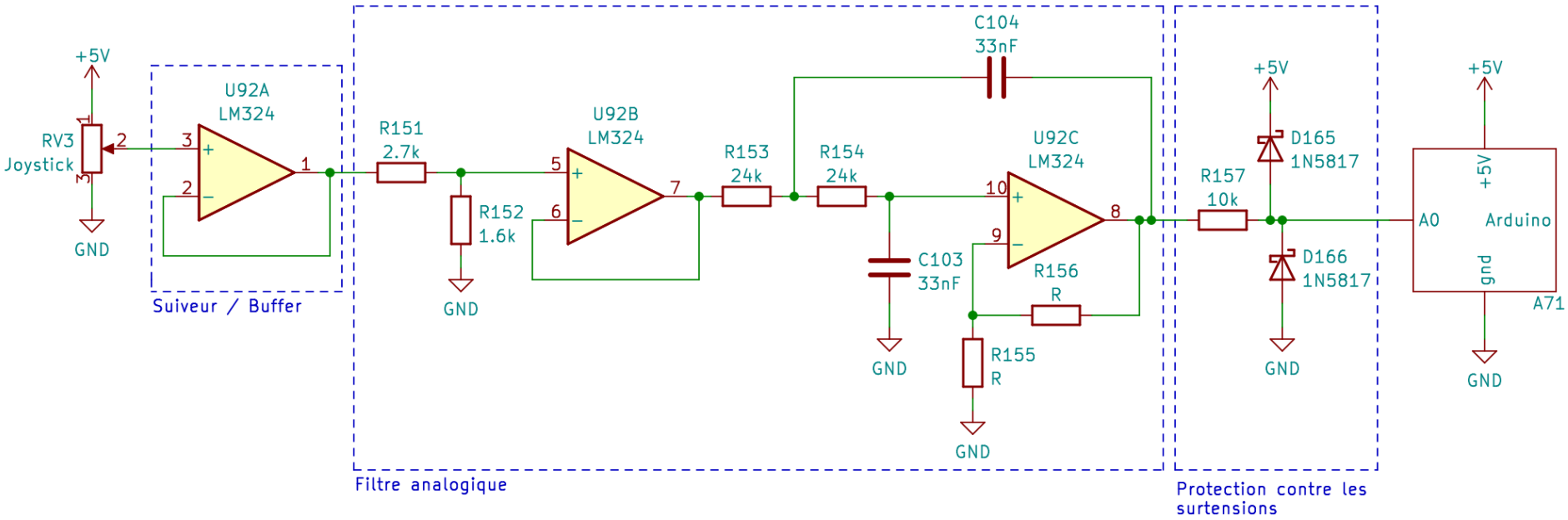

```
The maximum delay for the filtered signal is : 2.7757164928622804 ms
Cellule 0 Diviseur :

R1 = 2.0 kOhms
R2 = 1.2 kOhms

                                 _______
Vin ---[R2]----o--------| +      |
               |        |        |---o-- V0
             [R1]    |--| -      |   |
               |     |  ---------    |
               |     |               |
               |     ----------------
               |
             gnd
```

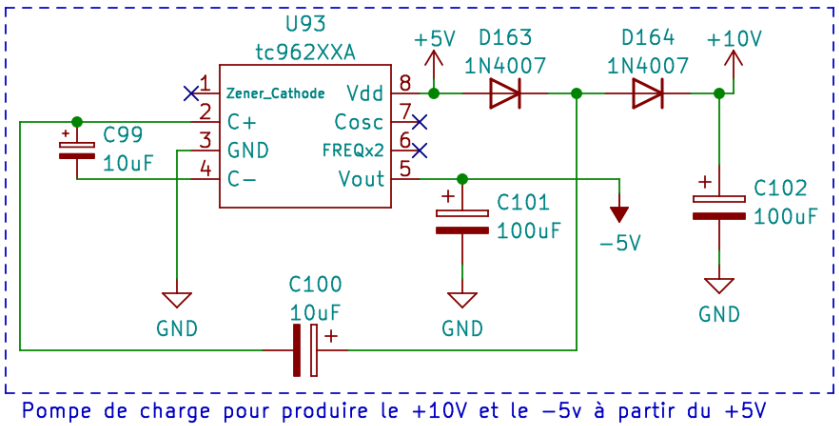
```
Cellule 1 de Sallen-key :

C1 = C2 = 150.0 nF
R1 = R2 = 13.0 kOhms
R : 2.0 kOhms
(K-1)R : 1.2 kOhms
Gain, K : 1.6026092864097345

                    C1
             -------||----------------------------
             |                                   |
                                  _______        |
     V0 --[R1]--o--[R2]--o-------------| +      |       |
                         |             | .      |---o-o-- Vout
                         = C2       |--| -      |   |
                         |          |  ---------    |
                         |          |               |
                         o-----[R]--o--[(K-1).R]----
                         |
                        gnd
```

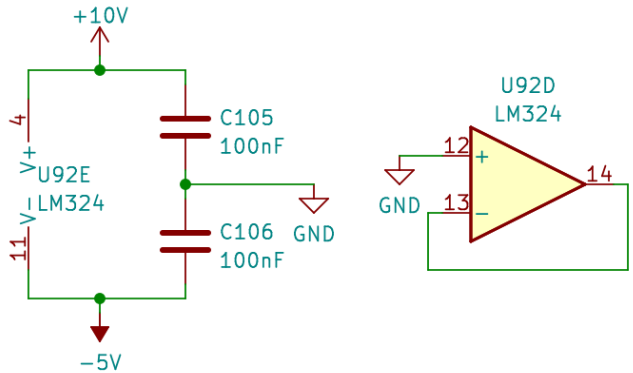
(version longue)

Implémentation du filtre analogique

Utilisez les schémas électriques générés par le script pour implémenter votre filtre ainsi :







(version longue)

Plan

La synthèse des filtres numériques 1/2

Téléchargez la bibliothèqe code/filtre/filter_tools.py du fichier code.zip et exécutez les lignes suivantes pour synthétiser un filtre analogique de Chebychev 2 qui réduit au moins d'un facteur 30 tout bruit de contenu fréquentiel supérieur à f_sys/2.

```
import filter_tools
import numpy as np

f_max = 20 # Hz
f_sys = 125 # Hz
f_sample = 2000 # Hz
# f_sample = 1000 # Hz

digital_order = 4
float_type = np.float32
wanted_digital_attenuation_at_fsys_over_2 = filter_tools.scaleToDb(1/50)

digital_filter = filter_tools.design_digital_filter(
    digital_order, wanted_digital_attenuation_at_fsys_over_2 , f_max, f_sample, f_sys, float_type,
    #filter_name = 'butterworth'
    filter_name = 'chebychev2'
)

f_cut_d = digital_filter['f_cut_high_attenuation']
filter_tools.check(f_max < digital_filter['f_cut_low_attenuation'] < f_sys/2 )
```

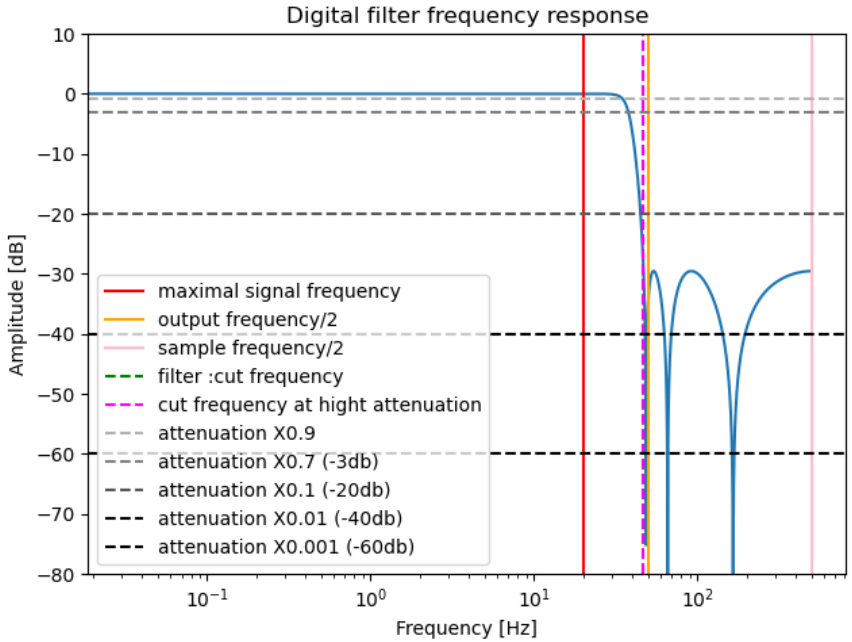
code/filtre/synthesis_of_analogic_and_digital_filters.py

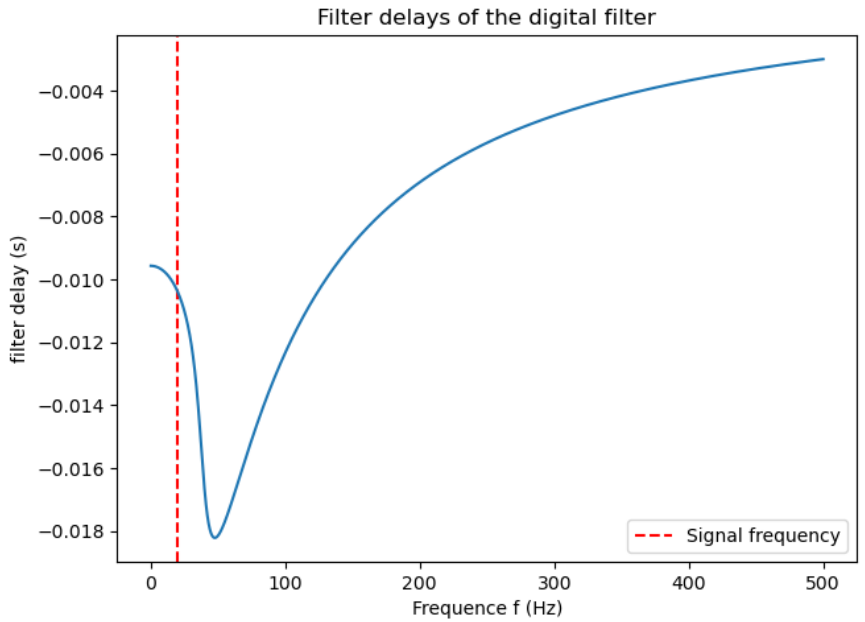
Ce programme dessine aussi la réponse fréquentiel du filtre (le diagrame de Bode), le retard du filtre en fonction du contenu fréquentiel, le retard maximal que l'on peut observer pour un signal de contenu fréquentiel borné par f_max mais aussi l'implémentation en C++ du filtre qu'il faudra réutiliser plus tard.

(version longue)

La synthèse des filtres numériques 2/2

On obtient les résultats suivants:






```
float type : <class 'numpy.float32'>


The maximum delay for the filtered signal is : 0.010400129425580268 seconds (96.15264955649393
Hz) ~= 10 samples at 1000 Hz

The filter order is : 6

The filter coefficients are :

b : [ 0.02780181 -0.1319144   0.28511503 -0.36157134  0.28511503 -0.1319144
  0.02780181]
a : [  1.          -4.93656     10.236738   -11.401082     7.1875944
  -2.4306488     0.34439158]


We recall that :

a[0]*y[n] = b[0]*x[n] + b[1]*x[n-1] + ... + b[M]*x[n-M]
                      - a[1]*y[n-1] - ... - a[N]*y[n-N]

where X is the input signal and Y is the filtered signal.
```

```
The C++ code is :

#include "filter.hpp"
// Chebychev2 filter :
//    Parameter :
//        order : 6
//        cut frequency : 47.0 Hz,
//        stop band attenuation : 29.542425094393252 dB,
//        sample frequency : 1.0 kHz
//    Analysis :
//        Maximal expected delay : 10.4 ms
const int order = 6;
RII_filter<float, order> filter{
    {0.02780181, -0.1319144, 0.28511503, -0.36157134, 0.28511503, -0.1319144, 0.02780181, }, //
 b : Numerator
    {1.0, -4.93656, 10.236738, -11.401082, 7.1875944, -2.4306488, 0.34439158, } // a : Denomina
tor
};
```

(version longue)

Vérification des retards

Il ne reste plus qu'à vérifier la contrainte de retard à ne pas dépasser :

```
total_delay = analog_filter['max_delay'] + digital_filter['max_delay']
```

Dans notre cas on obtient un retard maximal entre 12 et 13 ms ce qui correspond aux valeurs souhaitées.

Il n'est pas rare que le paramétrage des filtres ne permet pas de vérifier toutes les contraintes. Il faut allors changer le materiel utilisé pour de plus performant ou relâcher les contraintes en faisant des concessions sur les fonctionalités de l'application à développer.

Plan

Implémentation du filtre numérique

Le code source C++ du filtre numérique a été produit par le script de synthèse précédent. Il faut maintenant l'inclure à l'aide d'un copier/collé dans le code de votre microcontroleur. Son utilisation nécessite une mise à jour des données à une fréquence fixe et précise.

Deux méthodes sont possibles :

1. **mettre à jour le filtre, sans interruption, dans la fonction loop()**:
   La page 233, Implémenation du filtre numérique sans interruption. montre comment procéder.
   Ce code est simple à mettre en oeuvre. Cependant, le code du microcontrôleur doit être executé à la fréquence d'échantillonnage. Cela n'est pas toujours possible, surtout quand on doit comuniquer avec l'ordinateur à l'aide du port série.
2. **mettre à jour le filtre à l'aide d'une interruption** :
   La fonction loop() est alors intérronpue et mis à en pause à la fréquence choisie (celle d'échantillonnage) pour exécuter en priorité, le code de la fonction associée à l'interruption.
   Ce code est plus compliqué à mettre en oeuvre car il dépend de l'architecture choisie.
   Voici différentes implémentations du filtre utilisant des interruptions :
   - la page 235 est dédiée à l'Arduino Uno R4,
   - la page 238 est dédiée à l'ESP32,
   - la page 246 est dédiée à l'Arduino Uno R3.
   TRÈS TRÈS IMPORTANT : Vous devez vérifier que les calculs que vous réalisez se font en temps et en heure à la fréquence voulue. Pour ce faire, configurez deux broches digitales, (l'une pour l'interruption et l'autre pour la boucle) et allumez et éteingenez les broches au début et à la fin des sections de codes critiques et vérifiez à l'oscilloscope que les calculs se termine avant qu'une nouvelle interruption se lève.

(version longue)

Implémentation du filtre numérique sans interruption - 1/2

Pour cette impélmentation, vous devez utiliser les bibliothèques code/filtre/filter.hpp et code/filtre/queue.hpp du fichier code.zip.

```
#include "filter.hpp"

typedef double REAL_TYPE;
const unsigned int sample_frequence = 1000; // in Hz
const unsigned int output_frequence = 125; // in Hz
const unsigned int decimation_factor = sample_frequence / output_frequence;
// Chebychev2 filter :
//    Parameter :
//        order : 6
//        cut frequency : 58.25 Hz,
//        stop band attenuation : 26.020599913279625 dB,
//        sample frequency : 1.0 kHz
//    Analysis :
//        Maximal expected delay : 7.175 ms
const int order = 6;
RII_filter<REAL_TYPE, order> filter{
    {0.041494492, -0.17629611, 0.35559416, -0.43954557, 0.35559416, -0.17629611, 0.041494492, }, // b : Numerat
    {1.0, -4.604491, 8.970845, -9.438095, 5.6464767, -1.8195444, 0.2468492, } // a : Denominator
};

const unsigned int queue_capacity = 4; // have to be a power of two
Decimation_queue<REAL_TYPE, queue_capacity> output_queue(decimation_factor);
```

code/filtre/filter_signal.cpp

(version longue)

Implémentation du filtre numérique sans interruption - 2/2

```
unsigned int time, last_time;

void setup(){
  Serial.begin(115200);
  filter.reset();
  last_time = micros();
}

REAL_TYPE raw_value;
REAL_TYPE output_value;
bool loss_data;
const unsigned int sample_period = 1000000 / sample_frequence;

const int nb_bits_adc = 10; // Arduino Uno R3, R4,
// const int nb_bits_adc = 12; // Esp32
const int max_adc_value = (1<<nb_bits_adc) - 1;

void loop(){
  time = micros();
  if( time - last_time > sample_period ){
    raw_value = analogRead(A0);
    raw_value *= (((REAL_TYPE) 1.0)/max_adc_value);
    filter.append(raw_value);
    output_queue.append(filter.get_value());

    if( output_queue.get_avalaible_value(output_value, loss_data) ){
      Serial.println(output_value, 5);
      if(loss_data) Serial.println("Data lost");
    }
    last_time = time;
  }
}
```

code/filtre/filter_signal.cpp

(version longue)

Implémenation du filtre numérique avec interruption pour Arduino Uno R4 - 1/3

Pour cette impélmentation, vous devez utiliser les bibliothèques code/filtre/filter.hpp et code/filtre/queue.hpp du fichier code.zip.

code/filtre/filter_signal_with_interruption_uno_r4.cpp

```
#include "filter.hpp"
#include <FspTimer.h>

FspTimer timer_for_filter;

typedef float REAL_TYPE;
const unsigned int sample_frequence = 2000; // In Hz
const unsigned int output_frequence = 125; // In hz
const unsigned int decimation_factor = sample_frequence / output_frequence;
// Chebychev2 filter :
//    Parameter :
//        order : 4
//        cut frequency : 58.25 Hz,
//        stop band attenuation : 33.979400086720375 dB,
//        sample frequency : 2.0 kHz
//    Analysis :
//        Maximal expected delay : 11.09 ms
const int order = 4;
RII_filter<REAL_TYPE, order> filter{
    {0.018460825, -0.069107704, 0.10145059, -0.069107704, 0.018460825, }, // b : Numerator
    {1.0, -3.7085981, 5.16765, -3.2060149, 0.7471194, } // a : Denominator
};

const unsigned int queue_capacity = 4; // have to be a power of two
Decimation_queue<REAL_TYPE, queue_capacity> output_queue(decimation_factor);
Decimation_queue<REAL_TYPE, queue_capacity> output_queue_raw(decimation_factor);
```

(version longue)

Implémenation du filtre numérique avec interruption pour Arduino Uno R4 - 2/3

code/filtre/filter_signal_with_interruption_uno_r4.cpp

```
const int nb_bits_adc = 10; // Avalaible values for Arduino Uno R4 : 10, 11, 12, 13 or 14
const int max_adc_value = (1<<nb_bits_adc) - 1;

void make_a_sample(timer_callback_args_t __attribute((unused)) *p_args) {
    REAL_TYPE raw_value = analogRead(A0);
    raw_value *= (((REAL_TYPE) 1.0)/max_adc_value);
    filter.append(raw_value);
    output_queue.append(filter.get_value());
    output_queue_raw.append(raw_value);
}

bool beginTimer(float rate) {
  uint8_t timer_type = GPT_TIMER;
  int8_t tindex = FspTimer::get_available_timer(timer_type);
  if (tindex < 0){
    tindex = FspTimer::get_available_timer(timer_type, true);
  }
  if (tindex < 0) return false;
  FspTimer::force_use_of_pwm_reserved_timer();
  if( !timer_for_filter.begin(
      TIMER_MODE_PERIODIC, timer_type, tindex, rate, 0.0f, make_a_sample
  ) ) return false;
  if (!timer_for_filter.setup_overflow_irq()) return false;
  if (!timer_for_filter.open()) return false;
  if (!timer_for_filter.start()) return false;
  return true;
}
```

La partie du code concernant les interruptions est inspiré du blog blog de Phil Schatzmann (consulté le 03/11/2024).

(version longue)

Implémenation du filtre numérique avec interruption pour Arduino Uno R4 - 3/3

code/filtre/filter_signal_with_interruption_uno_r4.cpp

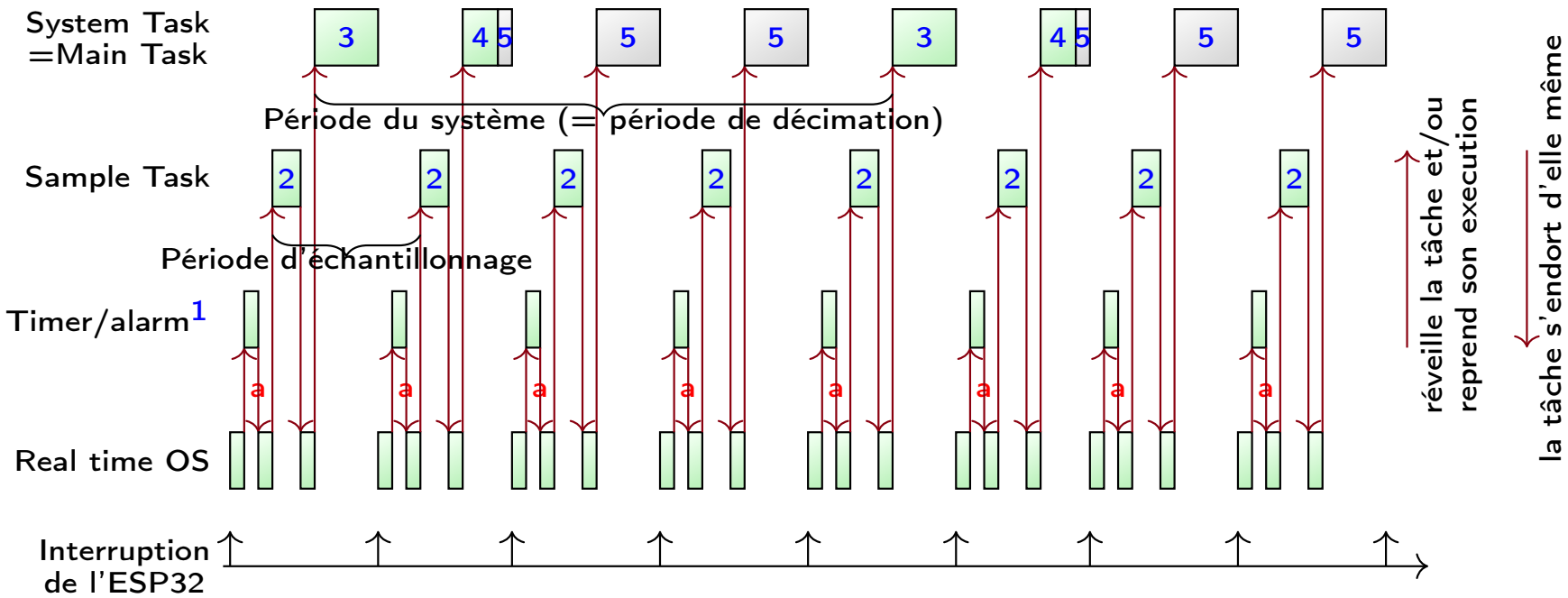
```
unsigned int new_time, last_time;

void setup() {
  Serial.begin(115200);
  if( !beginTimer(sample_frequence) ){
    Serial.println("Failed to init timer and interruption for filter.");
  }
  filter.reset();
  last_time = micros();
  analogReadResolution(nb_bits_adc);
}

REAL_TYPE output_value, output_value_raw;
bool loss_data, loss_data_raw;
const unsigned int sample_period = 1000000 / sample_frequence;

void loop(){
  new_time = micros();
  if( new_time - last_time > (sample_period/4) ){
    noInterrupts();
    bool have_new_value = output_queue.get_avalaible_value(output_value, loss_data);
    output_queue_raw.get_avalaible_value(output_value_raw, loss_data_raw);
    interrupts();
    if( have_new_value ){
      Serial.print(output_value, 6); Serial.print(", "); Serial.println(output_value_raw, 6);
      if(loss_data) Serial.println("Data lost");
    }
    last_time = new_time;
  }
}
```

(version longue)

Implémentation du filtre numérique avec interruption pour ESP32

L'ESP32 utilise FreeRTOS qui est un système d'exploitation temps réel pour ordonancer les tâches. Nous allons utiliser les timers de l'ESP32 et créer une tâche supplémentaire, la tâche d'échantillonnage (Sample Task) pour réaliser des mesures, les filtrer à grande fréquence et les décimer à une fréquence de décimation. Les calculs complexes nécessistant plus de temps sont réalisées dans la tâche dite "système" qui sera, dans notre cas particulier, la tâche principale du programme (Main Task = System Task).

Voici le chronograme des différentes tâches à implémenter :



1: exécute à chaque alarme la fonction qui réveille la tâche d'échantillonnage; 2: échantillonne, filtre et décime; 3/4: debut/fin du travail que doit exécuter la tâche "sytème"; 5: la tâche ne fait plus rien : loop() est exécuté mais ne fait rien.
a: demande à l'OS de réveiller la tâche d'échantillonage (Sample Task).

(version longue)

Implémentation du filtre numérique avec interruption pour ESP32 - 1/4

Pour cette impélmentation, vous devez utiliser les bibliothèques code/filtre/filter.hpp et code/filtre/queue.hpp du fichier code.zip.

```
#include <driver/gptimer.h>

#include "filter.hpp"
typedef float REAL_TYPE;
const unsigned int sample_frequence = 2000; // In Hz
const unsigned int output_frequence = 125; // In hz
const unsigned int decimation_factor = sample_frequence / output_frequence;
// Chebychev2 filter :
//    Parameter :
//        order : 4
//        cut frequency : 58.25 Hz,
//        stop band attenuation : 33.979400086720375 dB,
//        sample frequency : 2.0 kHz
//    Analysis :
//        Maximal expected delay : 11.09 ms
const int order = 4;
RII_filter<REAL_TYPE, order> filter{
    {0.018460825, -0.069107704, 0.10145059, -0.069107704, 0.018460825, }, // b : Numerator
    {1.0, -3.7085981, 5.16765, -3.2060149, 0.7471194, } // a : Denominator
};

const unsigned int queue_capacity = 4; // have to be a power of two.
Decimation_queue<REAL_TYPE, queue_capacity> decimated_queue(decimation_factor);
Decimation_queue<REAL_TYPE, queue_capacity> decimated_queue_raw(decimation_factor);
```

code/filtre/filter_signal_with_interruption_esp32.cpp

On commence prar déclarer le filtre digital et les files servant à la décimantion.

Implémentation du filtre numérique avec interruption pour ESP32 - 2/4

On commence par définir une tâche responsable de l'échantillonnage. Cette tâche est périodiquement réveillée pour faire produire, filtrer et décimer un échantillon.

code/filtre/filter_signal_with_interruption_esp32.cpp

```
const int ADC_PIN = 36; // On the boad, the pin name is "VP".
const int nb_bits_adc = 12; // Avalaible values for esp32 : 9, 10, 11, 12
const int max_adc_value = (1<<nb_bits_adc) - 1;

static portMUX_TYPE spinlock = portMUX_INITIALIZER_UNLOCKED;

void make_samples_task( void * pvParameters )
{
  for( ;; ){
    REAL_TYPE raw_value = analogRead(ADC_PIN);
    raw_value *= (((REAL_TYPE) 1.0)/max_adc_value);
    filter.append(raw_value);

    taskENTER_CRITICAL(&spinlock);
      decimated_queue.append(filter.get_value());
      decimated_queue_raw.append(raw_value);
    taskEXIT_CRITICAL(&spinlock);

    vTaskSuspend( NULL );
  }
}
```

La fonction taskENTER_CRITICAL(&spinlock) désactive toutes les interruptions, Cela permet de mettre à jour la structure de donnée partagée entre la tâche principale (exécutant le code de loop()) et la tâche d'échantillonnage. Le fonction taskEXIT_CRITICAL(&spinlock) rétablit les interruptions.
Enfin la fonction vTaskSuspend(NULL) rendort la tâche jusqu'à son prochain réveil.

(version longue)

Implémentation du filtre numérique avec interruption pour ESP32 - 3/5

```
const unsigned int tskMAIN_PRIORITY = 1, SAMPLE_PRIORITY = 2;
static_assert(SAMPLE_PRIORITY > tskMAIN_PRIORITY);
const int STACK_SIZE = 1000;
TaskHandle_t sample_task_handle = NULL;

void create_the_sample_task(){
  xTaskCreate(
    make_samples_task, "SampleTask", STACK_SIZE, NULL, SAMPLE_PRIORITY,
    &sample_task_handle
  );
  configASSERT( sample_task_handle );

  vTaskSuspend( sample_task_handle );
}
```

code/filtre/filter_signal_with_interruption_esp32.cpp

Ce code créer la tâche d'échantillonnage et la met en veille.

La priorité de cette tâche (SAMPLE_PRIORITY) est configurée avec une valeur plus grande que celle de la tâche principale (tskMAIN_PRIORITY) afin qu'au réveil de la tâche d'échantillonnage, le code de loop() s'interrompt pour laisser place à l'échantillonnage.

Implémentation du filtre numérique avec interruption pour ESP32 - 4/5

code/filtre/filter_signal_with_interruption_esp32.cpp

```
static bool IRAM_ATTR start_the_sample_task(
  gptimer_handle_t timer, const gptimer_alarm_event_data_t *edata,
  void *user_data
){
    BaseType_t high_task_awoken;
    high_task_awoken = xTaskResumeFromISR(sample_task_handle);
    return high_task_awoken == pdTRUE;
}

gptimer_handle_t gptimer = NULL;
const uint64_t timer_resolution = 1000000; // 1MHz
const uint64_t alarm_count = timer_resolution / sample_frequence;
void set_a_periodic_alarm_to_resume_the_samle_task(){
  gptimer_config_t timer_config = {
      .clk_src = GPTIMER_CLK_SRC_DEFAULT,
      .direction = GPTIMER_COUNT_UP,
      .resolution_hz = timer_resolution,
  };
  ESP_ERROR_CHECK(gptimer_new_timer(&timer_config, &gptimer));
  gptimer_event_callbacks_t call_backs = {
    .on_alarm = start_the_sample_task, };
  ESP_ERROR_CHECK(gptimer_register_event_callbacks(
    gptimer, &call_backs, NULL));
  ESP_ERROR_CHECK(gptimer_enable(gptimer));
  gptimer_alarm_config_t alarm_config = {
      .alarm_count = alarm_count,
      .reload_count = 0,
  };
  alarm_config.flags.auto_reload_on_alarm = true;
  ESP_ERROR_CHECK(gptimer_set_alarm_action(gptimer, &alarm_config));
  ESP_ERROR_CHECK(gptimer_start(gptimer));
}
```

Ce code configure un timer et une alarme qui, périodiquement, à la fréquence d'échantillonnage (ici 2kHz), interrompt l'exécution du code des différents tâches pour réveiller la tâche d'échantillonnage présentée à la page précédente.

Le code exécuté périodiquement par l'alarme ne peut pas réaliser l'échantillonnage et le filtrage car il doit être le plus rapide possible pour ne pas perturber le bon fonctionnement du système temps réeel FreeRTOS. C'est pourquoi il réveille à la place la tâche qui est chargée d'échantilloner.

(version longue)

Implémentation du filtre numérique avec interruption pour ESP32 - 5/5

```
void setup() {
  Serial.begin(115200);
  analogReadResolution(nb_bits_adc);
  pinMode(ADC_PIN, INPUT);

  create_the_sample_task();
  set_a_periodic_alarm_to_resume_the_samle_task();

  filter.reset();
}

REAL_TYPE filtered_and_decimed_value, decimed_raw_value;
bool data_was_loss, have_new_value;

void loop(){
  taskENTER_CRITICAL(&spinlock);
    have_new_value = decimated_queue.get_avalaible_value(
      filtered_and_decimed_value, data_was_loss
    );
    decimated_queue_raw.get_avalaible_value(decimed_raw_value);
  taskEXIT_CRITICAL(&spinlock);

  if( have_new_value ){
    Serial.print("filtre : ");
    Serial.print(filtered_and_decimed_value, 6);
    Serial.print(", raw : "); Serial.println(decimed_raw_value, 6);
    if(data_was_loss){ Serial.println("Some Data was Loss."); }
  }
}
```

code/filtre/filter_signal_with_interruption_esp32.cpp

La fonction loop() affiche à la fréquence de décimation (125 Hz) le signal filtré et le signal non filtré.

On retouve la section critique, qui doit être la plus courte possible et qui sert à récupérer les échantillons filtrés et décimés.

Il faut noter qu'en temps normal, vous allez vouloir exécuter à la fréquence de décimation, des tâches précises et importantes qui ne pourront pas être retardées par l'execution d'autres codes présents dans la boucle loop(). Il faudra créer pour cela une tâche dédiée qui sera réveillée directement par la tâche chargée de l'échantillonnage.

Pour créer cette tâche, vous vous inspirerez de la création et du réveil de la tâche d'échantillonnage vu précédement. Vous pourrez utilisez la méthode decimed_queue.has_value() pour déterminer à quel moment il faut réveiller cette nouvelle tâche sans vider la file de décimation.

(version longue)

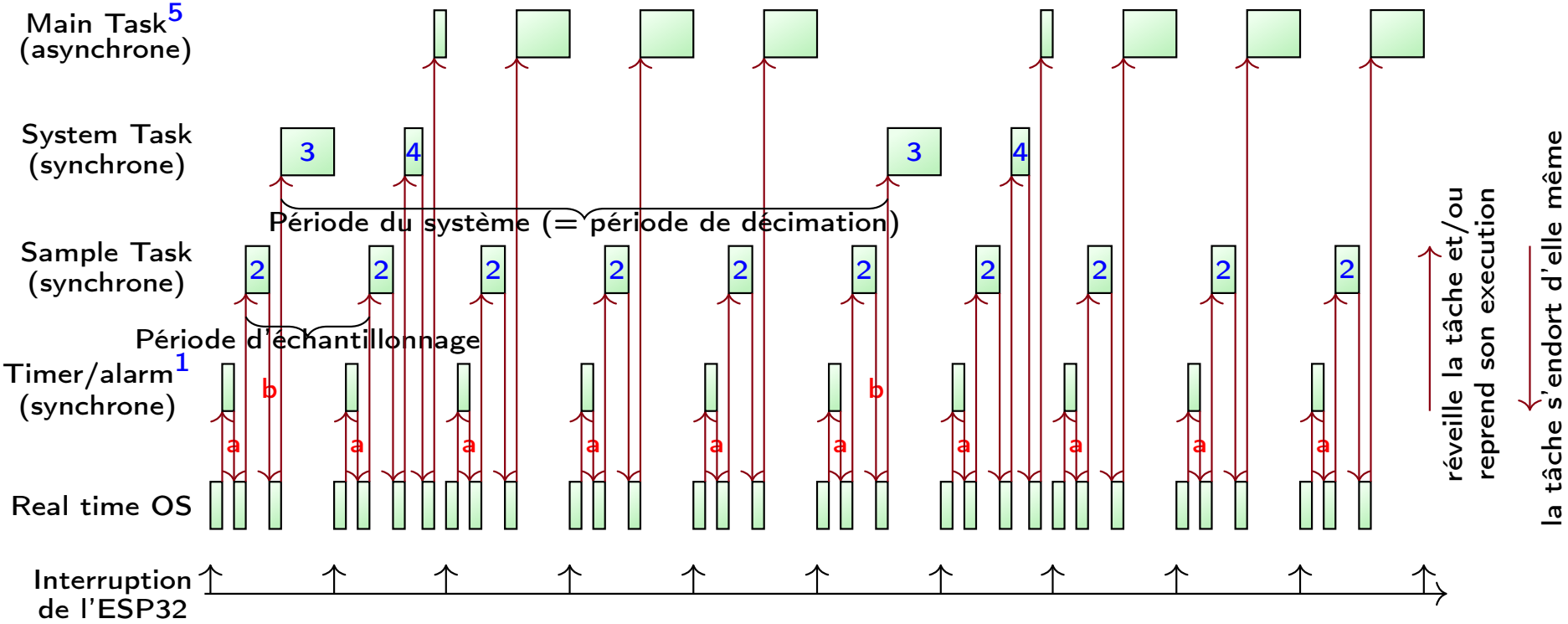
Implémentation du filtre numérique avec interruption pour ESP32

Il est souvant préférable de créer une tâche système (System Task) différente de la tâche principale (Main Task).

La tâche système réalise des tâches de manière synchrone à la fréquence de décimation. À l'aide des échantillons filtrés et décimés, elle s'occupe des tâches temps réelles comme le contrôle de moteurs par exemple.

La tâche principale est utilisée de manière asynchrone et réalise, en tâche de fond, au fil de l'eau et de la disponibilité du microncontrolleur, des tâches non temps réels, comme faire tourner un serveur Web par exemple.

Voici le chronograme de ce système mêlant tâches synchrones et tâches asynchrones :



1: exécute à chaque alarme la fonction qui réveille la tâche d'échantillonnage; 2: échantillonne, filtre et décime; 3/4: debut/fin de l'éxcution de la tâche "sytème"; 5: exécution au fil de l'eau du code de loop();
a: demande à l'OS de réveiller la tache *Sample Task*; b: demande à l'OS de réveiller la tâche *System Task*.

Vous trouver un exemple d'implémentation d'un tel système dans le fichier code/filtre/filter_signal_with_interruption_esp32_with_synchronous_and_asynchronous_tasks.cpp du code source associé code.zip à ce cours.

(version longue)

Implémenation du filtre numérique avec interruption pour Arduino Uno R3/Nano/Mega - 1/4

Nous utilisons la bibliothèque TimerInterrupt de Khoih. Vous devez donc installez cette bibliothèque en plus des bibliothèques code/filtre/filter.hpp et code/filtre/queue.hpp du fichier code.zip.

Attention. l'implémentation qui suit utilise le Timer1. Il se peut que d'autres bibliothèques utilisent déjà ce timer et se mette donc à dysfonctionner. Il faut alors utiliser un autre timer.

Pour ce faire, vous pouvez vous inspirer de l'exemple de la bibliothèque TimerInterrupt présent à la page suivante : github.com/khoih-prog/TimerInterrupt/.../Change_Interval_HF.ino.

Implémenation du filtre numérique avec interruption pour Arduino Uno R3/Nano/Mega - 2/4

code/filtre/filter_signal_with_interruption_unor3_nano_mega.cpp

```
// These define's must be placed at the beginning before #include "TimerInterrupt.h"
// Don't define _TIMERINTERRUPT_LOGLEVEL_ > 0. Only for special ISR debugging only. Can hang the system.
#define TIMER_INTERRUPT_DEBUG      0
#define _TIMERINTERRUPT_LOGLEVEL_  0
#define USE_TIMER_1 true
#include <TimerInterrupt.h> // To be included only in main(), .ino with setup()
#include "filter.hpp"

typedef float REAL_TYPE;
const unsigned int sample_frequence = 2000; // In Hz
const unsigned int output_frequence = 125; // In hz
const unsigned int decimation_factor = sample_frequence / output_frequence;
// Chebychev2 filter :
//    Parameter :
//        order : 4
//        cut frequency : 58.25 Hz,
//        stop band attenuation : 33.979400086720375 dB,
//        sample frequency : 2.0 kHz
//    Analysis :
//        Maximal expected delay : 11.09 ms
const int order = 4;
RII_filter<REAL_TYPE, order> filter{
    {0.018460825, -0.069107704, 0.10145059, -0.069107704, 0.018460825, }, // b : Numerator
    {1.0, -3.7085981, 5.16765, -3.2060149, 0.7471194, } // a : Denominator
};

const unsigned int queue_capacity = 4; // have to be a power of two
Decimation_queue<REAL_TYPE, queue_capacity> output_queue(decimation_factor);
Decimation_queue<REAL_TYPE, queue_capacity> output_queue_raw(decimation_factor);
```

(version longue)

Implémenation du filtre numérique avec interruption pour Arduino Uno R3/Nano/Mega - 3/4

```
const int nb_bits_adc = 10; // Avalaible values for Arduino Uno R4 : 10, 11, 12, 13 or 14
const int max_adc_value = (1<<nb_bits_adc) - 1;

void make_a_sample() {
    REAL_TYPE raw_value = analogRead(A0);
    raw_value *= (((REAL_TYPE) 1.0)/max_adc_value);
    filter.append(raw_value);
    output_queue.append(filter.get_value());
    output_queue_raw.append(raw_value);
}

unsigned int new_time, last_time;

void setup() {
  Serial.begin(115200);
  ITimer1.init();
  if( !ITimer1.attachInterrupt(sample_frequence, make_a_sample)){
    Serial.println(F("Fail to set ITimer1. Select another freq. or timer"));
  }
  filter.reset();
  last_time = micros();
}
```

code/filtre/filter_signal_with_interruption_unor3_nano_mega.cpp

(version longue)

Implémenation du filtre numérique avec interruption pour Arduino Uno R3/Nano/Mega - 4/4

```
REAL_TYPE output_value, output_value_raw;
bool loss_data, loss_data_raw;
const unsigned int sample_period = 1000000 / sample_frequence;

void loop(){
  new_time = micros();
  if( new_time - last_time > (sample_period/4) ){
    noInterrupts();
    bool have_new_value = output_queue.get_avalaible_value(output_value, loss_data);
    output_queue_raw.get_avalaible_value(output_value_raw, loss_data_raw);
    interrupts();
    if( have_new_value ){
      Serial.print(output_value, 6); Serial.print(", "); Serial.println(output_value_raw, 6);
      if(loss_data) Serial.println("Data lost");
    }
    last_time = new_time;
  }
}
```

code/filtre/filter_signal_with_interruption_unor3_nano_mega.cpp

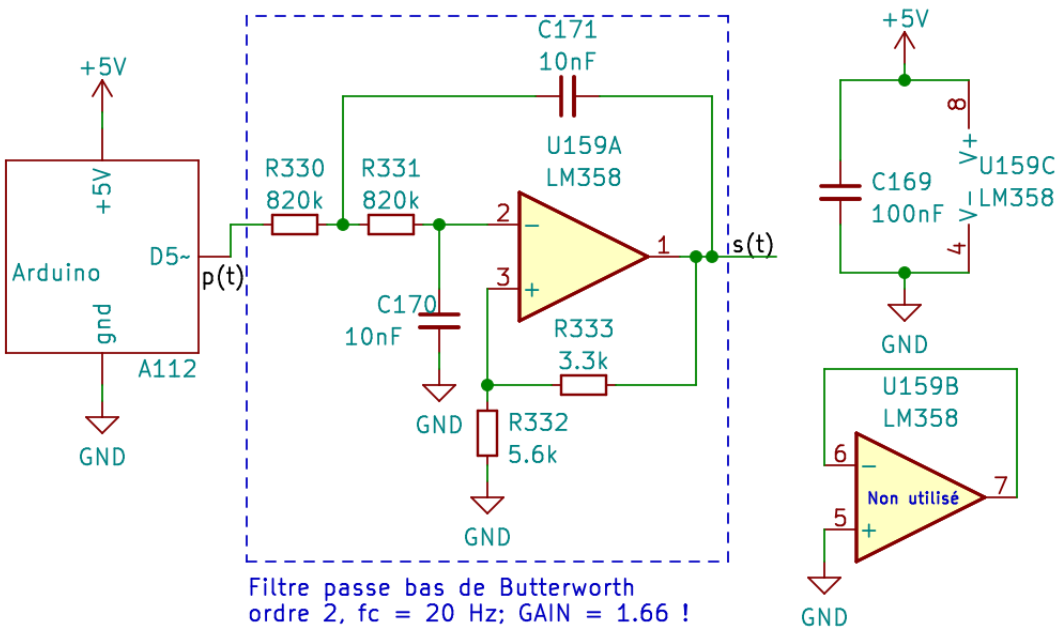
(version longue)

Plan

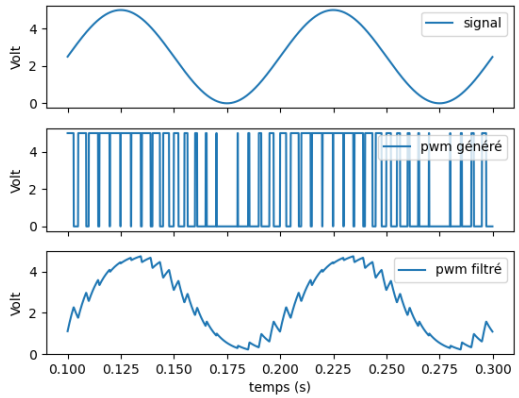
- Références
- La théorie et la chaîne de filtrage
- Exemple : dimensionner une chaîne de filtrage pour le signal d'un Joystick
- La synthèse et l'implémentation des filtres analogiques
- La synthèse des filtres numériques
- L'implémentation des filtres numériques
- Exemple : Générer des signaux analogiques avec des PWMs
- Exemple : limiter l'accélération d'un moteur

Produire des signaux analogiques à partir de PWMs

L'idée est de produire un signal analogique $s(t)$ de contenu fréquentiel maximal f_{max} à partir d'un signal $p(t)$ en PWM de rapport cyclique $r(t) \approx s(t)$ et de fréquence $f_{pwm} >> f_{max}$ que l'on filtre à l'aide d'un filtre passe bas de fréquence de coupure f_c où $f_{pwm} > f_c > f_{max}$. Voici un schéma électronique possible :



Attention : comme l'AOP est alimenté de 0V à 5V, il ne peut générer que des signaux de tension compris entre 20 mV et 3.5V ! Il faut donc que vous génériez des PWMs pour des signaux entre 20mv/1.66 et 2.7V/1.66.



Cette figure montre le résultat d'un autre circuit, mal dimensionné, mais qui permet de voir correctement le fonctionnement du filtre et de la pwm.

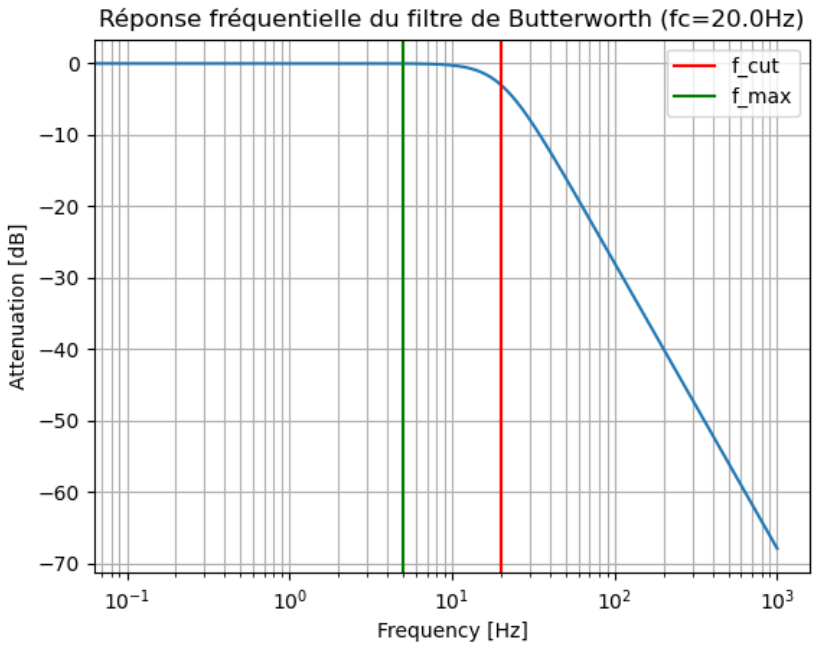
L'arduino génére une pwm de fréquence f_{pwm} où l'on met à jour le rapport cyclique à une fréquence $f_{maj} << f_{pwm}$ de sorte que son rapport cyclique $r(t_k)$ soit égal au signal à produire, à savoir $s(t_k)$ avec $t_k = k/f_{maj}$ et $k \in \mathbb{N}$.

Lorsque l'on filtre cette pwm à l'aide d'un filtre passe-bas de fréquence de coupure f_c avec $f_{pwm} >> f_c > f_{max}$ on obtient en sortie le signal souhaité $s(t)$ avec du bruit qui dépend du filtre et des différents choix de fréquences.

La slide suivante donne une procédure pour dimensionner correctement ce circuit. (version longue)

Procédure pour dimnentionner le générateur de signaux analogiques

1. on évalue le contenu fréquentiel maximal du signal s(t) (avec une transformée de fourrier).
   Par exemple, si $s(t) = 1V \times \sin(2\pi \cdot 5\text{Hz} \cdot t) + 2V \times \cos(2\pi \cdot 1\text{Hz} \cdot t)$ alors $f_{max} = 5\text{Hz}$.



2. on choisit un filtre, avec son ordre et sa fréquence de coupure f_c telle que $f_{max} < f_c$.
   Avec un filtre de Butterworth d'ordre 2 et de fréquence de coupure $f_c = 20\text{Hz}$ on obtient le diagramme de bode ci-contre et le contenu fréquentiel de s à f_{max} sera atténué d'un facteur > 0.99.
3. On détermine le nombre de points nécessaires pour approximer $s(t)$ avec $r(t)$. D'après Shanon, $f_{maj} > 2 \cdot f_{max}$. En pratique, on prend une marge, à savoir, $f_{maj} > 10 \times f_{max}$.
   Par exemple, on choisit $f_{maj} = 20 \cdot f_{max} = 100\text{Hz}$.
4. on fixe une amplitude de bruit b qui sera produit par le filtrage de la pwm.
   Par exemple, un besoin pourrait fixer le bruit maximal à l'amplitude $b = 100\text{mV}$.
5. On choisit la fréquence de la PWM de sorte que $f_{pwm} >> f_{maj}$ et que l'atténuation du filtre à f_{max} transforme une ampitude de Vcc à b.
   Par exemple, on choisit d'une part $f_{pwm} > 20 \cdot f_{maj} = 2\text{kHz}$. D'autre part, l'atténuation recherchée à f_{pwm} est $-20 \cdot \log_{10}(100\text{mV}/5\text{V}) = 34\text{dB}$. Le diagramme de Bode donne $f_{pwm} > 142\text{Hz}$. On choisit donc $f_{pwm} = 2\text{kHz}$.

(version longue)

Code pour générer le circuit électronique du filtre analogique

```
from matplotlib import pyplot as plt
import numpy as np
from scipy import signal
import filter_tools

Hz = 1; kHz = 10**3 * Hz

order = 2; f_max = 5*Hz; f_cut = 20 * Hz; w_cut = 2 * np.pi * f_cut
b, a = signal.butter(order, w_cut, 'low', analog=True, output='ba')

w, h = signal.freqs(b, a, np.arange(0, 2*np.pi*1000, 2*np.pi*.1))
plt.semilogx(w/(2*np.pi), 20 * np.log10(abs(h)))
plt.title(f"Butterworth filter frequency respons (fc={f_cut/Hz}Hz)")
plt.xlabel('Frequency [Hz]'); plt.ylabel('Attenuation [dB]')
plt.grid(which='both', axis='both')
plt.axvline(f_cut, color='red', label=f"f_cut")
plt.axvline(f_max, color='green', label=f"f_max")
plt.legend()
plt.show()

zeros, poles, k = signal.butter(
    order, w_cut, 'low', analog=True, output='zpk'
)
resistors=filter_tools.E24_minus_12
capacitors=filter_tools.C1
filter_tools.print_low_pass_filter(poles, resistors, capacitors)
```

code/filtre/generation_signal_pwm/filter_synthesis_signal_generation.py

Ce code calcule d'abord le numérateur et le dénominateur de la fonction de transfert du filtre analogique (a/b sont les coefficients des polynomes du dénominateur/numérateur de la fonction de transfert).

Ensuite, il dessine son diagramme de bode.

Enfin, il produit en ASCII, sur le terminal, le schéma électronique du filtre.

Codes pour générer la pwm

Pour générer la PWW à 4 kHz, on utilise la section 430.

Pour mettre à jour son rapport cyclique à 100 Hz, on utilise des interruptions vues à la section 416.

Dans l'archive code.zip, vous trouverez une implémentation :

- pour esp32, dans le fichier :
  code/filtre/generation_signal_pwm/generate_signal_with_pwm_esp32.cpp
- pour Arduino Uno R3/Leonardo/nano, dans le fichier :
  A FAIRE :
- pour Arduino Uno R4, dans le fichier :
  A FAIRE :

Plan

Limiter l'accélération d'un moteur – synthèse du filtre

Lorsque l'on controle un moteur, la consigne envoyée par l'utilistateur ne peut pas être appliquée directement car cela impose des courants trop importants pour les drivers et des accélérations trop fortes pour la mécanique.

La solution consiste à lisser les courbes. Généralement, on utilise des interpolations polynomiales (trapèze, courbe de beziers), mais ces méthodes sont compliquée à implémenter.

Une solution plus simple consiste à filtrer la consigne utilisateur à l'aide d'un filtre digital contenant N filtres RC en séries, où N est l'ordre du filtre final.

Pour faire cela, on utilise le code de droite pour synthétiser les paramètres b et a du filtre digital.

On choisit la fréquence de coupure et l'ordre, de façon à limiter l'accélération selon les besoins de l'application en observant la réponse du filtre à une consigne sous la forme d'un heaviside (le pire cas) comme le montre la slide suivante.

code/filtre/limiter_acceleration_moteur/synthesis_digital_filters_to_limit_motor_acceleration.py

```
from scipy import signal
import numpy as np
from matplotlib import pyplot as plt

Hz=1
sample_freq = 200 * Hz
real_type = np.float32

# Synthèse d'un simple filtre digital RC
f_cut = 5 * Hz
zero, pole, gain = signal.butter(
    1, f_cut, analog=False, output='zpk', fs=sample_freq
)

# Filtre contenant 3 filtrs RC en série
order = 3
final_zeros = np.hstack( order*[zero] );
final_poles = np.hstack( order*[pole] );
final_gain = gain**order
final_b, final_a = real_type(
    signal.zpk2tf(final_zeros, final_poles, final_gain)
)

print(f"Numérateur du filtre (b) : {list(final_b)}")
print(f"Dénominateur du filtre (a) : {list(final_a)}")
```

(version longue)

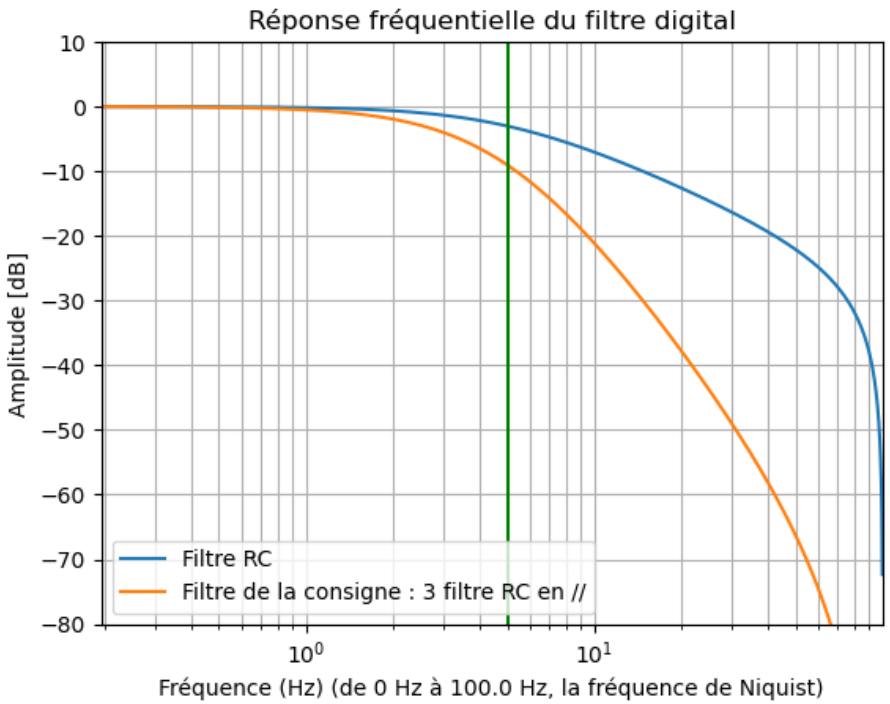
Limiter l'accélération d'un moteur – réponse à un Heaviside

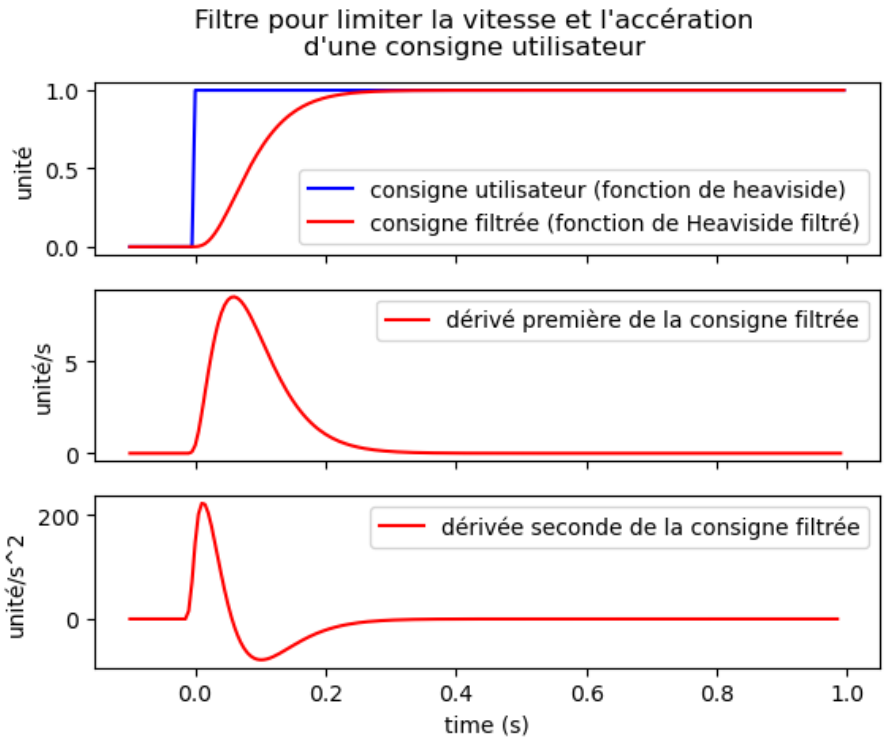
Si vous éxecutez le prgramme python du fichier suivant :
code/filtre/code/filtre/limiter_acceleration_moteur/
synthesis_digital_filters_to_limit_motor_acceleration.py

vous obtenez, sur le terminal, les coefficients du numérateur et dénominateur du filtre digital qui sont :

$b = [0.0003883724, 0.0011651171, 0.0011651171, 0.0003883724]$ (numerateur)
$a = [1.0, -2.562242, 2.1883614, -0.6230124]$ (dénominateur).

Puis, vous obtenez, à gauche, le diagramme de bode du filtre et à droite, la réponse à une consigne utilisateur sous la forme d'une fonction heaviside.





(version longue)

Limiter l'accélération d'un moteur : implémentation du filtre

Il ne vous reste plus qu'à implémenter le filtre sur arduino de la manière suivante (en utilisant les bibliothèques code/filtre/filter.hpp et code/filtre/queue.hpp) :

```
#include "filter.hpp"

typedef float REAL_TYPE;
const unsigned int sample_frequence = 200; // In Hz
const int order = 3;
RII_filter<REAL_TYPE, order> filter{
    {0.0003883724, 0.0011651171, 0.0011651171, 0.0003883724}, // b : Numerator
    {1.0, -2.562242, 2.1883614, -0.6230124} // a : Denominator
};

void setup() {
  filter.reset();
}

unsigned int new_time = 0, last_time = 0;
const unsigned int sample_period = 1000000 / sample_frequence;
void loop(){
  new_time = micros();
  if( new_time - last_time >= sample_period ){

    REAL_TYPE user_consign = 0; // Write here the code to get user consign

    filter.append(user_consign);
    REAL_TYPE filtered_consign = filter.get_value();

    last_time = new_time;
  }
}
```

code/filtre/limiter_acceleration_moteur/filtrer_consigne_utilisateur.cpp

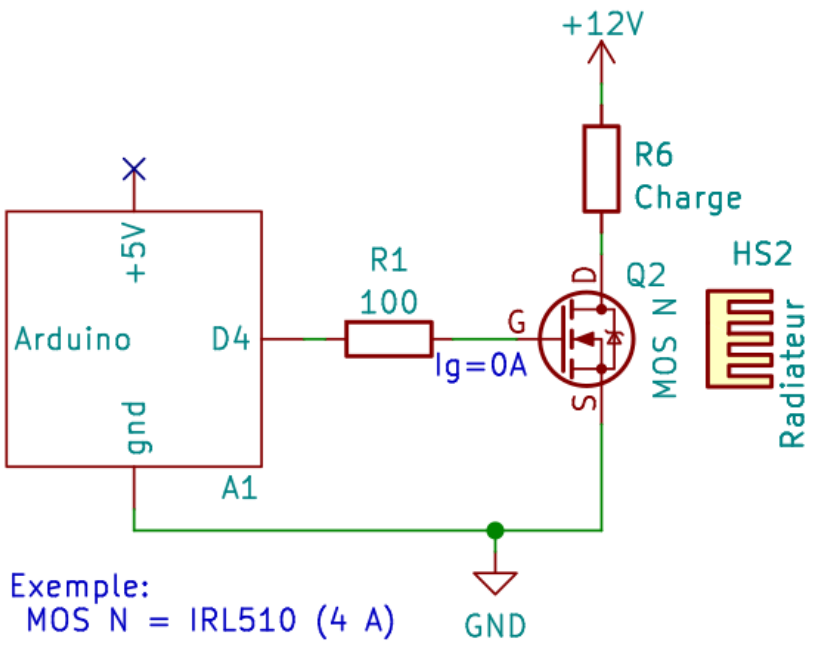
(version longue)

Plan

(présent dans la version courte)

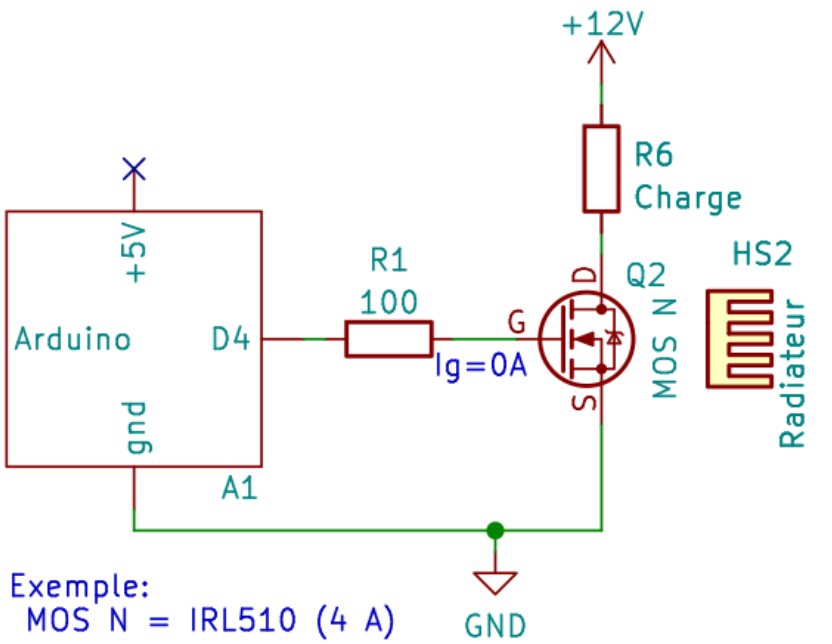
Arduino - Commander un MOSFET canal N

S=source G = grille D = Drain



Arduino - Commander un MOSFET canal N

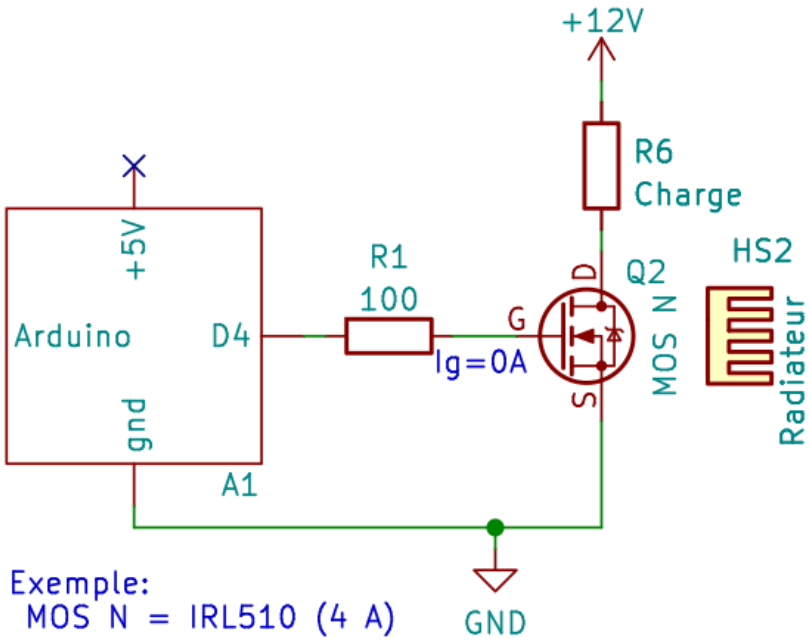
S=source G = grille D = Drain



MOSFET N = LA résistance entre le drain et la source est ajustable avec la tension de la grille.

Arduino - Commander un MOSFET canal N

S=source G = grille D = Drain



MOSFET N = LA résistance entre le drain et la source est ajustable avec la tension de la grille.

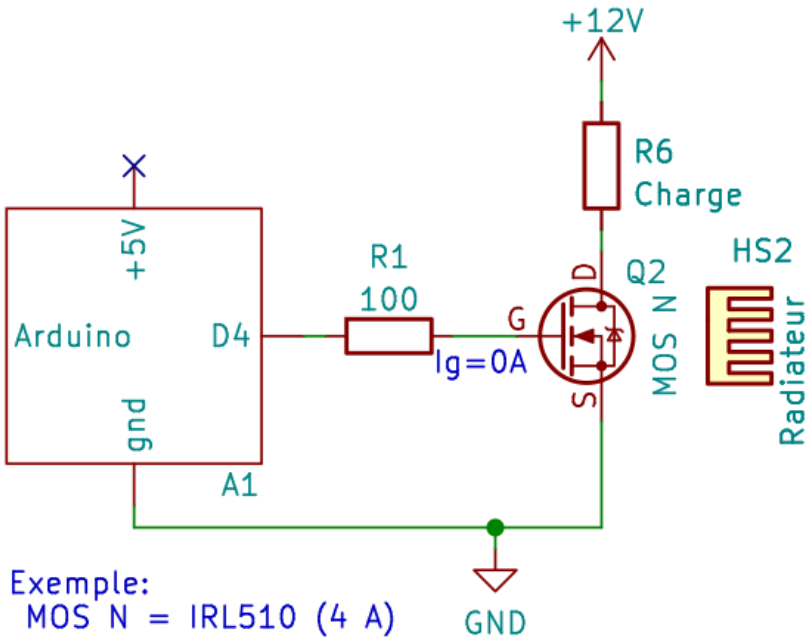
si $V_{gs} \leq 0 \quad \Rightarrow \quad R_{DS,\text{off}} > 400k\Omega$

si $V_{gs} \geq V_{\text{seuil}} \Rightarrow \quad R_{DS,\text{on}} < 2\Omega$

(présent dans la version courte)

Arduino - Commander un MOSFET canal N

S=source G = grille D = Drain



MOSFET N = LA résistance entre le drain et la source est ajustable avec la tension de la grille.

si $V_{gs} \leq 0 \quad \Rightarrow \quad R_{DS,\text{off}} > 400k\Omega$

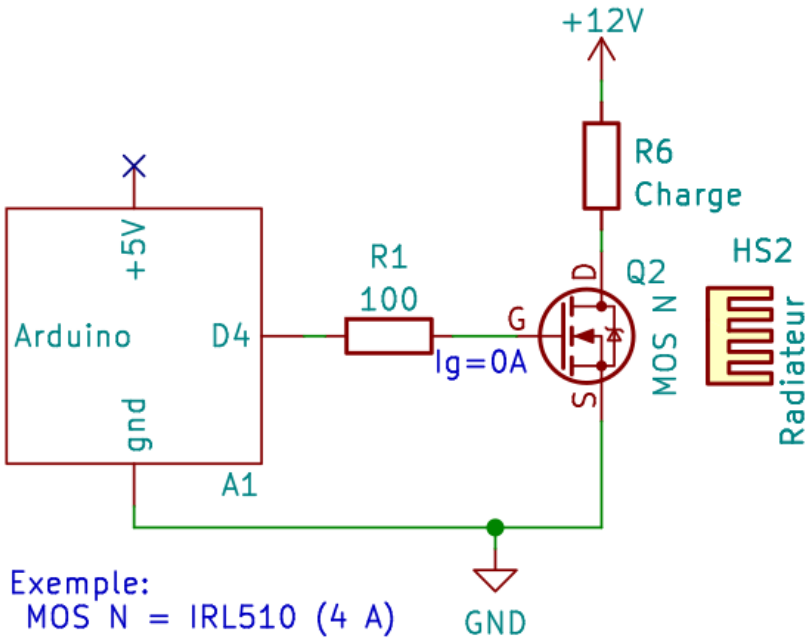
si $V_{gs} \geq V_{\text{seuil}} \Rightarrow \quad R_{DS,\text{on}} < 2\Omega$

La tension de grille ne nécessite pas de courant : $I_g \approx 0$.

(présent dans la version courte)

Arduino - Commander un MOSFET canal N

S=source G = grille D = Drain



MOSFET N = LA résistance entre le drain et la source est ajustable avec la tension de la grille.

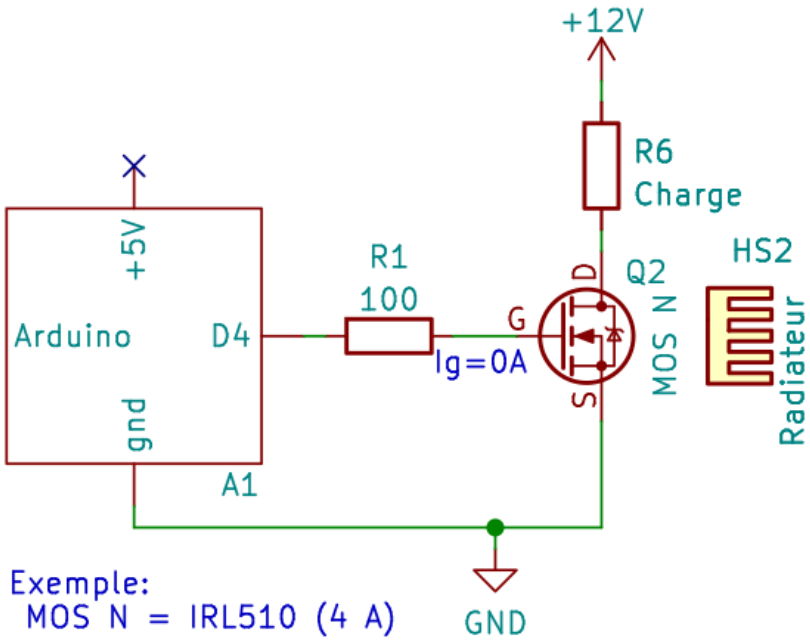
si $V_{gs} \leq 0 \quad \Rightarrow \quad R_{DS,\text{off}} > 400k\Omega$
si $V_{gs} \geq V_{\text{seuil}} \Rightarrow \quad R_{DS,\text{on}} < 2\Omega$

La tension de grille ne nécessite pas de courant : $I_g \approx 0$.

La résistance $R1$ protège $D4$ des appels de courant provoqués par le condensateur parasite entre la source S et la grille G.

Arduino - Commander un MOSFET canal N

S=source G = grille D = Drain



MOSFET N = LA résistance entre le drain et la source est ajustable avec la tension de la grille.

si $V_{gs} \leq 0 \quad \Rightarrow \quad R_{DS,\text{off}} > 400k\Omega$
si $V_{gs} \geq V_{\text{seuil}} \Rightarrow \quad R_{DS,\text{on}} < 2\Omega$

La tension de grille ne nécessite pas de courant : $I_g \approx 0$.

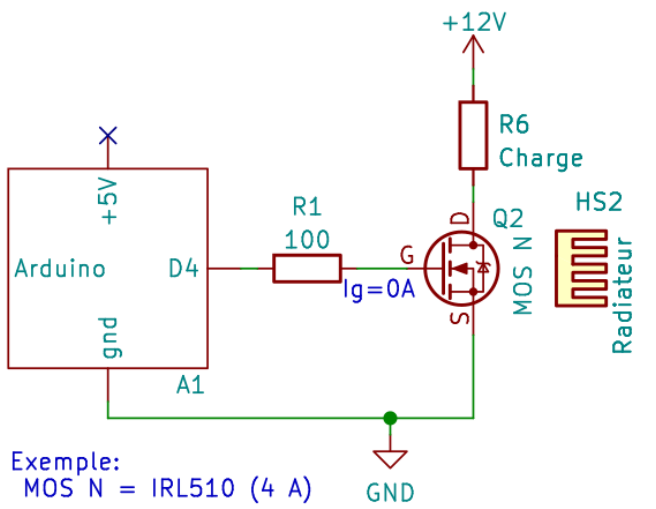
La résistance $R1$ protège $D4$ des appels de courant provoqués par le condensateur parasite entre la source S et la grille G.

Comment fonctionne les MOSFETs : www.youtube.com/watch?v=lyfx8CL7AkI
(consulté le 21/03/2025)

Arduino - Dissipation thermique sans radiateur !

Mosfet : IRL501
(boîtier TO220)
$R_{DS,on} = 0.54\Omega$

$I_{charge} = 2A$

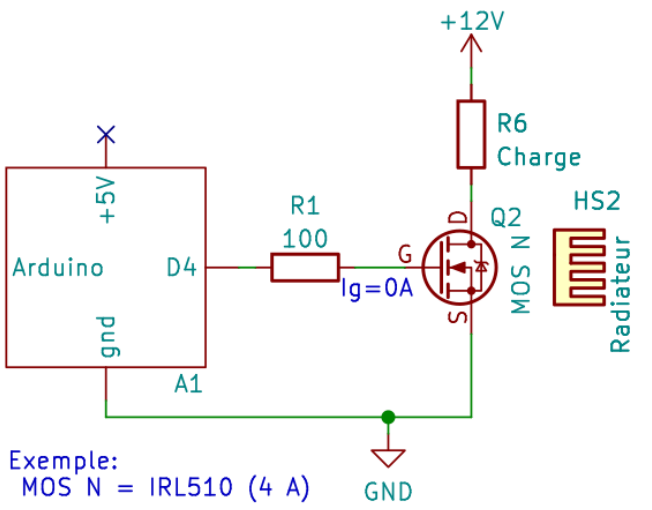


Arduino - Dissipation thermique sans radiateur !

Mosfet : IRL501
(boîtier TO220)
$R_{DS,on} = 0.54\Omega$

$V_{gs} = +5V \quad \Rightarrow \quad R_{DS} = 0.54\Omega.$

$I_{charge} = 2A$



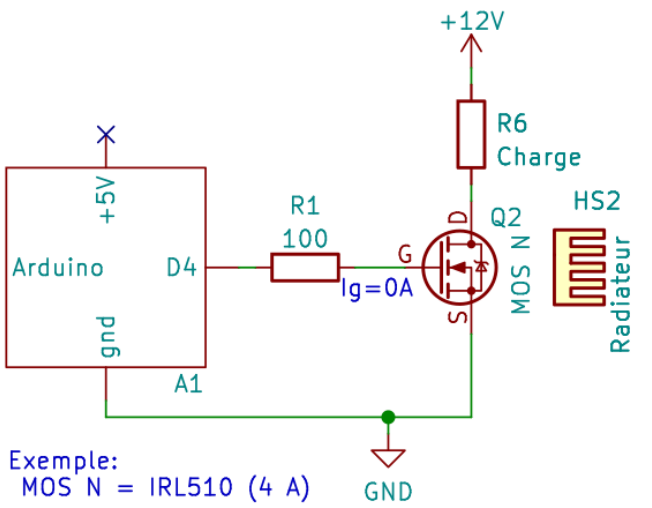
Arduino - Dissipation thermique sans radiateur !

Mosfet : IRL501
(boîtier TO220)
$R_{DS,on} = 0.54\Omega$

$V_{gs} = +5V \quad \Rightarrow \quad R_{DS} = 0.54\Omega.$

Loi d'ohms : $U_{DS} = R_{DS} \times I = .54 \times 2 \approx 1V.$

$I_{charge} = 2A$



Arduino - Dissipation thermique sans radiateur !

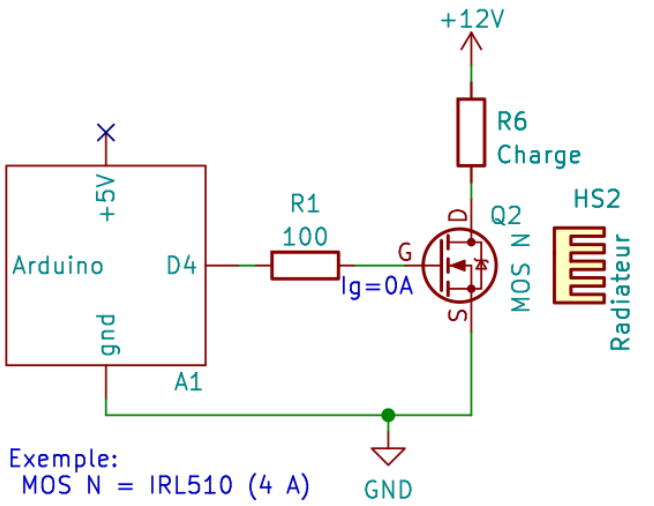
Mosfet : IRL501
(boîtier TO220)
$R_{DS,on} = 0.54\Omega$

$V_{gs} = +5V \quad \Rightarrow \quad R_{DS} = 0.54\Omega.$

Loi d'ohms : $U_{DS} = R_{DS} \times I = .54 \times 2 \approx 1V$.

Puissance dissipée par le mosfet :
$P_{mos} = U_{DS} \times I \approx 1V \times 2A = 2W$

$I_{charge} = 2A$



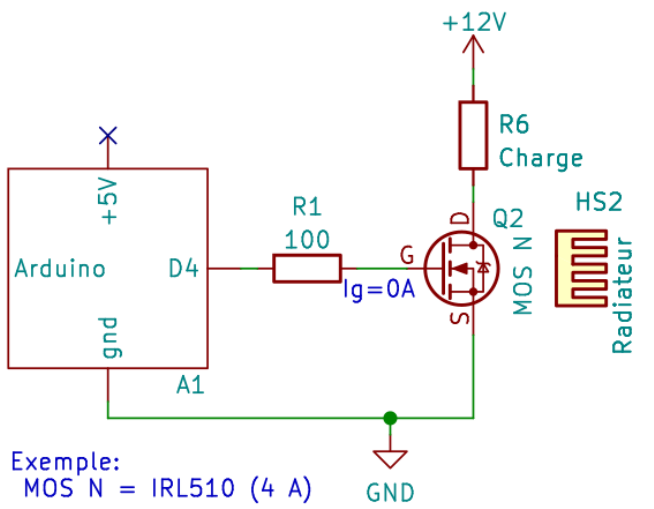
Arduino - Dissipation thermique sans radiateur !

Mosfet : IRL501
(boîtier TO220)
$R_{DS,on} = 0.54\Omega$

$R_{\text{jonction,boîtier}} = 2.5^o C/W$

$R_{\text{boîtier,air}} = 60^o C/W$

$I_{charge} = 2A$



$V_{gs} = +5V \quad \Rightarrow \quad R_{DS} = 0.54\Omega.$

Loi d'ohms : $U_{DS} = R_{DS} \times I = .54 \times 2 \approx 1V$.

Puissance dissipée par le mosfet :
$P_{mos} = U_{DS} \times I \approx 1V \times 2A = 2W$

Résistance thermique :
$R_{th} = R_{\text{jonction,boîtier}} + R_{\text{boîtier,air}} = (2.5 + 60)^o C/W$

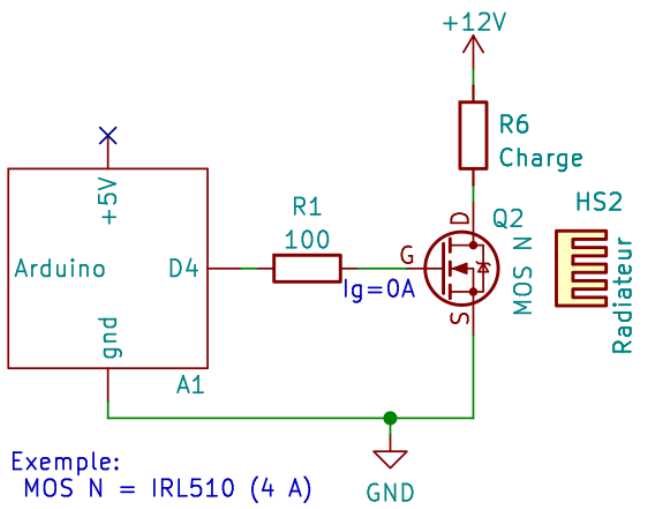
Arduino - Dissipation thermique sans radiateur !

Mosfet : IRL501
(boîtier TO220)
$R_{DS,on} = 0.54\Omega$

$R_{\text{jonction,boîtier}} = 2.5^o C/W$

$R_{\text{boîtier,air}} = 60^o C/W$

$I_{charge} = 2A$



$V_{gs} = +5V \quad \Rightarrow \quad R_{DS} = 0.54\Omega.$

Loi d'ohms : $U_{DS} = R_{DS} \times I = .54 \times 2 \approx 1V$.

Puissance dissipée par le mosfet :
$P_{mos} = U_{DS} \times I \approx 1V \times 2A = 2W$

Résistance thermique :
$R_{th} = R_{\text{jonction,boîtier}} + R_{\text{boîtier,air}} = (2.5 + 60)^o C/W$

Température du mosfet:
$T = T_{amb} + R_{th} \times P_{mos} \approx 40^o C + 62.5^o C/W \times 2W = 165^o C$

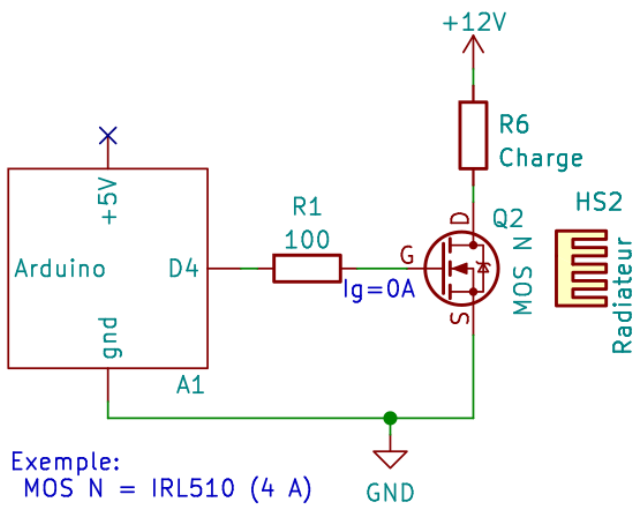
Arduino - Dissipation thermique sans radiateur !

Mosfet : IRL501
(boîtier TO220)
$R_{DS,on} = 0.54\Omega$

$R_{\text{jonction,boîtier}} = 2.5^o C/W$

$R_{\text{boîtier,air}} = 60^o C/W$

$I_{charge} = 2A$



$V_{gs} = +5V \quad \Rightarrow \quad R_{DS} = 0.54\Omega.$

Loi d'ohms : $U_{DS} = R_{DS} \times I = .54 \times 2 \approx 1V$.

Puissance dissipée par le mosfet :
$P_{mos} = U_{DS} \times I \approx 1V \times 2A = 2W$

Résistance thermique :
$R_{th} = R_{\text{jonction,boîtier}} + R_{\text{boîtier,air}} = (2.5 + 60)^o C/W$

Température du mosfet:
$T = T_{amb} + R_{th} \times P_{mos} \approx 40^o C + 62.5^o C/W \times 2W = 165^o C$

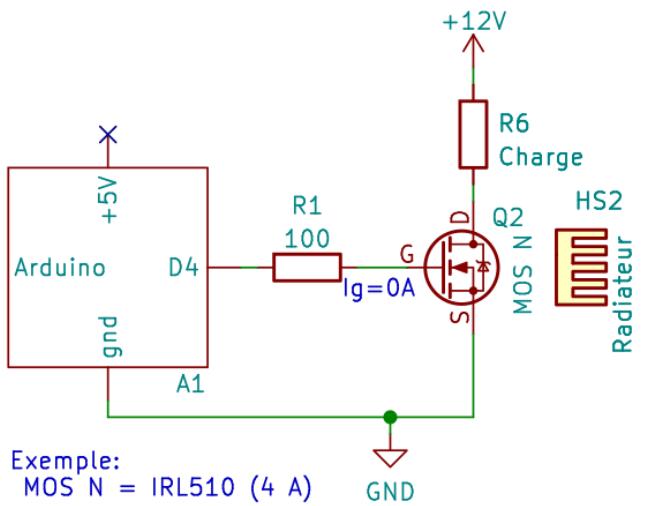
Le MOSFET brûle ! Il faut un radiateur.

Arduino - Dissipation thermique avec radiateur !

Mosfet : IRL501
(boîtier TO220)
$R_{DS,on} = 0.54\Omega$

$R_{\text{jonction,boîtier}} = 2.5^o C/W$

$I_{charge} = 2A$



$V_{gs} = +5V \quad \Rightarrow \quad R_{DS} = 0.54\Omega.$

Loi d'ohms : $U_{DS} = R_{DS} \times I = .54 \times 2 \approx 1V.$

Puissance dissipée par le mosfet :
$P_{mos} = U_{DS} \times I \approx 1V \times 2A = 2W$

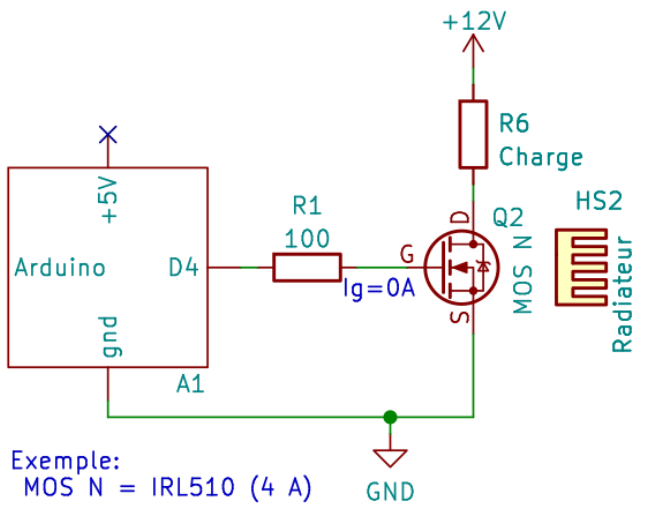
Arduino - Dissipation thermique avec radiateur !

Mosfet : IRL501
(boîtier TO220)
$R_{DS,on} = 0.54\Omega$

$R_{\text{jonction,boîtier}} = 2.5^o C/W$

$R_{\text{radiateur}} = 12^o C/W$

$I_{charge} = 2A$



$V_{gs} = +5V \quad \Rightarrow \quad R_{DS} = 0.54\Omega.$

Loi d'ohms : $U_{DS} = R_{DS} \times I = .54 \times 2 \approx 1V.$

Puissance dissipée par le mosfet :
$P_{mos} = U_{DS} \times I \approx 1V \times 2A = 2W$

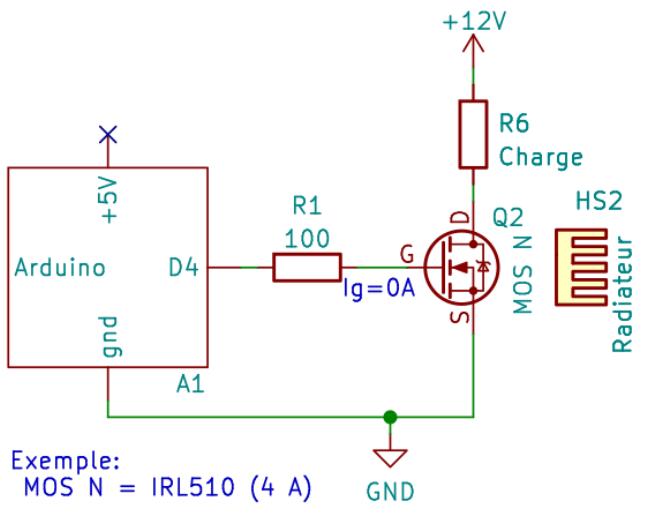
Arduino - Dissipation thermique avec radiateur !

Mosfet : IRL501
(boîtier TO220)
$R_{DS,on} = 0.54\Omega$

$R_{\text{jonction,boîtier}} = 2.5^o C/W$

$R_{\text{radiateur}} = 12^o C/W$

$I_{charge} = 2A$



$V_{gs} = +5V \quad \Rightarrow \quad R_{DS} = 0.54\Omega.$

Loi d'ohms : $U_{DS} = R_{DS} \times I = .54 \times 2 \approx 1V.$

Puissance dissipée par le mosfet :
$P_{mos} = U_{DS} \times I \approx 1V \times 2A = 2W$

Résistance thermique :
$R_{th} = R_{\text{jonction,boîtier}} + R_{\text{radiateur}} = (2.5+12)^o C/W$

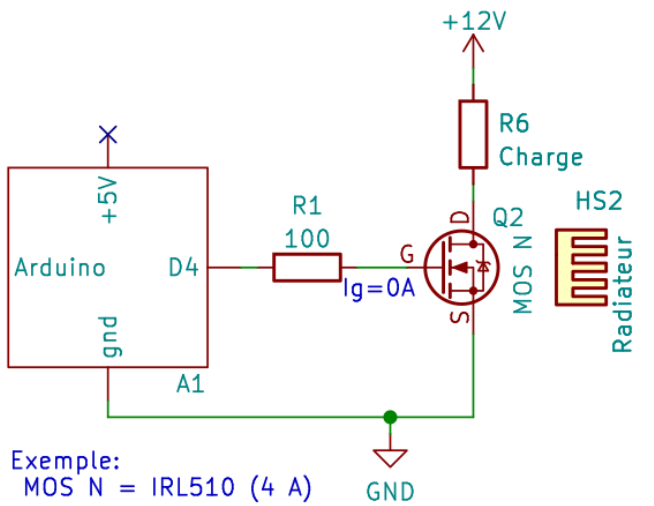
Arduino - Dissipation thermique avec radiateur !

Mosfet : IRL501
(boîtier TO220)
$R_{DS,on} = 0.54\Omega$

$R_{\text{jonction,boîtier}} = 2.5^o C/W$

$R_{\text{radiateur}} = 12^o C/W$

$I_{charge} = 2A$



$V_{gs} = +5V \quad \Rightarrow \quad R_{DS} = 0.54\Omega.$

Loi d'ohms : $U_{DS} = R_{DS} \times I = .54 \times 2 \approx 1V.$

Puissance dissipée par le mosfet :
$P_{mos} = U_{DS} \times I \approx 1V \times 2A = 2W$

Résistance thermique :
$R_{th} = R_{\text{jonction,boîtier}} + R_{\text{radiateur}} = (2.5+12)^o C/W$

Température du mosfet:
$T = T_{amb} + R_{th} \times P_{mos} \approx 40^o C + 14.5^o C/W \times 2W = 68.5^o C$

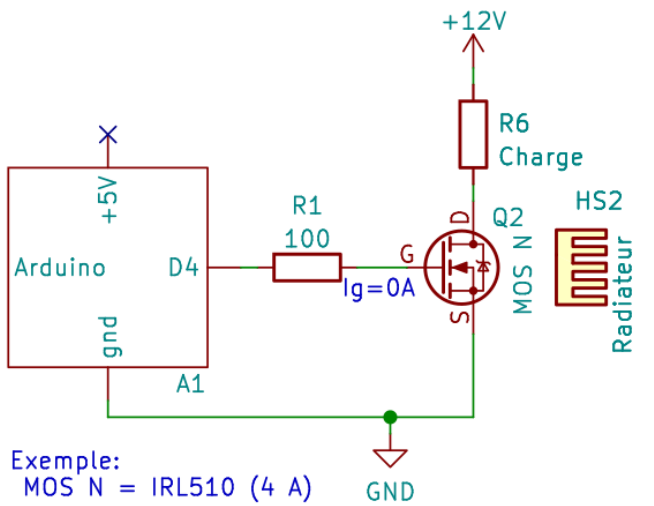
Arduino - Dissipation thermique avec radiateur !

Mosfet : IRL501
(boîtier TO220)
$R_{DS,on} = 0.54\Omega$

$R_{\text{jonction,boîtier}} = 2.5^o C/W$

$R_{\text{radiateur}} = 12^o C/W$

$I_{charge} = 2A$



$V_{gs} = +5V \quad \Rightarrow \quad R_{DS} = 0.54\Omega.$

Loi d'ohms : $U_{DS} = R_{DS} \times I = .54 \times 2 \approx 1V.$

Puissance dissipée par le mosfet :
$P_{mos} = U_{DS} \times I \approx 1V \times 2A = 2W$

Résistance thermique :
$R_{th} = R_{\text{jonction,boîtier}} + R_{\text{radiateur}} = (2.5+12)^o C/W$

Température du mosfet:
$T = T_{amb} + R_{th} \times P_{mos} \approx 40^o C + 14.5^o C/W \times 2W = 68.5^o C$

Le MOSFET se porte bien !

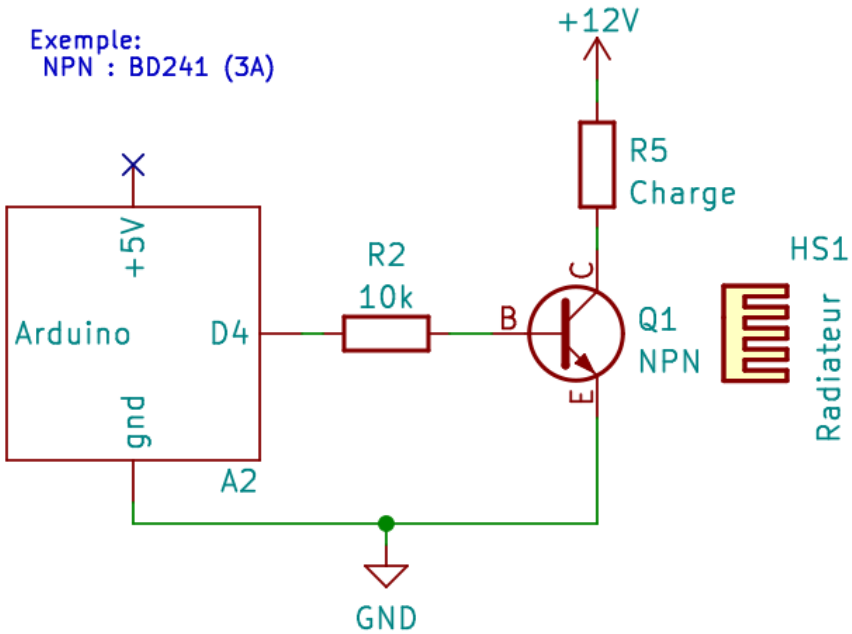
Arduino - Commander un transistor NPN

$$U_{eb} = Vb - Ve$$

Gain du transistor : h_{fe}

Courant de saturation du collecteur : $I_{c,max}$

B = Base C = collecteur E = Émetteur



Polarisation du transistor :

$$\begin{aligned} U_{eb} = 0.6V &\Rightarrow \text{polarisé} \\ U_{eb} < 0.6V &\Rightarrow \text{non polarisé} \end{aligned}$$

Si $D4$ est à 5V $\Rightarrow$ le transistor est polarisé.

Saturation du transistor :

$$h_{fe} \times I_b > I_{c,max} \Rightarrow \text{saturé}$$

Si $R2$ est petite le courant de la base I_b sera grand et le transistor saturé.

État de la jonction Emeteur-colleteur :

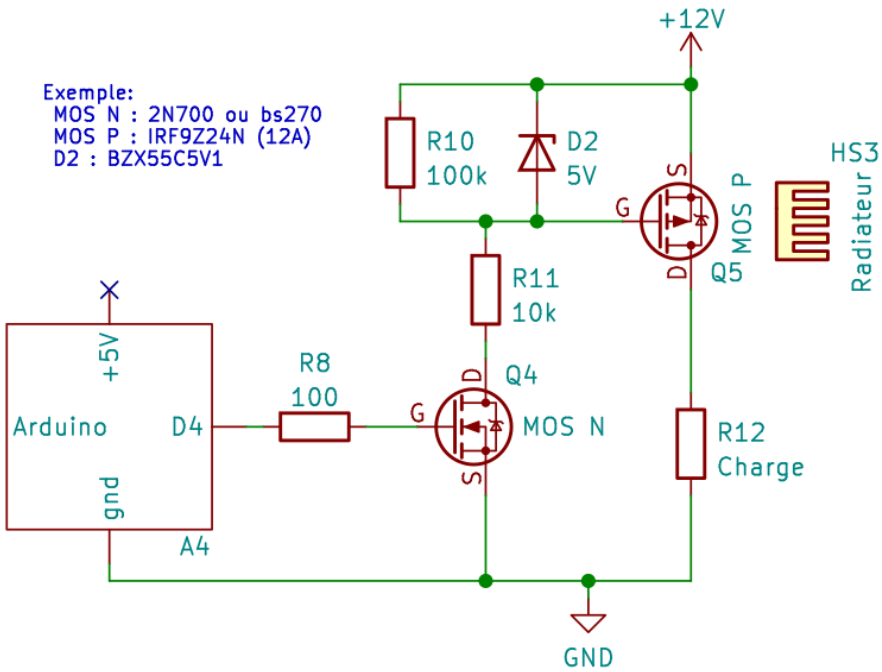
$$\begin{aligned} \text{non polarisé} &\Rightarrow \text{EC est ouvert.} \\ \text{polarisé} &\Rightarrow I_c = h_{fe} \times I_b. \\ \text{polarisé et saturé} &\Rightarrow \text{EC est fermé} \end{aligned}$$

V_{EC} dépend de la charge et du courant de saturation.

Comment fonctionne les transistors BJT : www.youtube.com/watch?v=2uowMENwiHQ (consulté le 21/03/2025)

(version longue)

Arduino - Commander un MOSFET canal P

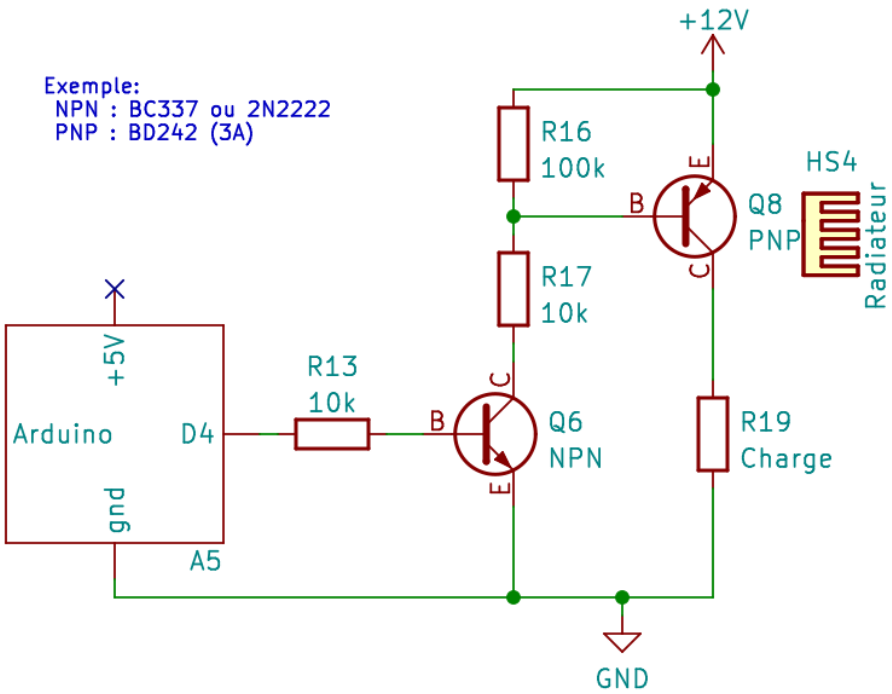


Si $D4 = 0V$, la tension $V_g - V_s$ du mosfet N Q4 vaut 0V. Il est donc ouvert. Grâce à R10, la tension $V_g - V_s$ du Mosfet P Q5 vaut 0V. Il est donc ouvert. La charge est déconnectée.

Si $D4 = 5V$, la tension $V_g - V_s$ du mosfet N Q4 vaut 0V . Il est donc fermé. Grâce à R11, le potentiel V_g du Mosfet P Q5 vaut +5V. La diode zener limite alors la tension et on a $V_g - V_s = -5V < -V_{seuil}$. Il est donc fermé. La charge est connectée au +12V.

La diode zener protège la grille du mosfet canal P. En effet la tension de grille ne doit pas dépasser une valeur dite de claquage.

Arduino - Commander un transistor PNP

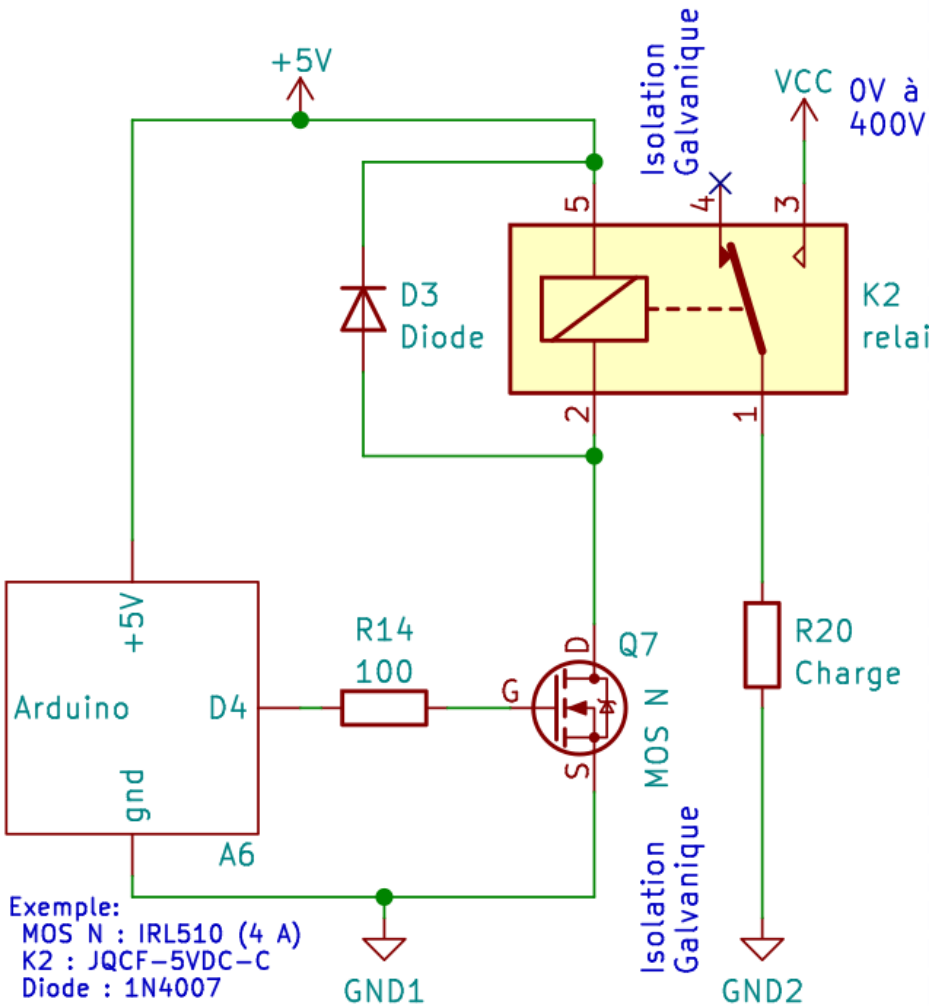


Quand le transistor NPN est ouvert, la résistance R16 impose un tension nulle entre la base et l'émetteur de Q8. Il est donc ouvert. La charge est déconnecté du =12V.

Quand le transistor NPN est fermé, le diviseur de tension R17, R16 polarise le transistor PNP Q8. Il devient fermé. La charge est connecté au +12V.

Il faut prévoir une chute importante de tension entre le collecteur et l'émetteur du transistor Q8, jusqu'à 2V-3V selon les courants.

Arduino - Commander un Relai avec un mosfet N



Un relai contient un bobine. Quand elle est alimentée, un champs magnétique apparaît et actionne un aimant qui fait ouvrir/fermer un interrupteur mécanique.

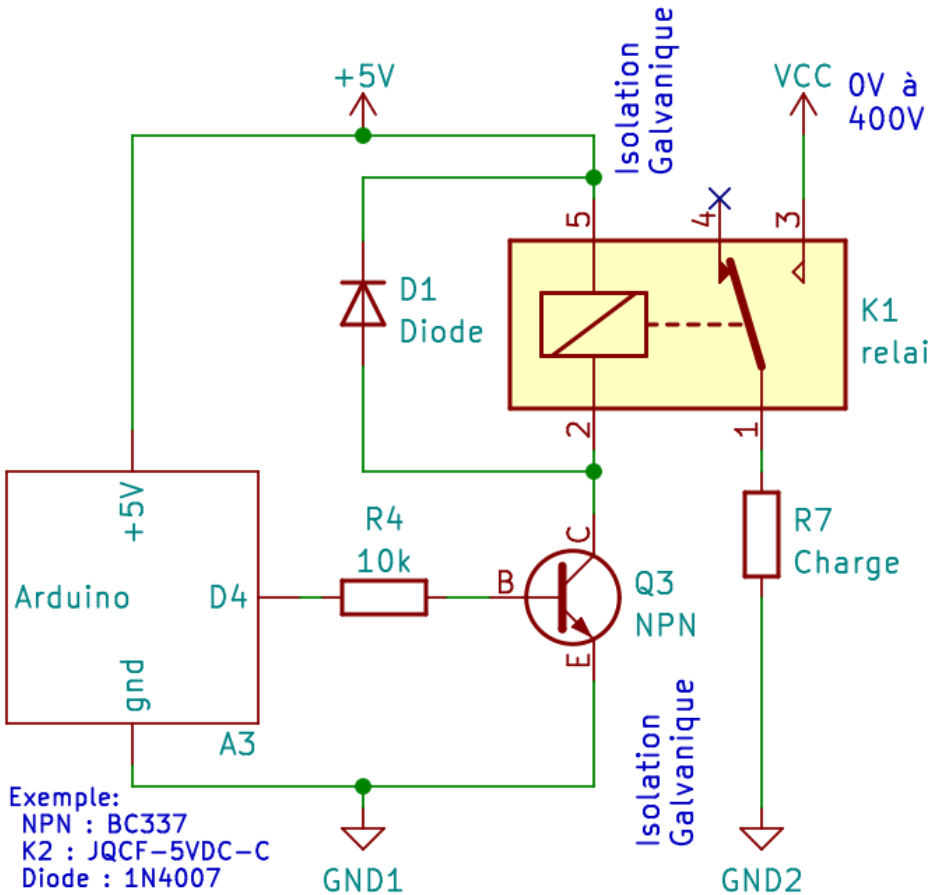
Le courant d'une bobine est continue. Quand on ouvre le MOSFET, le champ magnétique maintient le courant pendant un certain temps.

La diode D3, dite de route libre, sert à évacuer ce courant

Ce circuit permet d'isoler galvaniquement le circuit de contrôle du circuit de l'actionneur. On peut ainsi commander des circuits sous 230V.

(présent dans la version courte)

Arduino - Commander un relai avec un NPN



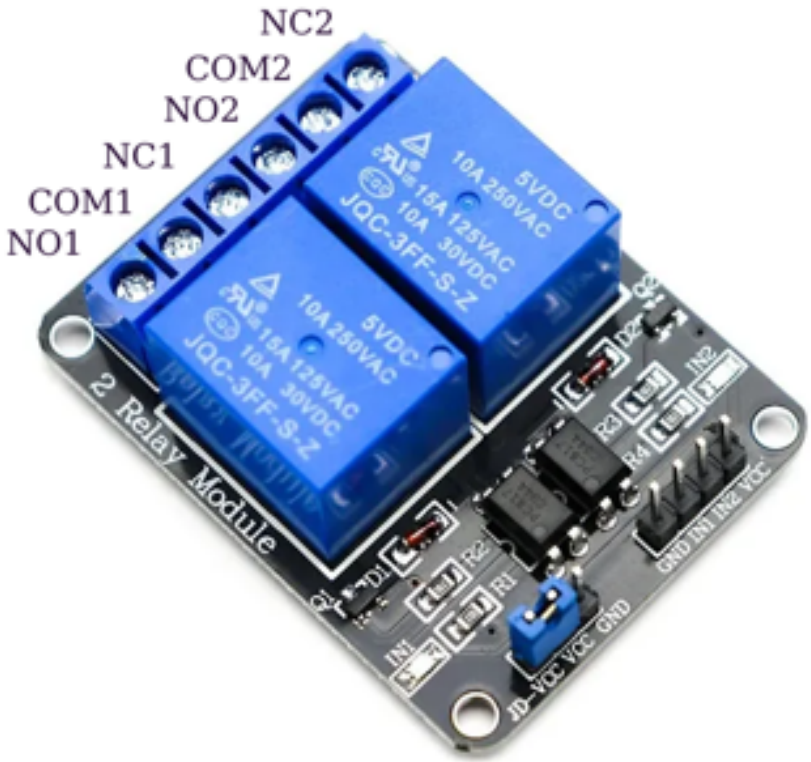
Il s'agit du même montage mais avec un transistor bipolaire NPN au lieu d'un mosfet canal N.

Commander deux relais avec un module arduino - 1/4

Il existe des modules arduinos pour commander entre 1 et 8 relais à la fois.

A droite est présenté un exemple de module qui permet de commander 2 relais.

Pour commander plusieurs modules, il est nécessaire d'utiliser une alimentation externe pour fournir le courant nécessaire pour faire commuter un relai. En effet, chaque relai nécessite entre 80 mA sous 5V et 150 mA sous 3V3 pour commuter.



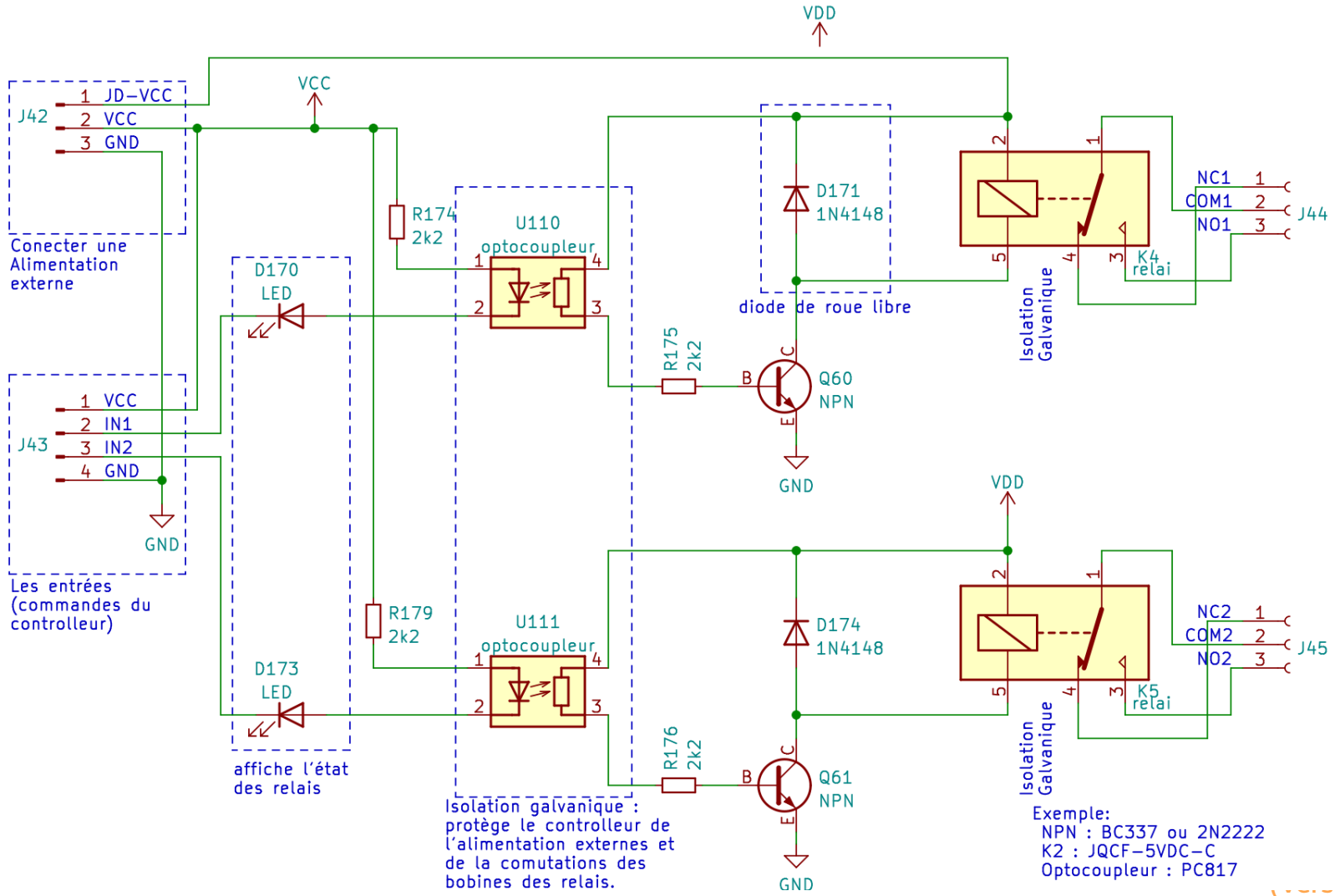
Pour protéger le microcontrolleur de cette alimentation externe et de la commutation des relais, les modules utilisent des optopcoupleurs. Un optocoupleur permet de transmettre un signal d'un circuit électrique à un autre en utilisant la lumière empéchant tout transfert d'energie électrique entre le controlleur et l'actionneur.

Les pages suivantes montrent comment utiliser ces modules et détaille aussi leur fonctionnement.

Schéma électrique du module arduino à deux relais - 2/4

Dans ce schema si on met un jumper entre les broches JD-VCC et VCC du connecteur J42 alors l'alimentation du microcontrolleur (VCC) est utilisée pour commuter les relais.

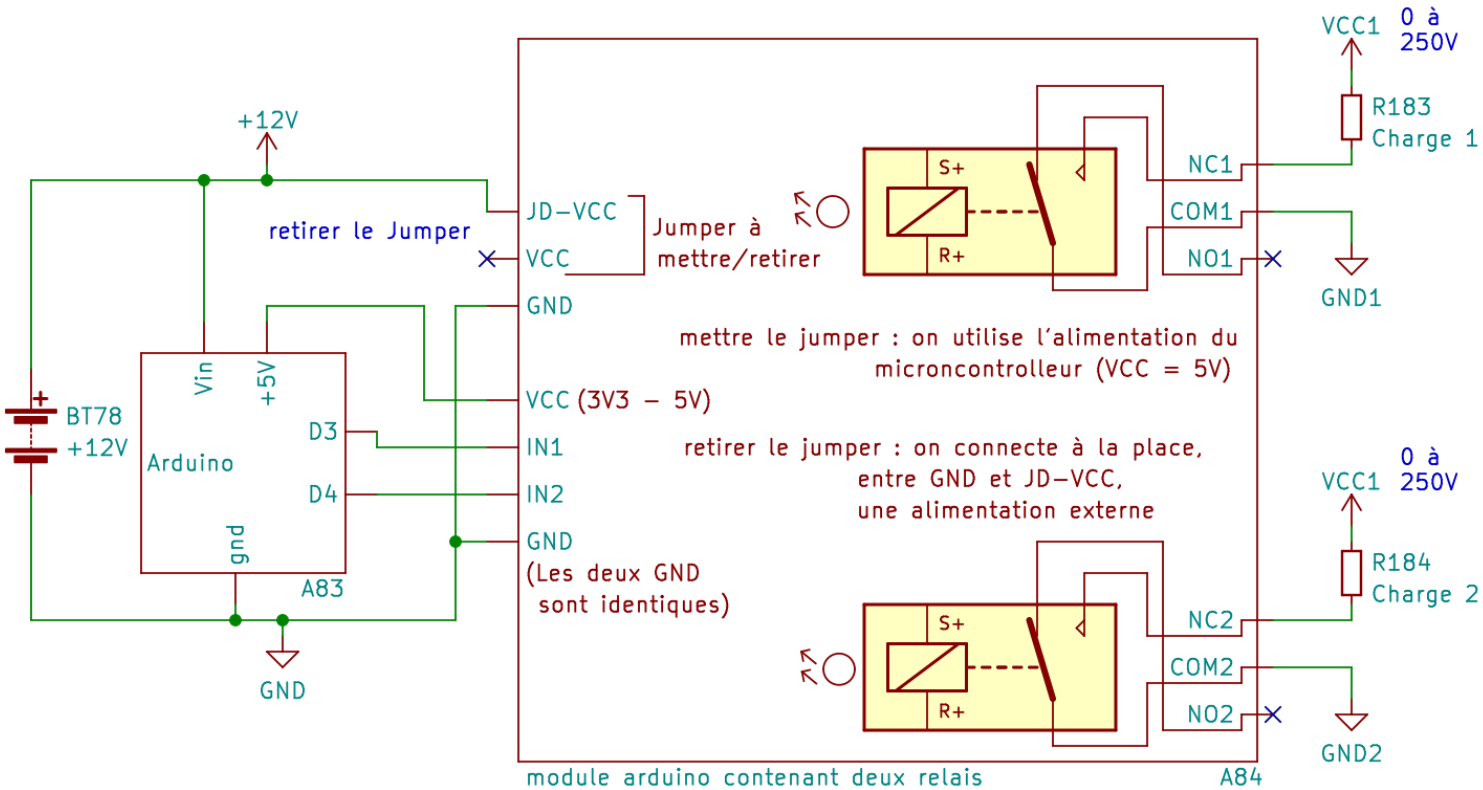
Si on retire ce jumper et que l'on met à la place une alimentation séparée entre le GND et le JD-VCC du connecteur J42 alors les relais sont alimentés uniqument par l'alimentation externe.



(version longue)

Utiliser le module à deux relais avec une alimentation externe - 3/4

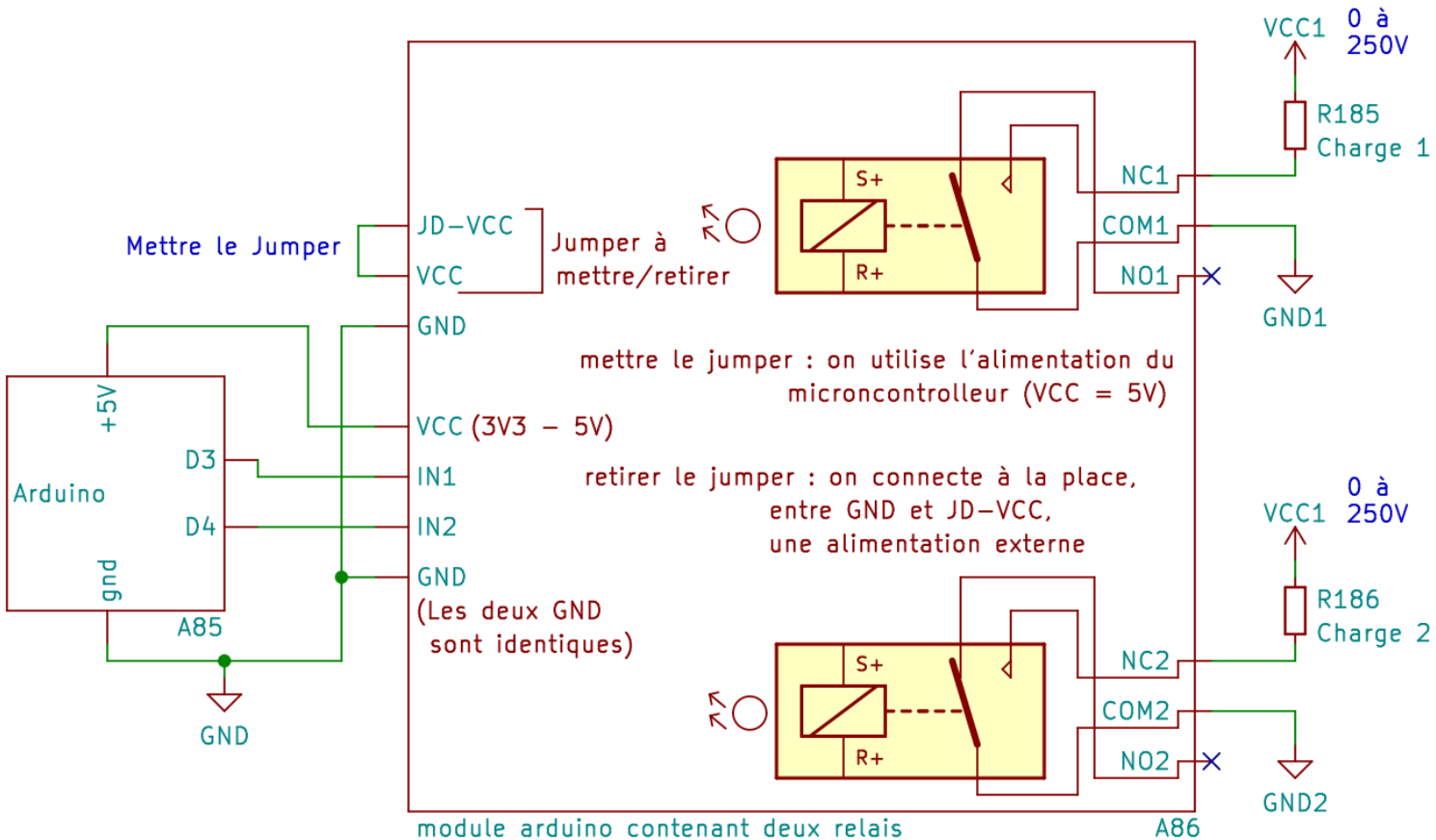
Voici comment utiliser le module avec une alimentation externe de 12V.



(version longue)

Utiliser le module à deux relais sans alimentation externe - 4/4

Voici comment utiliser le module sans alimentation externe. Une carte arduino 5V peut fournir jusqu'à 250 mA. On peut donc alimenter au plus 3 relais en même temps.



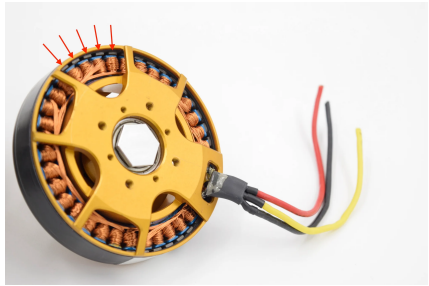
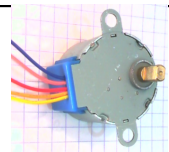
(version longue)

Plan

(présent dans la version courte)

Plan

Présentation des différentes types de moteur

famille	Servomoteur	Brushless	Moteur DC	Pas à pas
				 
Nb fils	3	3	3 (4-rare)	2	4	5
mise en oeuvre	Facile		Difficile	Moyen		Facile
Driver	Intégré	ESC	Ponts en H			ULN2003
Commande	position vitesse (rare)	vite-sse	vitesse position : difficile			position
Protocole	PWM Modélisme		Ad Hoc			
Précision sans capteur	Moyen/bon	aucune				Excellente
Possède une réduction	oui (coupleux)	souvent non		souvent oui		oui (coupleux)

(présent dans la version courte)

Mise en garde sur l'utilisation des moteurs

Les moteur peuvent devenir des générateurs de courant. Leurs utilisations peuvent provoquer des surtensions capables de détruire les circuits électroniques.

Leurs utilisations nécessitent l'ajout de protections electroniques particulières présentées à la page 317 .

Pour les petits moteurs du kit Arduino, ces protections ne sont pas nécessaires.

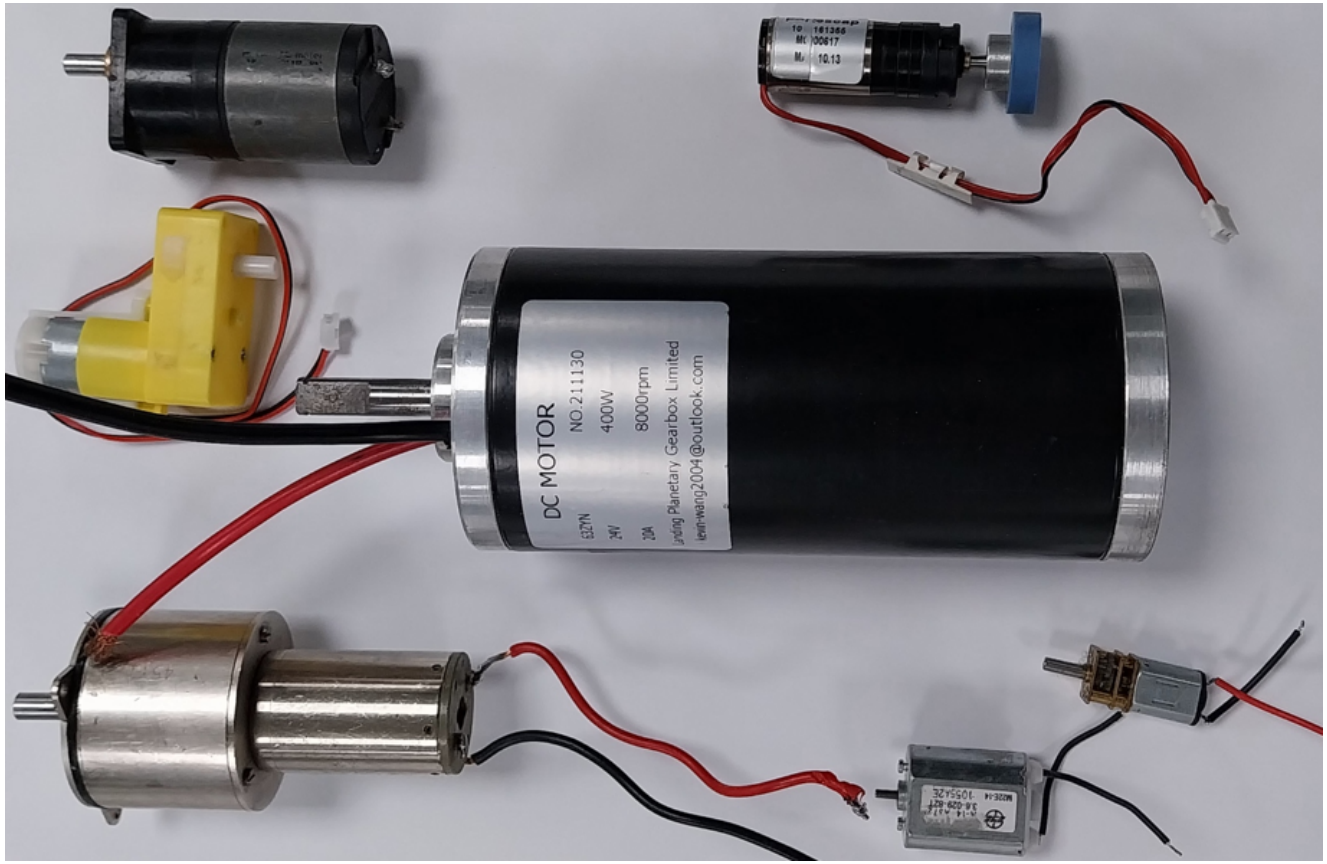
Référence concernant les moteurs et la robotique

Pour étudier plus sérieusement la théorie des moteurs et leur contrôle, vous pouvez consulter les ressources documentaires suivantes :

1. (moyen) Moteurs électriques pour la robotique, Pierre Mayé, 4eme Édition, 2023, Dunod.
2. (difficile) Theory of Robot Control, Carlos Canudas de Wit, Bruno Siciliano and Georges Bastin, 1996, Springer.

Plan

Présentation des moteurs DC



Pour faire tourner le moteur avec une alimentation et des interrupteurs, consultez la page 293

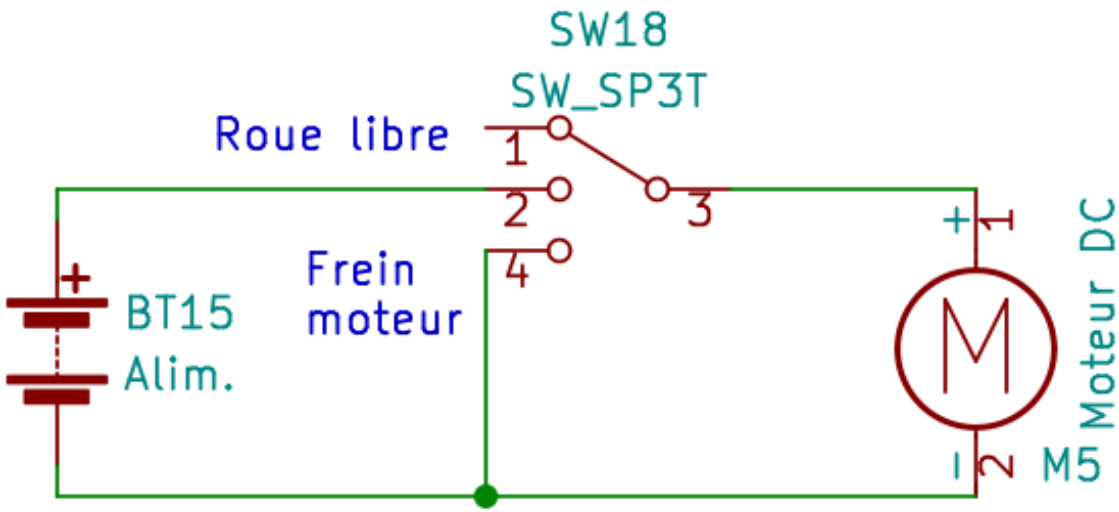
Pour le faire tourner dans un seule sens, avec un Mosfet allez à la page 316

Pour le faire tourner dans les deux sens avec un pont en H, continuer à la page 342.

Remarque, pour le faire tourner dans un seul sens vous pourvez juste utiliser un demi-pont.

(présent dans la version courte)

Les moteurs DC - commande manuelle – Faire tourner le moteur dans un seul sens



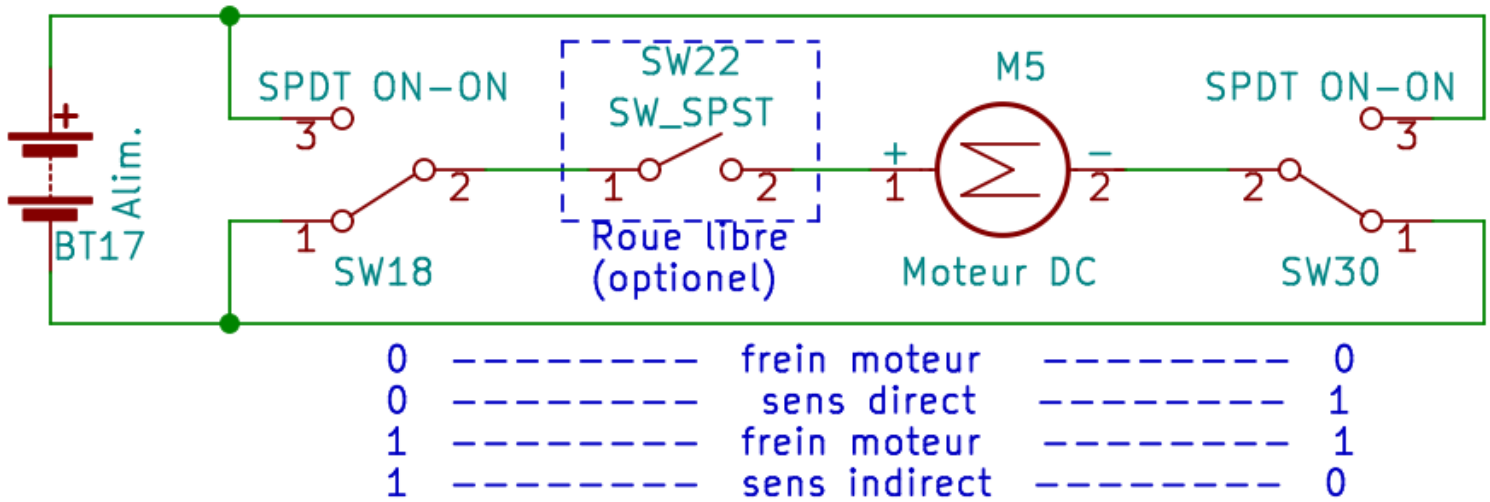
(présent dans la version courte)

Les moteurs DC - commande manuelle – Faire tourner dans les deux sens - 1/3

Pour activer le frein moteur, il faut appuyer sur un seul interrupteur parmi SW18 et SW30.

Pour changer de sens, il faut toujours appuyer sur les deux interrupteurs SW18 et SW30.

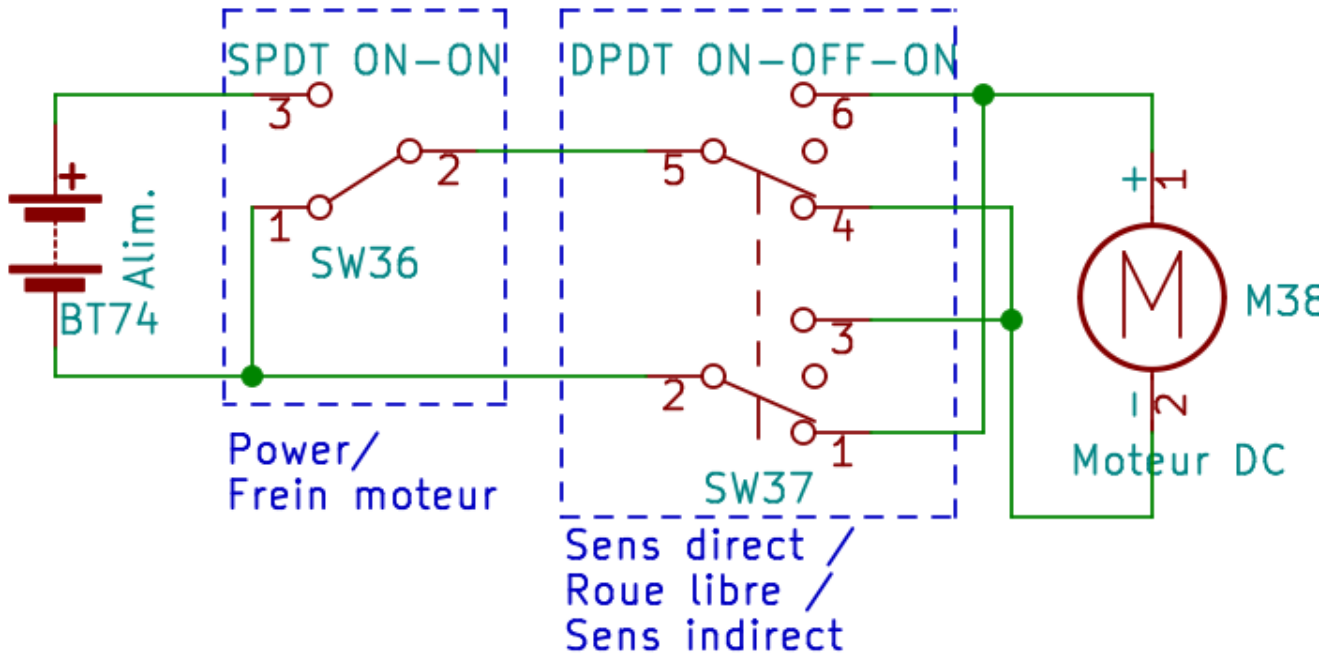
Ainsi, le frein moteur est obligatoirement activé quand on change de sens.



(version longue)

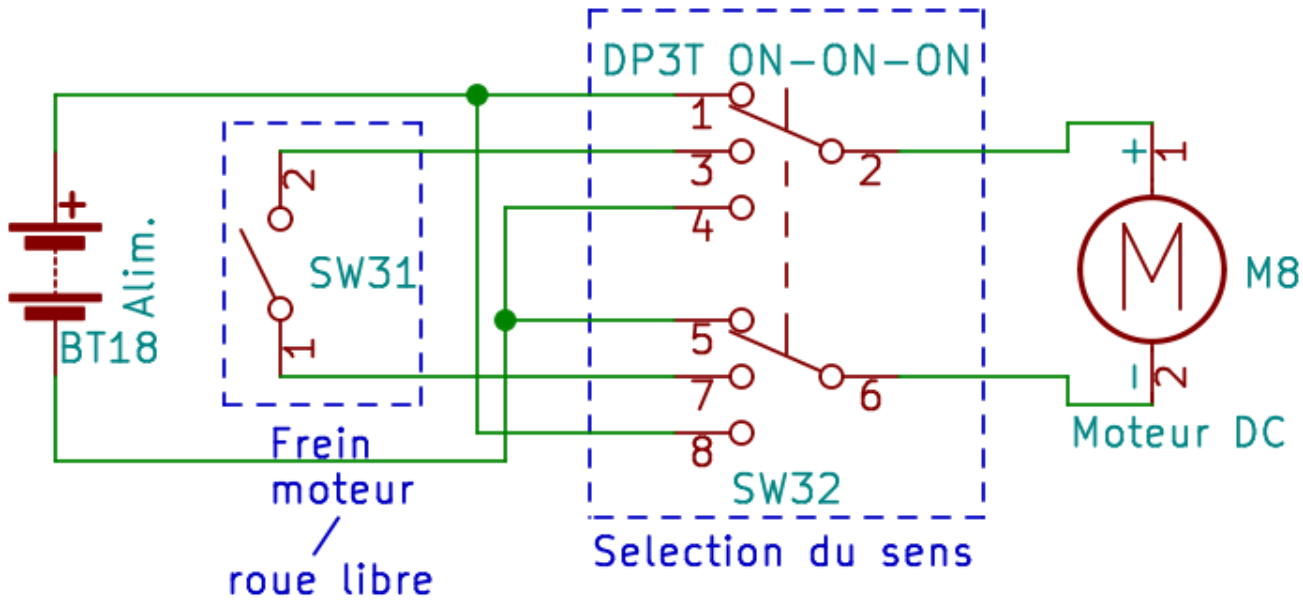
Les moteurs DC - commande manuelle – Faire tourner dans les deux sens - 2/3

L'interrupteur SW37 sert pour changer de sens le moteur ou le mettre en roue libre.
Cependant, quand on change de sens, on ne met pas le moteur en frein moteur, mais en roue libre.



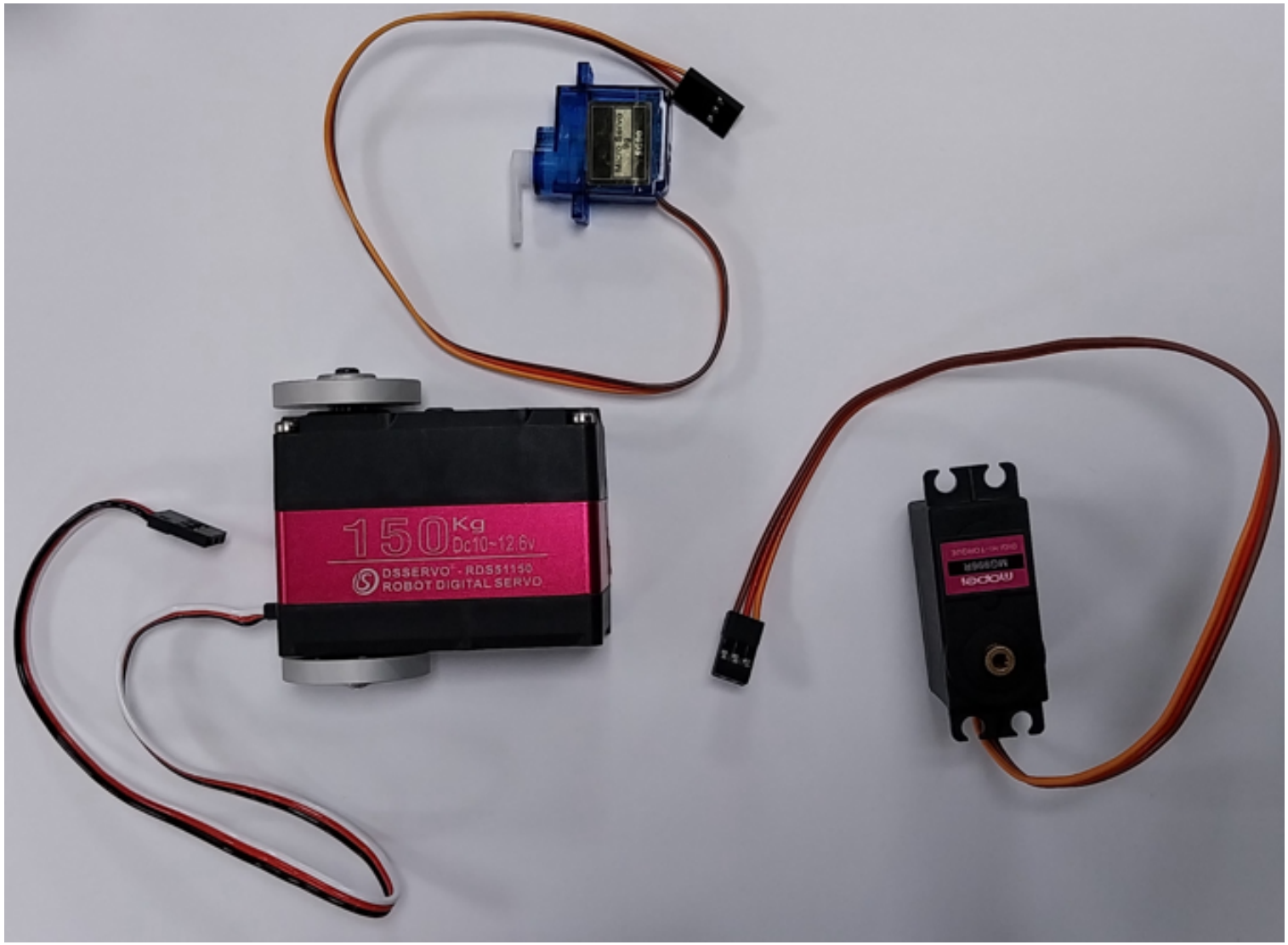
Les moteurs DC - commande manuelle – Faire tourner dans les deux sens - 3/3

Ce schéma est le plus pratique à utiliser. Cependant, il est bien plus onereux et il est difficile de trouver un interrupteur DP3T ON-ON-ON capable de passer suffisament de courant.



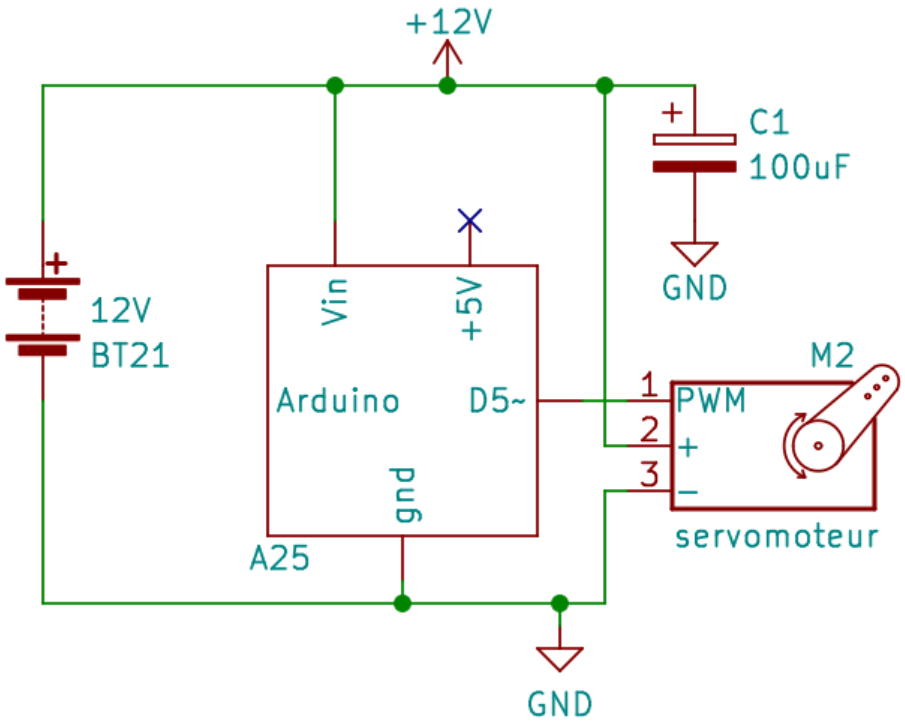
Plan

Présentation des servomoteurs

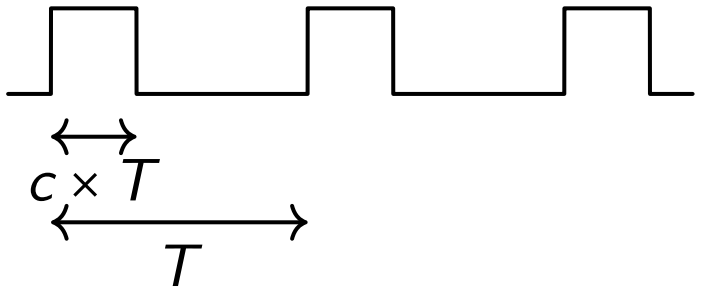


(présent dans la version courte)

Les servomoteurs



Signal de commande : Une PWM



Fréquence : 50Hz.

Période T : $20\text{ms} = \frac{1}{50\text{Hz}}$.

Rapport cyclique : $c \in [5\%, 10\%]$.

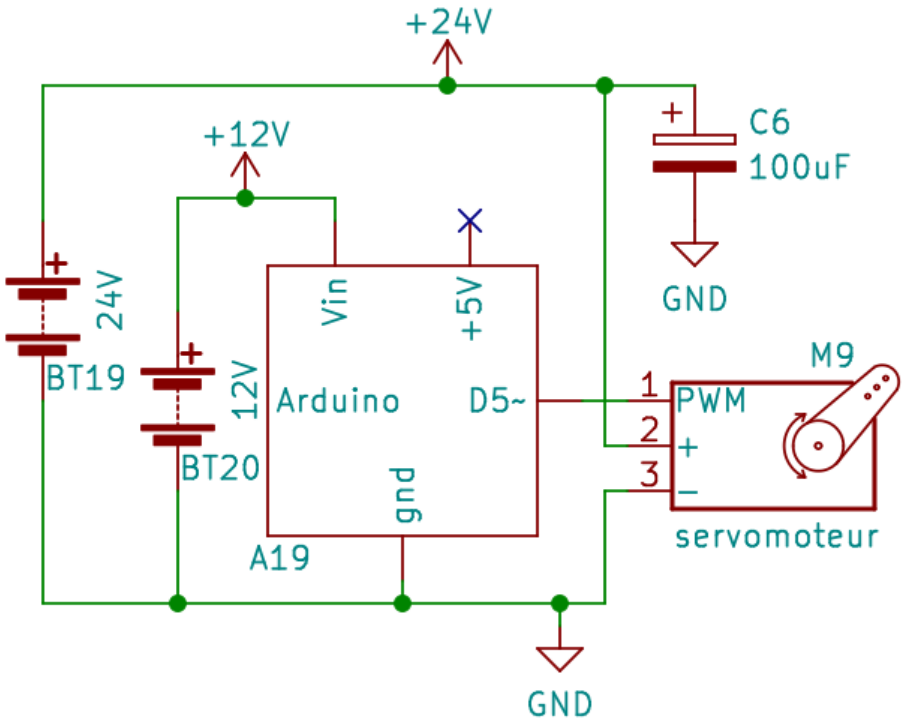
Durée ampl. max.: $c \times T \in [1,2]$ ms.

Position angulaire du moteur $\overset{\text{linéaire}}{\longleftrightarrow}$ rapport cyclique.

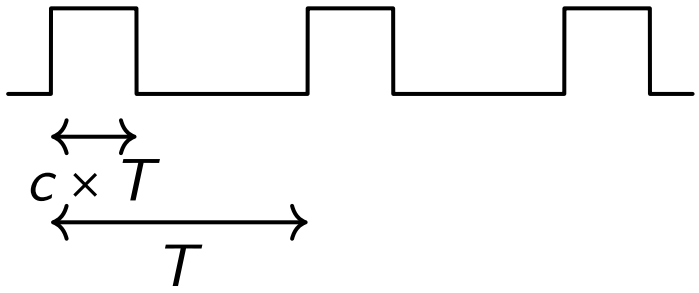
(présent dans la version courte)

Les servomoteurs

Avec deux alimentations séparées



Signal de commande : Une PWM



Fréquence : 50Hz.

Période T: $20\text{ms} = \frac{1}{50\text{Hz}}$.

Rapport cyclique : $c \in [5\%, 10\%]$.

Durée ampl. max.: $c \times T \in [1,2]$ ms.

Position angulaire du moteur $\overset{\text{linéaire}}{\longleftrightarrow}$ rapport cyclique.

(présent dans la version courte)

Programmer les servomoteurs sous Arduino

```
#include <Servo.h> // Installez la bibliotheque Servo de Michael Margolis 1.2.0
                   // ( allez dans arduino IDE -> tools -> manage Libraries )
const int pwm_pin = 5;
const int PWM_MIN = 1000; // 1000 us => angle minimal
const int PWM_MAX = 2000; // 2000 us => angle maximal
const int PWM_MID = 1500;
int pwm = PWM_MID;

Servo servomotor;

void setup() {
  servomotor.attach(pwm_pin);
  servomotor.writeMicroseconds(PWM_MID);
  delay(1000);
}

void loop() {
  for(pwm = PWM_MIN; pwm < PWM_MAX; pwm+=10){
    servomotor.writeMicroseconds(pwm);
    delay(200);
  }
  for(pwm = PWM_MAX; pwm >= PWM_MIN; pwm-=10){
    servomotor.writeMicroseconds(pwm); delay(200);
  }
}
```

code/servomotor.cpp

(présent dans la version courte)

Programmer les servomoteurs sous ESP32 (1/2)

On utilise ici l'api native, LED Control, qui permet de configurer simplement des PWMs.

```
const int pwm_pin = 5;
const int FREQUENCY = 50; // Hz
const int PERIOD = 1000000/FREQUENCY; // us
const int PWM_MIN = 1000; // 1000 us => angle minimal
const int PWM_MAX = 2000; // 2000 us => angle maximal
const int PWM_MID = (PWM_MIN + PWM_MAX)/2;
const int PWM_RESOLUTION = 10 ;
const int MAX_DUTY = (int)(pow(2,PWM_RESOLUTION)-1);

void writeMicroseconds(int pwm_pin, int pwm){
  uint32_t duty_cycle=map(pwm, 0, PERIOD, 0, MAX_DUTY);
  bool res = ledcWrite(pwm_pin, duty_cycle);
  if( !res ){ Serial.println("Failed to set duty cycle."); }
}

void setup() {
  bool res = ledcAttach(pwm_pin, FREQUENCY, PWM_RESOLUTION);
  if( !res ){ Serial.println("Failed to init the pwm."); }
  writeMicroseconds(pwm_pin, PWM_MID); delay(1000);
}
```

code/servomotor_esp32.cpp

A TESTER !

(présent dans la version courte)

Programmer les servomoteurs sous ESP32 (2/2)

Ce code fait tourner le servo moteur de son angle minimal à son angle maximal et vice versa.

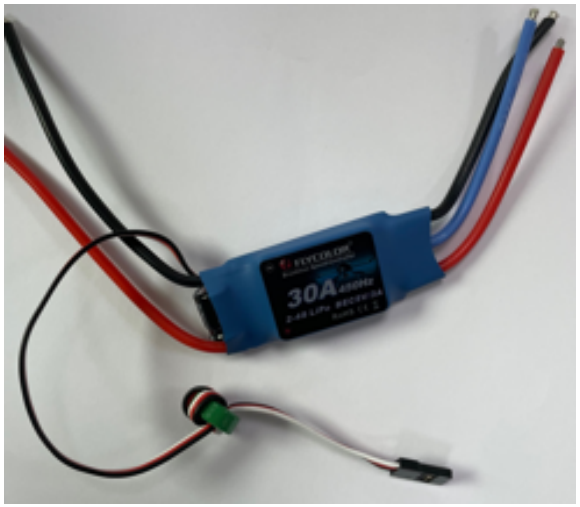
code/servomotor_esp32.cpp

```
void loop() {
  int pwm = PWM_MIN;
  for(pwm = PWM_MIN; pwm < PWM_MAX; pwm+=10){
    writeMicroseconds(pwm_pin, pwm); delay(200);
  }
  for(pwm = PWM_MAX; pwm >= PWM_MIN; pwm-=10){
    writeMicroseconds(pwm_pin, pwm); delay(200);
  }
}
```

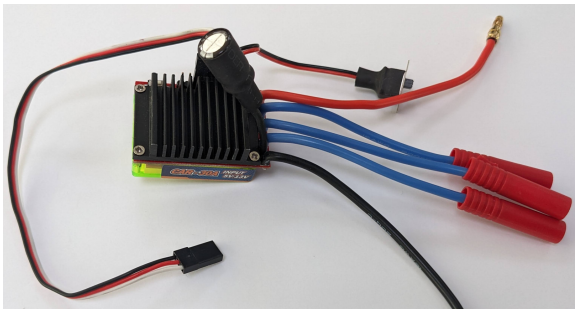
A TESTER !

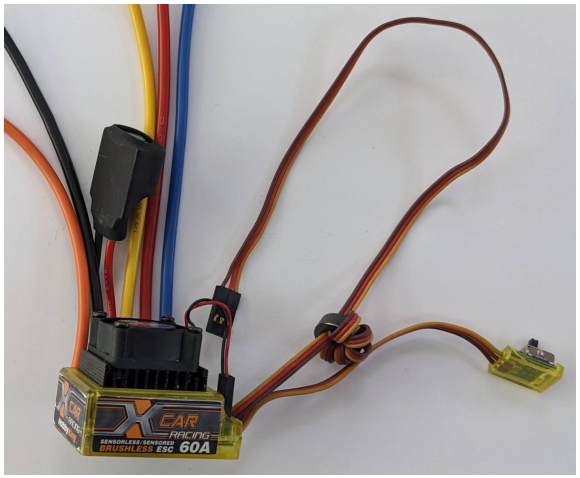
(présent dans la version courte)

Présentation des ESC



Les ESC des images à gauche sont utilisés pour les drones et avions réduits. Il n'ont pas de radiateur car le flux de l'air les refroidit. De-plus, ils font tourner les moteurs dans un seul sens. Si vous les utilisez pensez à refroidir à la fois l'ESC et le moteur.

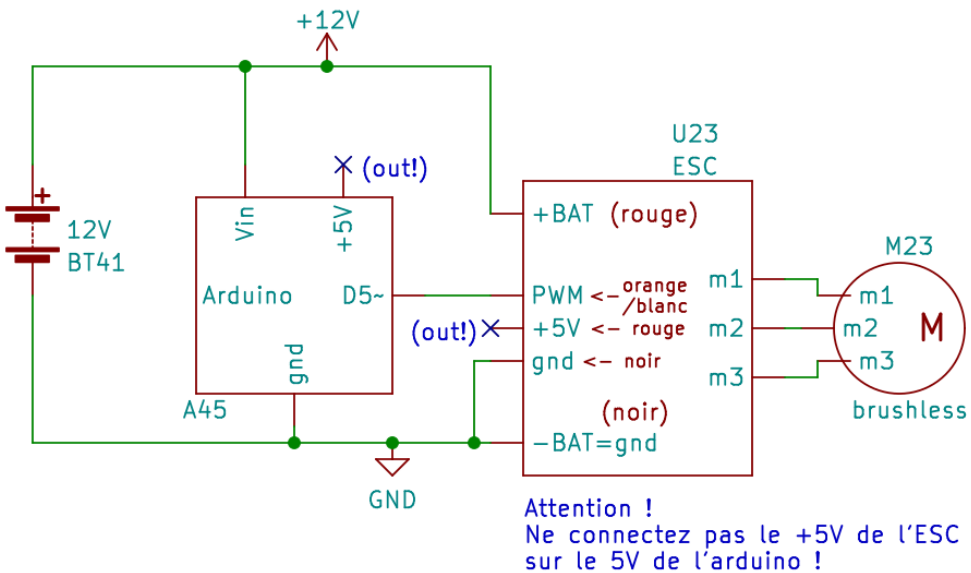




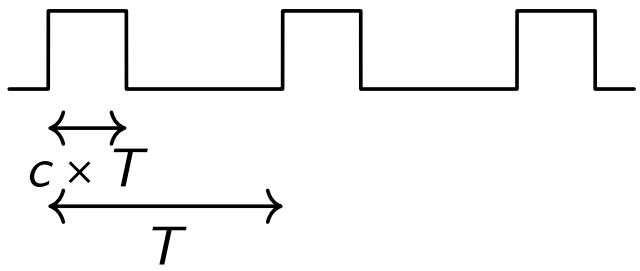
Les ESC des images de gauches sont utilisés dans les voitures modèle réduit. Ils possèdent un radiateur pour mieux dissiper la chaleur et ils permettent de faire tourner le moteur dans les deux sens. Dans ce cas, n'oubliez pas de refroidir le moteur s'il chauffe trop.

Les ESC

Les moteurs sans balais, tourne très vite et chauffe. Il faut donc bien fixer les moteurs et les refroidir tout comme l'ESC! En aero-modélisme le refroidissement est assuré par le flux de l'air de l'hélice.



Signal de commande : Une PWM



Fréquence : 50Hz.

Période T: $20\text{ms} = \frac{1}{50\text{Hz}}$.

Rapport cyclique : $c \in [5\%, 10\%]$.

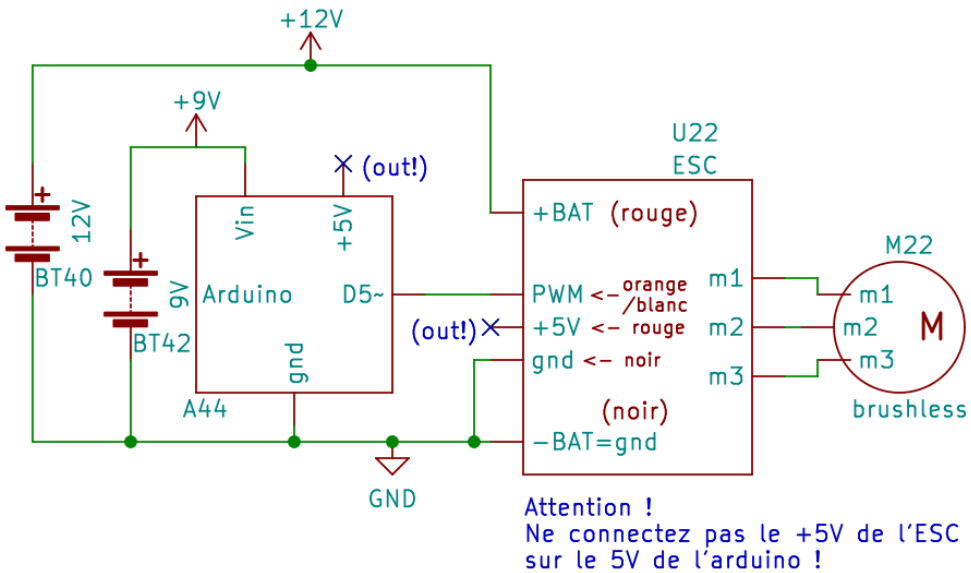
Durée ampl. max.: $c \times T \in [1,2]$ ms.

Vitesse angulaire du moteur $\xleftrightarrow{\text{linéaire}}$ rapport cyclique.

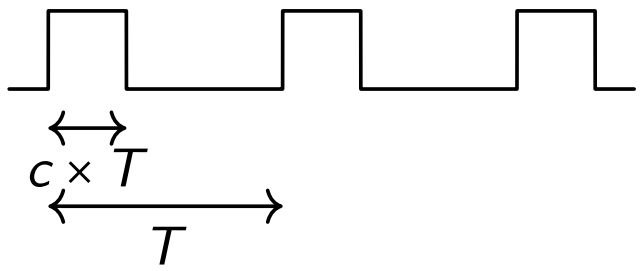
(version longue)

Les ESC

Les moteurs sans balais, tourne très vite et chauffe. Il faut donc bien fixer les moteurs et les refroidir tout comme l'ESC! En aero-modélisme le refroidissement est assuré par le flux de l'air de l'hélice.
Avec deux alimentations séparées



Signal de commande : Une PWM



Fréquence : 50Hz.

Période T: $20\text{ms} = \frac{1}{50\text{Hz}}$.

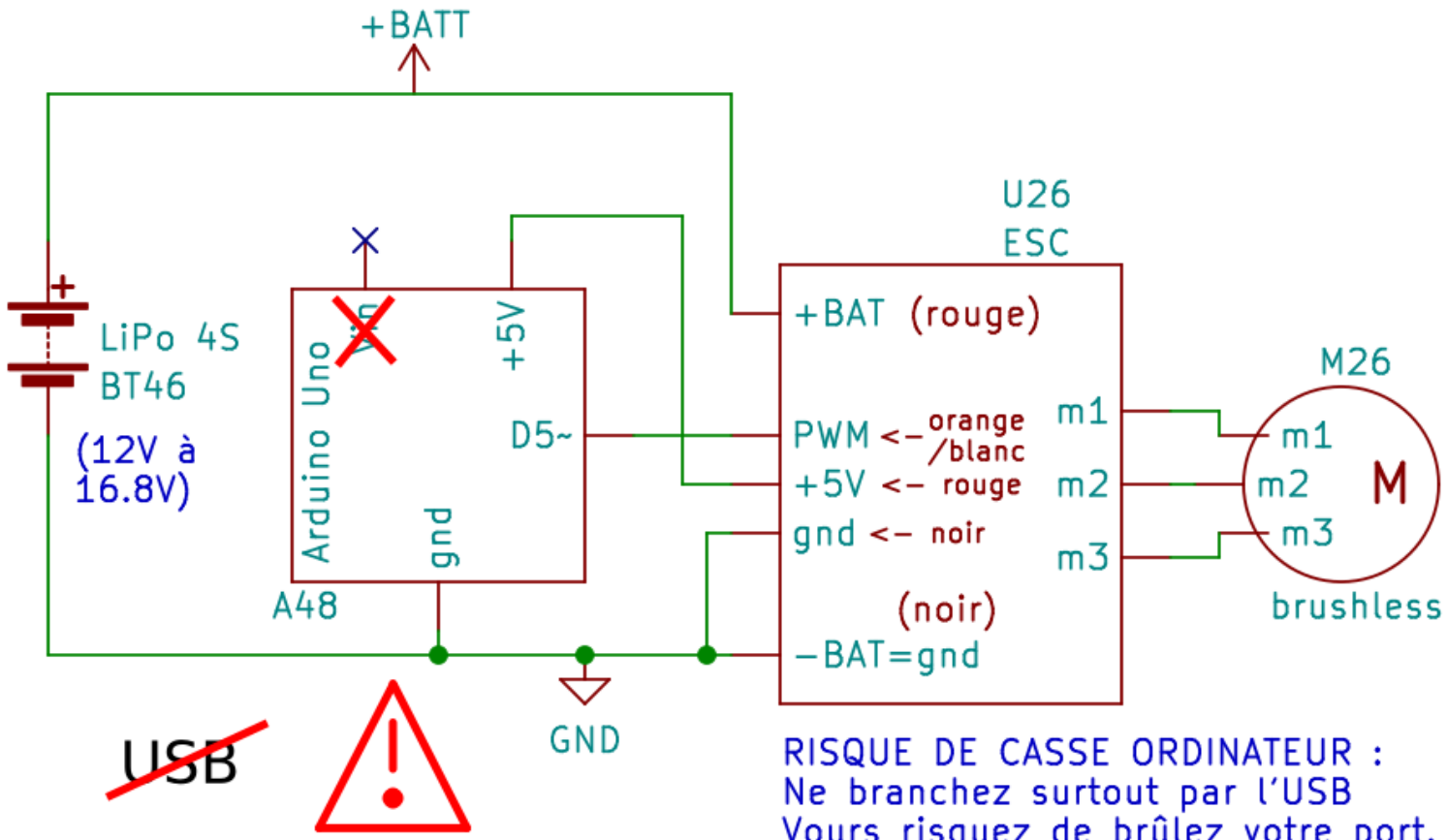
Rapport cyclique : $c \in [5\%, 10\%]$.

Durée ampl. max.: $c \times T \in [1,2]$ ms.

Vitesse angulaire du moteur $\xleftrightarrow{\text{linéaire}}$ rapport cyclique.

(version longue)

Montage dangereux - NE PAS CONNECTER l'USB



Consulez la page 628 pour comprendre pourquoi il ne faut pas connecter l'USB (ni le Vin) dans ce cas.

Configurer les ESC

Les ESC peuvent être utilisés pour différentes fonctions qui dépendent du constructeur.

Selon les drivers, on peut configurer, le sens de rotation, la gestion de la batterie, les coefficients des PID. etc ...
Souvent, la configuration par défaut est de faire tourrner le moteur dans un seul sens.

Pour connaître et configurer les options, il faut consulter la documentation du constructeur. Le choix des options se fait par une procédure particulièrement pénible où il faut écouter les beep pour savoir dans quel option on se trouve et appliquez des PWM MIN et PWM MAX pour se déplacer dans lei menu et choisir choisir les différentes option. C'est tout un poème ! En modélisme, c'est plus facile, car cela se fait avec la gâchette de la télécommande (gachette en bas = PWM MIN et gâchette en haut = PWM MAX).

Priez pour que la configuration par default soit celle que vous vouliez !

Programmer un ESC pour arduino qui tourne dans un seul sens (1/2)

La programmation est identique à celle des servomoteurs (cf. page 301). Les ESCs nécessitent d'être armés pour démarrer. Voici un exemple de procédure – ATTENTION : le moteur tourne très vite. Fixez-le bien :

```
#include <Servo.h> // Installez la bibliotheque Servo de Michael Margolis 1.2.0
                   // ( allez dans arduino IDE -> tools -> manage Libraries )
const int pwm_pin = 5;
const int PWM_MIN = 1000; // 1000 us => vitesse min = vitesse nulle
const int PWM_MAX = 2000; // 2000 us => vitesse max (dans un seul sens)
const int PWM_STOP = PWM_MIN; // STOP : vitesse nulle.
Servo esc;

void setup() {
  esc.attach(pwm_pin);

  // Exemple d'initialisation de l'ESC flycolor 30A (cf. notice constructeur)
  // Restart the driver if it is stoped.
  esc.writeMicroseconds(PWM_STOP);
  delay(100);
  esc.writeMicroseconds(PWM_MAX);
  delay(100);
  // Arm the motor
  esc.writeMicroseconds(PWM_STOP);
  delay(2000 + 1000);
}
```

code/esc_un_sens.cpp

(version longue)

Programmer un ESC pour arduino qui tourne dans un seul sens (2/2)

Attention, les moteurs peuvent tourner très vite. Limitez la vitesse maximale et fixer solidement à la table le moteur lors des tests.

code/esc_un_sens.cpp

```
void loop() {
  int PWM_MAX_VELOCITY = ( 3*PWM_MIN + PWM_MAX )/4;
  for(int pwm = PWM_MIN; pwm < PWM_MAX_VELOCITY; pwm+=10){
    esc.writeMicroseconds(pwm);
    delay(200);
  }
  for(int pwm = PWM_MAX_VELOCITY; pwm >= PWM_MIN; pwm-=10){
    esc.writeMicroseconds(pwm); delay(200);
  }
}
```

Programmer un ESC pour ESP32 qui tourne dans un seul sens (1/2)

On utilise ici l'api native, LED Control, qui permet de configurer simplement des PWMs.

```
const int pwm_pin = 5;
const int FREQUENCY = 50; const int PERIOD = 1000000/FREQUENCY; // Hz, us
const int PWM_MIN = 1000; // 1000 us => vitesse min = vitesse nulle
const int PWM_MAX = 2000; // 2000 us => vitesse max (dans un seul sens)
const int PWM_STOP = PWM_MIN; // STOP : vitesse nulle.
const int PWM_RESOLUTION = 10;
const int MAX_DUTY = (int)(pow(2,PWM_RESOLUTION)-1);

void writeMicroseconds(int pwm_pin, int pwm){
  uint32_t duty_cycle=map(pwm, 0, PERIOD, 0, MAX_DUTY);
  bool res = ledcWrite(pwm_pin, duty_cycle);
  if( !res ){ Serial.println("Failed to set duty cycle."); }
}

void setup() {
  bool res = ledcAttach(pwm_pin, FREQUENCY, PWM_RESOLUTION);
  if( !res ){ Serial.println("Failed to init the pwm."); }
  // Exemple d'initialisation de l'ESC flycolor 30A (cf. notice constructeur)
  // Restart the driver if it is stoped.
  writeMicroseconds(pwm_pin, PWM_STOP); delay(100);
  writeMicroseconds(pwm_pin, PWM_MAX); delay(100);
  // Arm the motor
  writeMicroseconds(pwm_pin, PWM_STOP); delay(2000 + 1000);
}
```

A TESTER !

code/esc_un_sens_esp32.cpp

(version longue)

Programmer un ESC pour ESP32 qui tourne dans un seul sens (2/2)

Attention, les moteurs peuvent tourner très vite. Limitez la vitesse maximale et fixer solidement à la table le moteur lors des tests.

code/esc_un_sens_esp32.cpp

```
void loop() {
  int PWM_MAX_VELOCITY = ( 3*PWM_MIN + PWM_MAX )/4;
  for(int pwm = PWM_MIN; pwm < PWM_MAX_VELOCITY; pwm+=10){
    writeMicroseconds(pwm_pin, pwm);
    delay(200);
  }
  for(int pwm = PWM_MAX_VELOCITY; pwm >= PWM_MIN; pwm-=10){
    writeMicroseconds(pwm_pin, pwm); delay(200);
  }
}
```

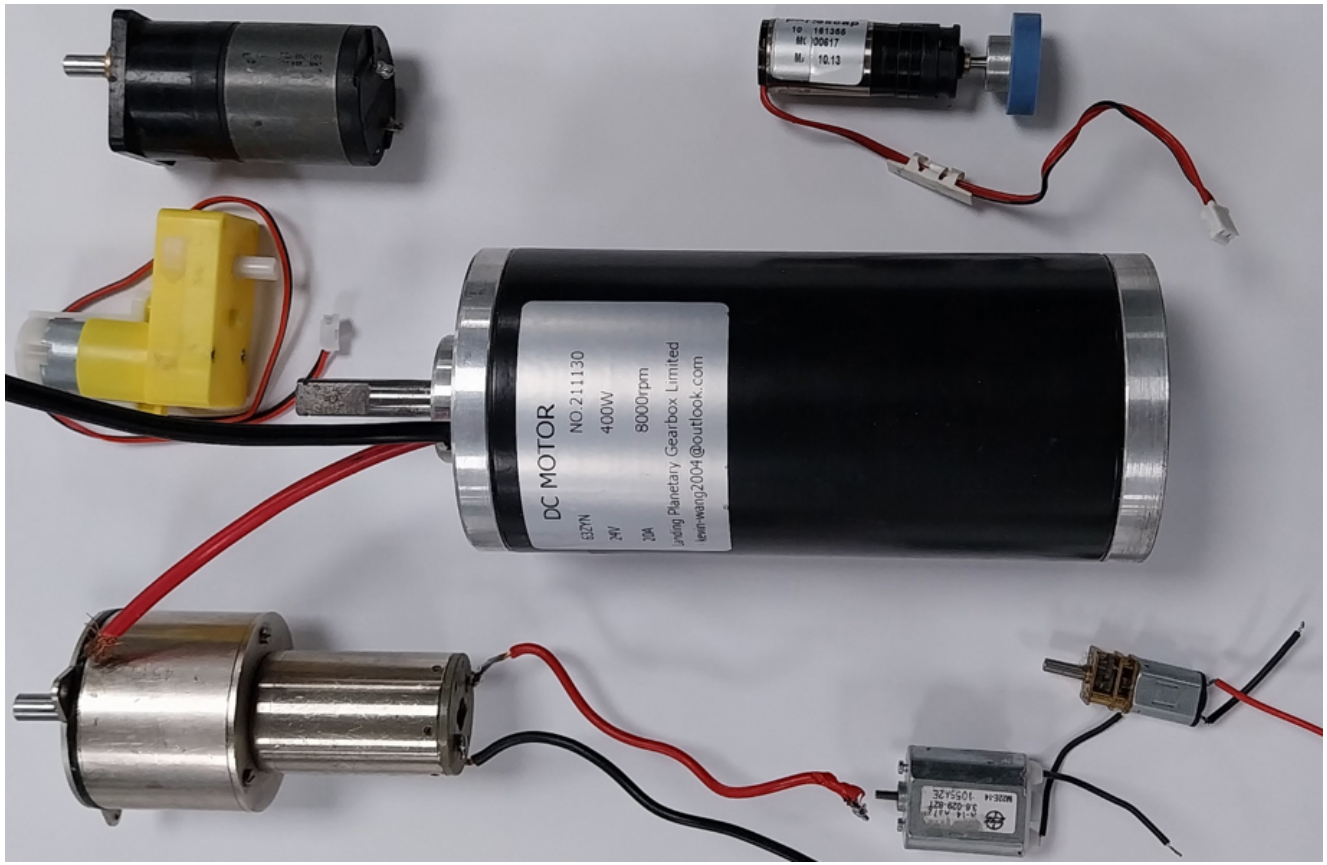
A TESTER !

Programmer un ESC qui tourne dans les deux sens

La programmation est identique à celle des servomoteurs (cf. page 301). Certains ESCs nécessitent une procédure de démarrage particulière qui change selon les moteurs. Il faut lire sa documentation. Voici un exemple de procédure :
A FAIRE : ECRIRE ET TESTER LE PROGRAMME

Plan

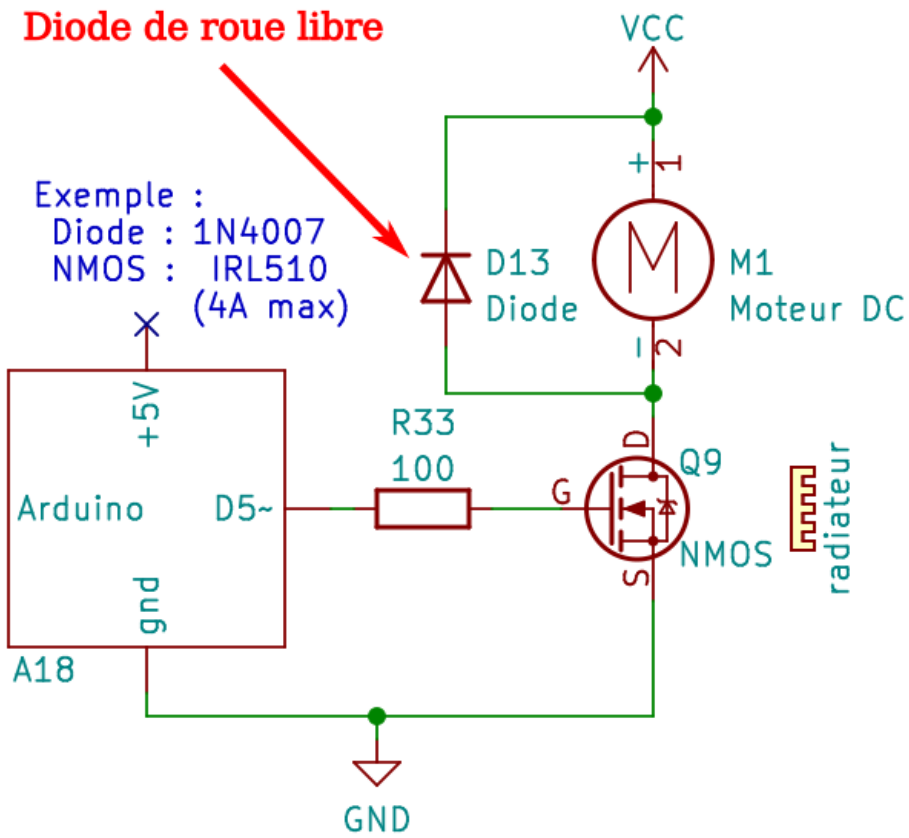
Rappel : presentation des moteurs DC



(présent dans la version courte)

Les moteurs DC – tourner dans un seul sens

Pour bien comprendre ce schema, il faut avoir lu les pages 259 à 272 sur commander un MOSFET canal N.



Le mosfet sert à ouvrir et fermer le circuit du moteur.

Un moteur est une bobine. Le courant qui le traverse est donc continue, même quand on ouvre le MOSFET. De plus, le moteur continue à tourner et à produire du courant. La diode D13, dite de roue libre, sert à évacuer ce courant.

Pour faire varier la vitesse du moteur, on utilise une PWM (cf page 104).

Pour ne pas entendre le moteur, la fréquence de la pwm doit valoir 25 kHz. A cette fréquence, il faut une résistance plus faible pour faire commuter la grille plus rapidement.

(présent dans la version courte)

Programmer un moteur qui tourne dans un seuls sens.

```
const int motor_pin = 5;
const int MAX_PWM = 255;
int motor_pwm = 0;

void setup() { }

void loop() {
  for( motor_pwm = 0; motor_pwm < MAX_PWM; motor_pwm += 10 ){
    analogWrite(motor_pin, motor_pwm);
    delay(120);
  }
  for( motor_pwm = MAX_PWM; motor_pwm >= 0; motor_pwm -= 10 ){
    analogWrite(motor_pin, motor_pwm);
    delay(120);
  }
}
```

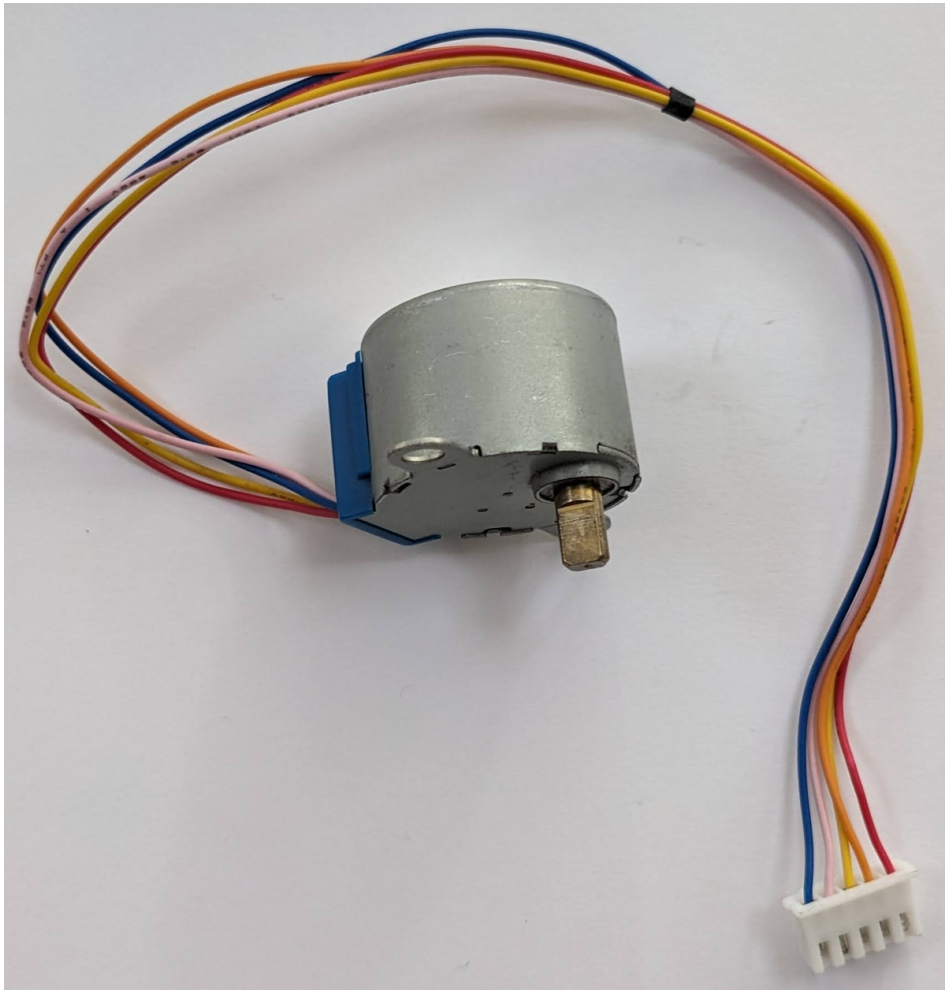
code/moteur_1_sens.cpp

Comme la fréquence par défaut de la PWM arduino varie entre 500 Hz et 1 KHz, on entends le moteur tourner. Pour ne plus l'entendre, il faut implémenter une pwm d'une fréquence > 22 KHz. (cf. page 434 à page 436).

(présent dans la version courte)

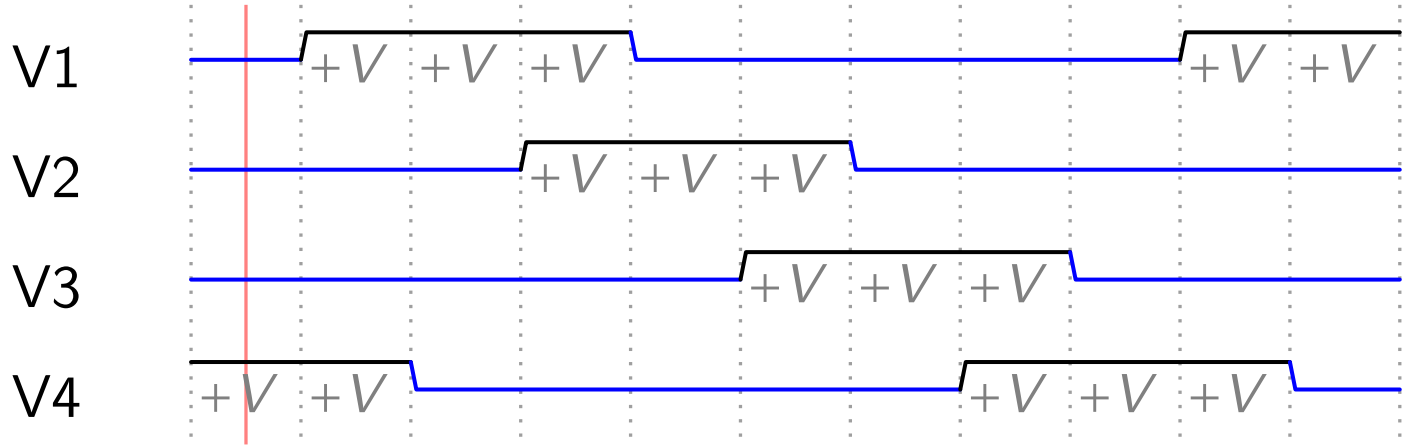
Plan

Présentation des moteurs pas à pas 5 fils

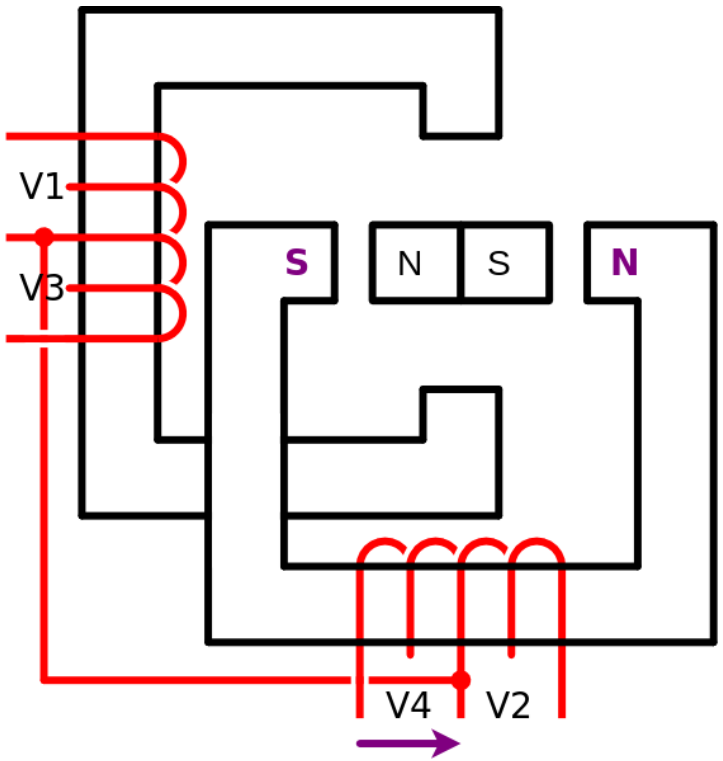


Le 28BYJ-48 des kits Arduino

Le moteur pas à pas 5 fils - Théorie

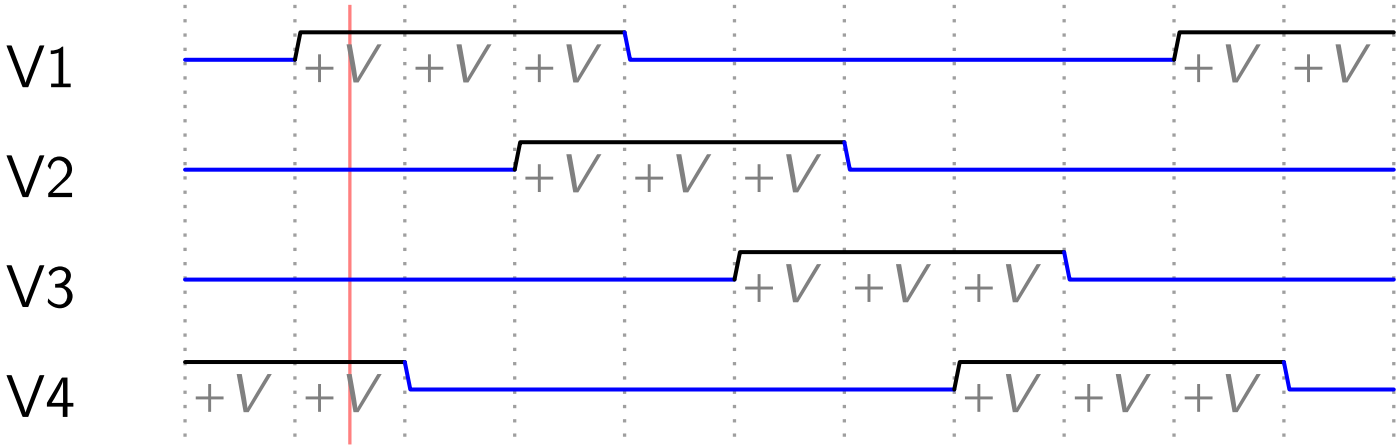


— : Haute impédance/ouvert — : $+V$

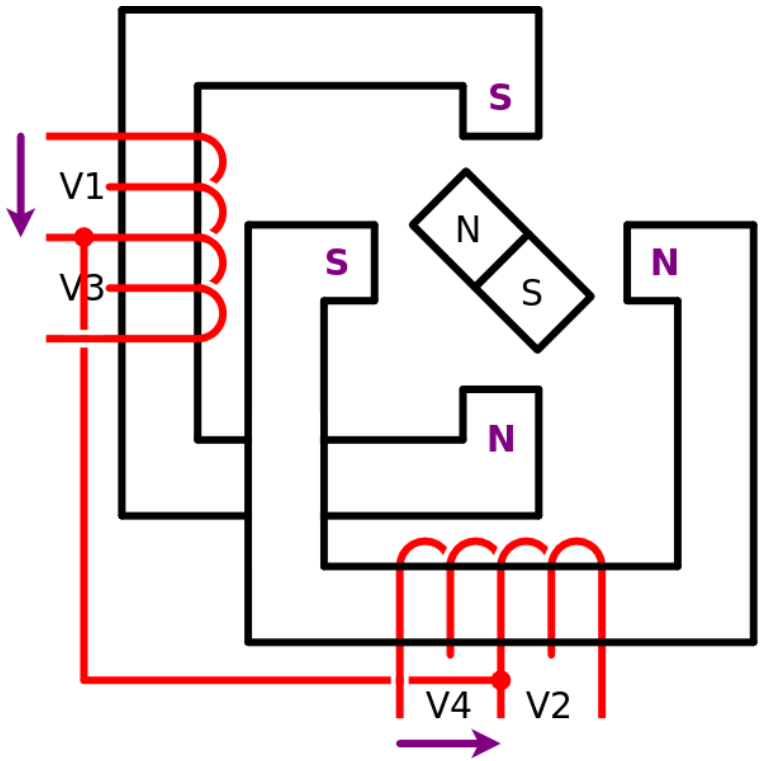


(présent dans la version courte)

Le moteur pas à pas 5 fils - Théorie

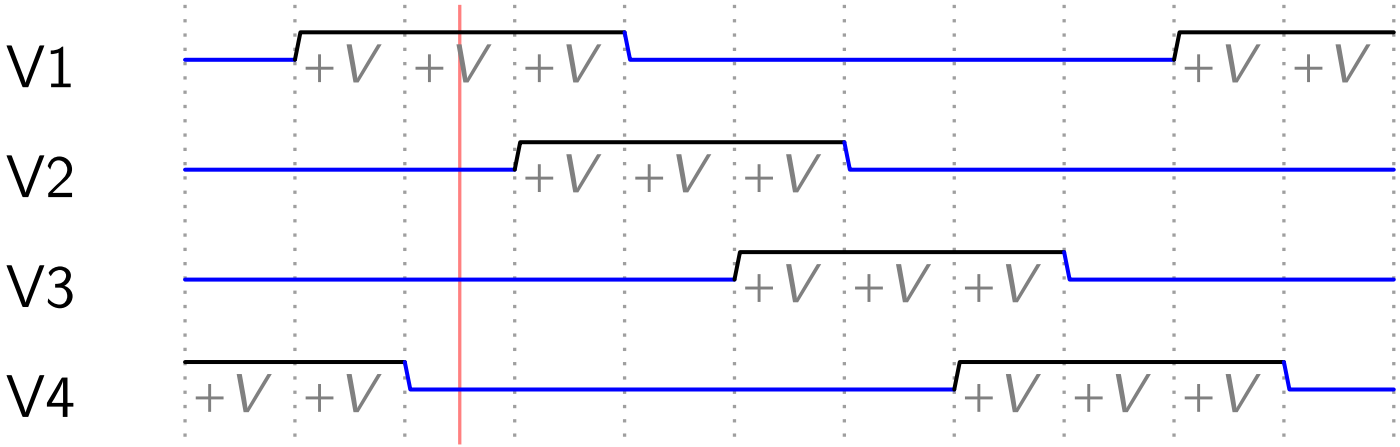


— : Haute impédance/ouvert — : $+V$

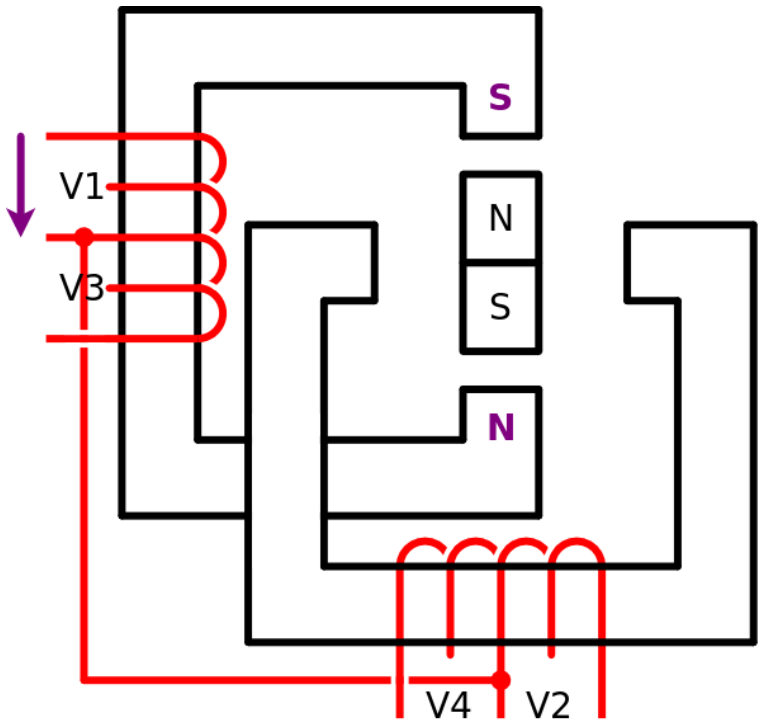


(présent dans la version courte)

Le moteur pas à pas 5 fils - Théorie

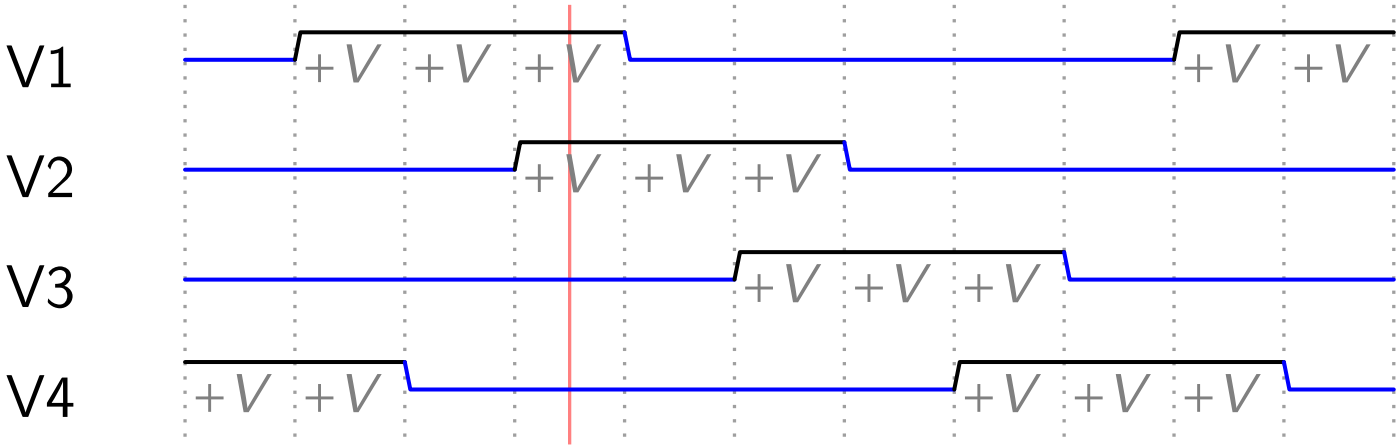


— : Haute impédance/ouvert — : $+V$

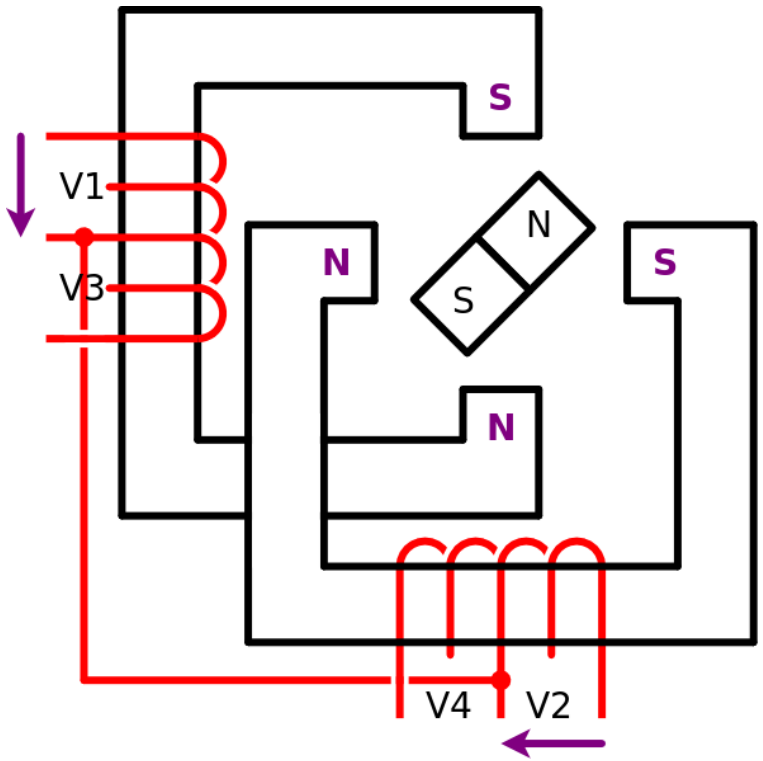


(présent dans la version courte)

Le moteur pas à pas 5 fils - Théorie

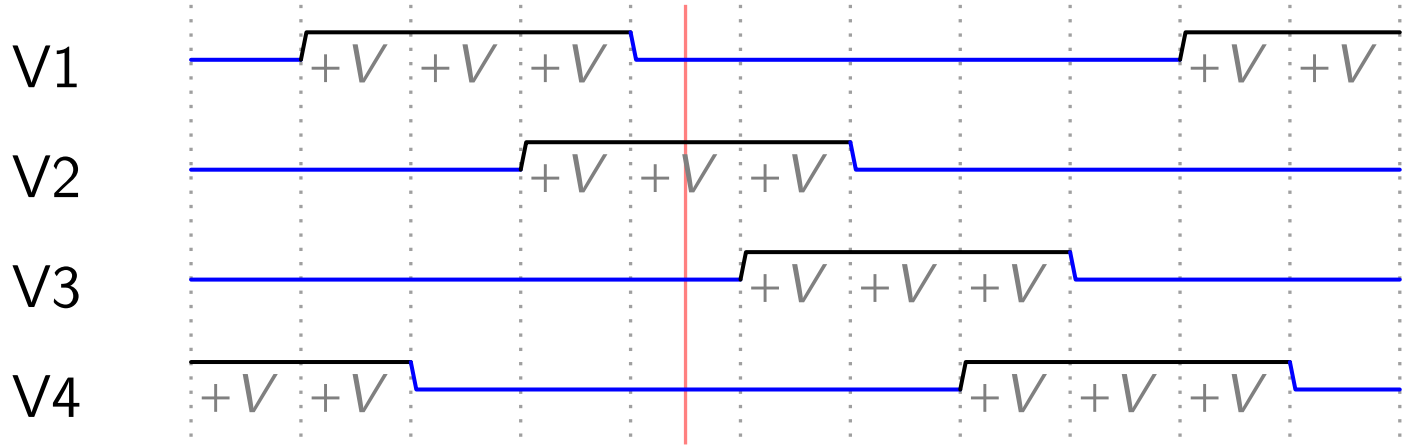


— : Haute impédance/ouvert — : $+V$

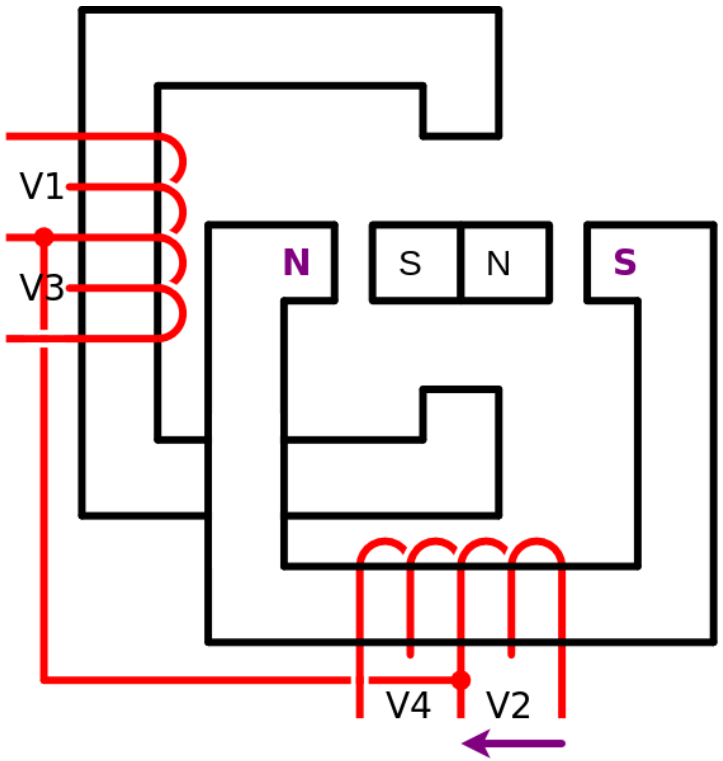


(présent dans la version courte)

Le moteur pas à pas 5 fils - Théorie

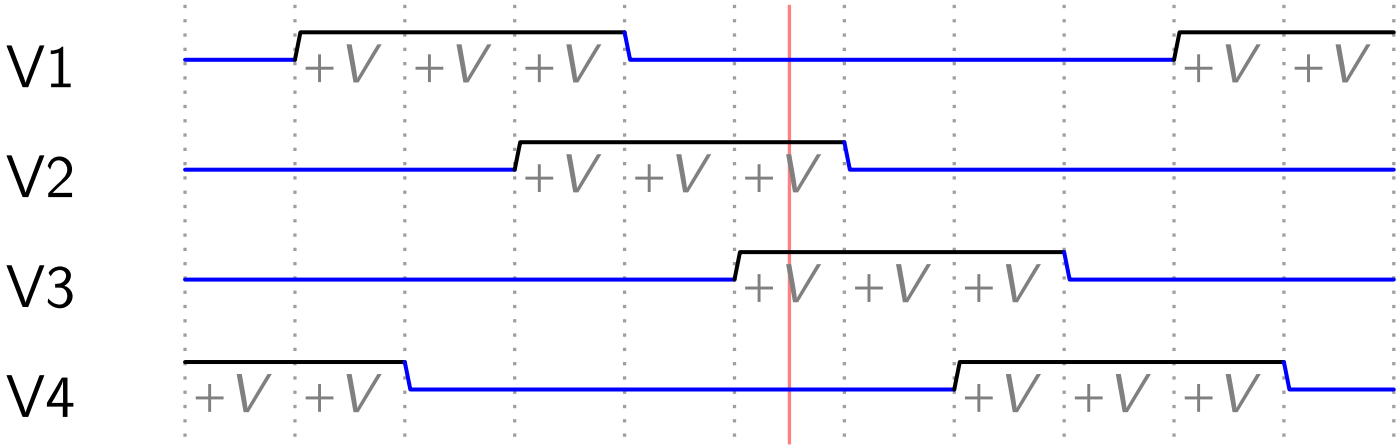


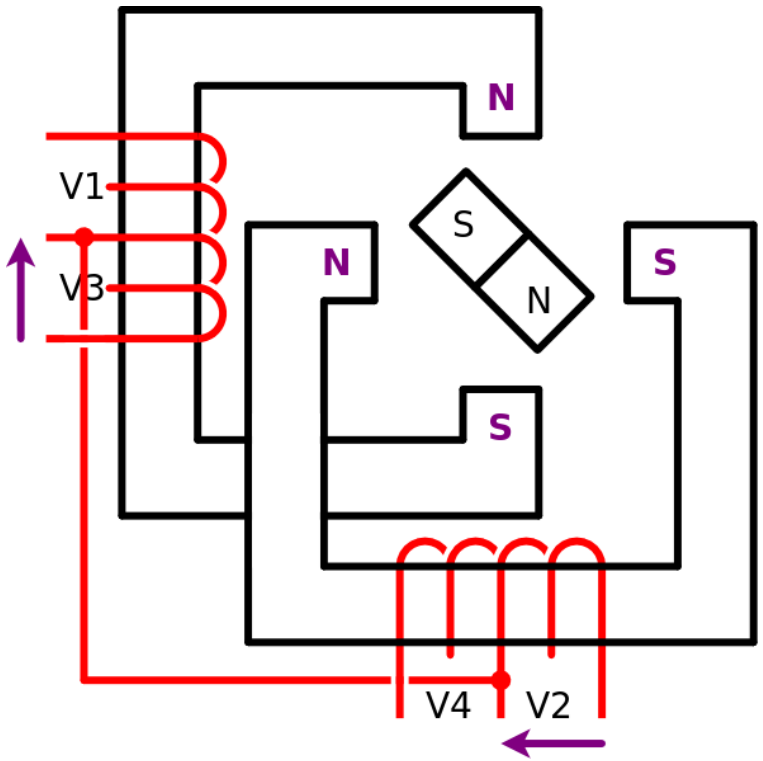
— : Haute impédance/ouvert — : $+V$



(présent dans la version courte)

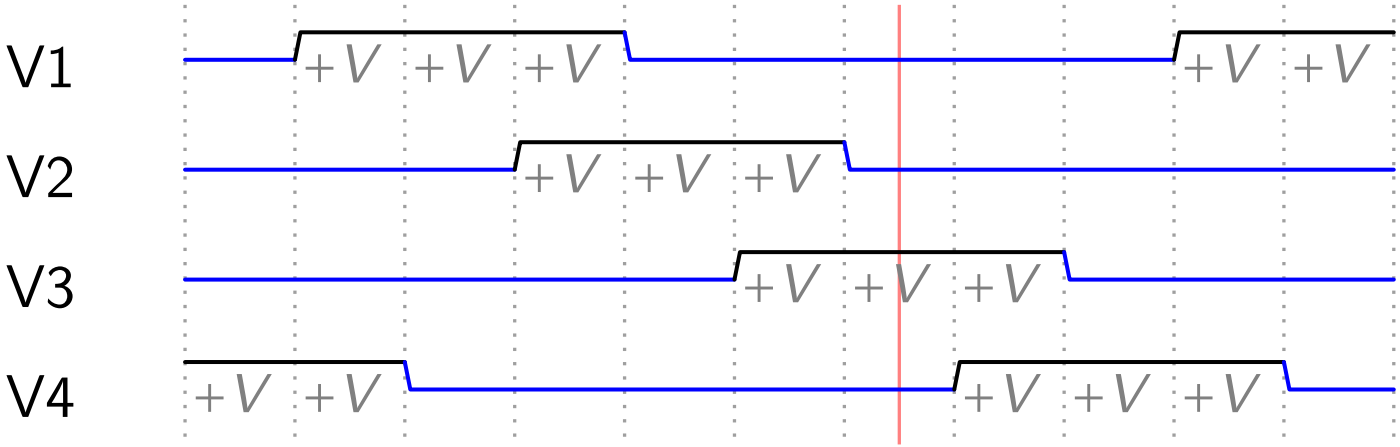
Le moteur pas à pas 5 fils - Théorie

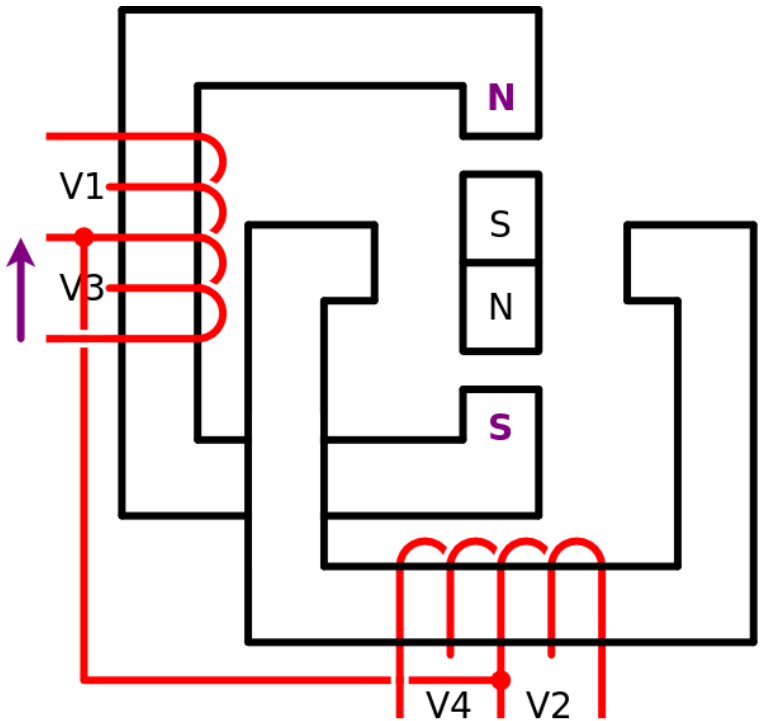




(présent dans la version courte)

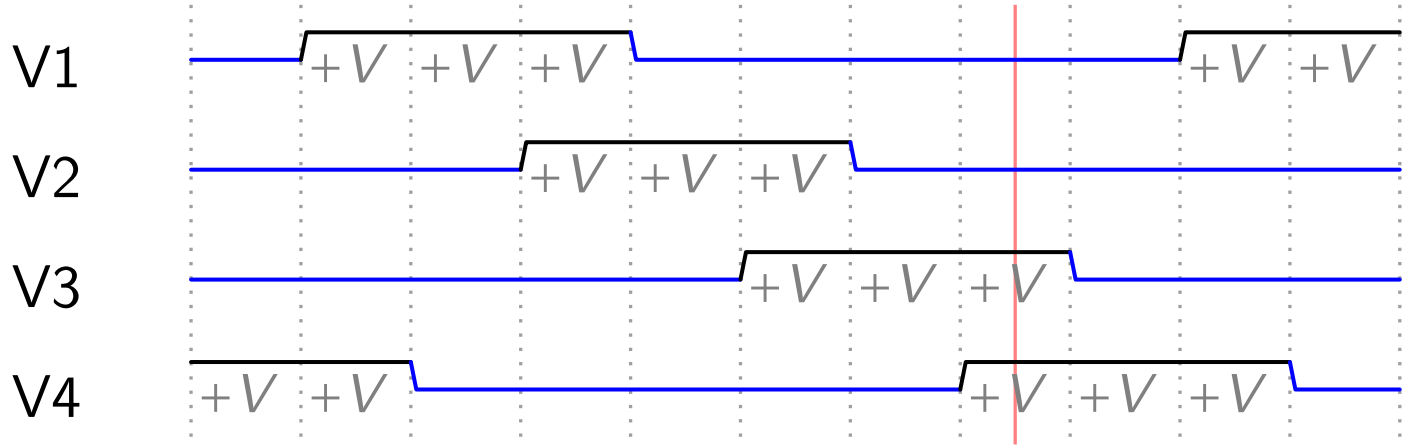
Le moteur pas à pas 5 fils - Théorie



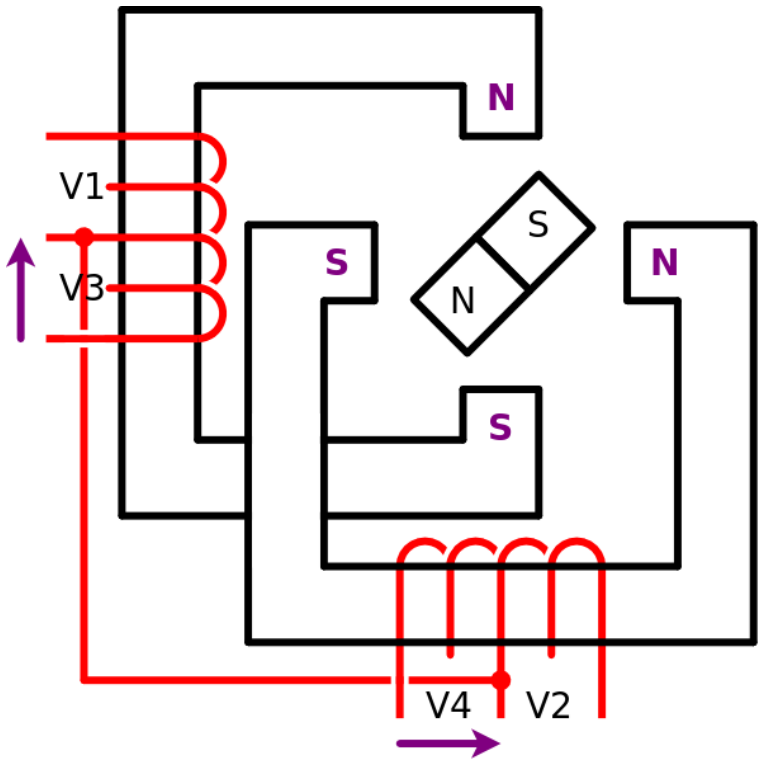


(présent dans la version courte)

Le moteur pas à pas 5 fils - Théorie

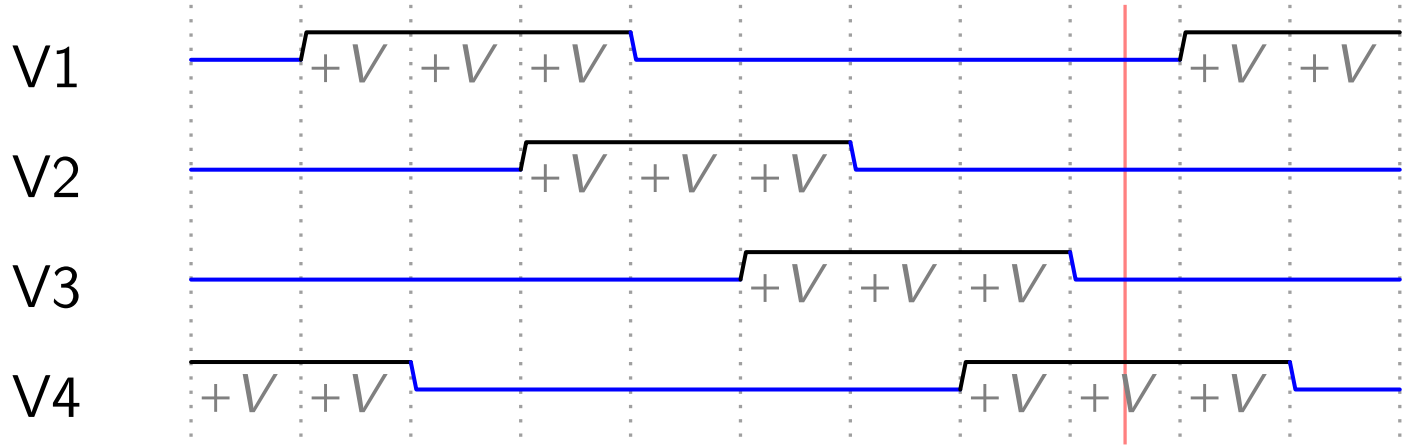


— : Haute impédance/ouvert — : $+V$

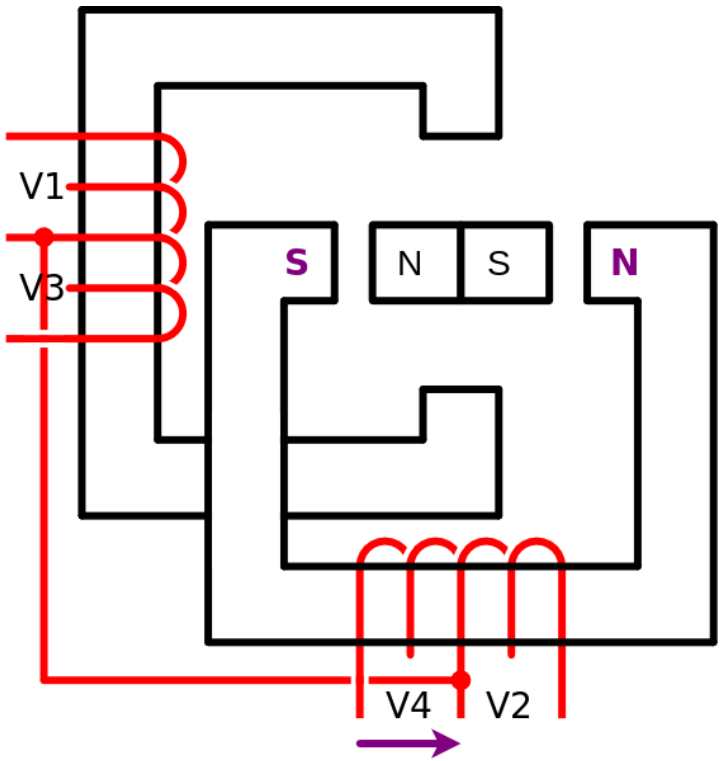


(présent dans la version courte)

Le moteur pas à pas 5 fils - Théorie

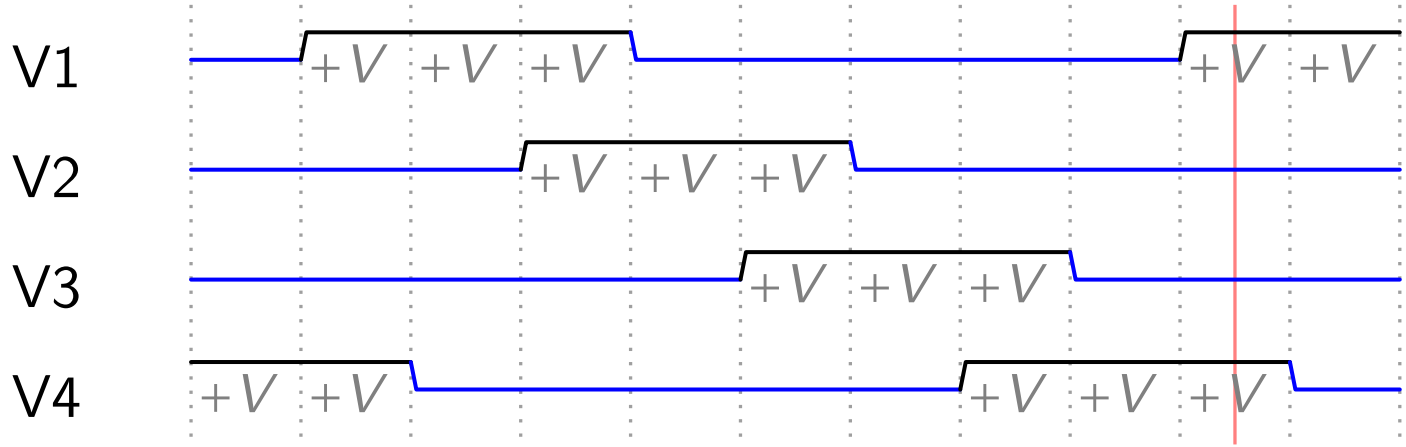


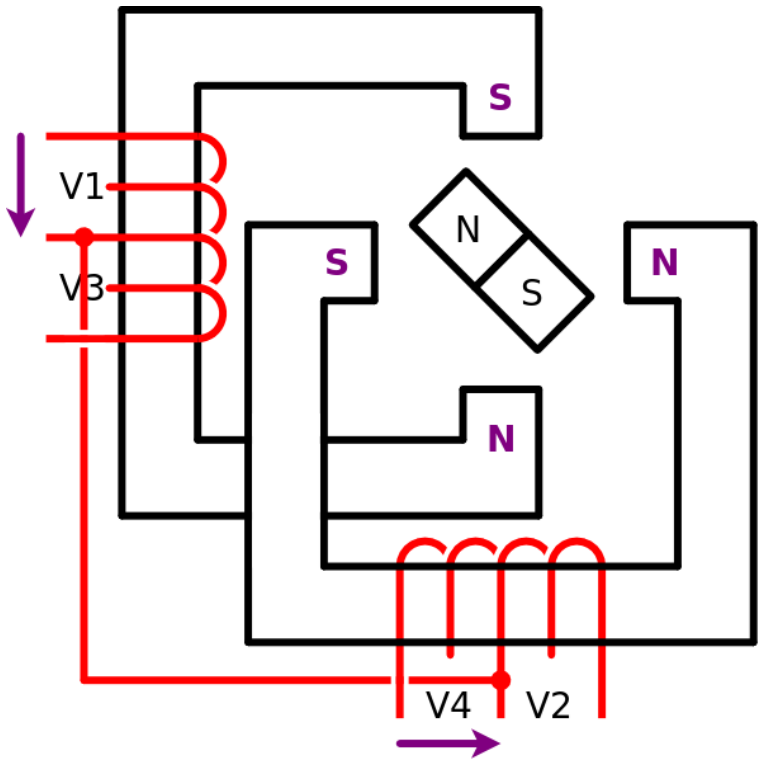
— : Haute impédance/ouvert — : $+V$



(présent dans la version courte)

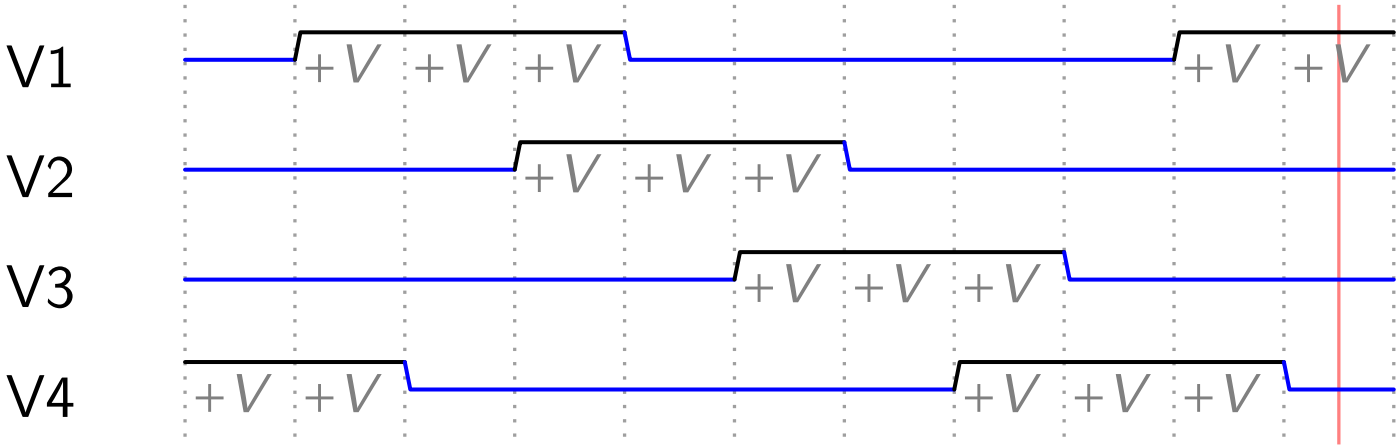
Le moteur pas à pas 5 fils - Théorie

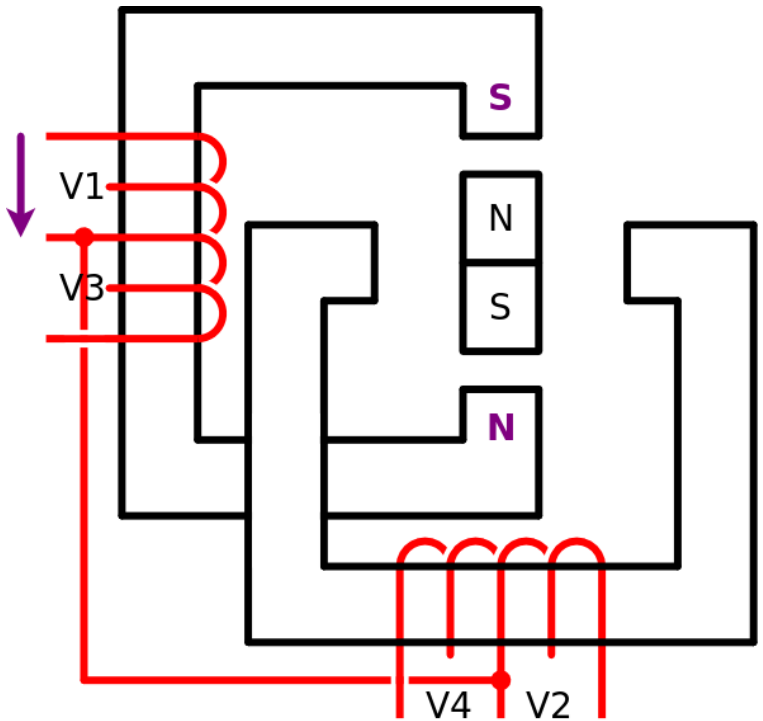




(présent dans la version courte)

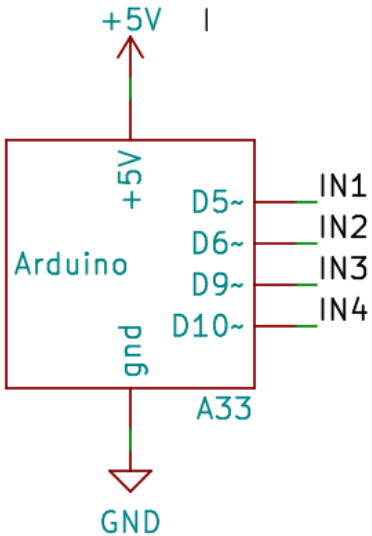
Le moteur pas à pas 5 fils - Théorie

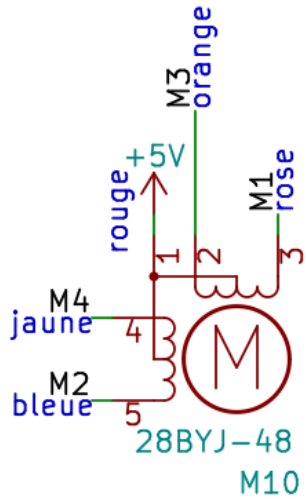




(présent dans la version courte)

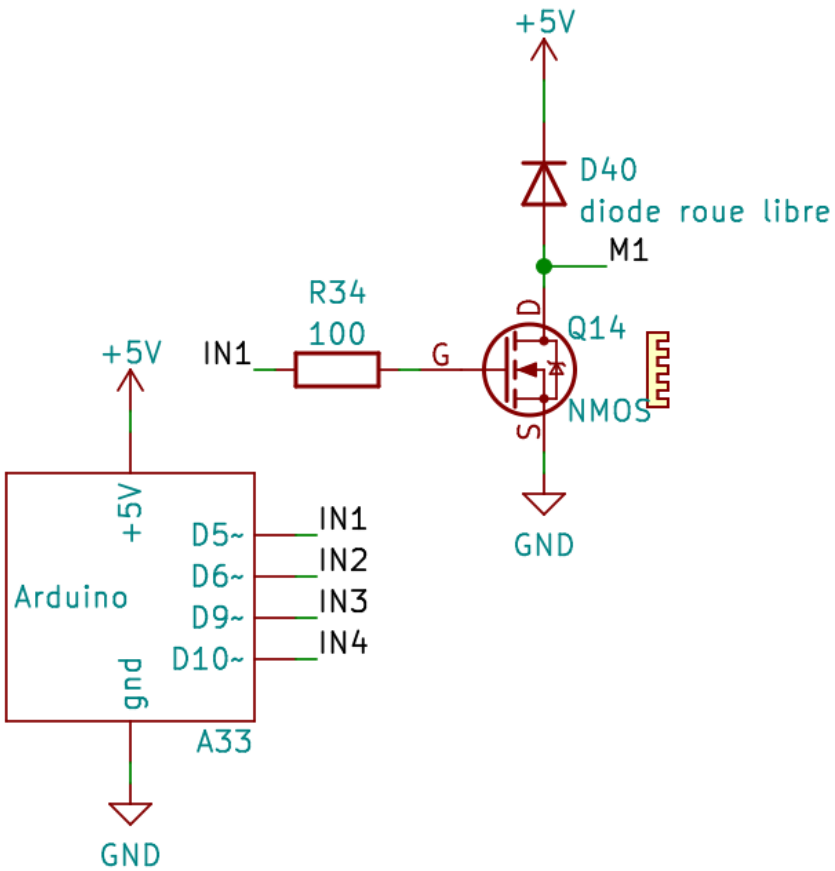
Le moteur pas à pas 5 fils - Schéma avec Mosfet

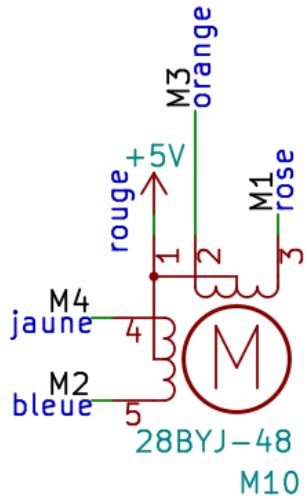




(présent dans la version courte)

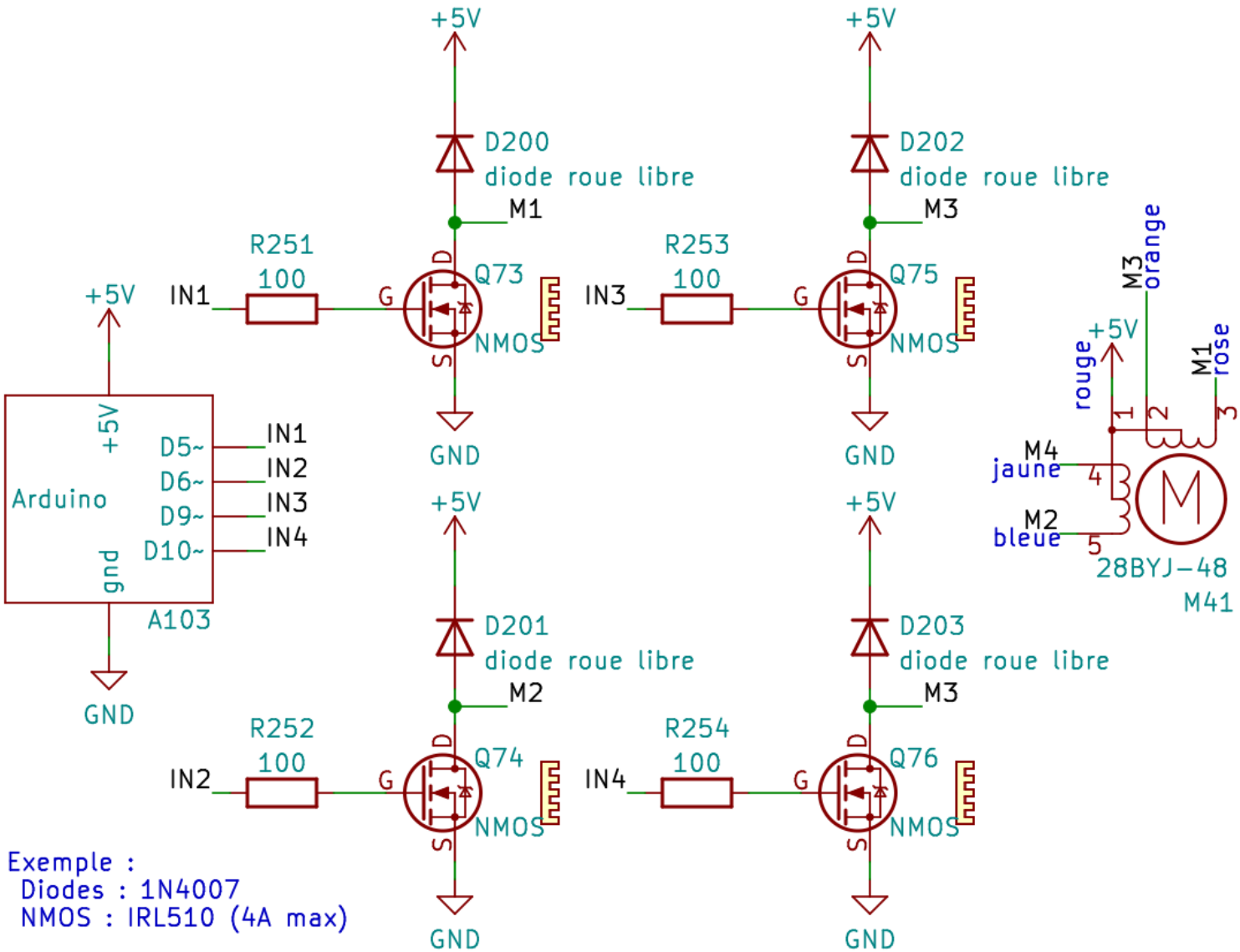
Le moteur pas à pas 5 fils - Schéma avec Mosfet





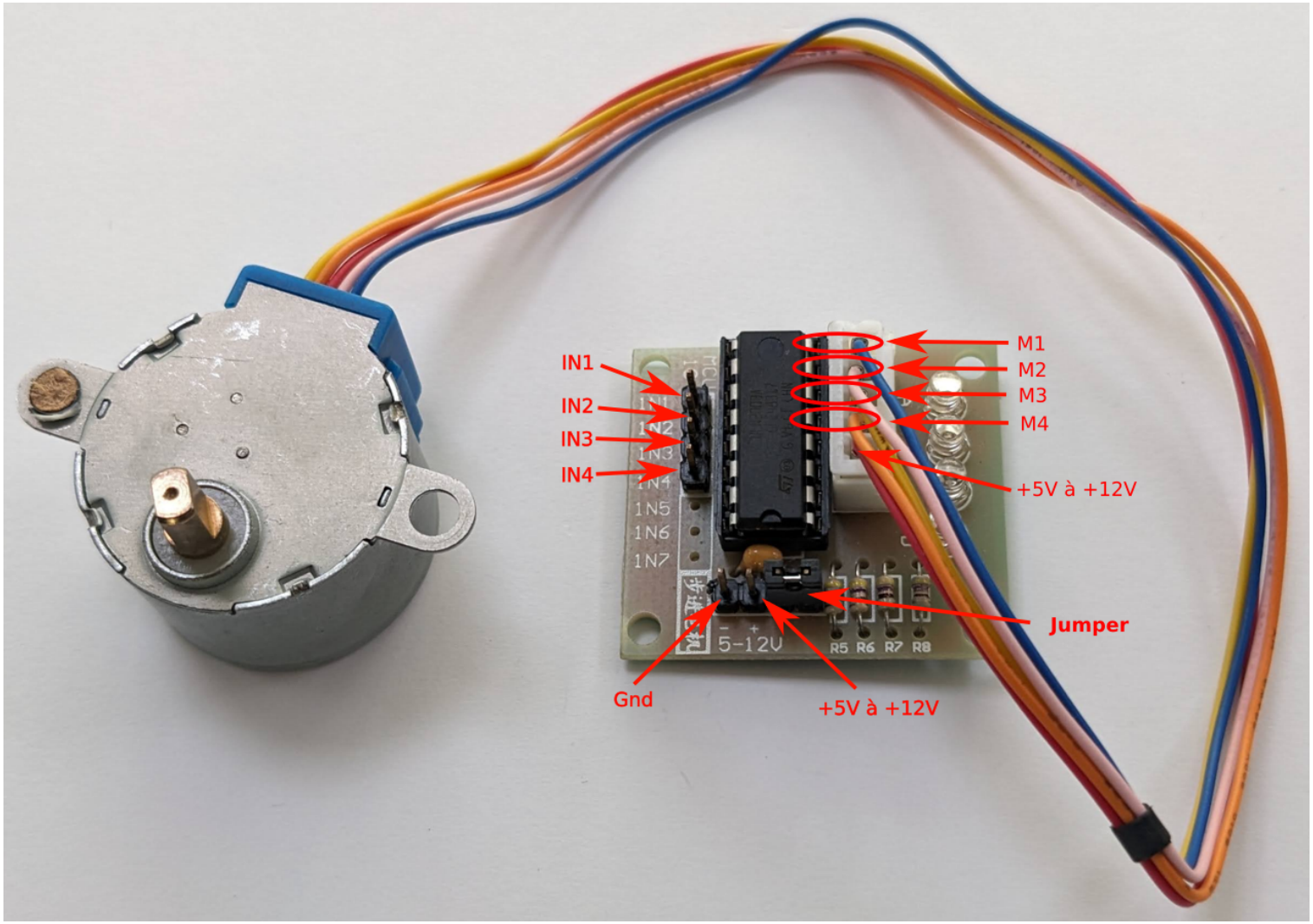
(présent dans la version courte)

Le moteur pas à pas 5 fils - Schéma avec Mosfet



(présent dans la version courte)

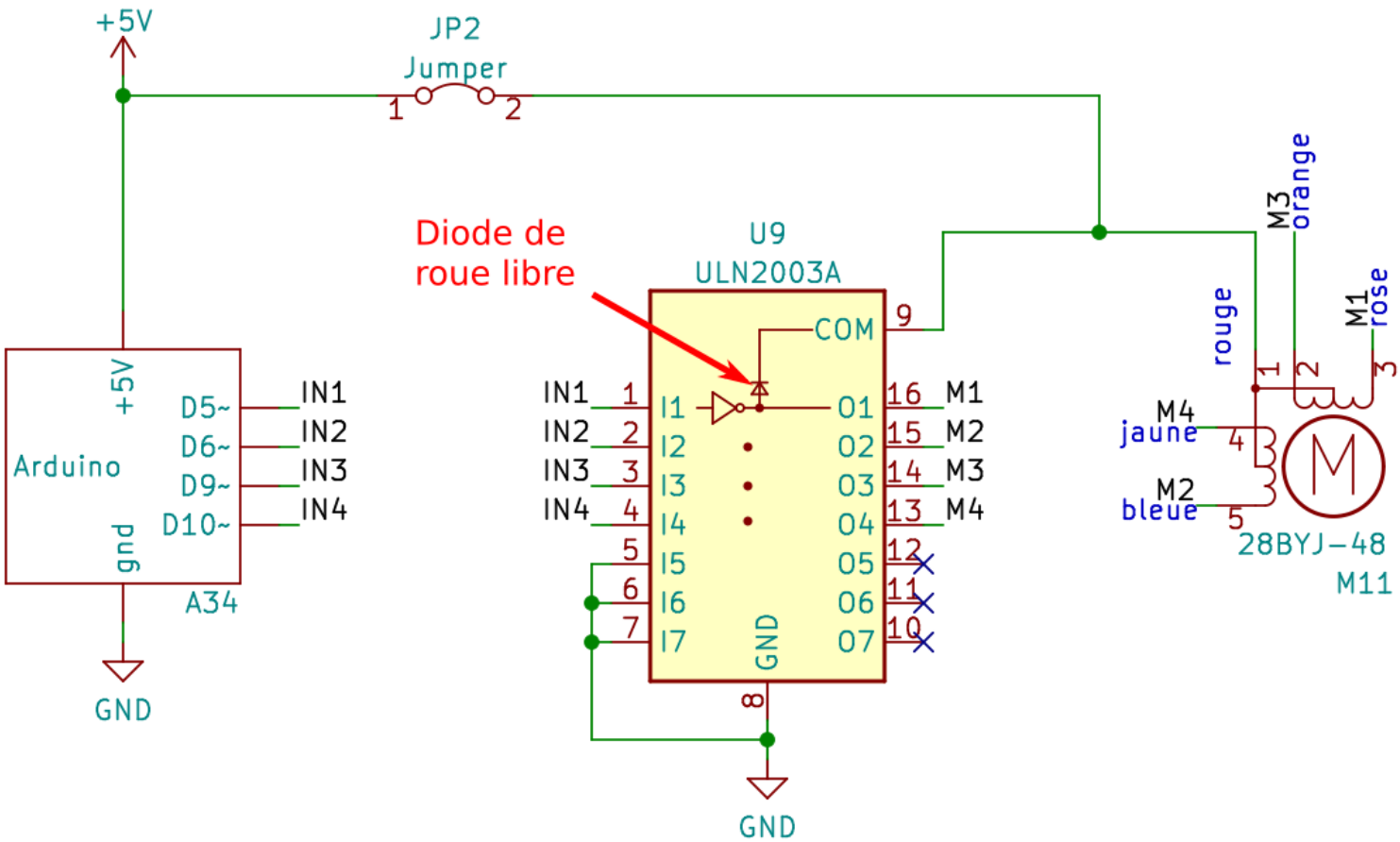
Présentation du module ULN2003 du kit Arduino



Une liste plus exhaustive de driver peut se trouver à la page 666.

(présent dans la version courte)

Le moteur pas à pas 5 fils - Schéma avec ULN2003



Programmer un moteur pas à pas 5 fils : le chronogramme.

```
enum State { Z=0, P=1, STATE_NB=2 };
// Z : high impedance, P : Positive

const int POLES_NB = 4, CYCLE_SIZE = 8;
int timing_table[POLES_NB][CYCLE_SIZE] = {
  {P, P, Z, Z, Z, Z, Z, P}, // Pole 1 : v1
  {Z, P, P, P, Z, Z, Z, Z}, // Pole 2 : v2
  {Z, Z, Z, P, P, P, Z, Z}, // Pole 3 : v3
  {Z, Z, Z, Z, Z, P, P, P} // Pole 4 : v4
};

enum { GATE=0, PINS_BY_POLES_NB=1 };
int control_pins[POLES_NB][PINS_BY_POLES_NB] = {
  {5}, // Pole 1 : IN3, (transistor gate)
  {6}, // Pole 2 : IN2, (transistor gate)
  {9}, // Pole 2 : IN1, (transistor gate)
  {10} // Pole 2 : IN4, (transistor gate)
};

int state_to_output[STATE_NB][PINS_BY_POLES_NB] = {
  {  LOW }, // Z,  (transistor is open)
  { HIGH } // P,  (transistor is closed)
};
```

code/stepper_5_wires.cpp

(présent dans la version courte)

Programmer un moteur pas à pas 5 fils : les roues libres

```
void free_wheeling(){
  for(int i=0; i<POLES_NB; i++){
    for(int j=0; j<PINS_BY_POLES_NB; j++){
      digitalWrite(
        control_pins[i][j],
        state_to_output[ Z ][j]
      );
    }
  }
}

// There is no break motor mode when using 5 wires.
void brake_motor(){ assert(false); }

void setup(){
  for(int i=0; i<POLES_NB; i++){
    for(int j=0; j<PINS_BY_POLES_NB; j++){
      pinMode(control_pins[i][j], OUTPUT);
    }
  }
  free_wheeling();
}
```

code/stepper_5_wires.cpp

Programmer un moteur pas à pas 5 fils : le mode pas à pas.

```
int step = 0;

void make_a_step(bool forward){

  if(forward){
    step = step + 1;
    if(step >= CYCLE_SIZE){ step = 0; }
  }else{
    step = step - 1;
    if(step < 0){ step = CYCLE_SIZE; }
  }

  for(int i=0; i<POLES_NB; i++){
    for(int j=0; j<PINS_BY_POLES_NB; j++){
      digitalWrite(
        control_pins[i][j],
        state_to_output[ timing_table[i][step] ][j]
      );
    }
  }

}
```

code/stepper_5_wires.cpp

(présent dans la version courte)

Programmer un moteur pas à pas 5 fils : fin.

code/stepper_5_wires.cpp

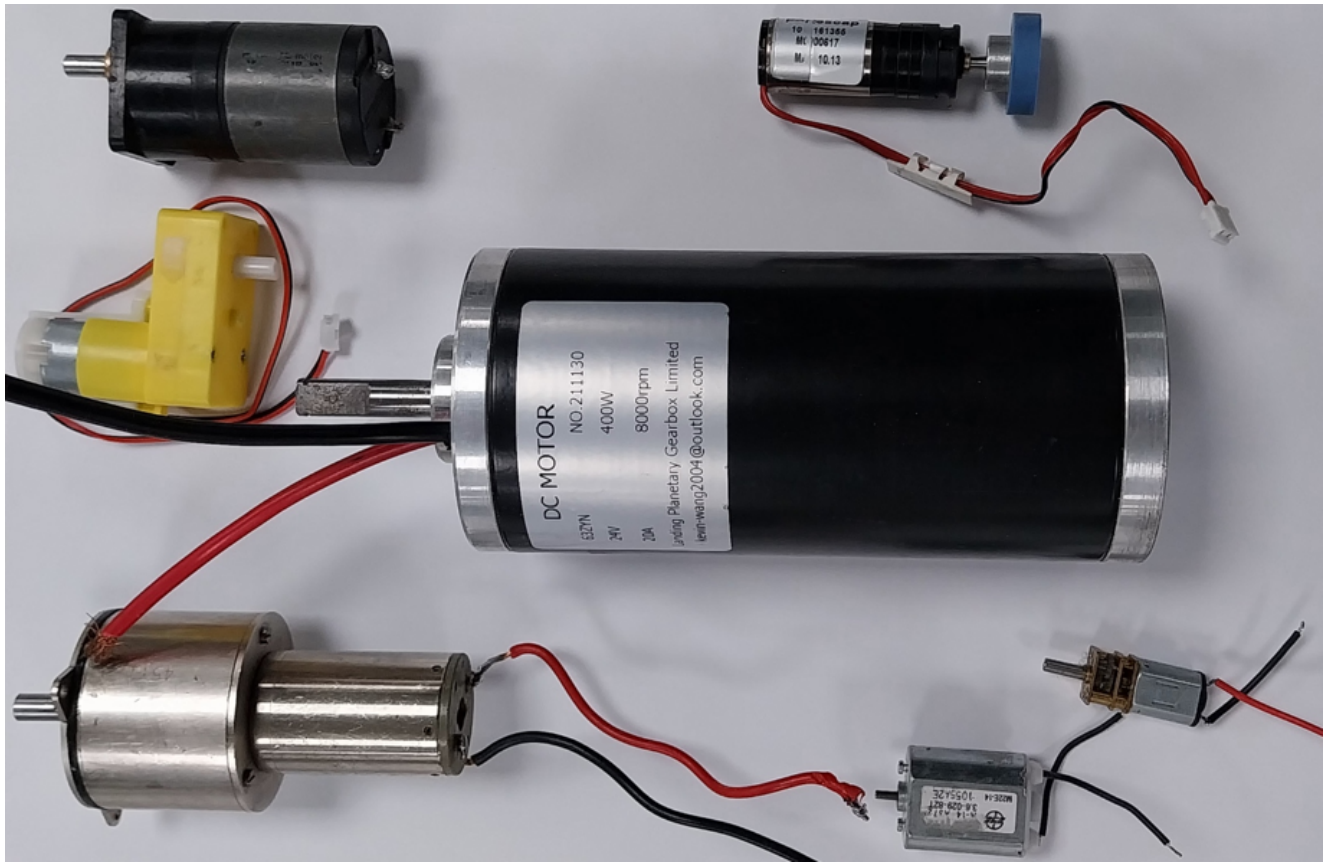
```
unsigned int step_delay = 50000; // in micro seconds
unsigned long last_step_time = 0;

void loop(){
  unsigned long now = micros();
  if( now - last_step_time >= step_delay ){
    last_step_time = now;
    make_a_step(true);
  }
}
```

(version longue)

Plan

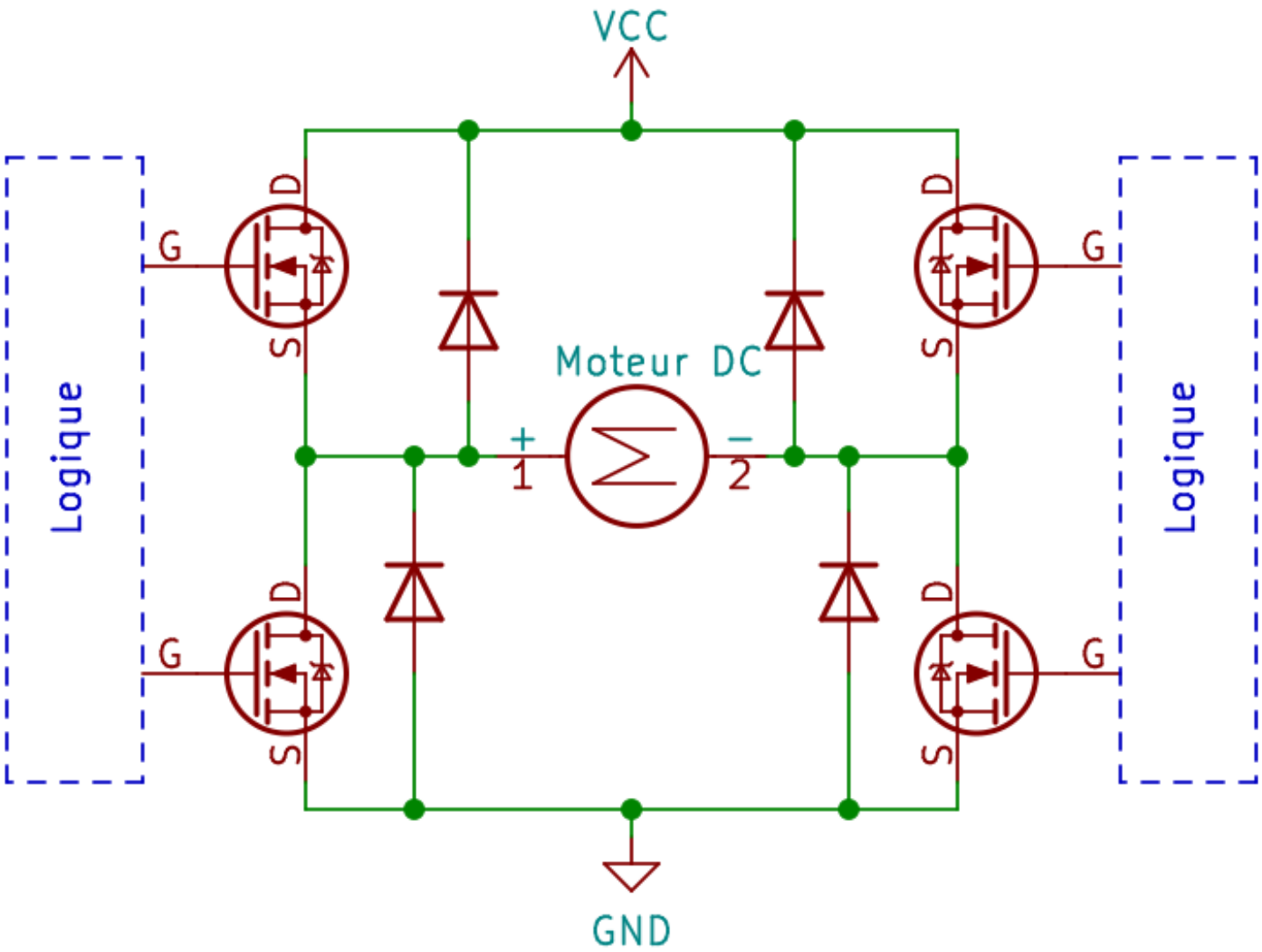
(présent dans la version courte)

Rappel : presentation des moteurs DC



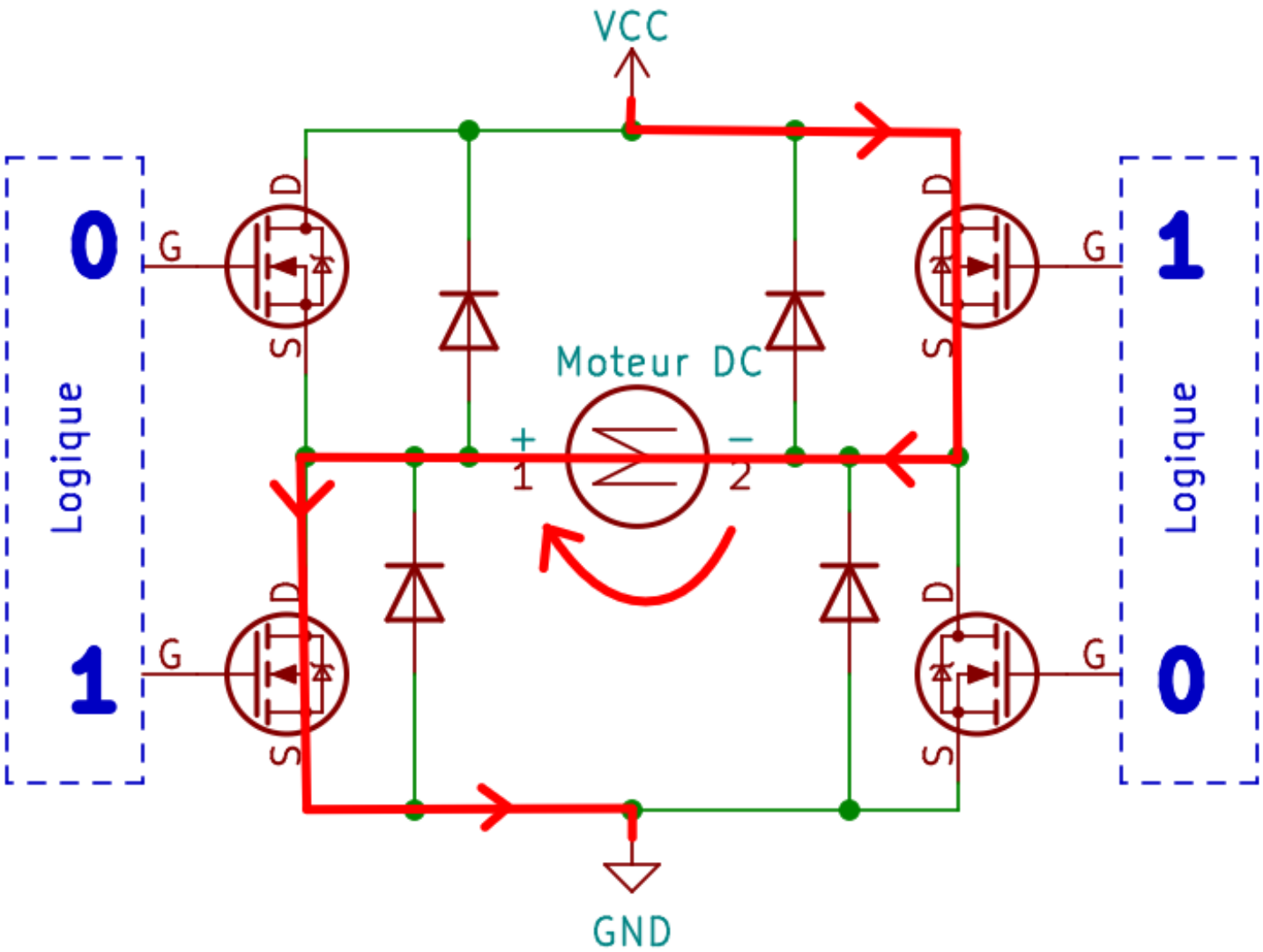
(présent dans la version courte)

Les moteurs DC - tourner dans les deux sens - le pont en H



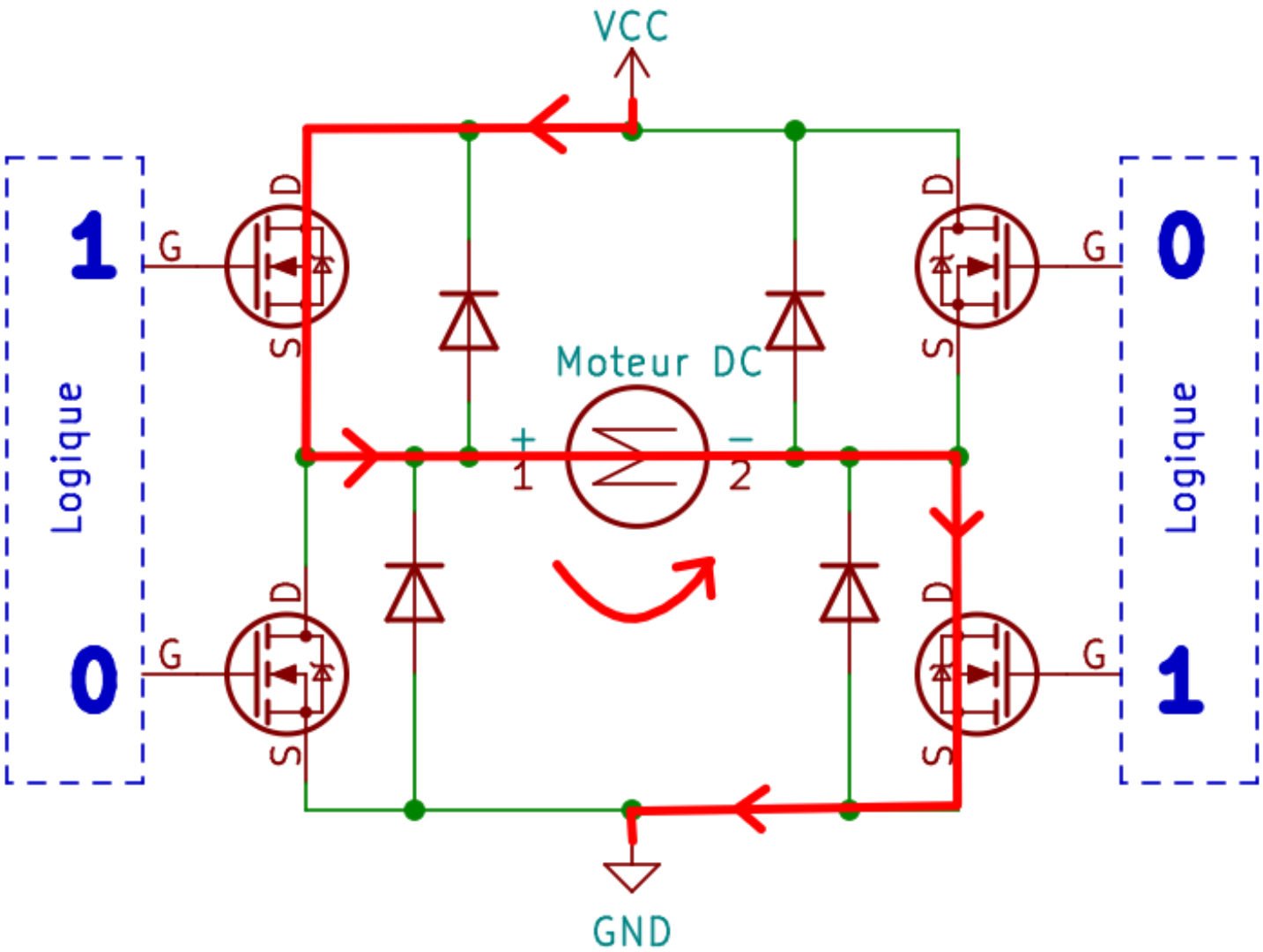
(présent dans la version courte)

Le pont en H : mode normal - sens indirect



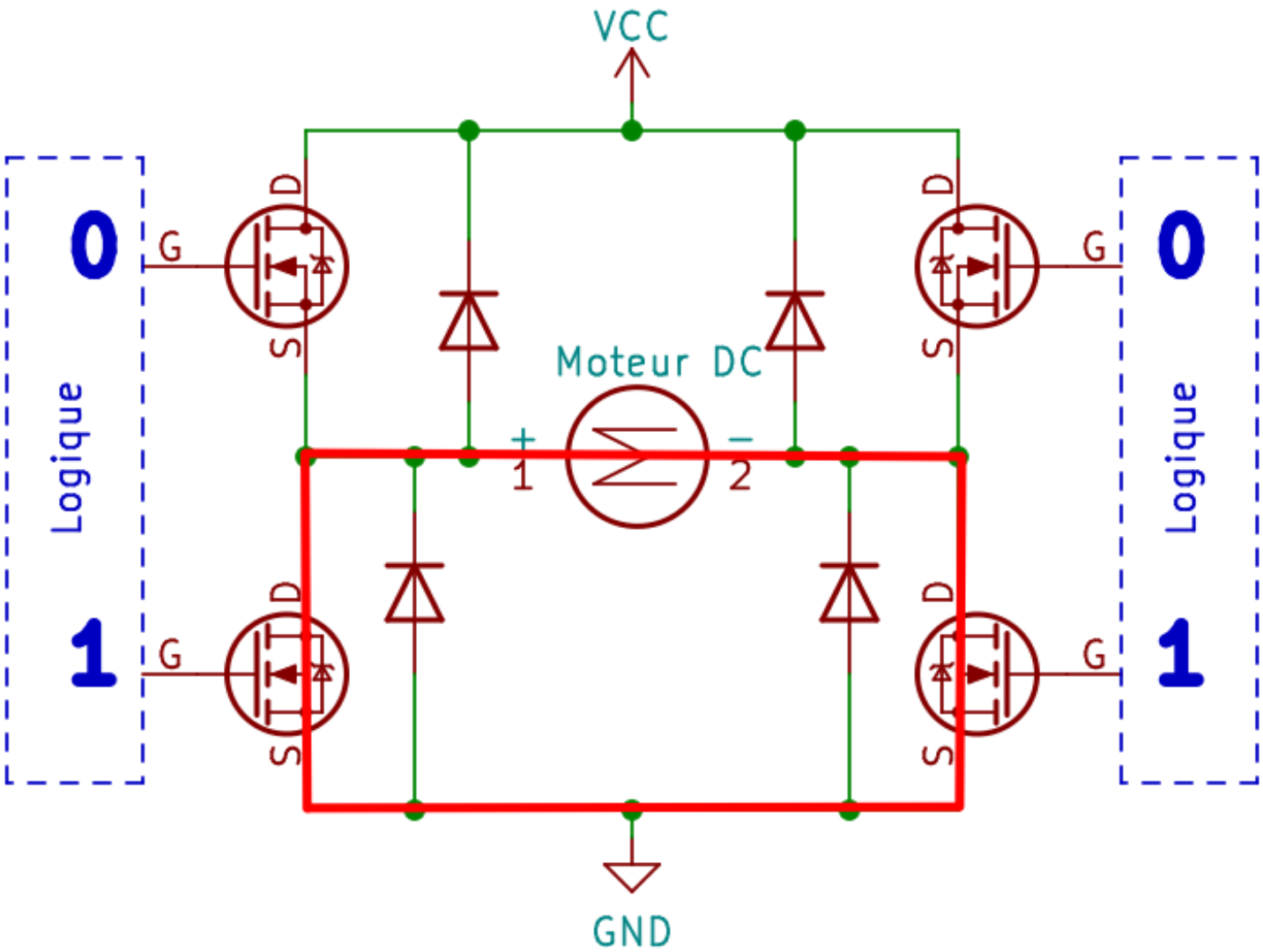
(présent dans la version courte)

Le pont en H : mode normal - sens direct



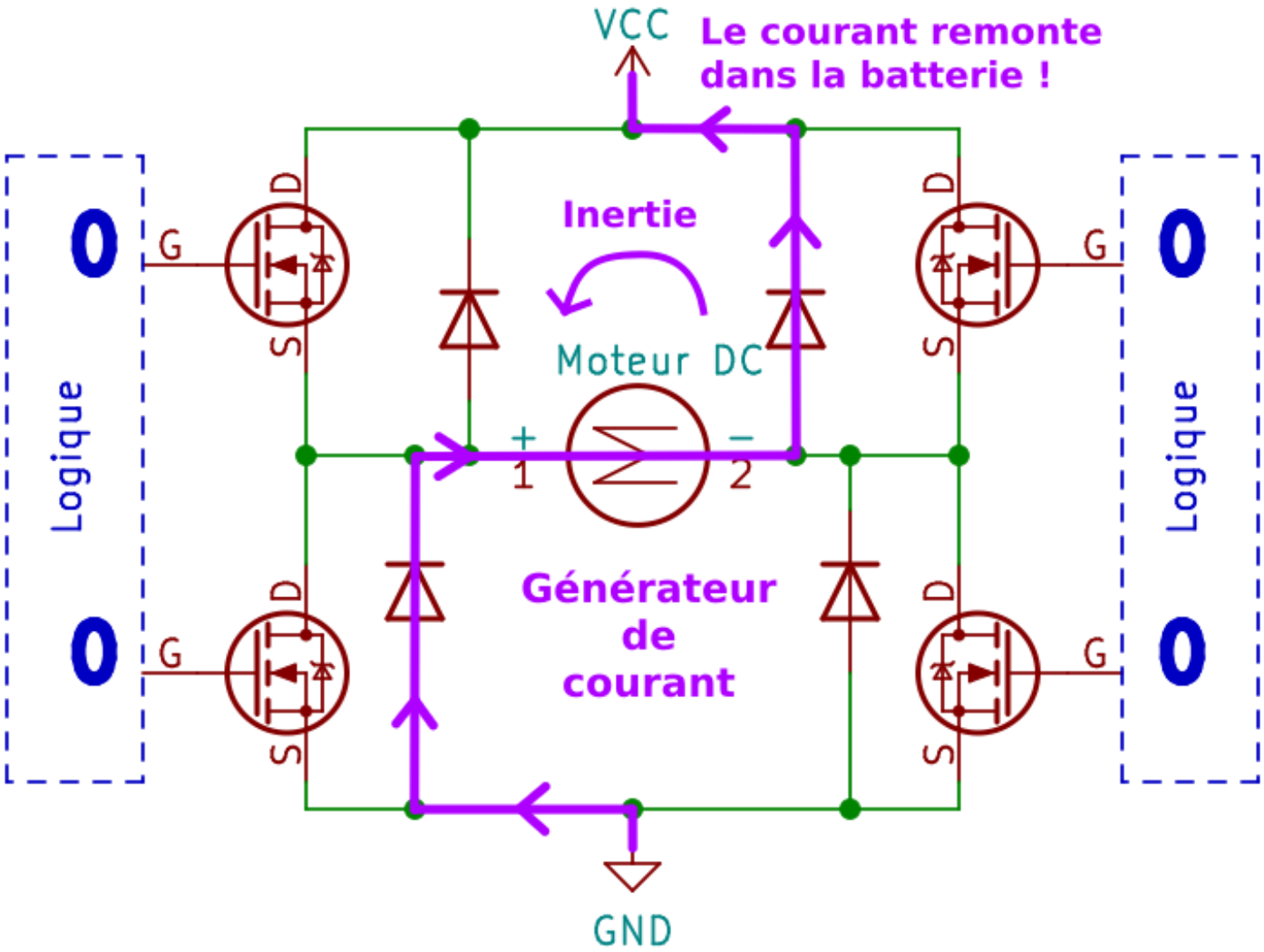
(présent dans la version courte)

Le pont en H : mode frein moteur



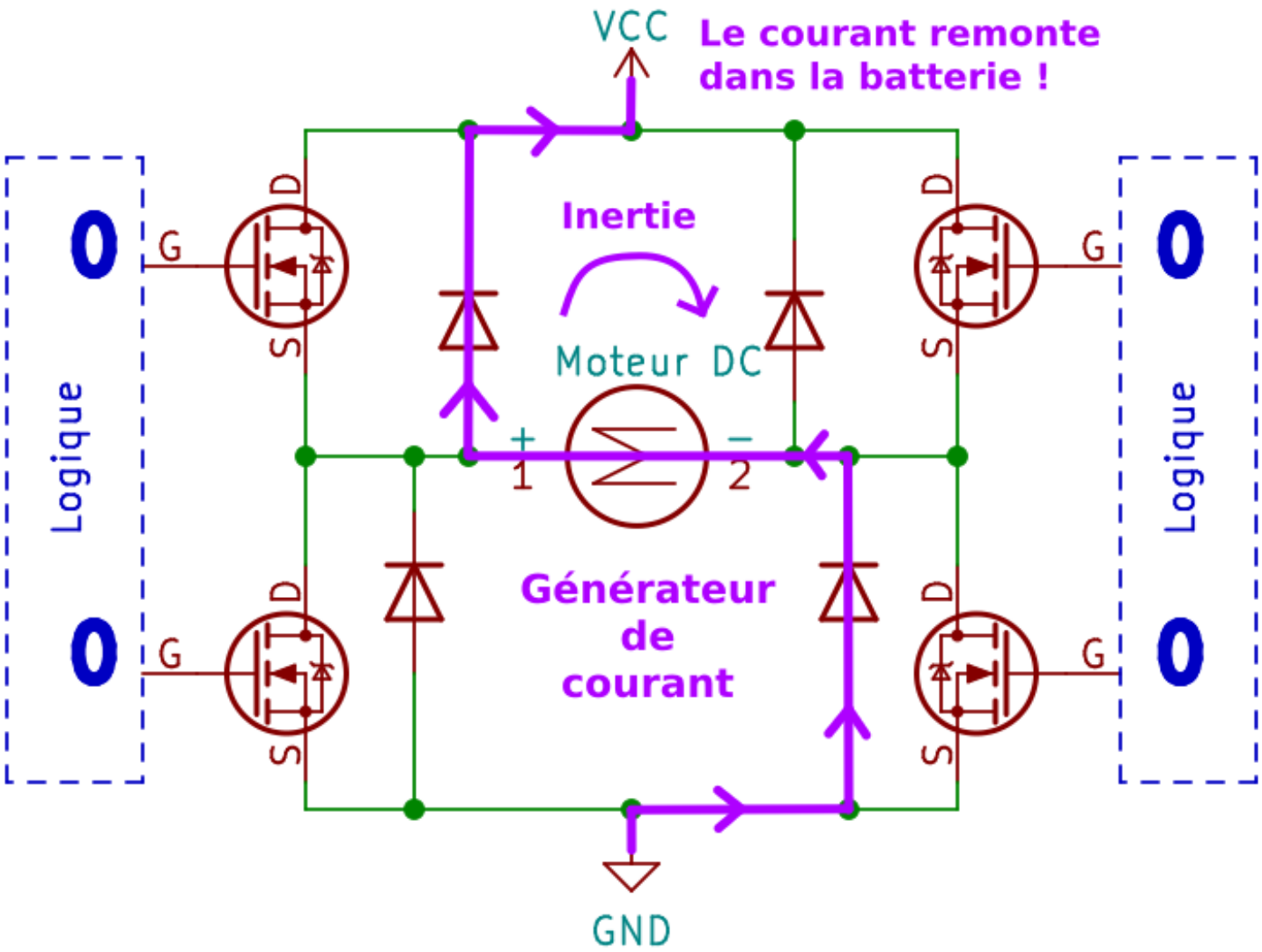
(présent dans la version courte)

Le pont en H : mode roue libre - générateur de courant



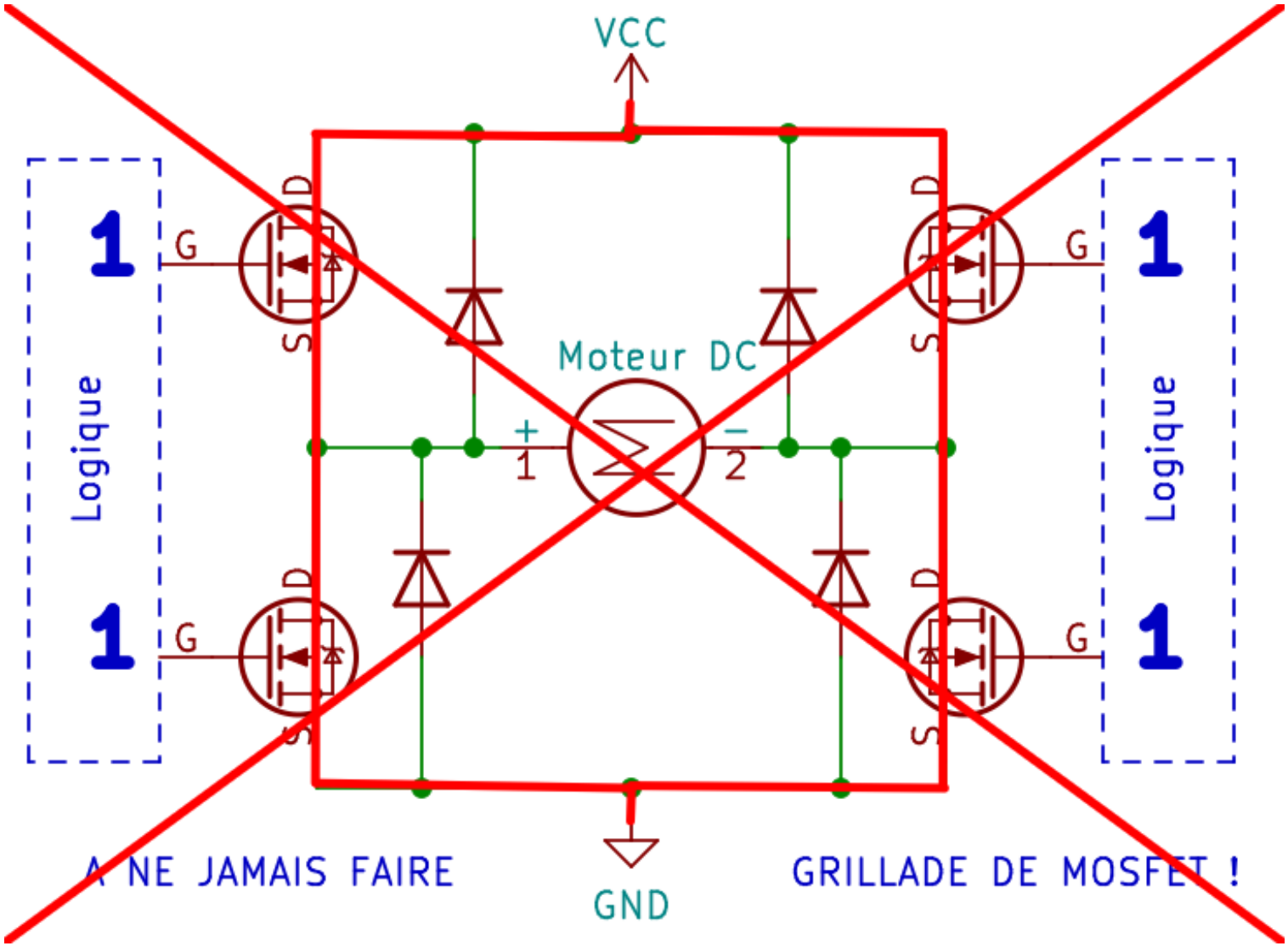
(présent dans la version courte)

Le pont en H : mode roue libre - générateur de courant



(présent dans la version courte)

Le pont en H : mode "autodestruction" !



(présent dans la version courte)

Les drivers de moteurs - le module Adafruit L298N

Faire un pont en H est compliqué : il faut éviter le mode autodestruction!

On utilise des drivers tout fait pour alimenter le pont en H sans erreur.

Avec ces drivers, on commande un circuit logique qui se charge de commander correctement les transistors.

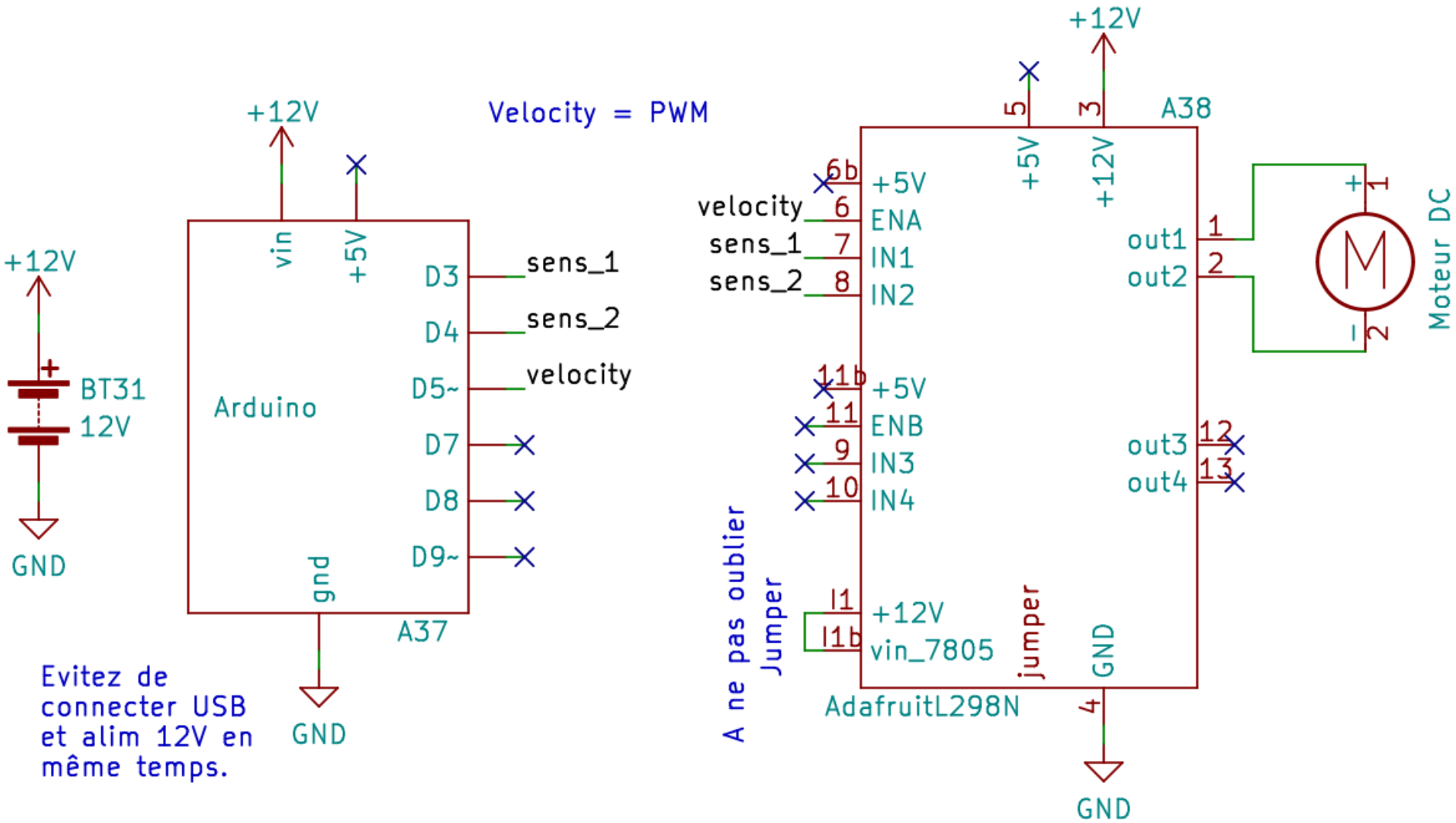
Chaque drivers est différent. Il faut consulter leur notice d'utilisation. Une liste de drivers et modules peut se trouver à la page 666.

Par exemple, le circuit L298N est un circuit pouvant commander deux moteurs. Chaque moteur se contrôle avec 3 entrées EN, IN1 et IN2 de la manière suivante :

EN	IN1	IN2	mode
0	0/1	0/1	roue libre
1	a	a	frein moteur ($a \in \{0,1\}$)
1	0	1	tourne en sens direct
1	1	0	tourne en sens indirect

(présent dans la version courte)

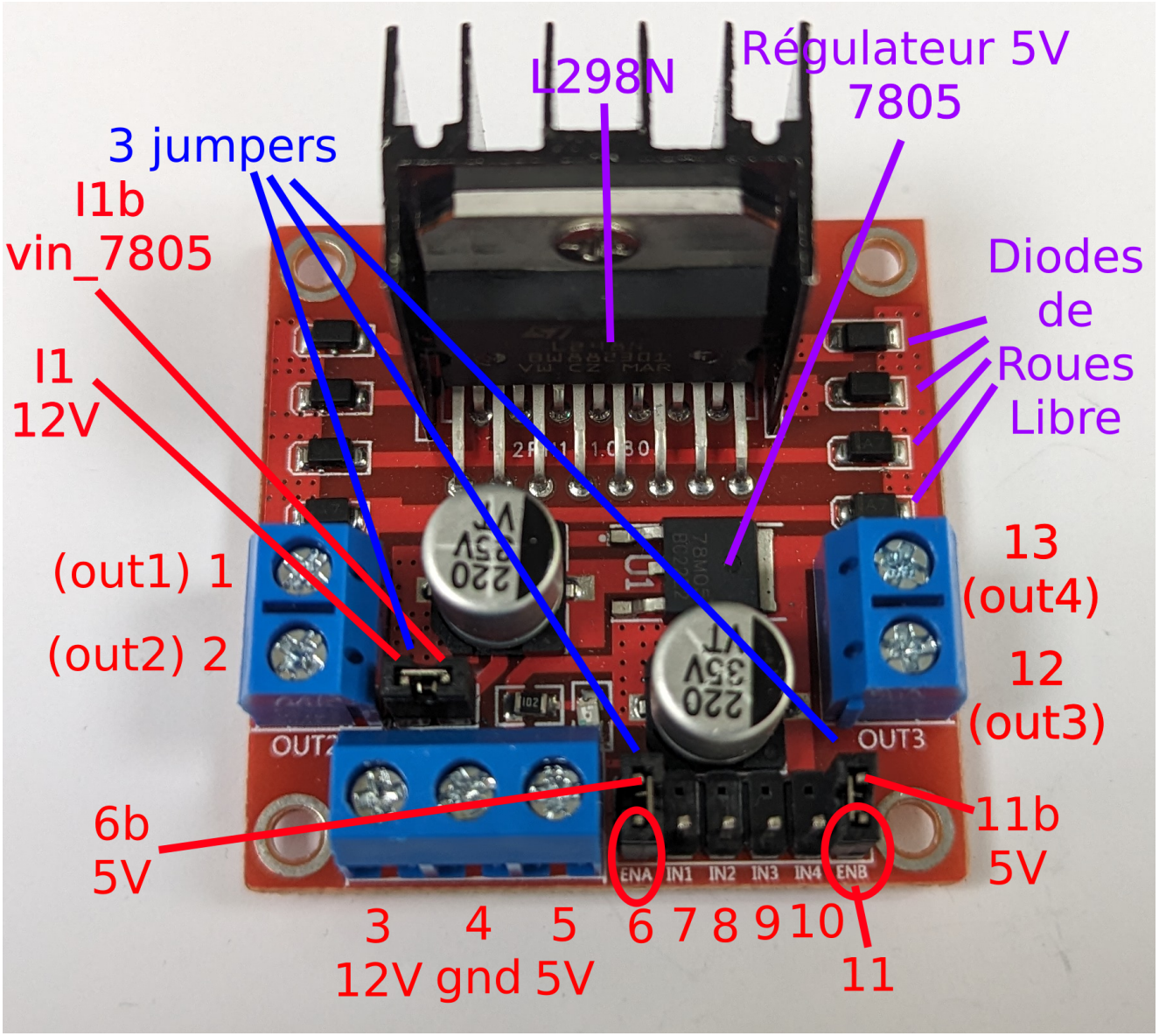
Les moteurs DC - le module Adafruit L298N



Faites attention à bien retirer les 2 cavaliers 6-6b et 11-11b et à laisser connecter le cavalier I1-I1b. À cause de la présence de quatre alimentations (l'USB, la batterie et les deux régulateurs internes de l'arduino ET du L298N), certains montages peuvent détruire le port USB.

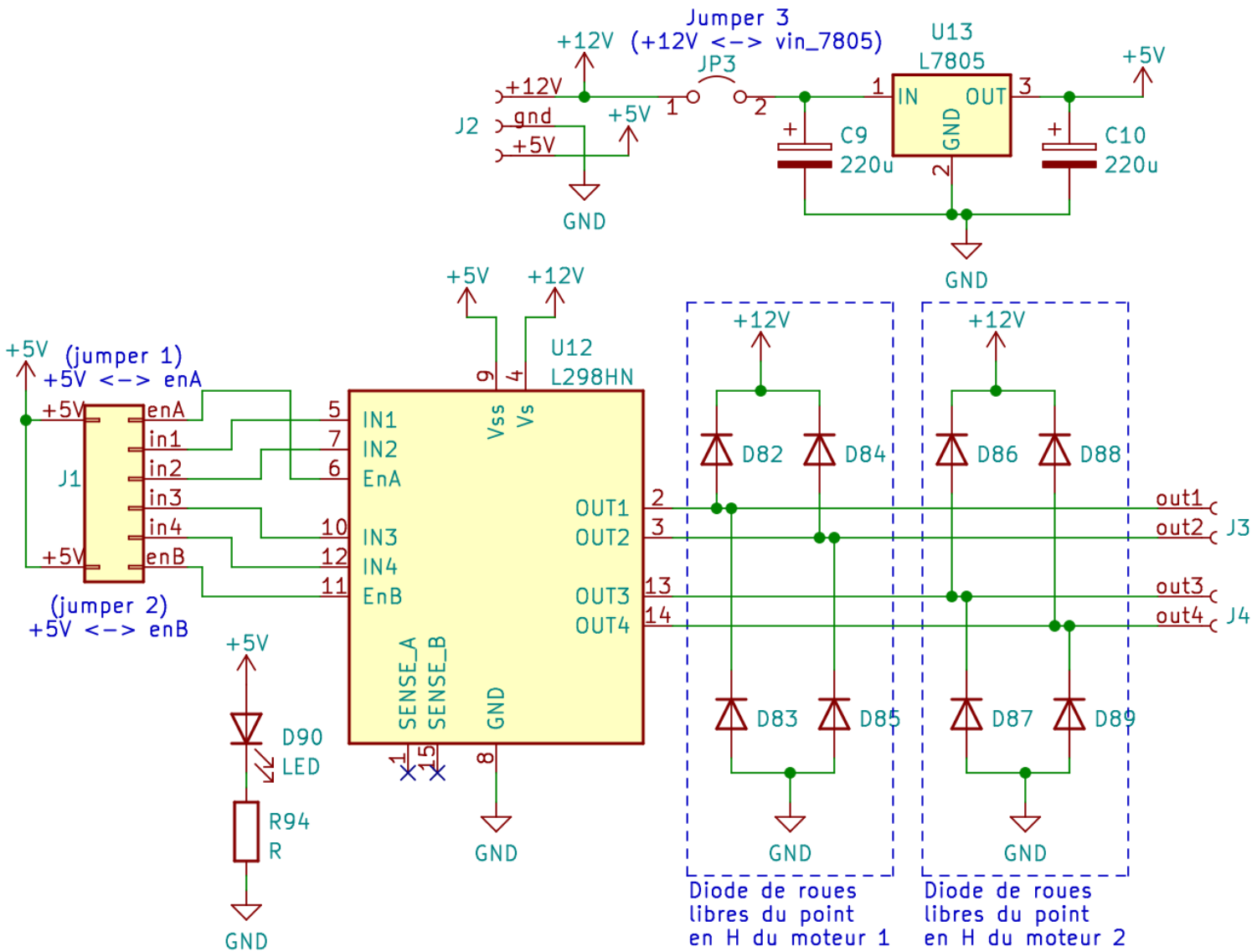
(présent dans la version courte)

Les drivers de moteurs - le module Adafruit L298N



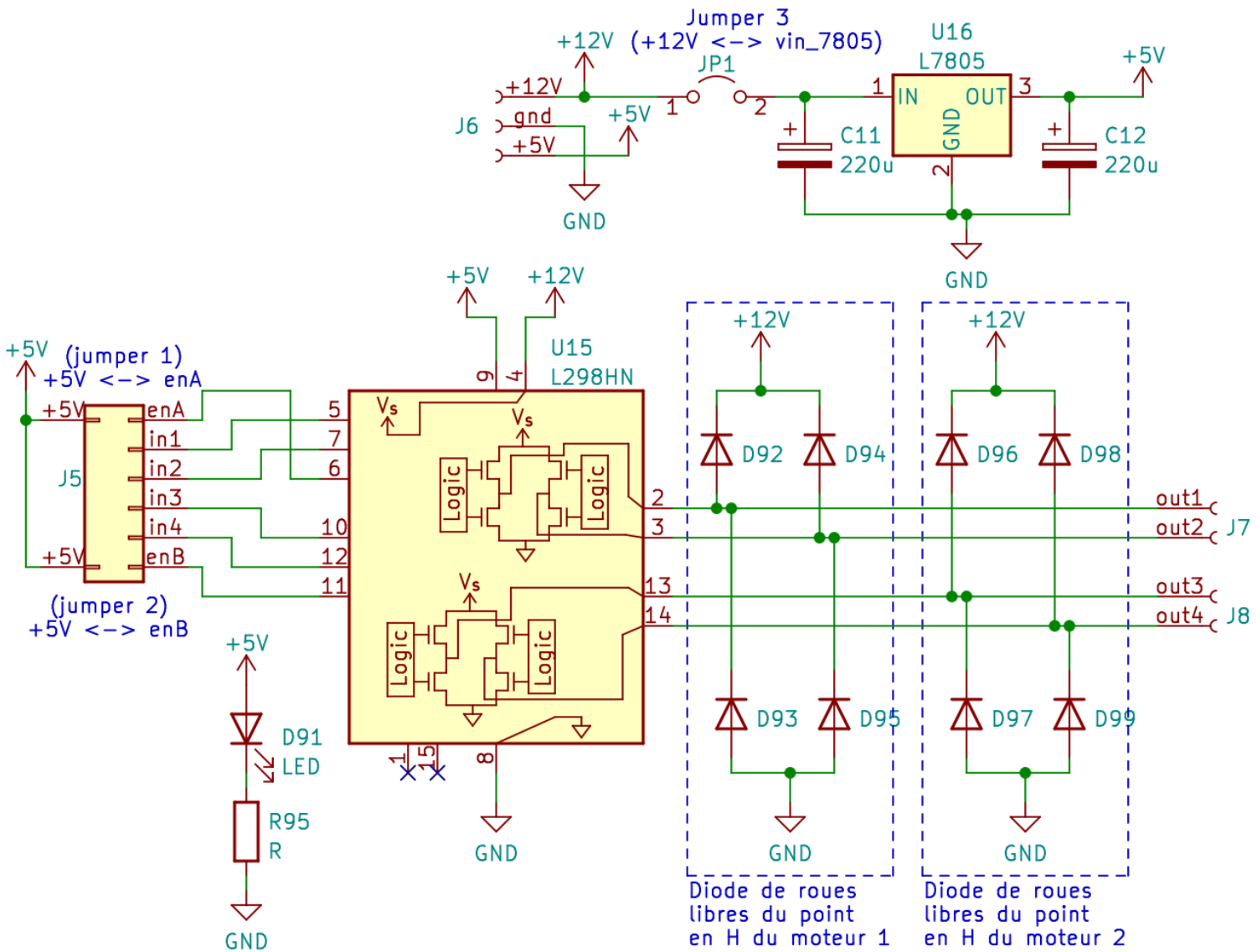
(présent dans la version courte)

Schéma électronique du module Adafruit L298N



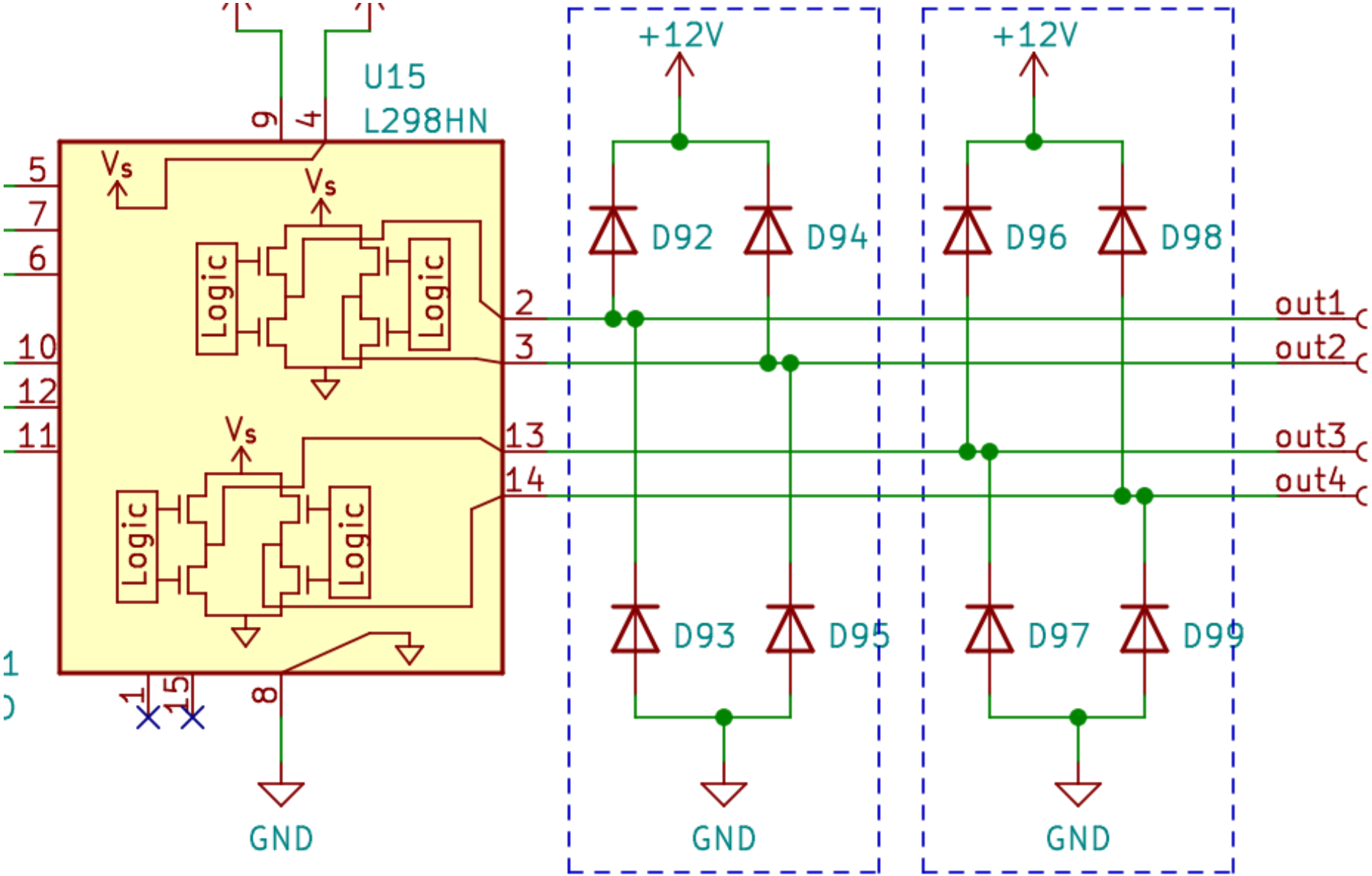
(présent dans la version courte)

Schéma électronique du module Adafruit L298N



(présent dans la version courte)

Schéma électronique du module Adafruit L298N



(présent dans la version courte)

Programmer le driver Adafruit L298N : mode roue libre

EN	IN1	IN2	mode
0	0/1	0/1	roue libre

```
const int DRIVER_ENA_PIN = 5; // PWM
const int DRIVER_IN1_PIN = 6; // Digital
const int DRIVER_IN2_PIN = 7; // Digital

const int PWM_MAX = 255;
const int PWM_MAX_MOTOR = 247; // 97% PWM MAX. Never 100% for
                          // the charge pump of the driver.

void free_wheeling(){
  analogWrite(DRIVER_ENA_PIN, 0);
  digitalWrite(DRIVER_IN1_PIN, LOW);
  digitalWrite(DRIVER_IN2_PIN, LOW);
}
```

code/motor_dc_l298n.cpp

(présent dans la version courte)

Programmer le driver Adafruit L298N : mode frein moteur

EN	IN1	IN2	mode
1	a	a	frein moteur ($a \in \{0,1\}$)

```
void motor_break(){
  digitalWrite(DRIVER_IN1_PIN, LOW);
  digitalWrite(DRIVER_IN2_PIN, LOW);
  analogWrite(DRIVER_ENA_PIN, PWM_MAX);
}

void setup() {
  pinMode(DRIVER_IN1_PIN, OUTPUT);
  pinMode(DRIVER_IN2_PIN, OUTPUT);
  motor_break();
}
```

code/motor_dc_l298n.cpp

(présent dans la version courte)

Programmer le driver Adafruit L298N : mode sens direct

EN	IN1	IN2	mode
0	0/1	0/1	roue libre
1	0	1	tourne en sens direct

```
void set_pwm(unsigned int pwm){
  pwm = (pwm < PWM_MAX_MOTOR ) ? pwm : PWM_MAX_MOTOR;
  analogWrite(DRIVER_ENA_PIN, pwm);
}
void motor_forward(unsigned int pwm){
  set_pwm(0);
  digitalWrite(DRIVER_IN1_PIN, HIGH);
  digitalWrite(DRIVER_IN2_PIN, LOW);
  set_pwm(pwm);
}
```

code/motor_dc_l298n.cpp

(présent dans la version courte)

Programmer le driver Adafruit L298N : mode sens indirect

EN	IN1	IN2	mode
0	0/1	0/1	roue libre
1	1	0	tourne en sens indirect

```
void motor_backward(unsigned int pwm){
  set_pwm(0);
  digitalWrite(DRIVER_IN1_PIN, LOW);
  digitalWrite(DRIVER_IN2_PIN, HIGH);
  set_pwm(pwm);
}

void run(int pwm){
  if( pwm >= 0 ){
    motor_forward(pwm);
  }else{
    motor_backward(-pwm);
  }
}
```

code/motor_dc_l298n.cpp

(version longue)

Programmer le driver Adafruit L298N : exemple

Le moteur accélère et décélère dans un sens, puis un autre. Enfin, il passe en frein moteur, puis en roue libre avant de recommencer le cycle.

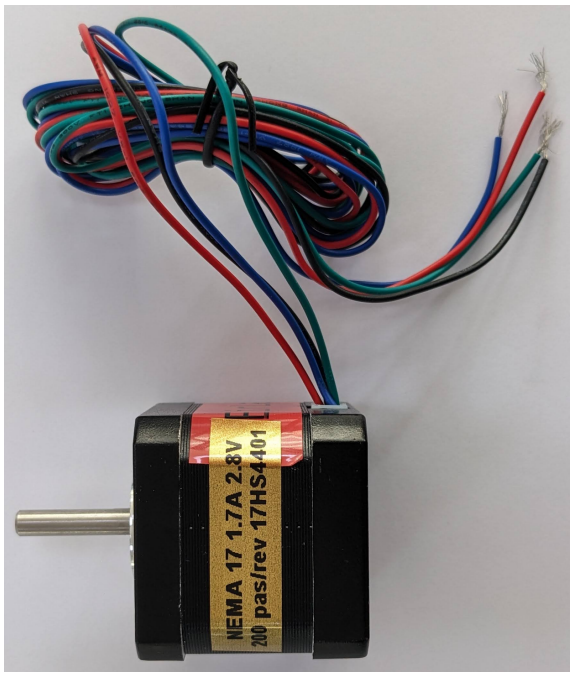
```
void loop(){
  int velocity; // in pwm (maximal velocity = +- PWM_MAX_MOTOR)
  for(velocity = 0; velocity<PWM_MAX_MOTOR; velocity += 10){
    run(velocity);
    delay(160);
  }
  for(velocity = PWM_MAX_MOTOR; velocity>-PWM_MAX_MOTOR; velocity -= 10){
    run(velocity);
    delay(160);
  }
  for(velocity = -PWM_MAX_MOTOR; velocity<0; velocity += 10){
    run(velocity);
    delay(160);
  }
  motor_break();
  delay(3000);
  free_wheeling();
  delay(3000);
}
```

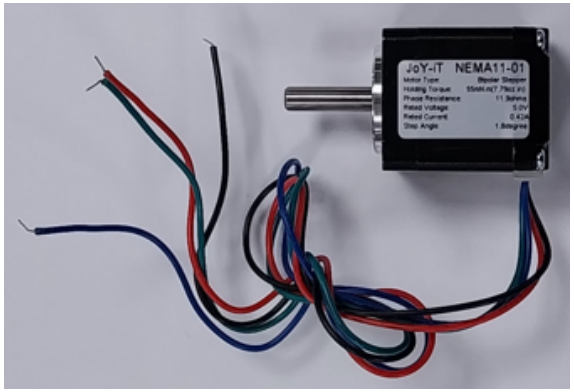
code/motor_dc_l298n.cpp

(version longue)

Plan

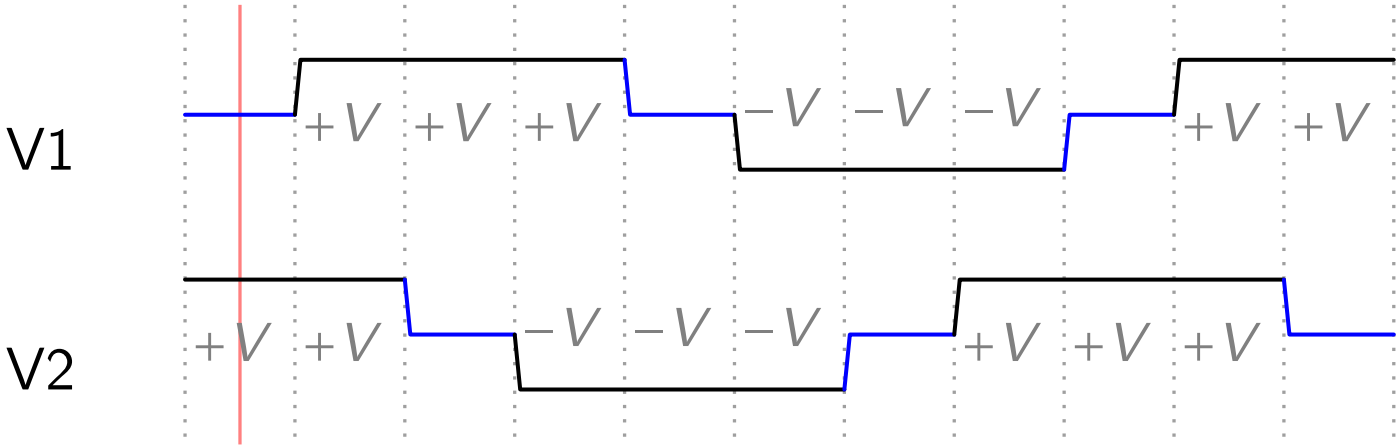
Présentation des moteurs pas à pas 4 fils



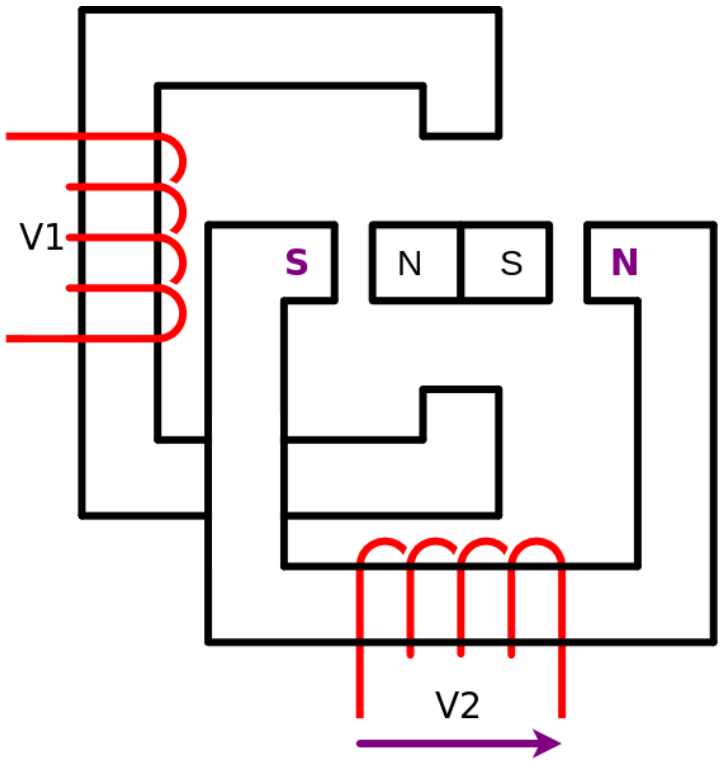


(présent dans la version courte)

Le moteur pas à pas 4 fils - la théorie

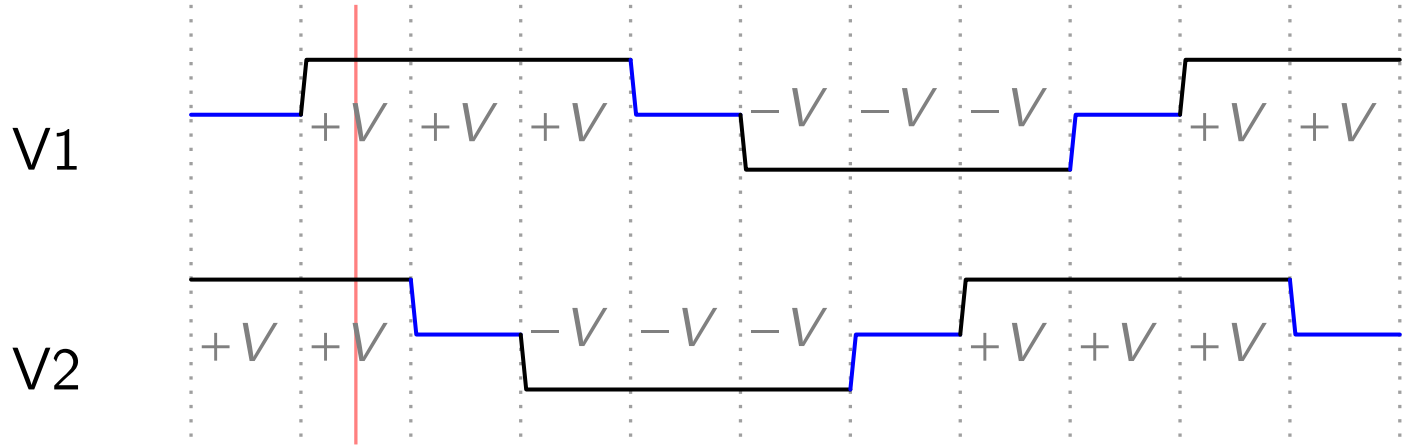


— : Haute impédance/ouvert — : $\pm V$

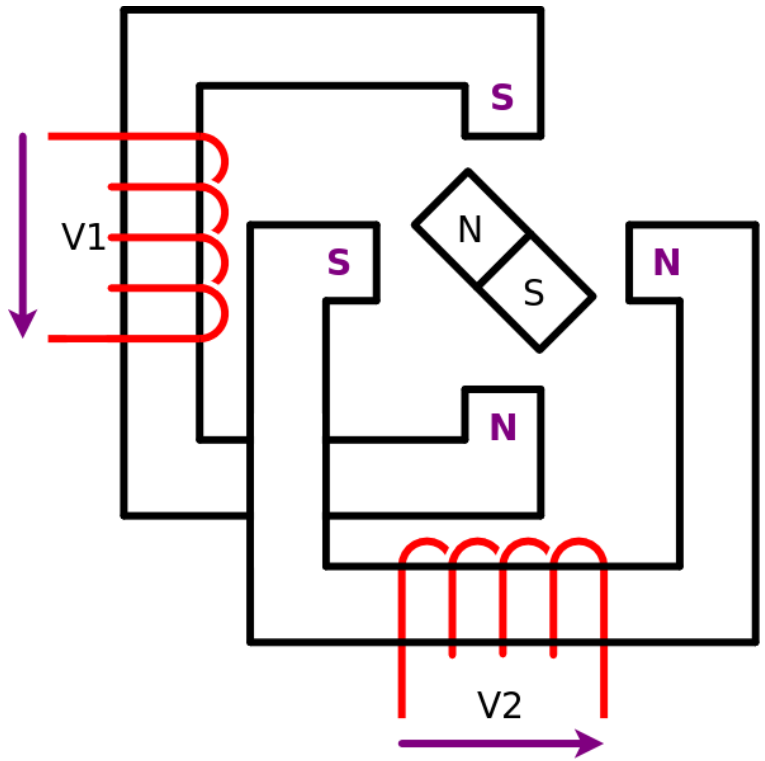


(présent dans la version courte)

Le moteur pas à pas 4 fils - la théorie

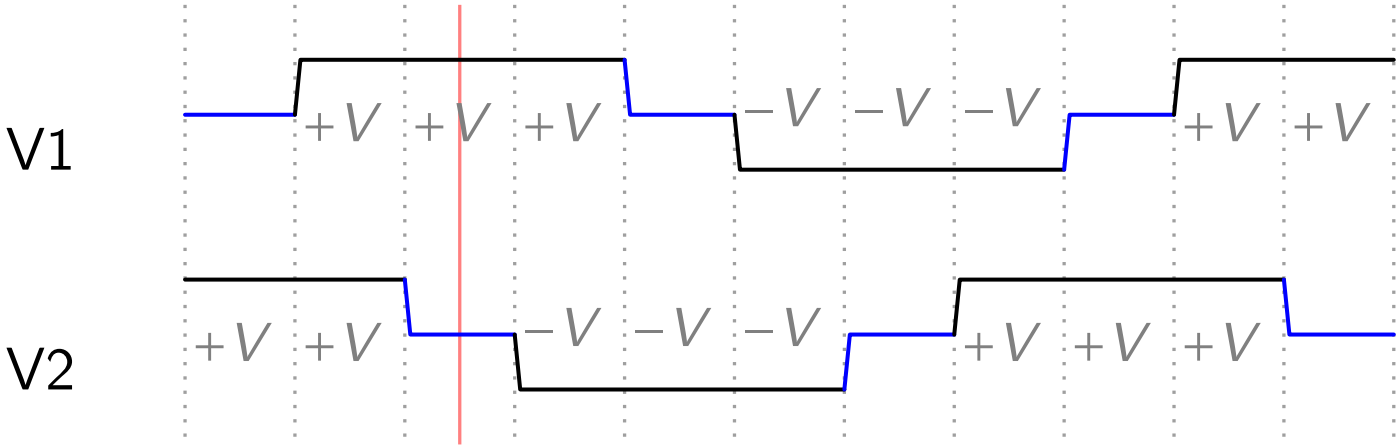


— : Haute impédance/ouvert — : $\pm V$

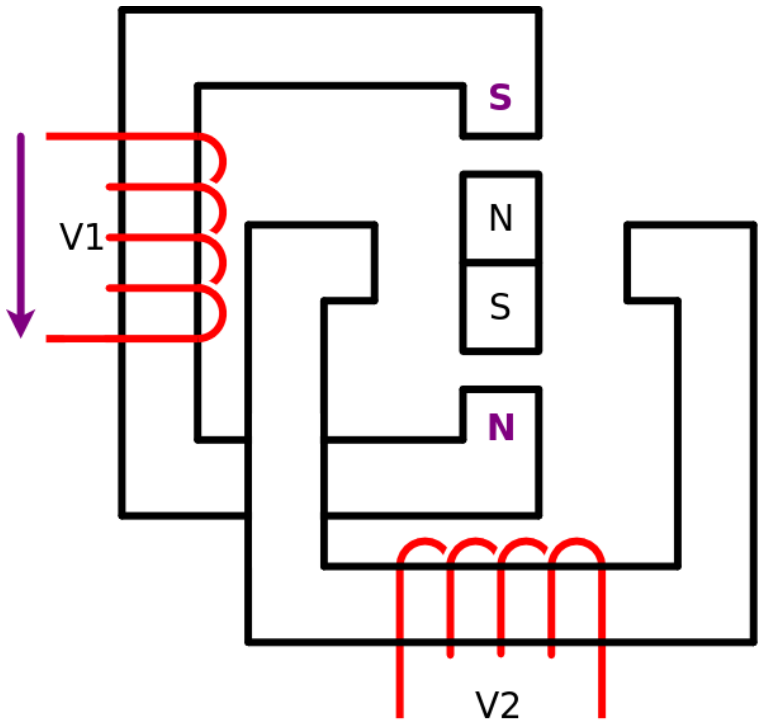


(présent dans la version courte)

Le moteur pas à pas 4 fils - la théorie

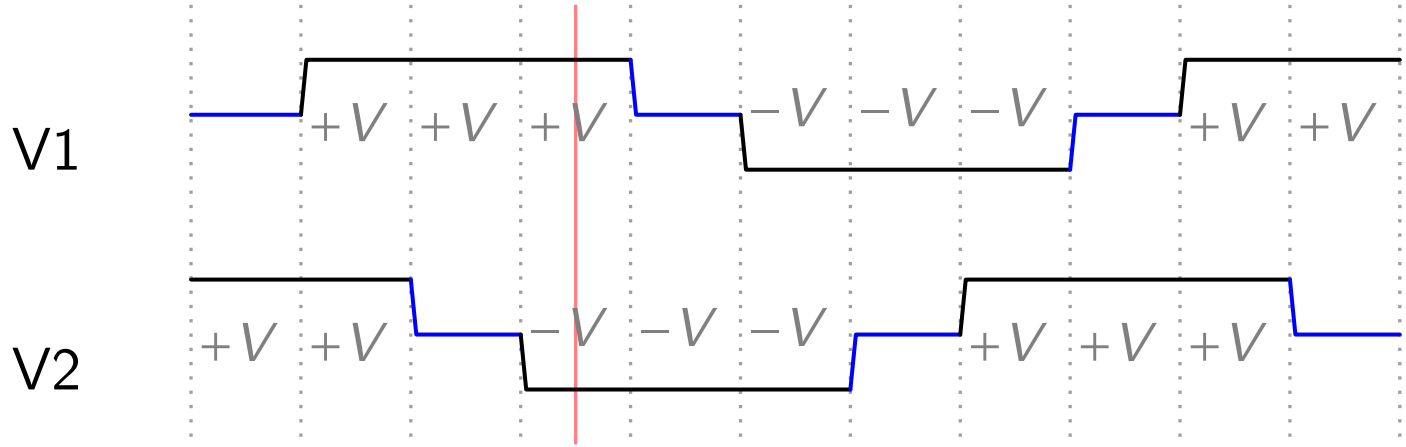


— : Haute impédance/ouvert — : $\pm V$

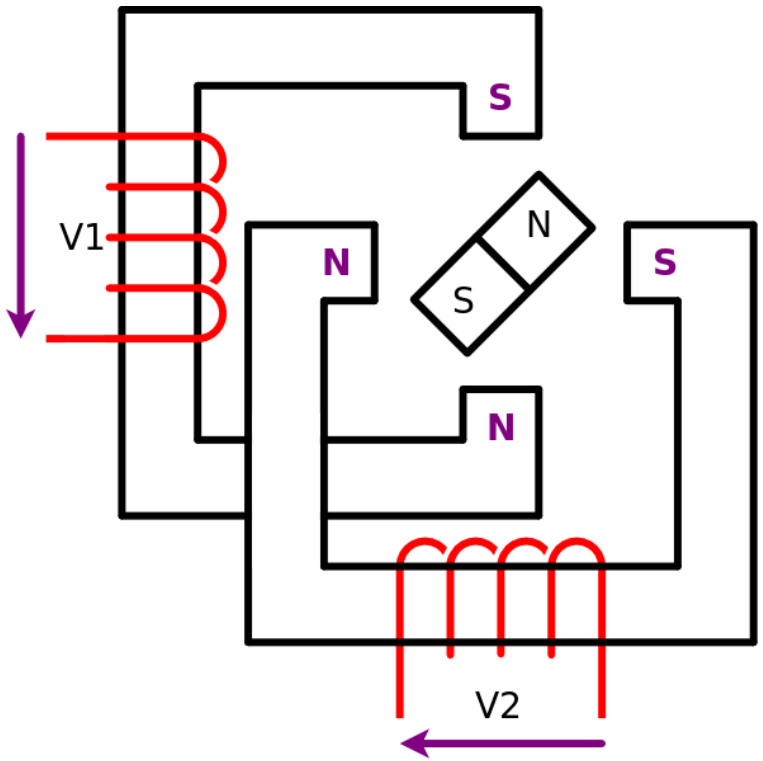


(présent dans la version courte)

Le moteur pas à pas 4 fils - la théorie

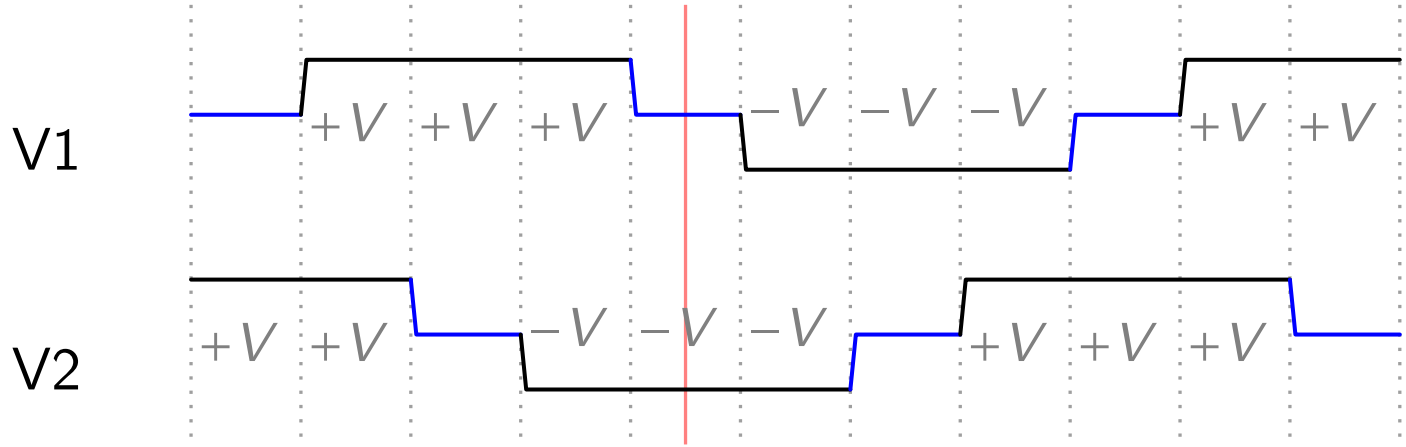


— : Haute impédance/ouvert — : $\pm V$

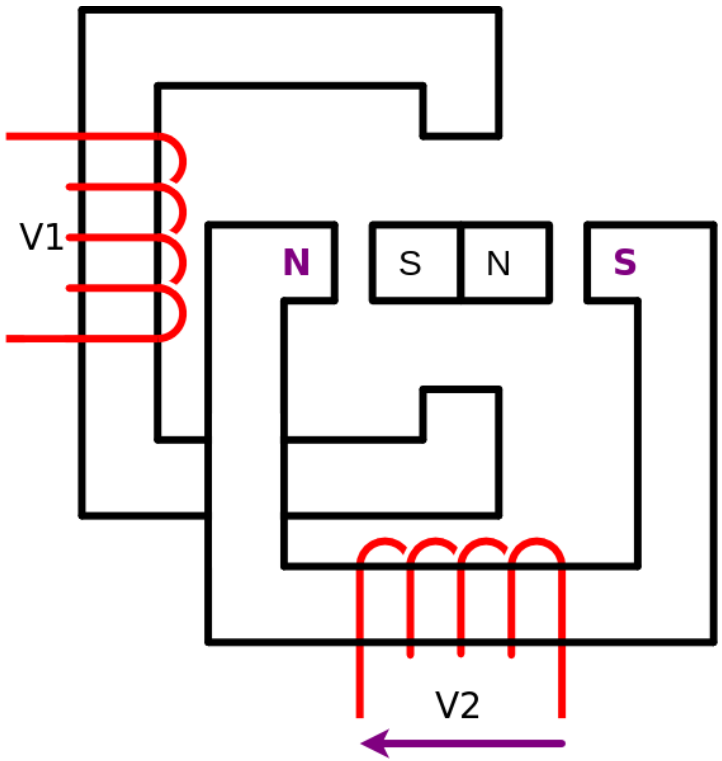


(présent dans la version courte)

Le moteur pas à pas 4 fils - la théorie

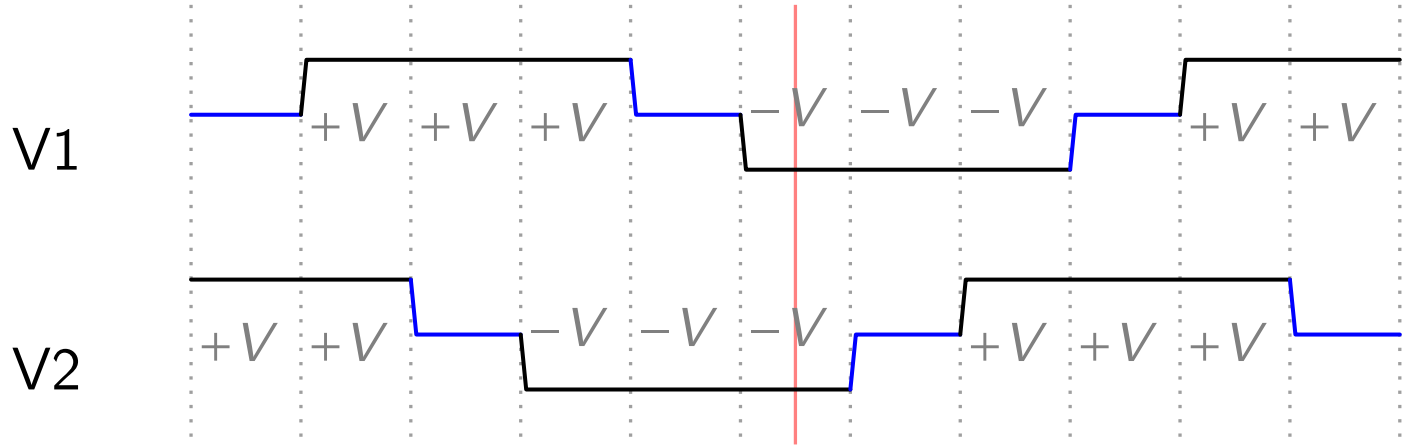


— : Haute impédance/ouvert — : $\pm V$

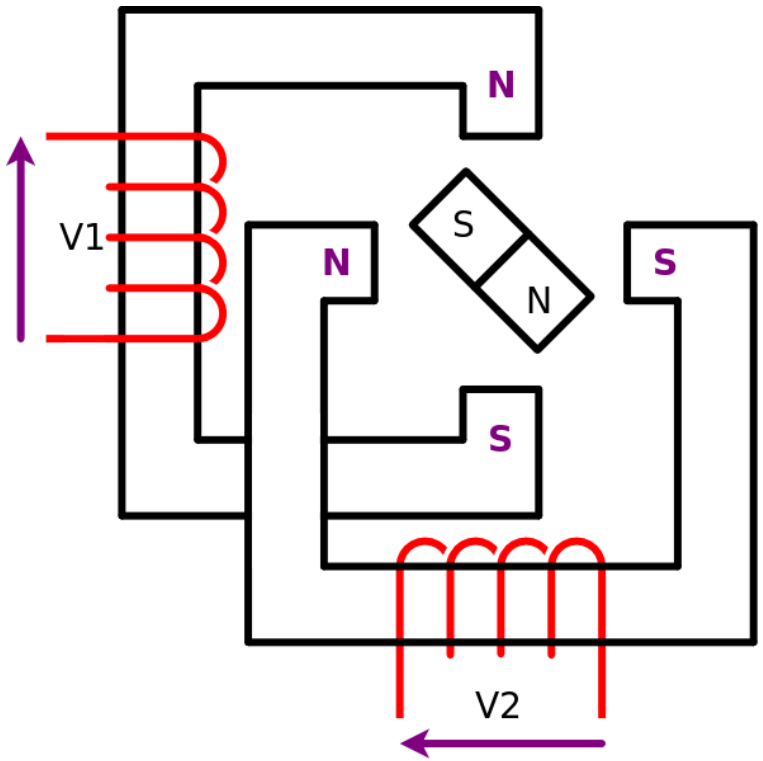


(présent dans la version courte)

Le moteur pas à pas 4 fils - la théorie

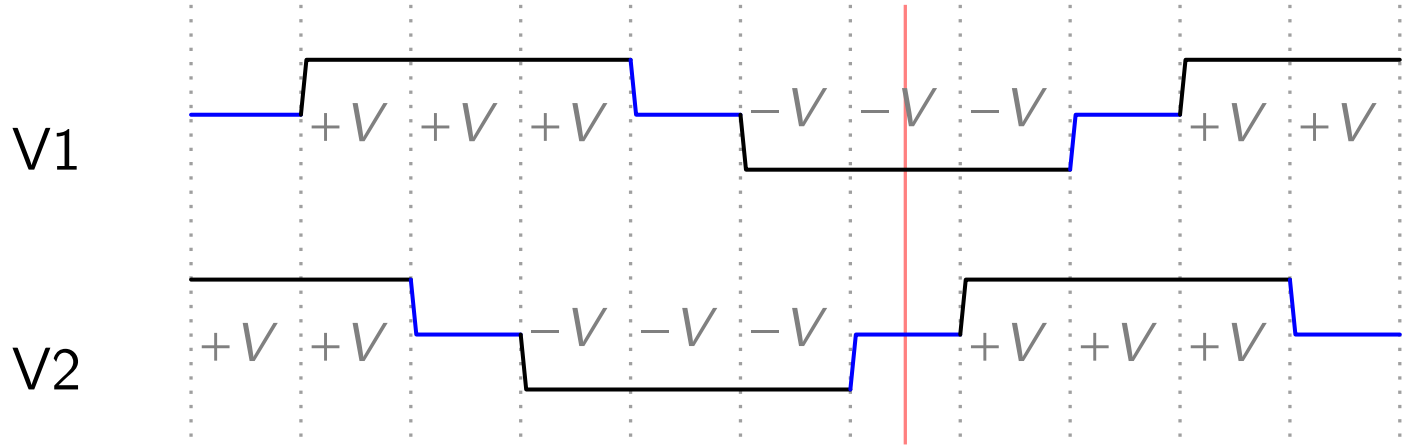


— : Haute impédance/ouvert — : $\pm V$

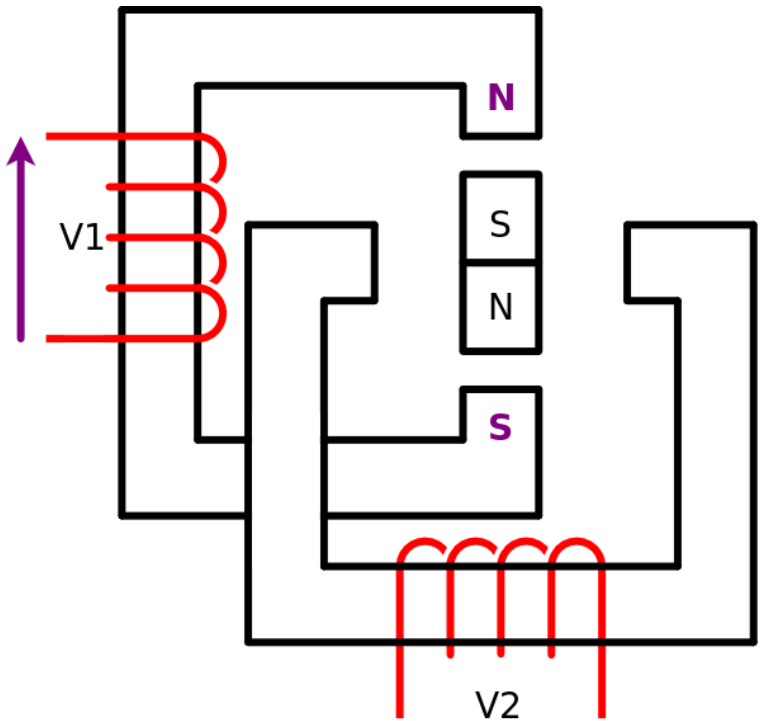


(présent dans la version courte)

Le moteur pas à pas 4 fils - la théorie

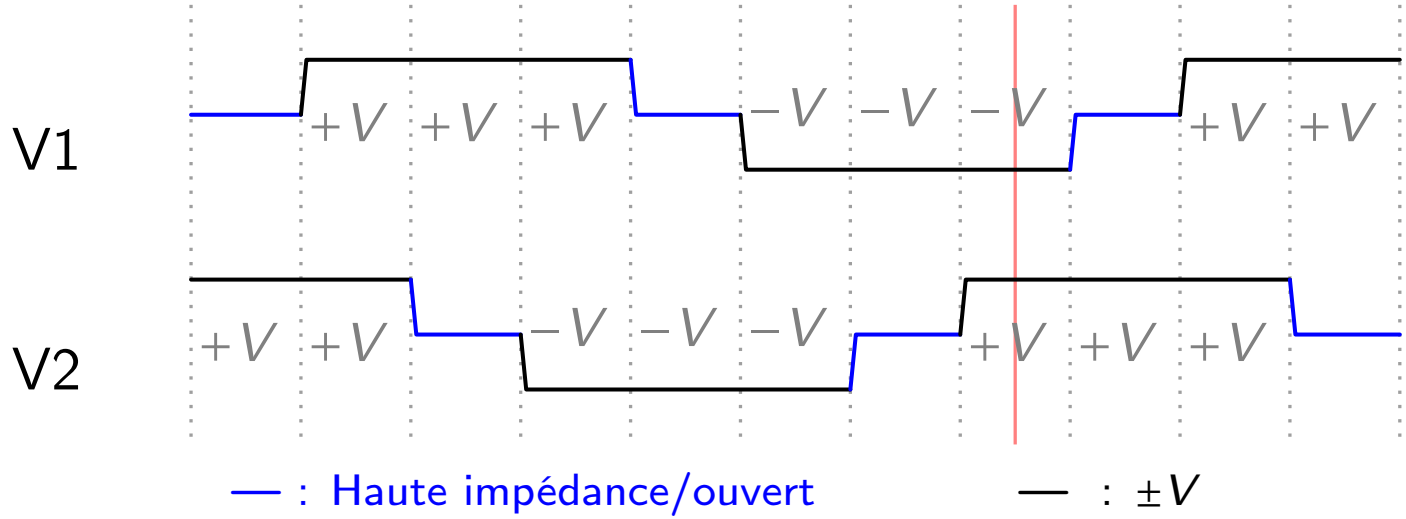


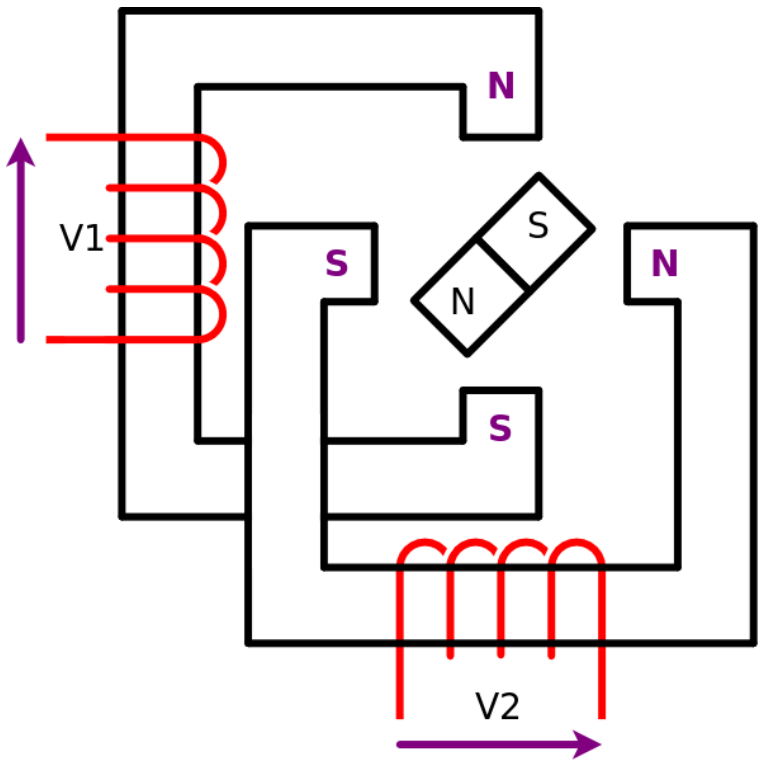
— : Haute impédance/ouvert — : $\pm V$



(présent dans la version courte)

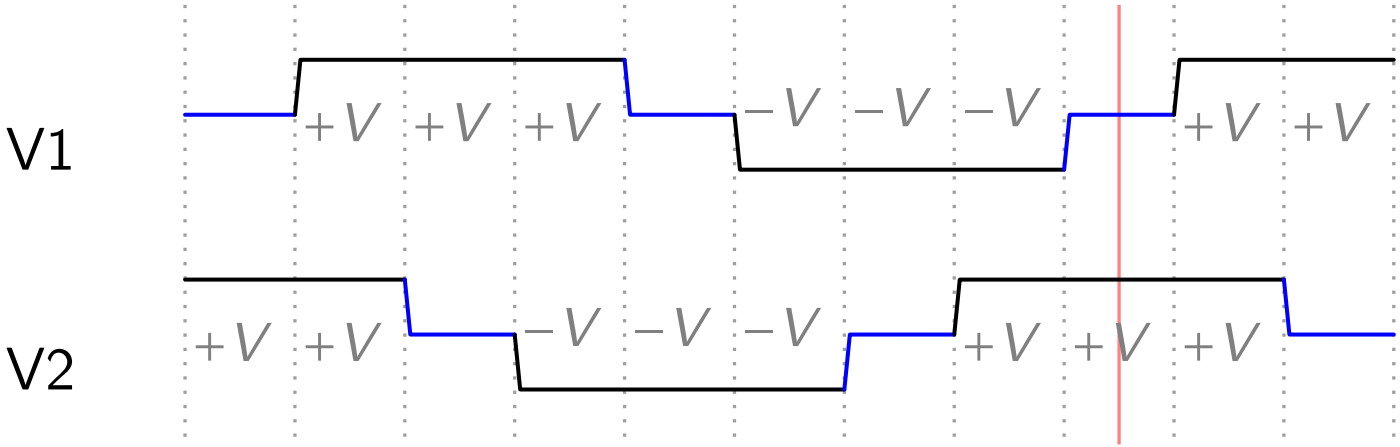
Le moteur pas à pas 4 fils - la théorie



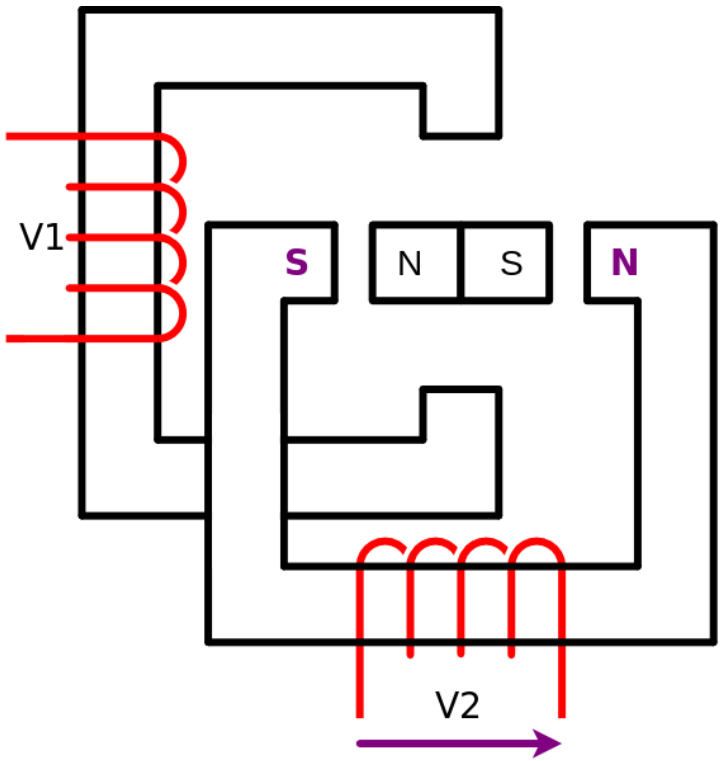


(présent dans la version courte)

Le moteur pas à pas 4 fils - la théorie

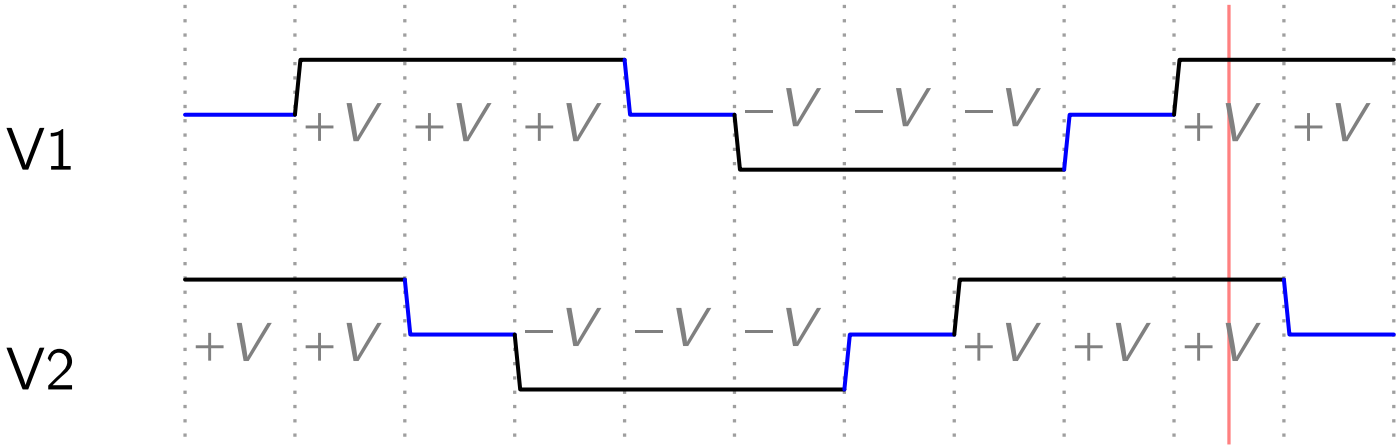


— : Haute impédance/ouvert — : $\pm V$

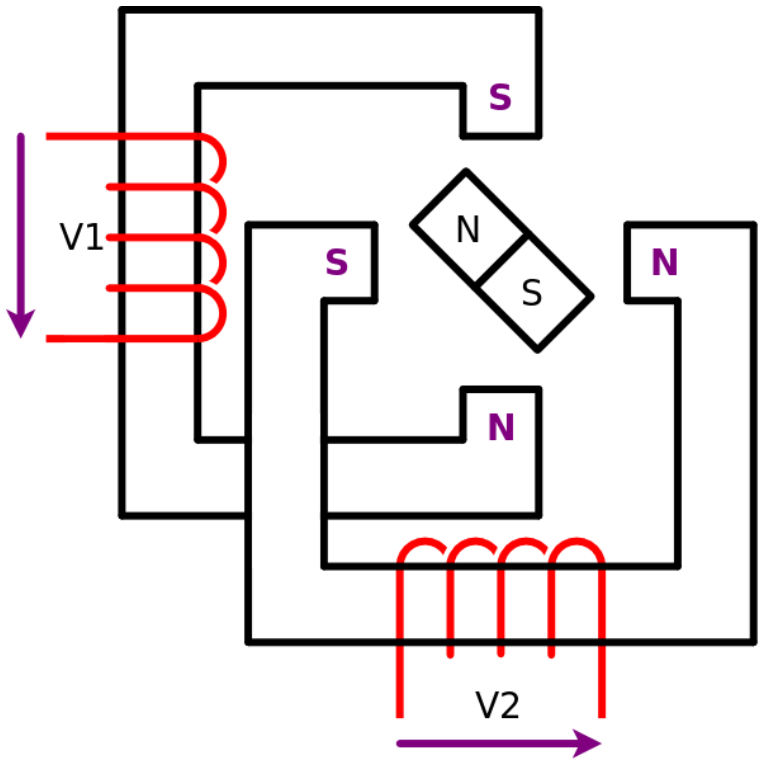


(présent dans la version courte)

Le moteur pas à pas 4 fils - la théorie

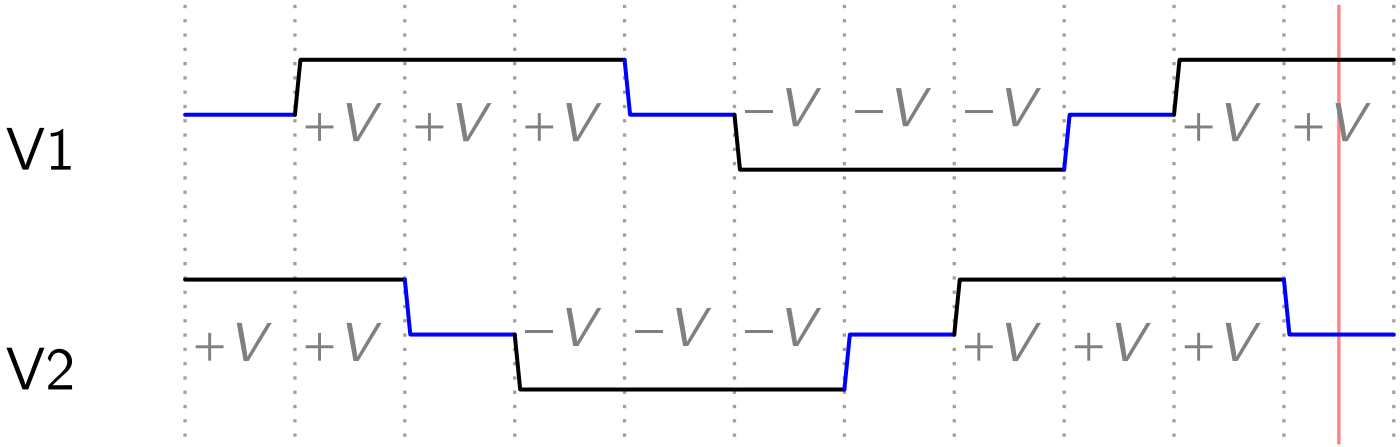


— : Haute impédance/ouvert — : $\pm V$

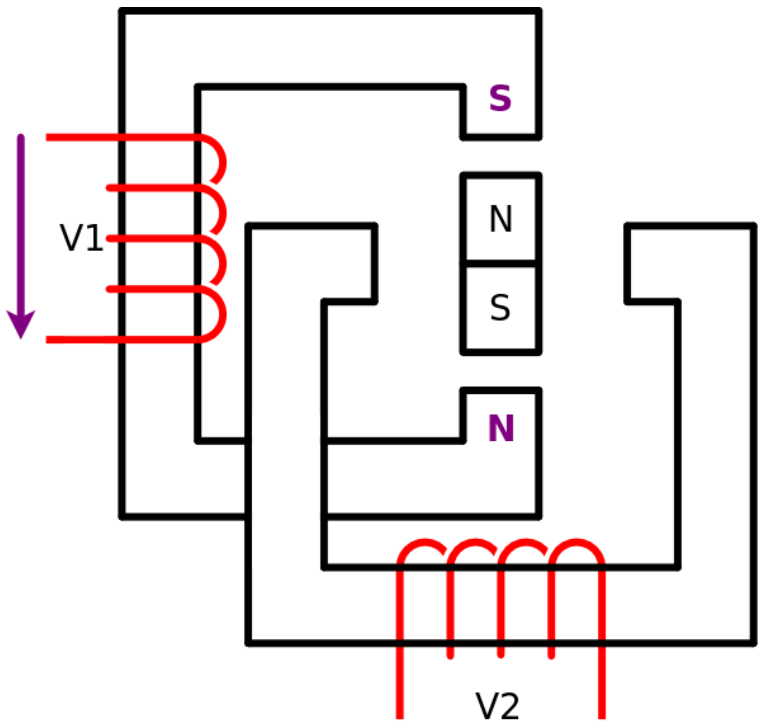


(présent dans la version courte)

Le moteur pas à pas 4 fils - la théorie



— : Haute impédance/ouvert — : $\pm V$



(présent dans la version courte)

Piloter des moteurs pas à pas 4 fils

Nous allons présenter 2 controlleurs pour commander les moteurs pas à pas à 4 fils :

Le Drivers Microstep à base de tb6600 est un controleur spécialisé pour moteur pas à pas 4 fils. Il se charge de générer pour vous le chronogramme des deux bobines du moteur et les PWM à envoyer à chacune d'elle pour que le courant reçu ne dépasse pas les spécifications du moteur.

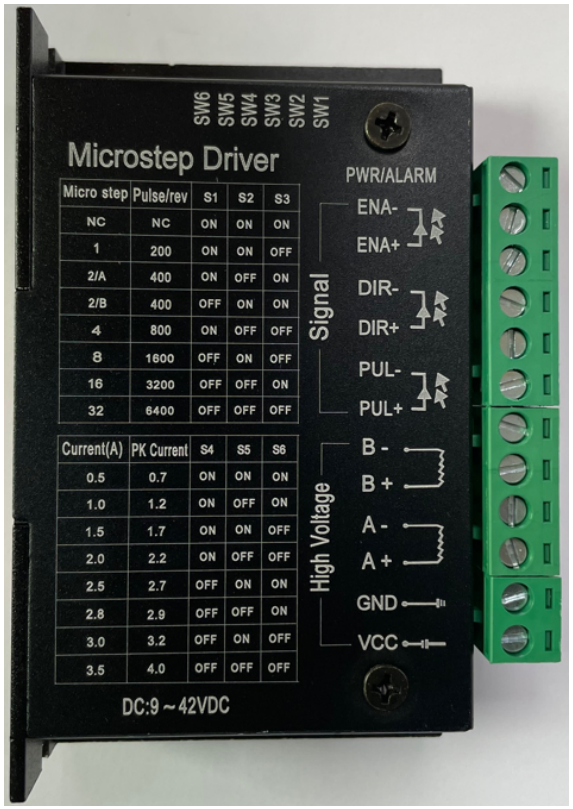
Le module adafruit L298N contient 2 ponts en H classiques. Son utilisation est plus ardue car la gestion du chronogramme et la programmation des PWM à envoyer à chacune des bobines est à votre charge !

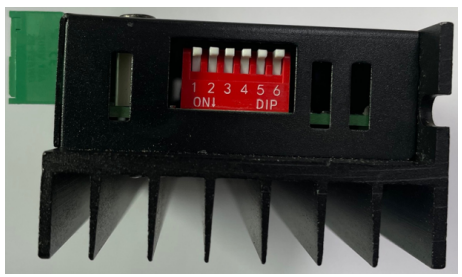
La première solution est bien plus facile à utiliser. La deuxième est moins chère et plus versatile car le module peut être utilisé avec tout type de moteur. C'est cependant bien plus difficile à programmer.

Une liste plus exhaustive de drivers et modules peut se trouver à la page 666.

Pas à pas 4 fils - Présentation du Microstep TB6600

Le driver génére pour vous le chronogramme et les PWM à envoyer aux moteurs.





Les broches d'alimentation sont GND et VCC.
Les entrées sont ENA±, DIR± et PUL±.
Les sorties pour les moteurs sont B± et A±.

Les modes du driver sont :

ENA+	DIR+	PUL+	(H=High, L=Low)
H	X	X	Roue libre
L	H	⌐↓	Tourne d'un micro-pas dans le sens direct
L	L	⌐↓	Tourne d'un micro-pas dans le sens indirect

ENA- = DIR- = PUL- = L

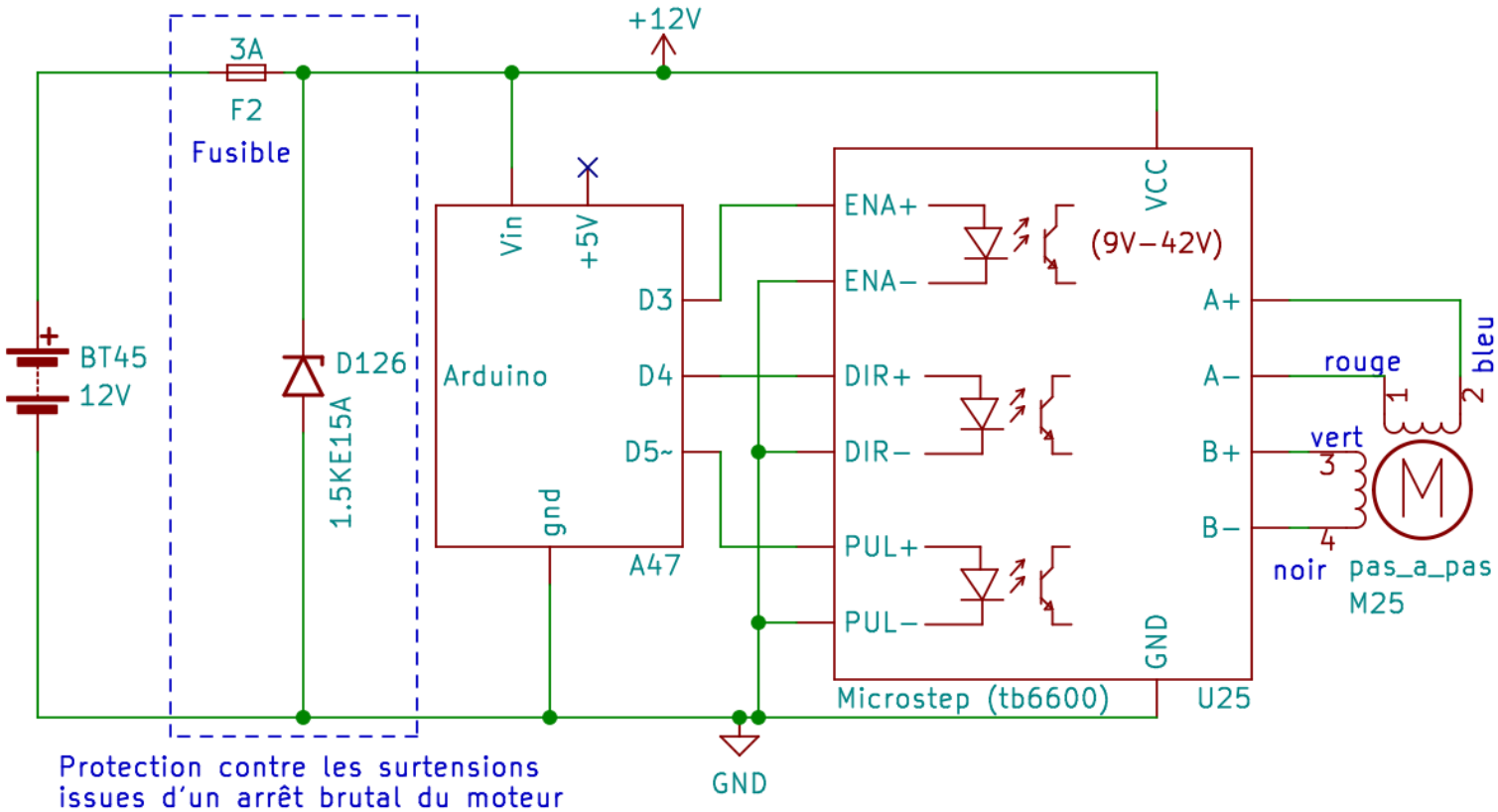
Les 6 cavaliers (de 0 à 6) servent à :

- configurer le courant max à appliquer au moteur (le rapport cyclique max des pwm).
- configurer le nb de micro-pas par tour (choisir le chronogramme et l'evolution des PWM). Le chronogramme prage 362 est celui des demi-pas ; "2/B" (cf datasheet du TB6600).

(version longue)

Pas à pas 4 fils - le Driver Microstep TB6600

Voici le schéma d'utilisation du drivers TB6600 :



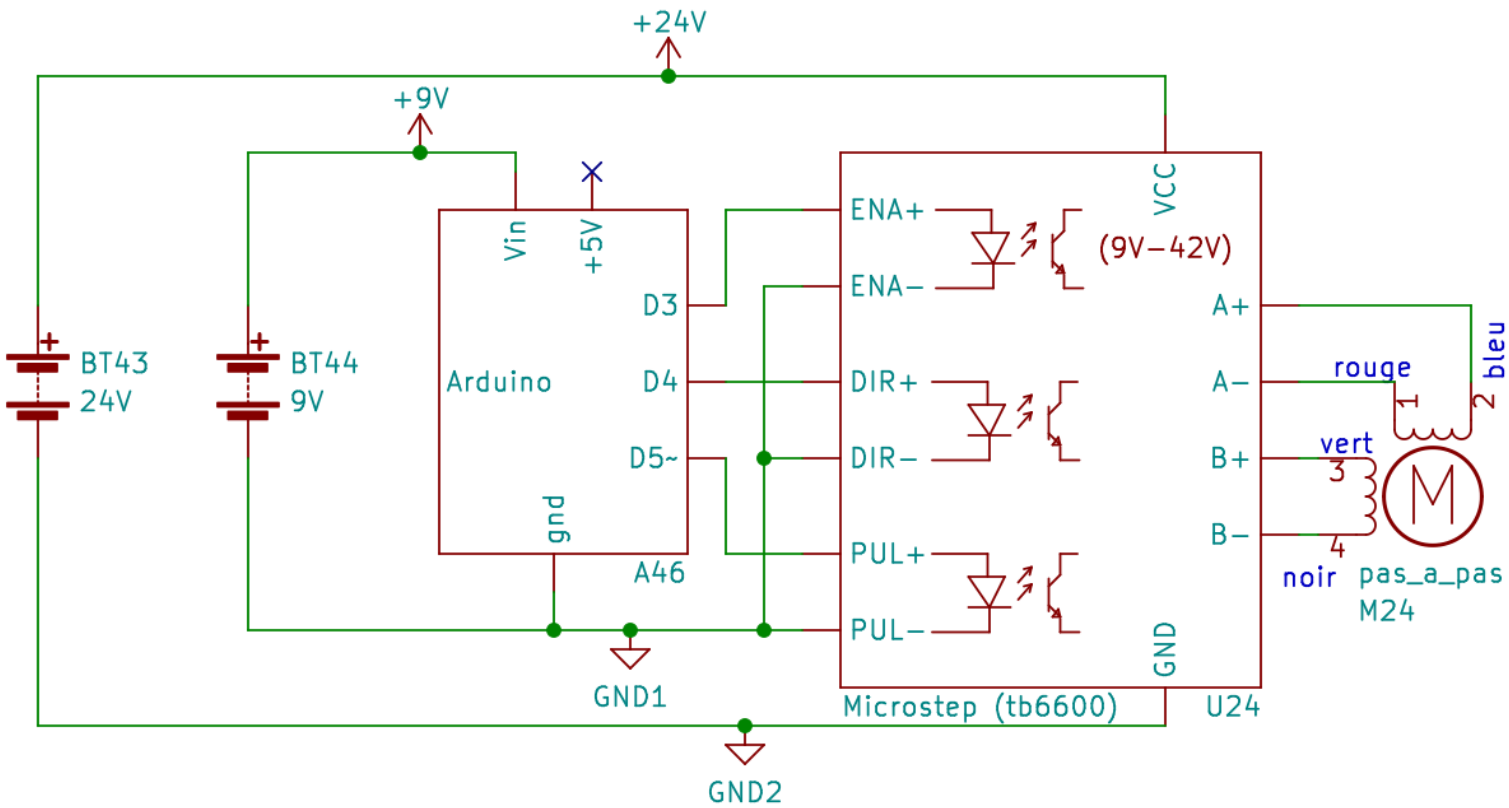
La tension de claquage de la diode TVS est de 15V, juste au dessus de 12V pour protéger le circuit des surtensions. Un moteur, dont chaque pôle consomme 1.7 A pour 1.6 Ω, consomme 9.25 $W = 2$ pôles $\times$ 1.6 $\Omega \times (1.7\ A)^2$. Le courant demandé à la batterie est de 1.3 $A = 12\ V / 9.25\ W$, d'où le fusible de 3A.

(version longue)

Pas à pas 4 fils - le Driver Microstep TB6600

Voici le schéma d'utilisation du drivers TB6600 :

Avec deux alimentations séparées



Gràce au 3 optocoupleurs internes et à la séparation des alimentations, le microcontrôleur est isolé de la partie puissance. Il est donc protégé des surtensions générés par le moteur.

(version longue)

Pas à pas 4 fils - Configurer le Driver Microstep TB6600

Il faut configurer le driver pour délivrer le courant adapté aux spécifications du moteur.

Par exemple, un moteur Nema 17 (17HS4401) fonctionne avec 1.7A. Il faut donc configurer le driver pour qu'il délivre 1.7A, à savoir, selectionner les cavaliers 4, 5 et 6 à respectivement ON, ON et OFF. .

Dans les slides de la page 362, le chronogramme présenté permet de faire tourner le moteur par demi-pas. On peut engendrer des chronomagrames qui font des pas entiers, mais aussi des 1/4, 1/8, 1/16, $\cdots$, 1/32 de pas.

Un moteur pas à pas Nema 17 (17HS4401) fait un tour tous les 200 pas. Ainsi, si le driver fait des micro-pas de 1/16 de pas, il faut faire 200×16 micro-pas pour que le moteur fasse 1 tour.

Par exemple, pour suivre le chronogramme donné à la page 362 et faire 1 tour tous les 400 micro-pas, selectionnez le mode "2/B" en configurant les cavaliers 1, 2 et 3 à respectivement OFF, ON et ON.

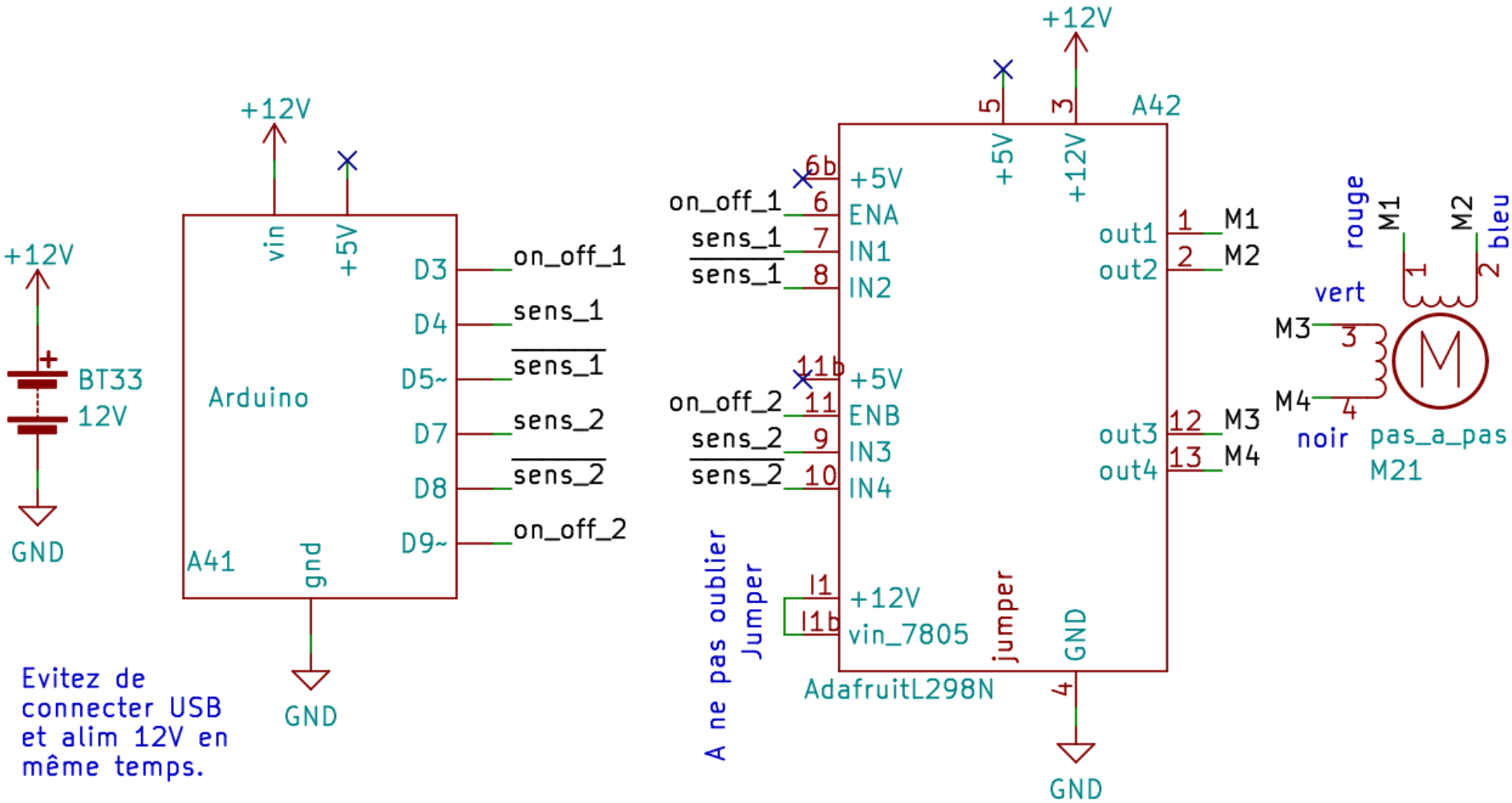
Tourner un pas à pas avec le driver Microstep TB6600

```
const int ena_pin = 3, dir_pin = 4, pul_pin = 5; int pul_value = 0;

void motor_on(bool on=true){ digitalWrite(ena_pin, on ? LOW: HIGH); }
void free_wheeling(){ motor_on(false); }
void move_clockwise(bool cw){ digitalWrite(dir_pin, cw ? LOW: HIGH); }

void setup() {
  pinMode(ena_pin, OUTPUT); pinMode(pul_pin, OUTPUT); pinMode(dir_pin, OUTPUT);
  free_wheeling(); move_clockwise(false); motor_on(true); pul_value=0;
}

unsigned long last_step_time = 0;
const uint32_t STEP_NB_BY_REV = 200, MSTEP_NB_BY_STEP = 2; // mode "2/B"
const uint32_t TIME_BY_REV = 1000000; // in micro seconds, tour d'une seconde
uint32_t micro_step_delay = TIME_BY_REV/(MSTEP_NB_BY_STEP*STEP_NB_BY_REV);

void loop() {
  uint32_t now = micros();
  if( now - last_step_time >= micro_step_delay/2 ){
    pul_value = 1-pul_value;
    digitalWrite(pul_pin, pul_value); // Fait un micro-pas a chaque front desc.
    last_step_time = now;
  }
}
```

code/microstepper.cpp

(version longue)

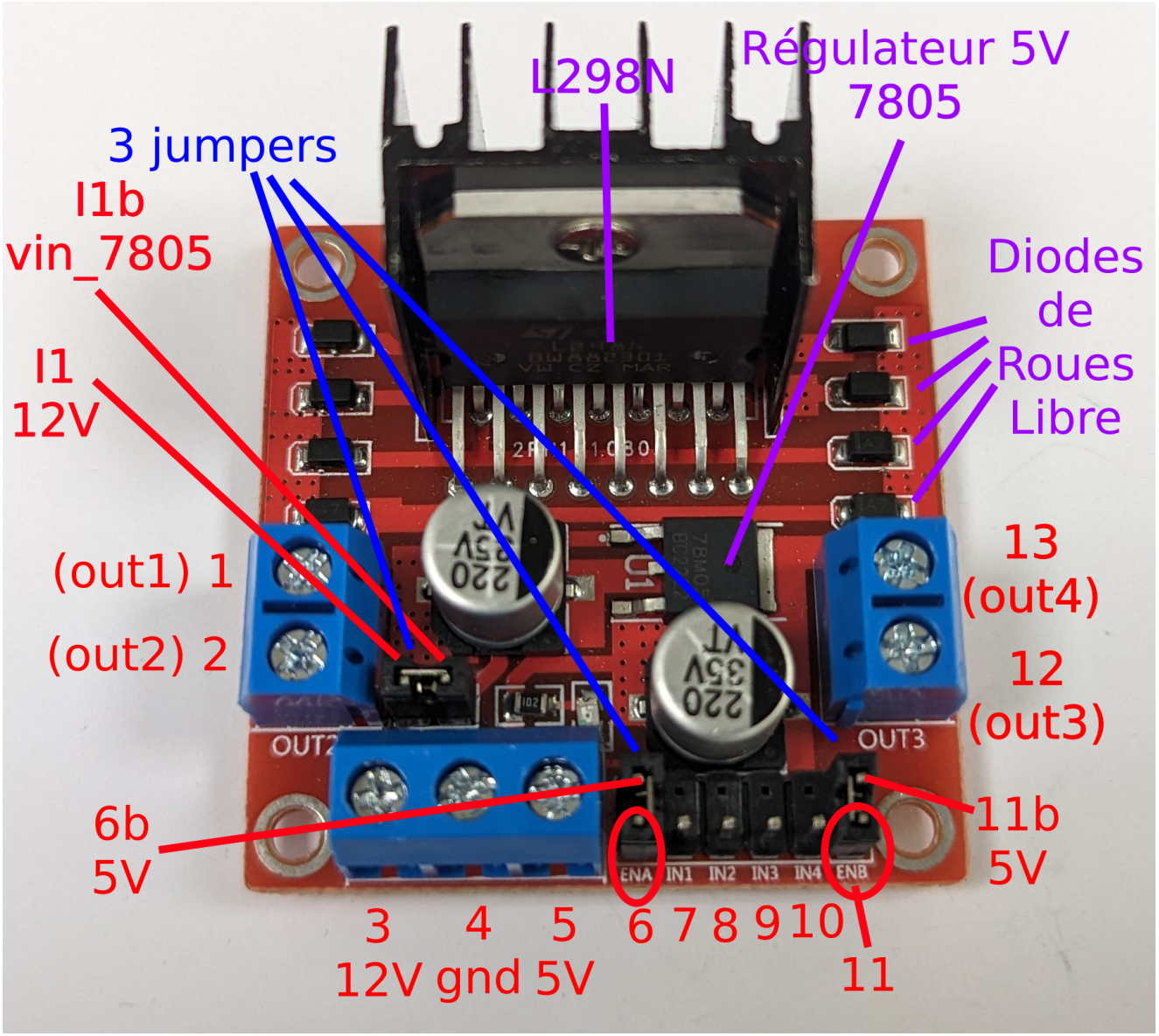
Le moteur pas à pas 4 fils - avec le module Adafruit L298N



Faites attention à bien retirer les 2 cavaliers 6-6b et 11-11b et à laisser connecter le cavalier I1-I1b. À cause de la présence de quatre alimentations (l'USB, la batterie et les deux régulateurs internes de l'arduino ET du L298N), certains montages peuvent détruire le port USB.

(présent dans la version courte)

Pas à pas 4 fils - Rappel sur le module Adafruit L298N



(présent dans la version courte)

Pas à pas 4 fils - Rappel sur le module Adafruit L298N

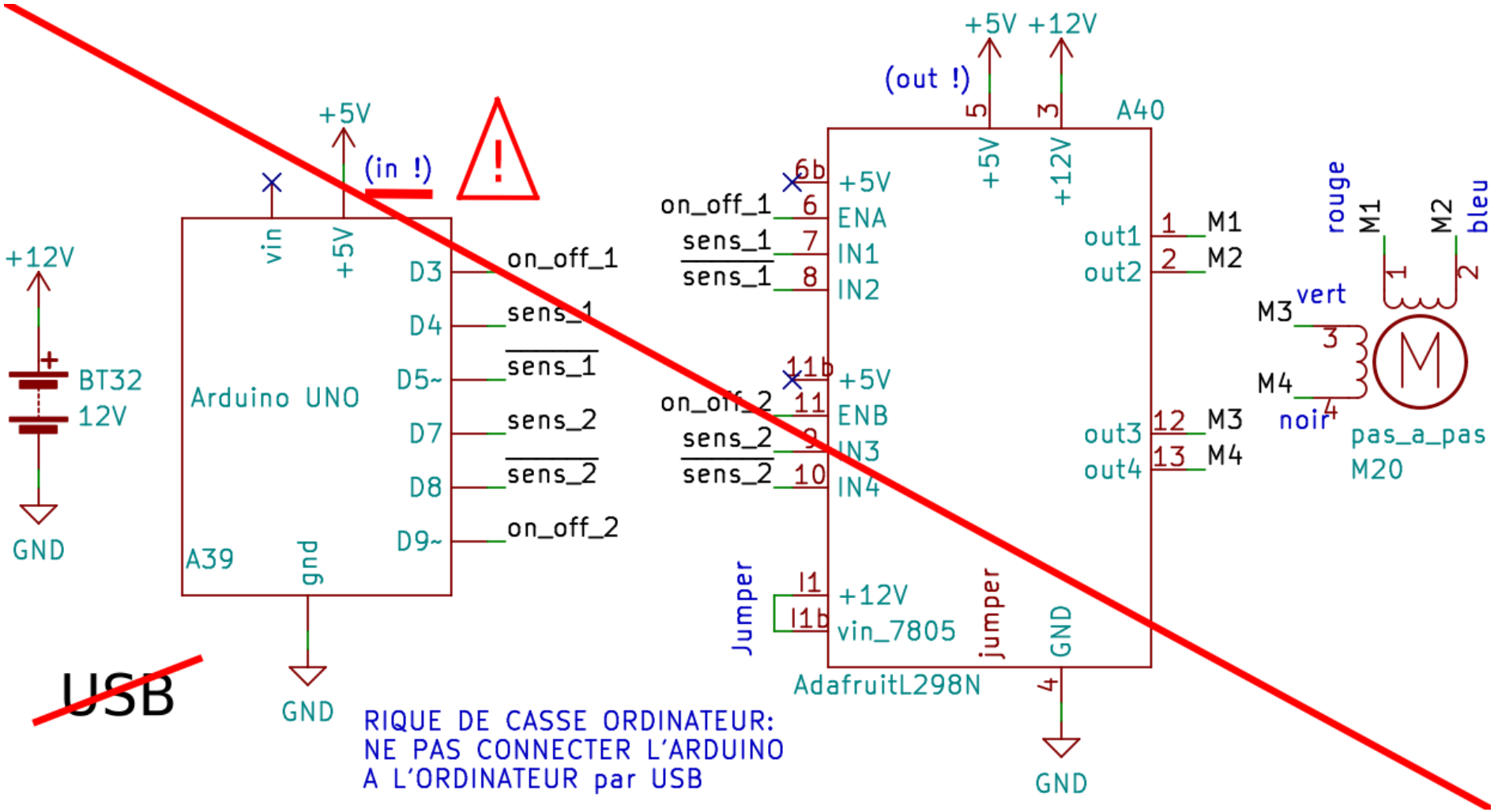
Chaque pôle se contrôle avec 3 entrées EN, IN1 et IN2 de la manière suivante :

EN	IN1	IN2	mode
0	0/1	0/1	roue libre
1	a	a	frein moteur ($a \in \{0,1\}$)
1	0	1	tourne en sens direct
1	1	0	tourne en sens indirect

Roue libre = pôles en hautes impédances.

Frein moteur = les bornes des pôles sont mis à 0V (court-circuités).

À ne pas faire : Alimenter l'Arduino UNO par le 5V



Consulez la page 628 pour comprendre pourquoi il ne faut pas connecter l'USB dans ce cas.

(version longue)

Determiner la PWM max que l'on peut envoyer à chaque bobine

A FAIRE :

Programmer un moteur pas à pas 4 fils : le chronogramme.

```
enum State { Z=0, P=1, N=2, B=3, STATE_NB=4 };
// Z : high impedance, P : Positive, N : Negative, B : motor Brake

const int POLES_NB = 2, CYCLE_SIZE = 8;
State timing_table[POLES_NB][CYCLE_SIZE] = {
  {P, P, Z, N, N, N, Z, P},
  {Z, N, N, N, Z, P, P, P}
};

enum { ENABLE=0, SENS_1=1, SENS_2=2, PINS_BY_POLES_NB=3 };
int control_pins[POLES_NB][PINS_BY_POLES_NB] = {
  {3,  4, 5}, // Pole 1 : ENA, IN1, IN2 (driver L259N)
  {9, 7, 8} // Pole 2 : ENB, IN3, IN4 (driver L259N)
};

const long int PWM_MAX = 255, V_ALIM = 120; // In DeciVolt
const long int H_DROP = 32, V_MOTOR = 28;// In Decivolt
const long int POLE_PWM_MAX = (V_MOTOR*PWM_MAX)/(V_ALIM-H_DROP);
int state_to_output[STATE_NB][PINS_BY_POLES_NB] = {
  //    PWM (EN)    SENS1   SENS2
  {           0,      LOW,    LOW }, // Z, roue libre (driver L259N)
  { POLE_PWM_MAX,    HIGH,    LOW }, // P, sens direct (driver L259N)
  { POLE_PWM_MAX,     LOW,   HIGH }, // N, sens indirec (driver L259N)
  {     PWM_MAX,     LOW,    LOW } // B, frein moteur (driver L259N)
};
```

ode/stepper_4_wires.cpp

(version longue)

Moteur pas à pas 4 fils : le mode roue libre et frein moteur

```
void apply_state_to_pole(unsigned int i, State state){
  analogWrite(
    control_pins[i][ ENABLE ], state_to_output[ state ][ ENABLE ]
  );
  digitalWrite(
    control_pins[i][ SENS_1 ], state_to_output[ state ][ SENS_1 ]
  );
  digitalWrite(
    control_pins[i][ SENS_2 ], state_to_output[ state ][ SENS_2 ]
  );
}

// Don't forget to decelerate before making a motor_brake.
void motor_brake(){
  for(int i=0; i<POLES_NB; i++){ apply_state_to_pole(i, B); }
}

// Be careful with the free-wheeling mode, as the motor becomes a
// generator, causing current to return to the power supply rail
// and potentially causing damage. It is advisable to avoid using
// this mode. Prefer to decelerate and use the motor_brake() mode.
void free_wheeling(){
  for(int i=0; i<POLES_NB; i++){ apply_state_to_pole(i, Z); }
}
```

code/stepper_4_wires.cpp

(version longue)

Moteur pas à pas 4 fils : le mode demi-pas à demi-pas

```
static volatile int step = 0;
void make_one_half_step(bool forward){

  if(forward){
    step = step + 1;
    if(step >= CYCLE_SIZE){ step = 0; }
  }else{
    step = step - 1;
    if(step < 0){ step = CYCLE_SIZE-1; }
  }

  for(int i=0; i<POLES_NB; i++){
    State state = timing_table[i][step];
    apply_state_to_pole(i, state);
  }

}
```

code/stepper_4_wires.cpp

(version longue)

Moteur pas à pas 4 fils : setup et loop

```
void setup(){
  for(int i=0; i<POLES_NB; i++){
    pinMode(control_pins[i][ SENS_1 ], OUTPUT);
    pinMode(control_pins[i][ SENS_2 ], OUTPUT);
  }
  free_wheeling();
}

const long int STEP_NB_BY_REVOLUTION = 200;
const long int TIME_BY_REVOLUTION = 1000000; // in us, (1 s)
unsigned int half_step_delay = TIME_BY_REVOLUTION/(2*STEP_NB_BY_REVOLUTION);
                    // in micro seconds
unsigned long last_step_time = 0;

void loop(){
  // In this example, the motor does not need to operate at high
  // speed, so there is no requirement for a ramp to facilitate
  // acceleration or deceleration.
  unsigned long now = micros();
  if( now - last_step_time >= half_step_delay ){
    last_step_time = now;
    make_one_half_step(true);
  }
}
```

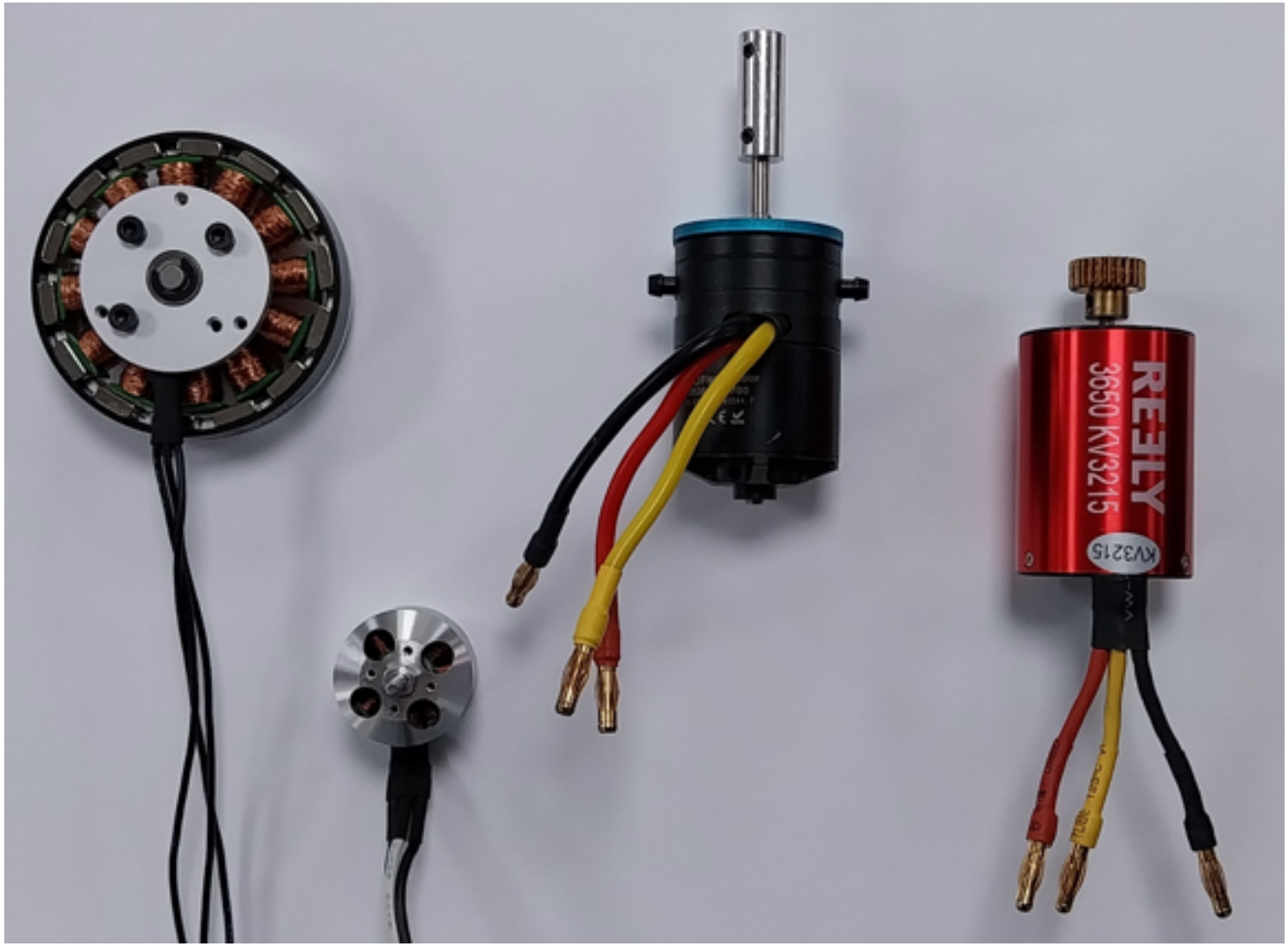
code/stepper_4_wires.cpp

(version longue)

Plan

(présent dans la version courte)

Présentation des moteurs sans balais (bushless motor)



(présent dans la version courte)

Les moteurs sans balais (bushless motor)

Le contrôle en vitesse et en position des moteurs sans balais (brushless) est difficile car il faut contrôler précisément l'orientation du champs électromagnétique dans l'entrefer du moteur (FOC: Field Oriented Control). Cela nécessite des connaissances en mécanique, électromagnétique, électronique et automatique assez avancées qui sorte du champs de cette introduction.

Utilisez donc des modules prêt à l'emploi, comme par exemple, les ESC vu à la page 304.

Si vous souhaitez tout de même mettre en oeuvre, avec des ponts en H, un régulateur FOC de votre moteur sans balais, consultez le site : SimpleFOCDocs . Ce site propose des cartes et une bilbiothèque pour Arduino implémentant un régulateur FOC en position et en vitesse.

Une liste de drivers et modules peut se trouver à la page 666.

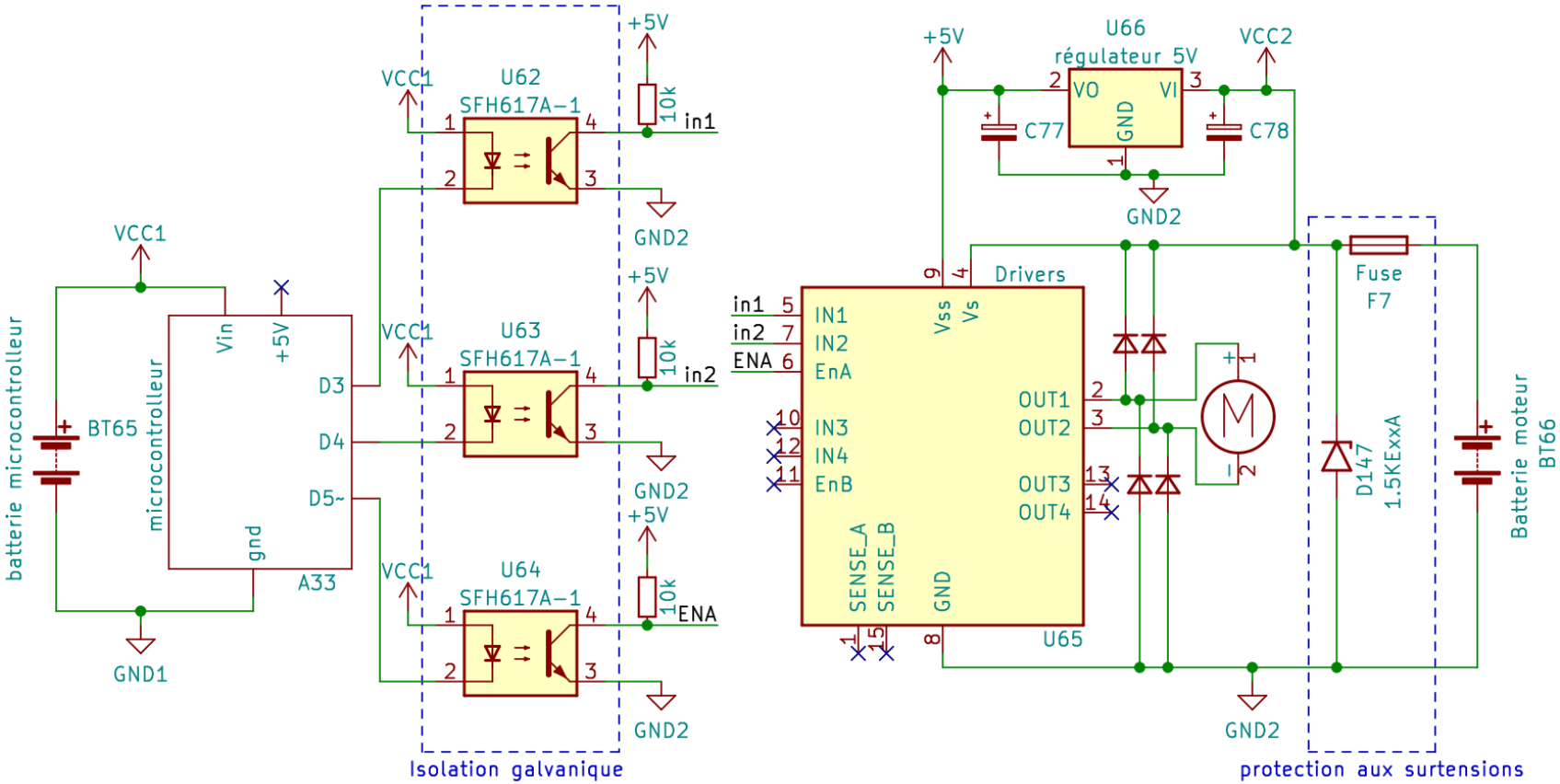
(présent dans la version courte)

Plan

Les mesures de protections contre les surtensions et le bruit

Trois mesures peuvent être appliquées :

- absorber les surtensions avec une diode TVS et son fusible (D147 + F7);
- séaprer les alimentations (BT68/VCC1 et BT70/VCC2);
- réaliser une isolation galvanique. (U62, U63 et U64)



(version longue)

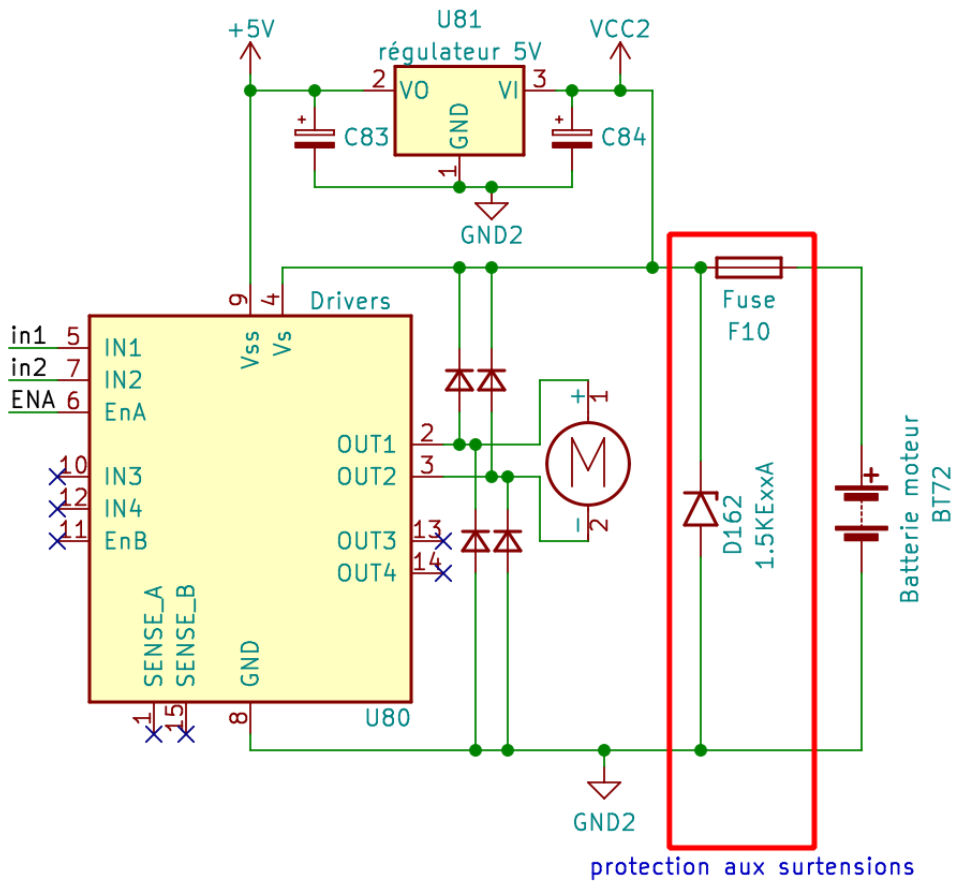
Les mesures de protections contre les surtensions - la diode TVS

La moins onéreuse, c'est la première protection à mettre en place ! La diode TVS absorble toutes les tensions au dessus d'une tension seuille qu'il faut choisir la plus proche au dessus de la tension maximale de la batterie.

Le fusible est indispensable : si la diode brûle, elle se transforme en fil (contrairement aux diodes zener), le courant augmente alors et brûle le fusible qui ouvre et protège le circuit.

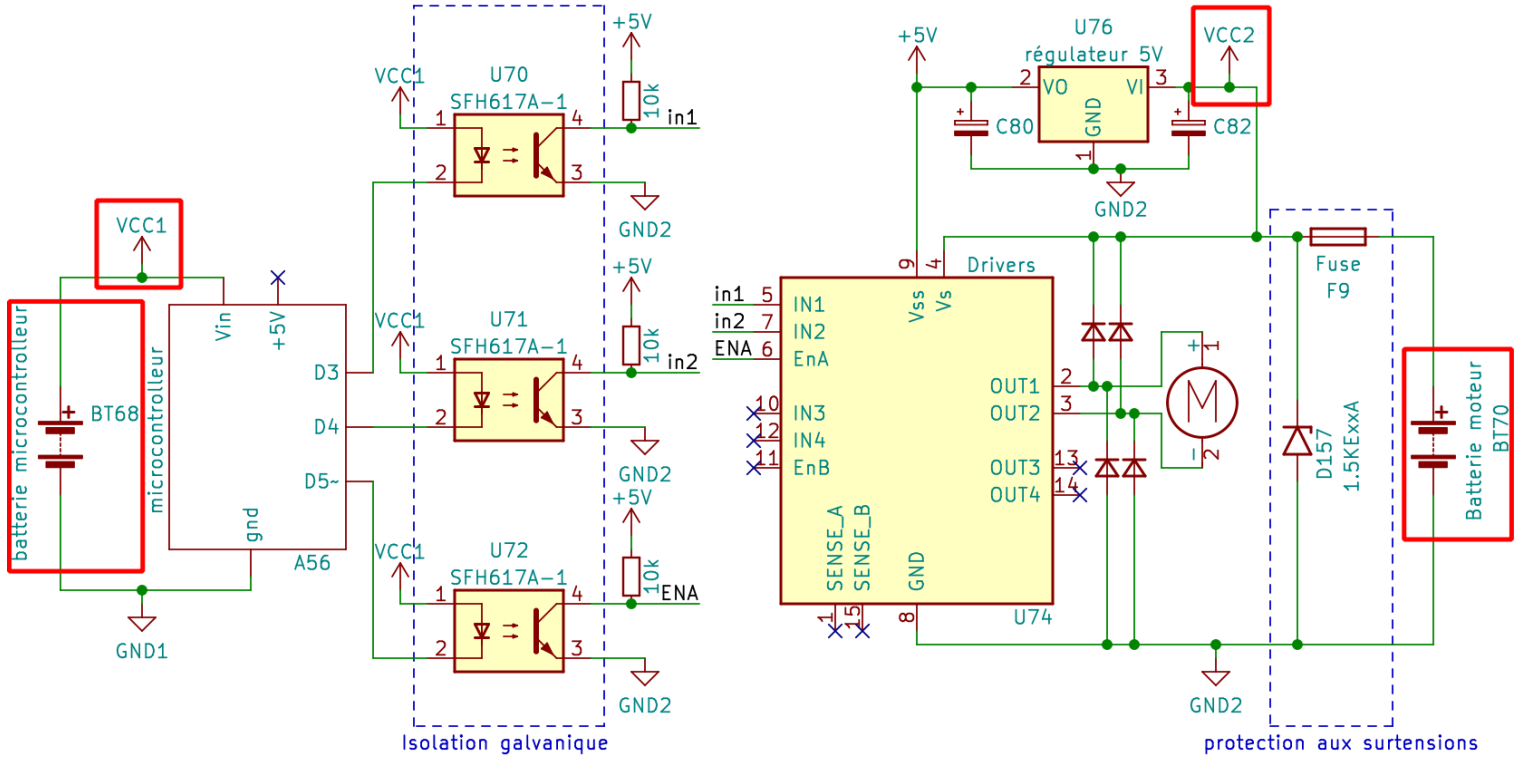
La valeur du fusible doit être choisie au dessus du courant maximal consomé par votre circuit.

Déterminer expérimentatlement ce courant en alimentant votre circuit avec une alimentation de Laboratoire qui mesure le courant consomé.



Les mesures de protections contre les surtensions - Séparer les batteries

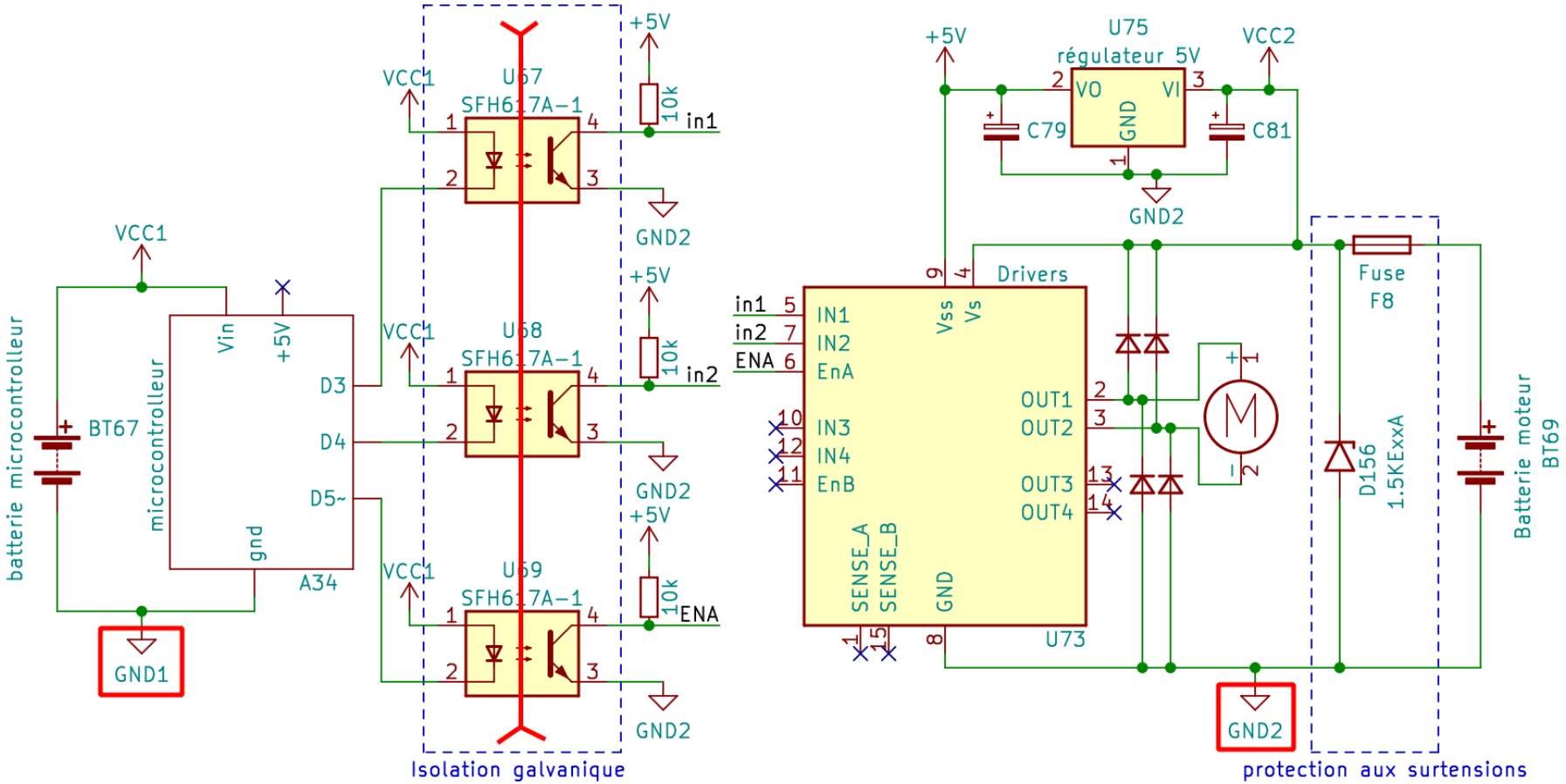
Séparer les batteries permet de réduire la propagation du bruit et des surtensions provoquées par respectivement le driver et le moteur.



(version longue)

Les mesures de protections contre les surtensions - l'isolation galvanique

L'isolation galvanique permet d'isoler complètement la partie puissance de la partie contrôle. Il permet de ne pas propager à la fois le bruit et les surtensions. Le circuit possède donc deux masse distincts : GND1 et GND2.



Les mesures de protections contre les surtensions - protéger le port USB

Une mesure permettant d'empêcher une surtension sur le port USB est l'isolation galvanique.

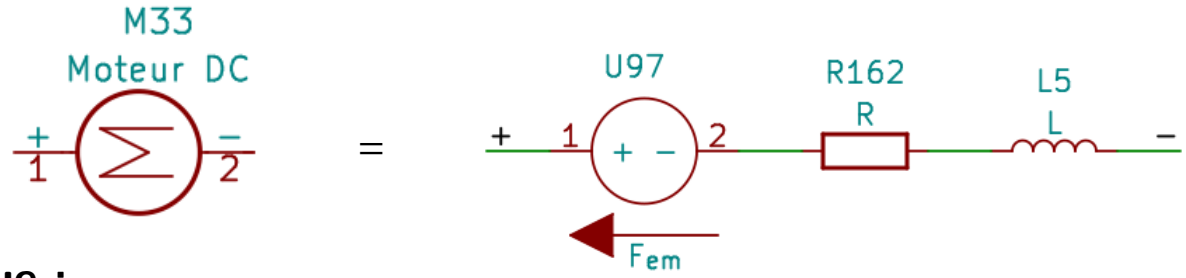
Par exemple, si le driver d'un moteur n'est pas isolé galvaniquement, une surtention peut endommager le port USB. Pour éviter ce problème, à defaut d'isoler galvaniquement le driver, vous pouvez isoler le port USB du reste du circuit avec une clef USB utilisant le circuit ADMU3160. On trouve ces clefs sous le nom de "USB Digital Isolator".

(version longue)

Plan

- Présentation des moteurs
- Les moteurs DC – commande manuelle
- Les servomoteurs et les ESC
- Les moteurs DC – tourner dans un seul sens
- Les moteurs pas à pas 5 fils
- Les moteurs DC – tourner dans les deux sens – Le pont en H
- Les moteurs pas à pas 4 fils
- Les moteurs sans balais (brushless)
- Les mesures de protections contre les surtensions et le bruit
- Modèle d'un moteur à courant continu et identification de ses paramètres
- Limiter la commande d'entrée d'un moteur

Modèle d'un moteur à courant continu



Modèle électrique :

$$\begin{cases} U & = & f_{em} + R \cdot I + L \partial_t I \\ f_{em} & = & \lambda \cdot \omega \end{cases}$$

ω : vitesse angulaire du moteur (rad/s)
U : tension au borne du moteur (V)
I : courant traversant le moteur (A)
F_{em} : force électromotrice (V)

L: inductance du moteur (H)
R : résistance électrique du moteur (Ω)
λ : constante de couple $(N.M/A)$

Modèle mécanique :

$$\begin{cases} J \cdot \partial_t \omega & = & C_{em} + C_{visqueux} + C_{sec} + C_{charge} \\ C_{em} & = & \lambda \cdot I \\ C_{visqueux} & = & -\alpha_f \cdot \omega \\ C_{sec} & = & \begin{cases} -C_{em} & \text{si } \omega = 0 \quad (\text{alors } c_{em} \leq c_0); \\ -\alpha_s \cdot \text{signe}(\omega) & \text{si } \omega \neq 0 \quad (\text{alors } c_{em} \geq c_0). \end{cases} \end{cases}$$

J : inertie du moteur $(kg.m^2)$
C_{em} : couple électromagnétique $(N.m)$
$C_{visqueux}$: couple frottement visqueux $(N.m)$
C_{sec} : couple frottement sec $(N.m)$
C_{charge} : charge appliquée au moteur $(N.m)$

λ : constante de couple $(N.m/A)$
α_f : constante frottement visqueux $(N.m.s)$
α_s : constante frottement sec $(N.m)$
c_0 : couple minimal pour lutter contre les frottements $(N.m)$

(version longue)

Formulaire pour le régime transitoire des moteurs

Équation différentielle temporelle si $\omega \neq 0$:

$$H_0 \cdot U = \begin{array}{l} \tau_e \cdot \tau_m \cdot \partial_t^2 \omega + (\tau_m + \frac{L \cdot \alpha_f}{\lambda^2}) \cdot \partial_t \omega + (1 + \frac{R \cdot \alpha_f}{\lambda^2}) \cdot \omega \\ + \ \frac{R \cdot \alpha_s}{\lambda^2} . \text{signe}(\omega) - \frac{R}{\lambda^2} C_{charge} - \frac{L}{\lambda^2} \partial_t C_{charge} \end{array}$$

Équation différentielle simplifiée (on néglige l'effet inductif du moteur) si $\omega \neq 0$:

$$H_0 \cdot U \underset{\tau_m >> \tau_e}{\approx} \tau_m \cdot \partial_t \omega + (1 + \frac{R \cdot \alpha_f}{\lambda^2}) \cdot \omega + \ \frac{R \cdot \alpha_s}{\lambda^2} . \text{signe}(\omega) - \frac{R}{\lambda^2} C_{charge}$$

Constante de couple ($N.m/A$) : λ
Constante de vitesse H_0 ($rad.V^{-1}.s^{-1}$) ou K_n ($kv = rpm/V$) :

$$H_0 = \frac{1}{\lambda} = K_n \times \frac{2 \cdot \pi}{60}$$

Constante de temps mécanique τ_m (s) et électrique τ_e (s) :

$$\tau_m = \frac{R \cdot J}{\lambda^2} \qquad\qquad \tau_e = \frac{L}{R}$$

Fonction de transfert sans charge et sans frottement :

$$H(s) \ = \ \frac{\text{Laplace}(\omega)}{\text{Laplace}(U)} \ = \ \frac{H_0}{\tau_m \cdot \tau_e \cdot s^2 + \tau_m \cdot s + 1} \ \underset{\tau_m >> \tau_e}{\approx} \ \frac{H_0}{1 + \tau_m \cdot s}$$

(version longue)

Formulaire pour le régime permanent des moteurs

Équations en régime permanent ($\partial_t \omega = 0$ et $\partial_t C_{charge} = 0$) :

$$U = \lambda \cdot \omega + R \cdot I \qquad \text{et} \qquad C_{em} = \lambda \cdot I \qquad \text{et}$$

$$\text{si } \omega \neq 0, \qquad H_0 \cdot U = (1 + \frac{R \cdot \alpha_f}{\lambda^2}) \cdot \omega + \frac{R \cdot \alpha_s}{\lambda^2} . \text{signe}(\omega) - \frac{R}{\lambda^2} C_{charge}$$

Couple de décrochage (stall torque) (N.m) : C_{stall}
Courant de démarrage (starting current) (A) : I_{start}

$$C_{stall} = \lim_{\substack{\omega \to 0 \\ \omega \neq 0 \\ \text{sans charge}}} C_{em} = \lambda \cdot I_{start} = \alpha_s$$

$$\text{sans charge} \quad \Leftrightarrow \quad C_{charge} = 0$$

Couple de friction (friction torque) (N.M) : $C_{friction}$
Courant sans charge (No load current) (A) : $I_0 = \lim_{\substack{U \to U_{max} \\ \textbf{sans charge}}} I$

$$C_{friction} = \lim_{\substack{U \to U_{max} \\ \text{sans charge}}} C_{em} = \lambda \cdot I_0 = \alpha_f \cdot \omega_{max} + \alpha_s$$

avec

$$\omega_{max} = \lim_{\substack{U \to U_{max} \\ \text{sans charge}}} \omega = \frac{U_{max} - R \cdot I_0}{\lambda}$$

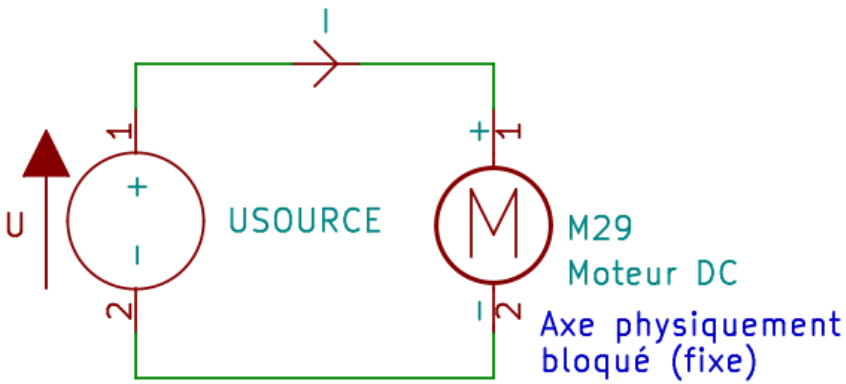
Identifier la résistance R d'un moteur DC

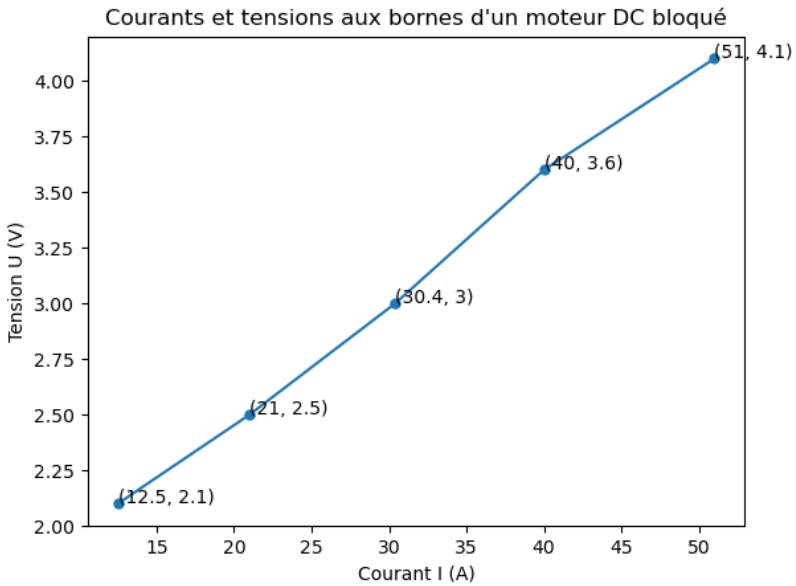
Bloquez physiquement l'axe du moteur ($\omega = 0$). Le modèle électrique du moteur en régime permanent devient :

$$U = R \cdot I.$$

Appliquez différentes tensions au borne du moteur en partant de faibles tensions puis mesurez les courants associés. La pente de la droite obtenue est la résistance R de votre moteur. Attention, selon les moteurs, les courants obtenus peuvent être très grand.

Exemple :





$$R = \frac{4.1V - 2.1V}{51A - 12.5A} \approx 50m\Omega$$

Pour ce moteur, le courant que l'on obtiendrai à 24V serait de 434 A ! Il faut donc impérativement utiliser des petites tensions sur de petites durées (le temps de faire les mesures) pour éviter que le moteur surchauffe.

(version longue)

Identifier la constante de couple λ d'un moteur DC

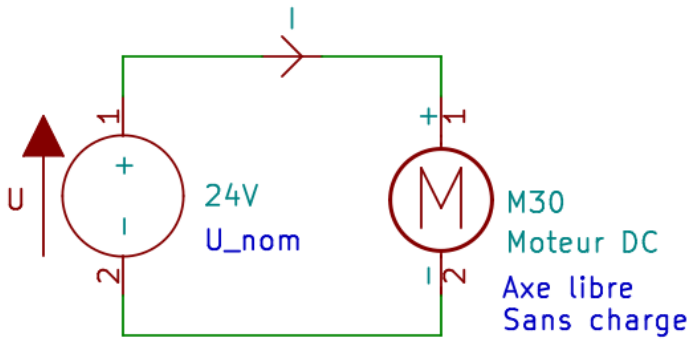
Nous allons utiliser le modèle éléctrique en régime permanent du moteur :

$$U = \lambda \cdot \omega + R \cdot I.$$

Faites tourner le moteur à vide (sans charge) à sa tension nominale U_{nom} (V) (en anglais : rated voltage, nominal voltage). Il s'agit de la tension d'utilisation optimale préconisée par le constructeur. Mesurez ensuite son courant I_{nom} (A) et aussi sa vitesse ω_{nom} (rad/s).
La constante de couple est donné par :

$$\lambda = \frac{U_{\text{nom}} - R \cdot I_{\text{nom}}}{\omega_{\text{nom}}}.$$

Exemple :



Le constructeur nous donne les valeurs nominales que l'on obtient aussi expérimentalement avec un moteur sans charge que l'on à sa tension nominale :

$U_{\text{nom}} = 24V$
$I_{\text{nom}} = 20A$
$\Omega_{\text{nom}} = 8000\text{rpm} = \frac{8000 \cdot 2\pi}{60} rad/s \approx 838\ rad/s$

$$\lambda = \frac{24V - 20A \cdot 0.05\Omega}{838 rad/s} \approx 0.0275 N \cdot m/A.$$

La résistance R a été calculée précédement : $R = 50m\Omega$.

Remarque : Il est maintenant facile de calculer le couple nominal à partir de ces données :
$C_{\text{nom}} = \lambda \cdot I_{\text{nom}} \approx 0.0275 N.m/A \times 20A \approx 0.55 N.m.$

(version longue)

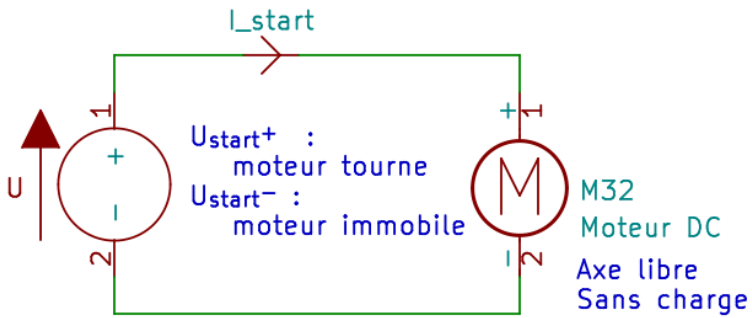
Identifier la constante de frottement sec α_s

On a vu qu'en Régime permanant, la constante de frottement sec ou couple de décorchage s'obtient ainsi :

$$C_{stall} = \lim_{\substack{\omega \to 0 \\ \omega \neq 0 \\ \text{sans charge}}} C_{em} = \lambda \cdot I_{start} = \alpha_s$$

Avec une alimentation de laboratoire, alimentez le moteur à vide (sans charge) et libre de mouvement à une tension de $0V$. Augmentez ensuite la tension et arrêtez dès que l'axe se met à tourner. Le courant de démarrage I_{start} est alors donné par le courant du bloc d'alimentation. Il ne reste plus qu'à calculer la constante de frottement sec à l'aide de λ.

Exemple :



$$\begin{aligned} C_{stall} = \alpha_s &= \lambda \cdot I_{start} \\ &= 0.0275 N.m/A \times 3A \\ &\approx 0.0825 N.m. \end{aligned}$$

Expérimentalement, notre moteur à commencé à tourner pour une tension de $V_{start} = 0.8V$.

Le courant mesuré était alors de $I_{start} = 3A$.

On a aussi déterminer précédement que la constante de couple était égale à $\lambda = 0.0275 N.m/A$.

Identifier la constante de frottement visqueux α_f

Nous allons utiliser le modèle mécanique du moteur :

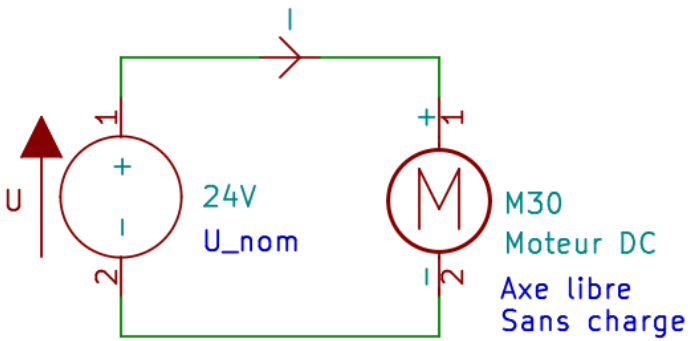
$$J \cdot \partial_t \omega = \lambda . I - \alpha_f \cdot \omega - \alpha_s \cdot \text{signe}(\omega) + C_{charge}$$

sans charge ($C_{charge} = 0$) et en régime permanent ($\partial_t \omega = 0$). En faisant tourner le moteur avec sa tension nominale, sans charge, libre de tout mouvement. On obtient alors:

$$\alpha_f = \frac{\lambda . I_{\text{nom}} - \alpha_s \cdot \text{signe}(\omega_{\text{nom}})}{\omega_{\text{nom}}}$$

où I_{nom} et ω_{nom} sont mesurés.

Exemple :



Le constructeur nous donne les valeurs nominales que l'on obtient aussi expérimentalement avec un moteur sans charge que l'on alimente à sa tension nominale :

$U_{\text{nom}} = 24V$
$I_{\text{nom}} = 20A$
$\Omega_{\text{nom}} = 8000\text{rpm} = \frac{8000 \cdot 2\pi}{60} rad/s \approx 838\ rad/s$

$$\begin{aligned} \alpha_f &= \frac{0.0275N.m/A \times 20A - 0.0825N.m \times 1}{838\ rad/s} \\ &\approx 5.579 \times 10^{-4} N.m.s \end{aligned}$$

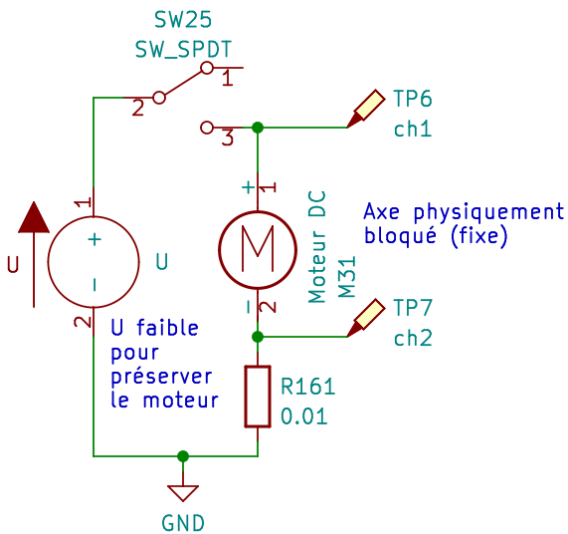
La constante de frottements a été déterminée précédement et vaut $\alpha_s = 0.0825N.m$.
De même pour la constante de couple : $\lambda = 0.0275N.m/A$.

(version longue)

Identifier l'inductance L d'un moteur DC

Pour identifier L, bloquez le moteur. Le modèle électrique devient $U = 0 + R \cdot I + L\partial_t I$.
Ensuite faites le montage ci-dessous. Quand, au temps t_0, vous fermez l'interrupteur SW25, le courant passe de 0 A à $I_{max} = \frac{U}{R}$ ampère suivant l'équation $I = \frac{U}{R} \cdot \left(1 - e^{-\frac{L}{R+R161}(t-t_0)}\right)$. Quand le temps écoulé $t - t_0$ est égal à $\frac{R+R161}{L}$, le courant vaut : $\frac{U}{R} \cdot (1 - \frac{1}{e}) = I_{max} \times 0.632$. Il ne reste donc plus qu'à mesurer le temps que met le courant pour atteindre 63 % de I_{max}.

Exemple :



Pour le moteur précédent et le montage de gauche, on a mesuré les tensions V_{ch1} et V_{ch2} et obtenu le résultat suivant :

A FAIRE :

Le temps de monté à 63% est égal à : $t_{63\%} - t_0 = ...$

$$L = \frac{R+R_{161}}{t_{63\%} - t_0} = \frac{...}{...} \approx ...$$

Attention, le moteur est bloqué, il faut donc utiliser une tension U faible pour ne pas endomager le moteur (cf. page 401).

Remarque : L'inductance des moteurs est souvant ignorée car son impact sur la dynamique mécanique est négligeable par rapport à celui du couple electromagnétique et celui de l'inertie de la partie mobile du moteur.

Identifier l'inertie J de la partie mobile du moteur

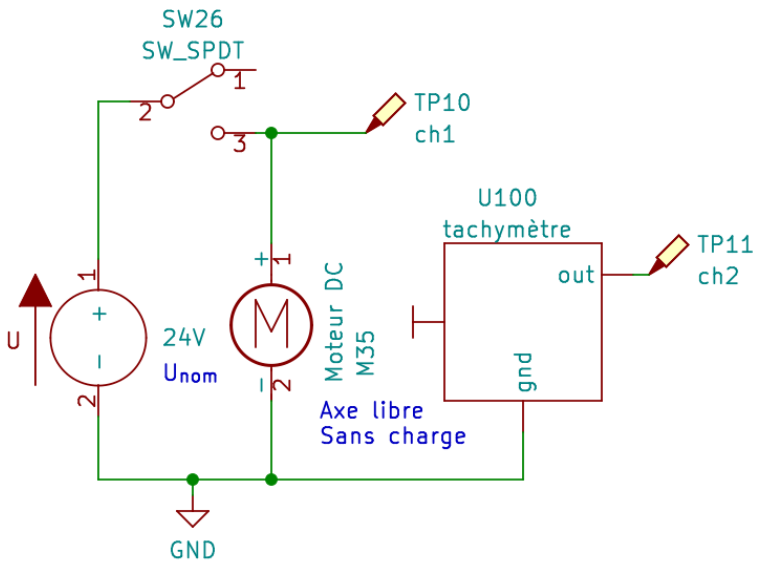
Utilisons l'équation différentiells simplifiée d'un moteur sans charge (on néglige L et $C_{charge}=0$):

$$\frac{1}{\lambda}\cdot U \underset{\tau_m >> \tau_e}{\approx} \tau_m \cdot \partial_t\omega + (1+\frac{R\cdot\alpha_f}{\lambda^2})\cdot\omega + \frac{R\cdot\alpha_s}{\lambda^2}.\text{signe}(\omega).$$

On réalise le montage ci-dessous, avec un moteur où l'axe est libre. Quand on ferme au temps t_0 l'interrupteur, la tension U passe de 0 V à Vcc et la vitesse ω (mesurée par le tachymètre) de 0 rad/s à $\omega_{max}=\frac{\lambda\cdot Vcc - R\cdot\alpha_s}{\lambda^2+R\cdot\alpha_f}$ en suivant l'équation : $\omega = U_{max}\cdot\left(1-e^{-\frac{\lambda^2+R\cdot\alpha_f}{\lambda^2\cdot\tau_m}(t-t_0)}\right)$.

Pour calculer J, il suffit de mesurer le temps $t_{63\%}-t_0$ nécessaire à la vitesse ω pour passer de 0 rad/s à 63.2% de ω_{max} car alors $t_{63\%}-t_0=\tau_m\cdot\frac{\lambda^2}{\lambda^2+R\cdot\alpha_f}=\frac{J}{\frac{\lambda^2}{R}+\alpha_f}$.

Exemple :



Voici, pour le moteur précédent, les mesures des tensions V_{ch1} et V_{ch2} :

A FAIRE :

Le temps de monté à 63% est égal à : $t_{63\%}-t_0=....$

Les constantes λ, R et α_f sont connues. Ainsi,

$$\begin{aligned} J &= \left(\frac{\lambda^2}{R}+\alpha_f\right)\cdot(t_{63\%}-t_0) = \frac{\cdots}{\cdots} \\ &\approx \ldots \end{aligned}$$

D'autres méthodes d'identification sont consultables au chapitre 11 du livre Moteurs électriques pour la robotique.

(version longue)

Plan

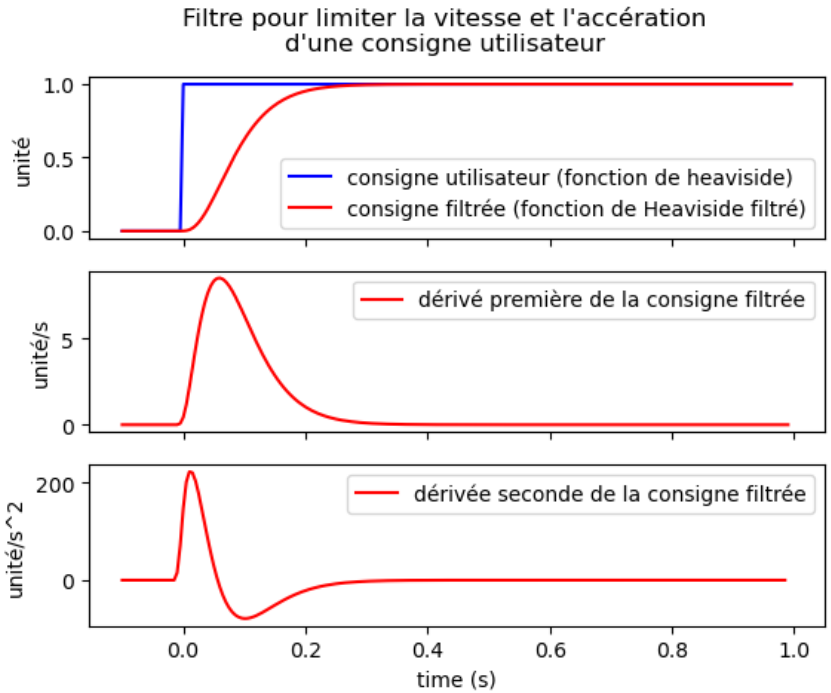
Limiter la commande d'entrée d'un moteur

Lorsque l'on controle un moteur, la consigne envoyée par l'utilistateur ne peut pas être appliquée directement car cela impose des courants trop importants pour les drivers et des accélérations trop fortes pour la mécanique.

Pour préserver la mécanique du robot et le driver du moteur il est nécessaire de limiter la commande d'entrée d'un moteur.

Cette problématique est traitée de la page 255 à la page 257 de ce cours.

Dans ces pages, on montre comment filtrer la consigne utilisateur afin de produire une commande lisse comme le montre la figure suivante :



(version longue)

Plan

(version longue)

Plan

- Les timers logiciels/software (non concurrentiel)
- Les timers matériels/hardware
- Les interruptions périodiques
- Les interruptions externes
- Les PWMs
- Calculs des fréquences et rapports cyclique des timers et PWMs

Les timers logiciels/software (non concurrentiel)

Un timer logiciel est un code qui permet d'exécuter des fonctions à des périodes données (ou à des temps spécifiques) de manière non concurrentielle : le timer et les fonctions sont executés les uns après les autres sans être intérrompus.

Ainsi, contrairement aux ordonanceurs temps réels, le timer logiciel doit attendre la fin de l'exécution des fonctions qu'il a démarrés pour pouvoir en démarrer d'autres.

Les fonctions sont donc executées avec plus ou moins de retard, c'est à dire quand le code du timer peut s'exécuter et que le temps de cooldown des fonctions est dépassé.

Pour un bon fonctionnement du programme, le temps d'execution des fonctions doit être faible vis à vis des périodes et des cooldowns du timer.

Si vous voulez avoir des garanties sur les temps d'exécution de vos fonctions, il faut utiliser des timers matériel (cf. section 2) et/ou utiliser un système d'écxécution temps réeel comme FreeRTOS pour ordonnancer les différentes tâches à exécuter.

Exemple d'utilisation d'un timer logiciel (non concurentiel)

```cpp
#include <arduino-timer.h>

Timer<10> software_timer; // this timer can register 10 tasks.

typedef struct { uint64_t loop_time; } data_t;
data_t data;
bool print_A(void *argument){
  data_t* data_ptr = static_cast<data_t*>(argument);
  Serial.print("A : "); Serial.println(data_ptr->loop_time);
  return true; // Continue to repeat (don't cancel the task).
}
bool print_B(void *argument){
  data_t* data_ptr = static_cast<data_t*>(argument);
  Serial.print("B : "); Serial.println(data_ptr->loop_time);
  return true; // ignored because it's not a periodic task.
}
bool print_C(void *argument){
  data_t* data_ptr = static_cast<data_t*>(argument);
  Serial.print("C : "); Serial.println(data_ptr->loop_time);
  return true; // ignored because it's not a periodic task.
}

uint32_t period = 1000, absolute_time = 3500, delay_time = 3500;
void setup(){
  Serial.begin(9600);
  software_timer.every(period, print_A, &data);
  software_timer.at(absolute_time, print_B, &data);
  delay(1000);
  software_timer.in(delay_time, print_C, &data);
}
void loop(){
  data.loop_time = millis();
  software_timer.tick();
}
```

ode/timers/software_timer.cpp

Ce programme utilise la bibliothèque : contrem/arduino-timer. Elle permet de créer facilement des timers logiciels.

Dans ce code, la fonction print_A() est exeecutée périodoiquement toutes les secondes. La fonction print_B() est exécutée 3.5 secondes après la début d'exécution du firmware. Enfin, la fonction print_B() est exécuté 3.5 secondes après l'execution de la 3ieme ligne du setup().

Consultez sa documentation en ligne, qui présente plus de fonctionalité, à l'adresse : documentation de arduino-timer.

(version longue)

Plan

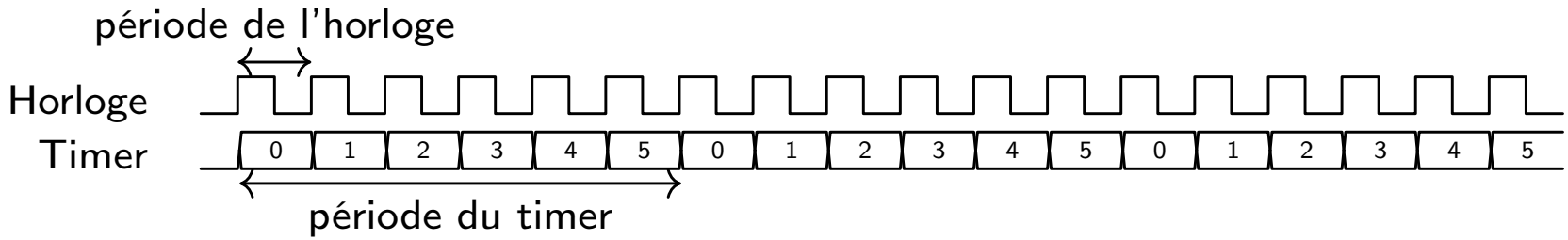
- Les timers logiciels/software (non concurrentiel)
- Les timers matériels/hardware
- Les interruptions périodiques
- Les interruptions externes
- Les PWMs
- Calculs des fréquences et rapports cyclique des timers et PWMs

Les timers matériels/hardware

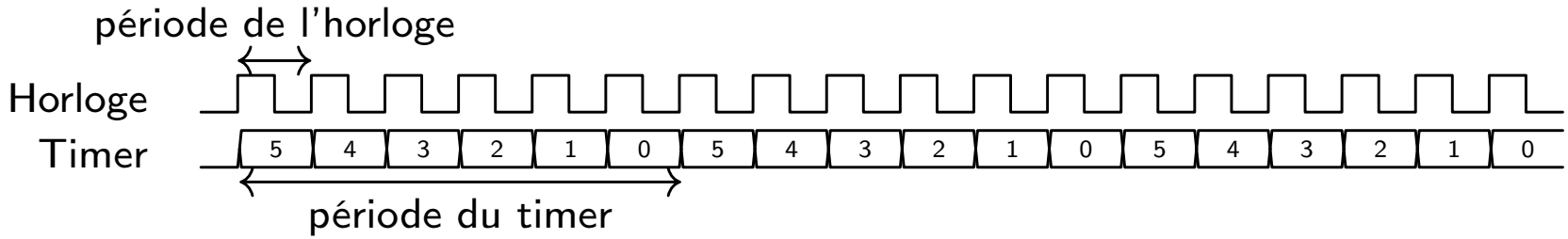
Un timer matériel est un compteur dont la valeur augmente ou diminue de 1 à une fréquence imposée par une horloge matériel. Cette fréquence f_{clk} est la fréquence de mise à jour du timer.

Il y a 3 modes de timers qui définissent l'évolution du compteur :

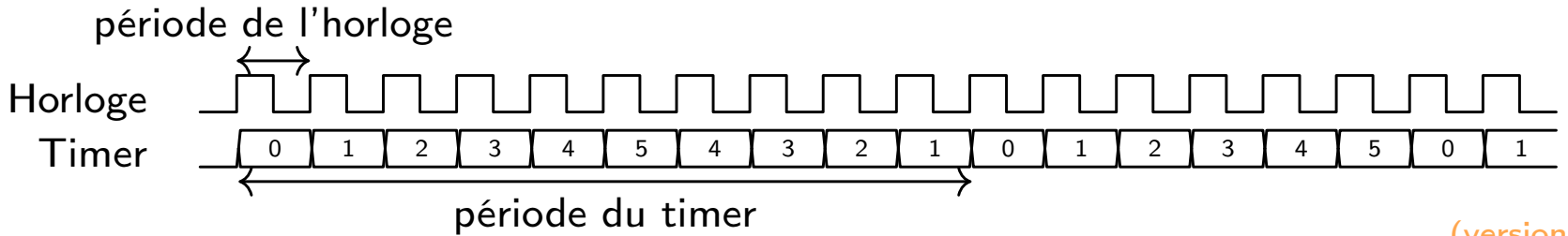
– le mode Up: le compteur augmente de 0 jusqu'à la valeur dite de prescalaire, noté p, puis recommance à 0. Exemple avec le préscalaire $p = 5$:



– le mode Down : le compteur diminue de la valeur de prescalaire p jusqu'à 0 puis recommance à la valeur de prescalaire. Exemple avec le préscalaire $p = 5$:



– Le mode Up-Down : le compteur augmente de 0 jusqu'à la valeur dite de prescalaire p puis diminue pour revenir à 0 avant de recommencer. Exemple avec le préscalaire $p = 5$:



(version longue)

Plan

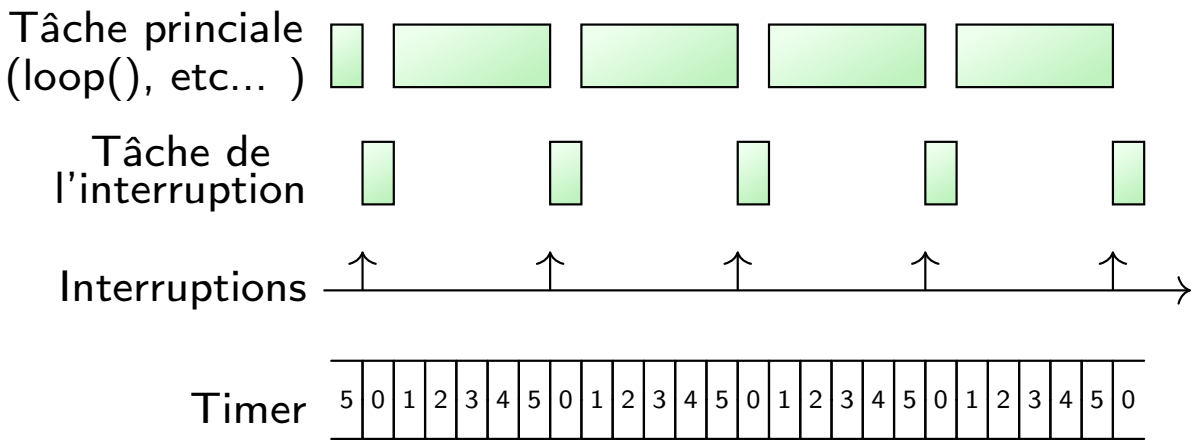
- Les timers logiciels/software (non concurrentiel)
- Les timers matériels/hardware
- Les interruptions périodiques
- Les interruptions externes
- Les PWMs
- Calculs des fréquences et rapports cyclique des timers et PWMs

Les interruptions

Pour chaque timer, on peut interrompre le processeur pour exécuter des tâches particulières. On appelle cela une interruption.

Grâe aux interruptions, on peut exécuter le code de certaines fonctions avec des garanties sur la périodicité de leurs exécutions.

Les interruptions sont déclenchées à chaque fois que le timer prend une valeur qui a été configurée à l'initialisation du microcontrolleur. Généralement, on choisit la valeur 0.



Configurer des timers est assez compliqué car cela dépend des différents microcontrôlleurs, dont voici quelques documentations :

- pour l'arduino UNO: document technique de référence de l'Atmega328P;
- pour l'esp32-C3 : document technique de référence de l'esp32-c3.

Expliquer le fonctionnement précis de ces timers dépasse le cadre de ce cours. Nous utilisons plutôt des bibliothèques prêtes à l'emploi pour configurer les timers et les interruptions.

(version longue)

Utiliser les interruptions sous Arduino Uno, nano et Mega

```
#define TIMER_INTERRUPT_DEBUG      0
#define _TIMERINTERRUPT_LOGLEVEL_  0
#define USE_TIMER_1 true
#include <TimerInterrupt.h> // To be included only
                            // in the main file.
struct { uint64_t counter; } data;
void interruption_function() {
  data.counter = data.counter + 1;
}

const unsigned int timer_frequence = 2000; // Hz
void setup() {
  Serial.begin(9600);
  ITimer1.init();
  if( !ITimer1.attachInterrupt(
    timer_frequence, interruption_function
  ) ){
    Serial.println(F("Fail to set ITimer1."
      "Select another freq. or timer"));
  }
}

int counter, old_counter = 0;
void loop(){
  noInterrupts(); // Start of critical section
  counter = data.counter;
  interrupts(); // End of critical section

  if(counter != old_counter){
    Serial.print("Update "); Serial.println(counter);
    old_counter = counter;
  }
  delay(1000);
}
```

code/timers/interruptions_unor3_nano_mega.cpp

Ce programme utilise la bibliothèque khoih-prog/TimerInterrupt de Khoi Hoang qui doit être installé.

Ce programme incrémente un compteur à 2 kHz et l'affiche toute les secondes.

On définit un timer pour executer la fonction interruption_function() à 2 kHz : la boucle principale (loop()) est interrompue toute les 500 μs pour exécuter interruption_function().

La fonction de l'interruption doit être la plus courte possible.

Dans la fonction loop(), il est important de créer une section critique pour désactiver les interruptions quand on récupére le contenu du compteur. Sans cette section critique, interruption_function() pourrait interrompre la copie et modifier le compteur, corrompant les données en cours de copie. La section critique doit aussi être la plus courte possible.

Un exemple concret de mis en oeuvre est présent dans la partie *Implémentation des filtres numériques* à la page 245

(version longue)

Utiliser les interruptions sous Arduino Uno R4 – 1/2

code/timers/interruptions_unor4.cpp

```
#include <FspTimer.h>

struct { uint64_t counter; } data;
void interruption_function(
  timer_callback_args_t __attribute((unused)) *p_args
) {
  data.counter = data.counter + 1;
}

FspTimer timer;
bool beginTimer(float frequence) {
  uint8_t timer_type = GPT_TIMER;
  int8_t tindex = FspTimer::get_available_timer(timer_type);
  if (tindex < 0){
    tindex = FspTimer::get_available_timer(timer_type, true);
  }
  if (tindex < 0) return false;
  FspTimer::force_use_of_pwm_reserved_timer();
  if( !timer.begin(
      TIMER_MODE_PERIODIC, timer_type, tindex, frequence,
      0.0f, interruption_function
  ) ) return false;
  if (!timer.setup_overflow_irq()) return false;
  if (!timer.open()) return false;
  if (!timer.start()) return false;
  return true;
}
```

Ce programme, découpé en 2 parties, incrémente un compteur à 2 kHz et l'affiche toute les secondes.

Ce programme est une adaptation du code publié par Phil Schatzmann sur sa page web : under the hood arduino uno r4 timers.

On définit un timer pour executer la fonction interruption_function() à 2 kHz : la boucle principale (loop()) est interrompue toute les 500 μs pour exécuter interruption_function().

La fonction de l'interruptions doit être la plus courte possible. Ici elle incrémente un compteur.

Utiliser les interruptions sous Arduino Uno R4 – 2/2

code/timers/interruptions_unor4.cpp

```
const unsigned int timer_frequence = 2000; // Hz
void setup() {
  Serial.begin(9600);
  if( !beginTimer(timer_frequence) ){
    Serial.println("Failed to init timer");
  }
}

int counter, old_counter = 0;
void loop(){
  noInterrupts(); // Start of critical section
  counter = data.counter;
  interrupts(); // End of critical section

  if(counter != old_counter){
    Serial.print("Update "); Serial.println(counter);
    old_counter = counter;
  }
  delay(1000);
}
```

Dans la fonction loop(), il est important de créer une section critique pour désactiver les interruptions quand on récupére le contenu du compteur. Sans cette section critique, interruption_function() pourrait interrompre la copie et modifier le compteur, corrompant les données en cours de copie. La section critique doit aussi être la plus courte possible.

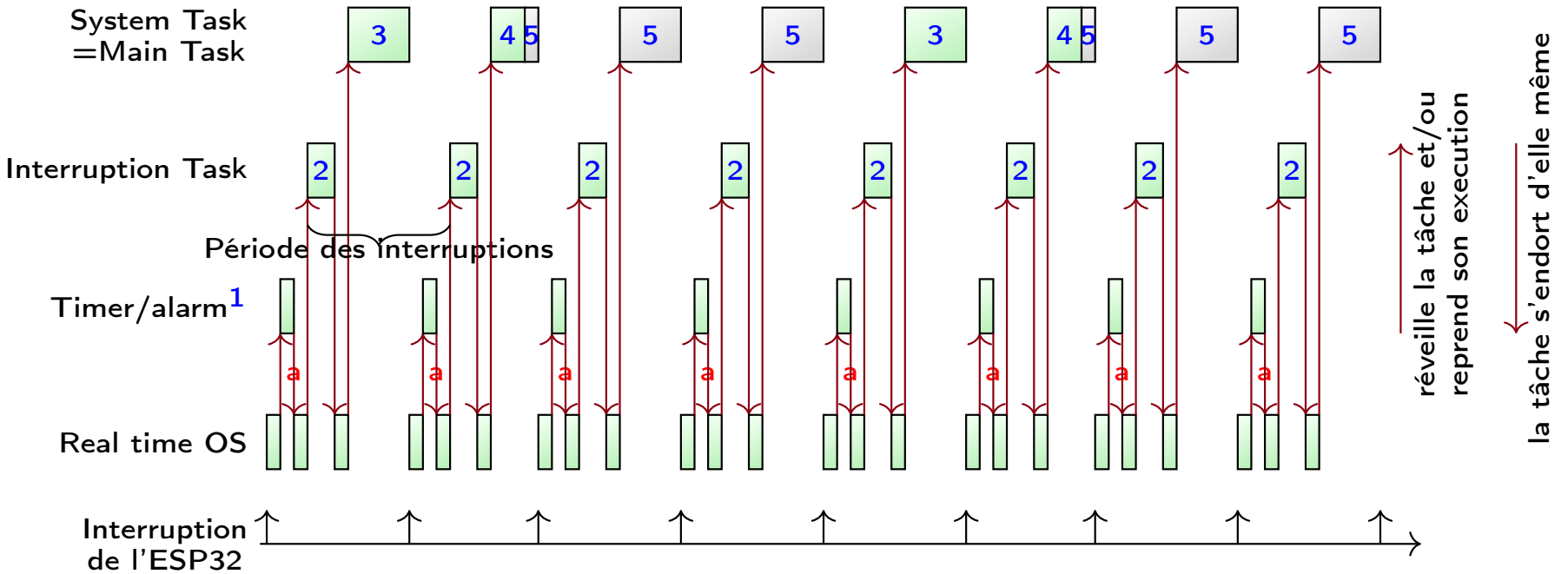
Un exemple concret de mis en oeuvre est présent dans la partie *Implémentation des filtres numériques* à la page 235

Utiliser les interruptions sous ESP32

L'implémentation pour ESP32 est plus compliquée. L'ESP32 ne permet pas d'exécuter les fonctions d'interruptions pendant suffisament de temps pour réaliser des tâches modérément complexes comme lire des broches analogiques.

Il faut donc utiliser le système temps réels FreeRTOS de l'ESP32 pour créer une tâche que l'on nomera "Interruption Task" qui sera réveillée par le timer. Cette tâche aura une priorité plus élevée vis à vis de la tâche principale ("System Task"), ce qui assurera son exécution périodique.

Voici le chronograme de ces différentes tâches à implémenter :



1: exécute à chaque alarme la fonction qui réveille la tâche d'intérruption; 2: réalise la tâche périodique associée à l'interruption; 3/4: debut/fin du travail que doit exécuter la tâche "sytème"; 5: la tâche attends (iutilisation de delay()) ou fait d'autres tâches.
a: demande à l'OS de réveiller la tâche d'échantillonage (Sample Task).

(version longue)

Utiliser les interruptions sous ESP32 – 1/3

```
#include <driver/gptimer.h>

struct { uint64_t counter; } data;
static portMUX_TYPE spinlock =
  portMUX_INITIALIZER_UNLOCKED;

void make_interruption_task( void * pvParameters ){
  int counter = 0;
  for( ;; ){
    counter = counter + 1;

    taskENTER_CRITICAL(&spinlock);
      data.counter = counter;
    taskEXIT_CRITICAL(&spinlock);

    vTaskSuspend( NULL );
  }
}

const unsigned int INTERRUPTION_TASK_PRIORITY = 2;
const int STACK_SIZE = 1000;
TaskHandle_t interruption_task_handle = NULL;

void create_the_interruption_task(){
  xTaskCreate(
    make_interruption_task, "Interruption Task",
    STACK_SIZE, NULL,
    INTERRUPTION_TASK_PRIORITY, &interruption_task_handle
  );
  configASSERT( interruption_task_handle );

  vTaskSuspend( interruption_task_handle );
}
```

Le code de ce programme, découpé en 3 parties, incrémente un compteur à 2 kHz et l'affiche toute les secondes.

Cete partie du code créé la tâche d'interruption ayant pour nom *"Interruption Task"*.

Cette tâche est constituée d'une boucle infinie dont l'exécution du coprs de boucle est interrompu à l'aide de la fonction *vTaskSuspend(NULL)* et reprise (à la ligne vTaskSuspend()) par le timer de l'ESP32.

On utilise *spinlock* et les fonctions taskENTER_CRITICAL() et taskEXIT_CRITICAL() pour désactiver les interruptions. Le loquet spinlock sert à désactiver les interruptions uniquement du coeur du microcontrolleur qui exécute la tâche. Cela permet d'empêcher le système temps réel d'interrompre la tâche quand elle écrit dans les données partagées avec d'autres tâches, comme *data*.

(version longue)

Utiliser les interruptions sous ESP32 – 2/3

code/timers/interruptions_esp32.cpp

```
static bool IRAM_ATTR start_the_interruption_task(
  gptimer_handle_t timer, const gptimer_alarm_event_data_t *edata, void *user_data
){
    BaseType_t high_task_awoken;
    high_task_awoken = xTaskResumeFromISR(interruption_task_handle);
    return high_task_awoken == pdTRUE;
}

const unsigned int interruption_frequence = 2000; // In Hz
const uint64_t timer_resolution = 1000000; // 1MHz
const uint64_t alarm_count = timer_resolution / interruption_frequence;
gptimer_handle_t gptimer = NULL;
void set_a_periodic_alarm_to_resume_the_interruption_task(){
  gptimer_config_t timer_config = {
      .clk_src = GPTIMER_CLK_SRC_DEFAULT, .direction = GPTIMER_COUNT_UP, .resolution_hz = timer_resolution,
  };
  ESP_ERROR_CHECK(gptimer_new_timer(&timer_config, &gptimer));
  gptimer_event_callbacks_t call_backs = {.on_alarm = start_the_interruption_task, };
  ESP_ERROR_CHECK(gptimer_register_event_callbacks(gptimer, &call_backs, NULL));
  ESP_ERROR_CHECK(gptimer_enable(gptimer));
  gptimer_alarm_config_t alarm_config = {.alarm_count = alarm_count, .reload_count = 0, };
  alarm_config.flags.auto_reload_on_alarm = true;
  ESP_ERROR_CHECK(gptimer_set_alarm_action(gptimer, &alarm_config));
  ESP_ERROR_CHECK(gptimer_start(gptimer));
}
```

Cette partie du code initialise le timer et démarre la tâche d'interruption.

La féquence de l'interruption est définie à l'aide de la variable *interruption_frequence*.

(version longue)

Utiliser les interruptions sous ESP32 – 3/3

code/timers/interruptions_esp32.cpp

```
void setup() {
  Serial.begin(9600);

  create_the_interruption_task();
  set_a_periodic_alarm_to_resume_the_interruption_task();
}

int counter, old_counter = 0;
void loop(){
  taskENTER_CRITICAL(&spinlock); // Start of critical section
    counter = data.counter;
  taskEXIT_CRITICAL(&spinlock); // End of critical section

  if(counter != old_counter){
    Serial.print("Update "); Serial.println(counter);
    old_counter = counter;
  }
  delay(1000);
}
```

Dans la fonction loop(), il est important de créer une section critique pour désactiver les interruptions quand on récupére le contenu du compteur. Sans cette section critique, la tâche *"Interruption Task"* pourrait interrompre la copie et modifier le compteur, corrompant les données en cours de copie. La section critique doit aussi être la plus courte possible.

Un exemple d'utilisation est donnée dans la partie *Implémentation des filtres numériques* à la page 238

Utiliser les interruptions sous ESP32 avec des tâches synchrones et asynchrones.

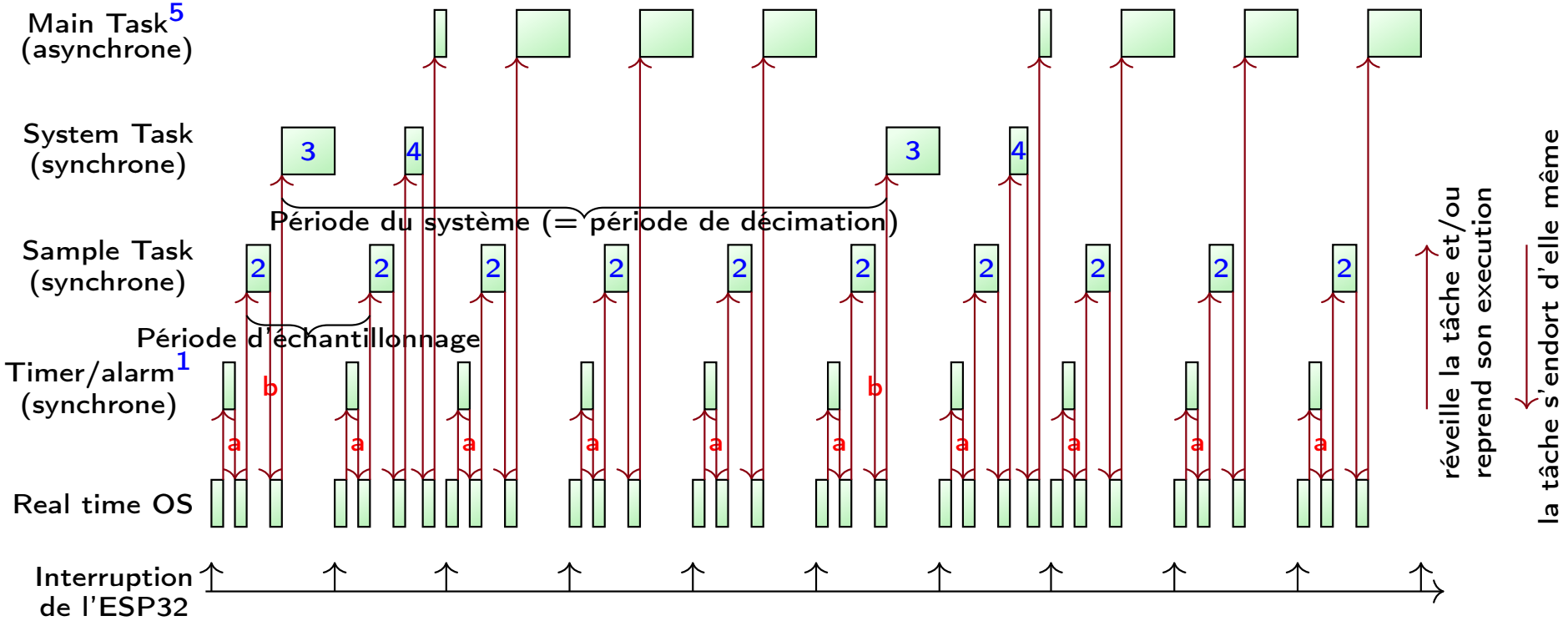
Il est souvant préférable de créer une tâche système (System Task) différente de la tâche principale (Main Task).

La tâche système réalise des tâches de manière synchrone une fréquence peu élevée (100 Hz à 200 Hz). Elle s'occupe des tâches temps réelles complexes et coûteux en temps de calculs comme le contrôle de moteurs par exemple.

La tâche d'interruption réalise des tâches rapides, comme l'échantillonage à une fréquence allant de 1 à 10kHz.

La tâche principale effectue de manière asynchrone, en tâche de fond, au fil de l'eau et de la disponibilité du microncontrolleur, des tâches non temps réels, comme comuniquer sur le port série, ou héberger un serveur Web !

Voici le chronograme de ce système mêlant tâches synchrones et tâches asynchrones :



1: exécute à chaque alarme la fonction qui réveille la tâche d'échantillonnage; 2: échantillonne, filtre et décime; 3/4: debut/fin de l'éxcution de la tâche "sytème"; 5: exécution au fil de l'eau du code de loop();
a: demande à l'OS de réveiller la tache *Sample Task*; b: demande à l'OS de réveiller la tâche *System Task*.

Vous trouver un exemple d'implémentation d'un tel système dans le fichier
code/filtre/filter_signal_with_interruption_esp32_with_synchronous_and_asynchronous_tasks.cpp (version longue)

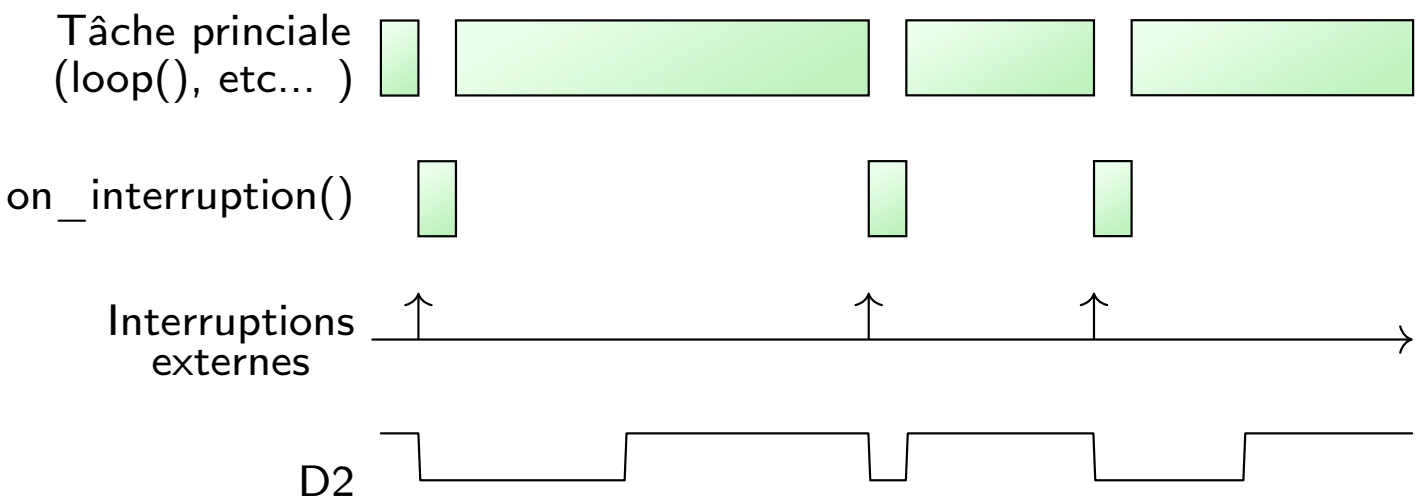
Plan

- Les timers logiciels/software (non concurrentiel)
- Les timers matériels/hardware
- Les interruptions périodiques
- Les interruptions externes
- Les PWMs
- Calculs des fréquences et rapports cyclique des timers et PWMs

Les interruptions externes

Les interruptions externes sont des interruptions générées sur des fronts montants et/ou descendants d'une broche d'un microconttolleur. Ces interruptions permettent d'executer des tâches en réponse à des évenements exterieurs, comme l'appuye d'une tocuhe d'un clavier par exemple.

Dans le chronogramme ci-dessous, sur le front descendant uniquement de la broche D2, l'éxecution de la tâche principale est interrompue le temps d'éxecuter le code d'une fonction d'interruption apellée on_interruption().



Le code des interruptions doivent être le plus cours possible afin de ne pas perturber le bon fonctionnement de la carte. Généralement, on se contente d'activer des flags (booléens) afin de les traiter ultèrieurement, dans la tâche principale.

Implémentation des interruptions externes pour toutes les cartes Arduino

Dans ce programme, à chaque fois que le signal connecté à la broche D2 passe de l'état 1 à l'état 0, la fonction on_interruption() est executée.

Les variables communes aux interruptions et à la tâche principale doivent être déclarées en "variable" pour empécher le compilateur d'optimiser le coder en retirant la ligne flag=true. En effet, pour le compilateur, cette ligne de code sert à rien car flag n'est pas utilisée dans la fonction alors qu'en fait, elle est utilisée en dehors, dans la fonction loop().

Dans la fonction loop(), les interruptions doivent être désactivées le temps de lire ou écrire les variables communes à loop() et à on_interruption() (ici, flag). Cette partie critique, qui commence par noInterrupts() et fini par interrupts() est obligatoire pour epêcher la fonction on_interruption() de corrompre les données que loop() est en train de lire ou écrire.

Pour tester ce programme, connectez un bouton sur la broche D2 et appuyez dessus. Attention ! Le rebond du bouton (cf. page 95) devrait activer plusieurs fois l'interruption.

```
const int interrupt_pin = 2;
volatile bool flag = false;

void on_interruption(){
  flag = true;
}

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

  pinMode(interrupt_pin, INPUT_PULLUP);

  // Avalaible mode : LOW, CHANGE, RISING, FALLING
  const int mode = FALLING;
  attachInterrupt(
    digitalPinToInterrupt(interrupt_pin),
    on_interruption, mode
  );
}

bool have_to_work = false;
void loop() {
  noInterrupts(); // Enter in the critical section
    have_to_work = flag;
    flag = false;
  interrupts(); // Exit the critical section

  if(have_to_work){
    Serial.println("Encore du travail ?!");
  }
}
```

code/timers/interruptions_externes_arduino.cpp

(version longue)

Implémentation des interruptions externes pour les ESP32

Dans ce programme, à chaque fois que le signal connecté à la broche 19 passe de l'état 1 à l'état 0, la fonction on_interruption() est executée.

Pour plus d'explications consultez la slide précédente.

Pour tester ce programme, connectez un bouton sur la broche 19 et appuyez dessus. (la broche 2 n'est pas disponnible pour les interruptions)

Attention ! Le rebond du bouton (cf. page 95) devrait activer plusieurs fois l'interruption.

code/timers/interruptions_externes_esp32.cpp

```
const int interrupt_pin = 19;
volatile bool flag = false;
static portMUX_TYPE spinlock =
  portMUX_INITIALIZER_UNLOCKED;

void ARDUINO_ISR_ATTR on_interruption() {
  flag=true;
}

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

  pinMode(interrupt_pin, INPUT_PULLUP);

  // Avalaible mode : LOW, CHANGE, RISING, FALLING
  attachInterrupt(
    interrupt_pin, on_interruption, FALLING
  );
}

bool have_to_work = false;

void loop() {
  taskENTER_CRITICAL(&spinlock);
    have_to_work = flag;
    flag = false;
  taskEXIT_CRITICAL(&spinlock);

  if(have_to_work){
    Serial.println("Encore du travail ?!");
  }
}
```

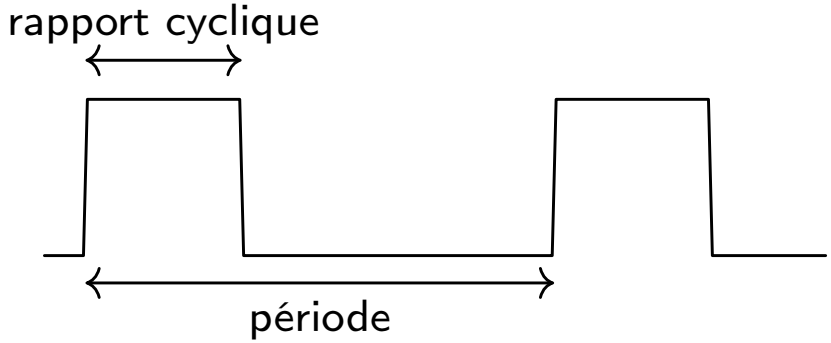
(version longue)

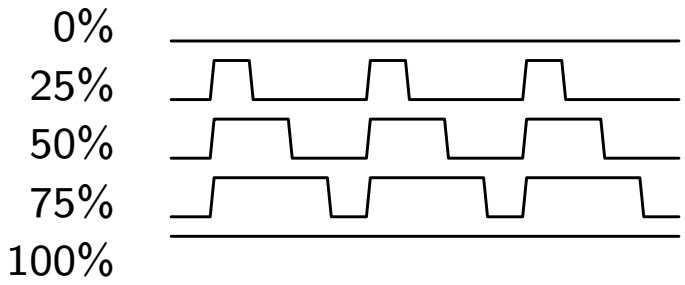
Plan

- Les timers logiciels/software (non concurrentiel)
- Les timers matériels/hardware
- Les interruptions périodiques
- Les interruptions externes
- Les PWMs
- Calculs des fréquences et rapports cyclique des timers et PWMs

Présentation des signaux en MLI (en PWM)

Un signal en MLI (Modulation de Largeur d'Impulsion) ou en PWM (Pulse With Modulation) est un signal carré (0 ou *Vcc* volt) périodique dont on fixe le rapport cyclique (duty cycle) qui est le rapport entre la durée à l'état actif et la période,



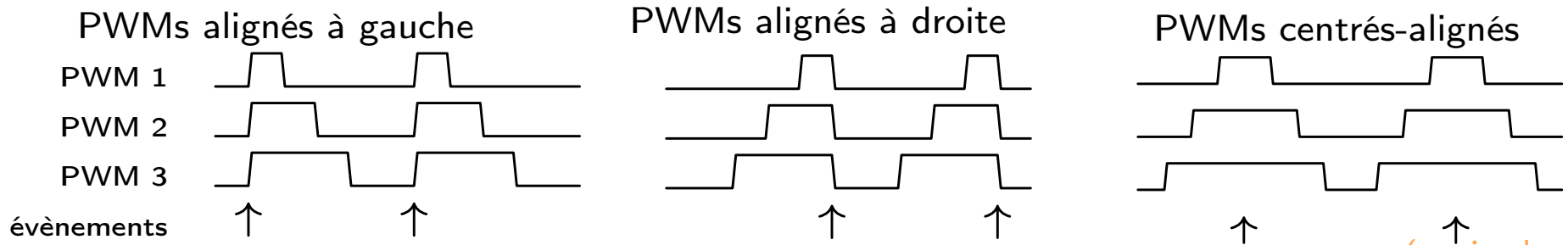


Le rapport cyclique est donc un réél entre 0 et 1.

La figure de gauche donne cinq signaux en pwm de rapport cycliques 0%, 25%, 50%, 75% et 100%.

Les trois types de timer (Up, Down, Up-Down) permettent de définir trois type de pwm, les pwms alignés à gauche (les fronts montants sont synchrones avec un évenement), les pwms alignés à droites (les fronts descendants sont synchrones avec un évenement) et les pwm centrées-alignées (le centre de l'état actif est synchrone avec un évenement).

Voici un schéma qui illustre ces trois types de PWM :

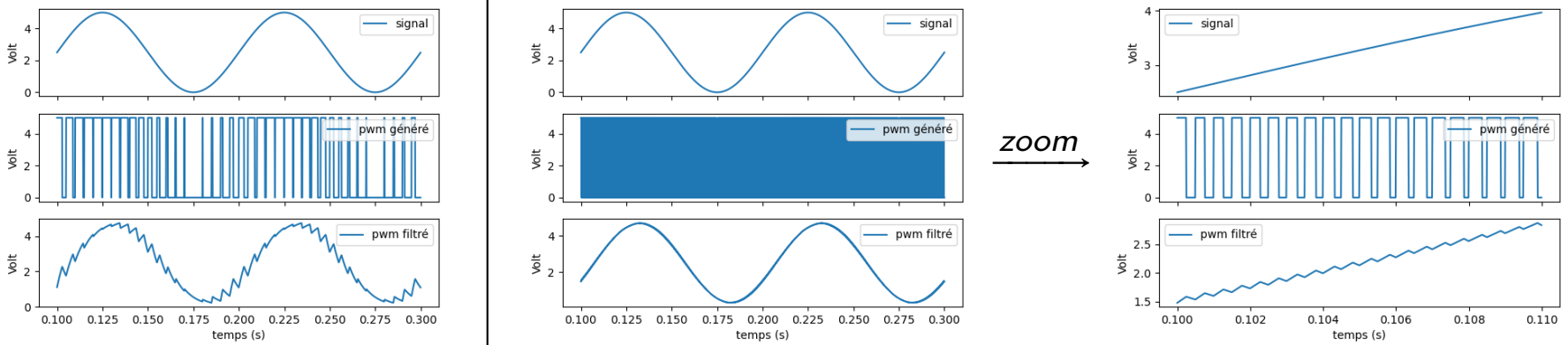


(version longue)

Où utiliser des PWMs dans vos projets

Il est compliqué de créer des signaux analogiques. Les signaux logiques sont faciles à produire.

Quand un système à commander se comporte comme un filtre passe-bas vis à vis de sa commande, il est plus facile de le commander avec une pwm dont le rapport cyclique suit le signal à produitre. On obtient les shémas de gauche et de droite suivants :



À gauche la pwm n'a pas été correctement dimentionnée afin de mieux voir la PWM et l'action du système commandé qui se comporte comme un filtre passe-bas. Au milieu la pwm a été mieux dimentionnée et à droite, un zoom a été réalisée sur la figure du milieu.

De nombreux systèmes physiques se comportent comme des filtres passe-bas. C'est le cas des moteurs par exemples que l'on commande à l'aide de PWMs (cf. page 399).

On utilise aussi les PWMs pour commander l'intensité des leds. C'est la persistance rétignienne de l'oeil humain qui joue le role de filtre passe bas. Les leds, elles, clignotent réellement.

Une balançoire est un filtre passe bande. de période $2\pi\sqrt{l/g}$ (g:cst de gravitation, l:longueur corde). Pour qu'elle oscille utilisez des impulsions à cette période. Toute autre impulsion est inutile.

Pour apprendre à dimentionner correctement une PWM et un filtre consultez la la page 250 : *Produire des signaux analogiques à partir de PWMs* .

(version longue)

Quel type de PWM choisir ?

Cette question dépasse le cadre de ce cours. La plupart du temps, le type de pwm importe peu. C'est le cas de tous les projets de ce cours.

Les pwms centré-alignés sont utilisés pour le contrôle moteur, notamment pour les moteurs à Trois phases. Cela permet :

- d'éviter que les fronts montants aient lieu en même temps. Cela réduit le bruit et limite le courant que doit apporter le driver pour faire commuter les mosfets des 3 demi-pont en H.
- en utilisant l'interruption du timer centré-aligné, on peut réaliser des mesures au centre des signaux carrés afin de réduire le bruit lié à la commutation des pwm. C'est utilie lorsque l'on souhaite mesurer le courant passant dans les phases des moteurs.

Les pwms alignés à gauche ou à droite permettent d'obtenir des fréquences plus élevées. Il sont aussi plus partiques à utiliser pour comuniquer entre différents périphériques où les lectures et écritures doivent souvant être déclanchées sur des fronts montants ou descendants.

Implémentation SIMPLIFIÉ des PWMs du framework arduino

On utilise ici l'implémentation par defaut des PWMs dans le framework d'Arduino.

```
const int pwm_pin = 5;
int duty_cycle = 0;

void setup() { }

void loop() {
  for(int i=0; i<=100; i++){
    duty_cycle = map(
      i, 0, 100, 0, 255
    ); // i % of duty cycle
    analogWrite(
      pwm_pin, duty_cycle
    );
    delay(100);
  }
}
```

code/pwm.cpp

La fréquence de la pwm est fixe et ne peut être changée. Elle dépend des cartes et des broches et vaut généralement soit 500 Hz, soit 1 kHz.

Seul le rapport cyclique qui est un entier entre 0 et 255 inclu peut être modifié en utilisant la fonction analogWrite().

Voici un exemple de code faisant allumer graduellement une led sur 10s.

Utilisation avancée des PWMs sous Arduino Uno R3

```
#include <TimerOne.h>

const int pwm_pin = 9;
const int period = 40; // us, = 1/(25 kHz)
const int MAX_DUTY = 1023;

void setup(){
  Timer1.initialize(period); Timer1.pwm(pwm_pin, 0);
}

uint32_t duty_cycle;
void loop(){
  for(int i=0; i<=100; i++){
    duty_cycle = map(i, 0, 100, 0, MAX_DUTY);
    Timer1.pwm(pwm_pin, duty_cycle);
    delay(100);
  }
}
```

code/timers/pwm_pour_arduino_r3.cpp

On utilise la bibliothèque TimerOne de Paul Stoffregen. Cette bibliothèque permet de programmer des PWM pour le timer 1 qui utilise la broche 9 et 10.

Pour programmer les broches 5 et 8, il faut utiliser le timer 0. Il n'y a pas de bibliothèque pour faire cela. Il faut le faire en code bear-metal. La page Arduino Timer Interrupts (consulté le 5/11/2024) vous sera d'une grande aide.

Pour programmer les broches 3 et 11, il faut utiliser le timer 2 et la bibliothèque theAndreas/TimerTwo (non testé, consulté le 5/11/2024).

Ce code, périodiquement, allume graduellement une led sur 10s. La led est sur la broche 9.

Attention ! Ces timers sont utilisés dans d'autre bibliothèque interne d'Arduino. Si vous modifiez leurs fréquences vous modifirez le fonctionnement de la carte arduino. Voici quelques bibliothèques et fonctionalités qui ront affectés :

- le timer 0 est utilisé pour les fonctions delay(), millis() and micros();
- le timer 1 est utilisé par la bibliothèque servo;
- le timer 2 est utilisé par la fonction Tone();

(version longue)

Utilisation avancée des PWMs sous Arduino R4 MINIMA

```cpp
#include <pwm.h>

const int pwm_pin = 9;
const unsigned int frequence = 25000; // Hz
const int MAX_DUTY = 1023;

PwmOut pwm(pwm_pin);

void setup() {
  float duty_cycle = 0;
  pwm.begin(frequence, duty_cycle);
}

void loop(){
  for(int i=0; i<=100; i++){
    float duty_cycle = i; // In %
    pwm.pulse_perc(duty_cycle);
    delay(100);
  }
}
```

code/timers/pwm_pour_arduino_r4.cpp

Les broches pwm de l'arduino UNO R4 MINIMAL sont 3, 4, 6, 9, 10 et 11.

La fréquence de la pwm a été configurée à 25 kHz.

Ce code, périodiquement, allume graduellement une led sur 10s. La led est sur la broche 9.

Utilisation avancée des PWMs sous ESP32

On utilise ici l'api native, LED Control, qui permet de configurer simplement des PWMs.

Périodiquement, ce code allume graduellement une led sur 10s. La led est sur la broche 19.

```
const int PWM_RESOLUTION = 8;
const int MAX_DUTY = (int)(pow(2,PWM_RESOLUTION)-1);
const int pwm_pin = 19;
const int frequency = 25000; // Hz

void setup(){
  Serial.begin(115200);
  bool res = ledcAttach(
      pwm_pin, frequency, PWM_RESOLUTION
  );
  if( !res ){
    Serial.println("Failed to init the pwm.");
  }
}

void loop(){
  for(int i=0; i<=100; i++){
    uint32_t duty_cycle=map(i, 0, 100, 0, MAX_DUTY);
    bool res = ledcWrite(pwm_pin, duty_cycle);
    if( !res ){
      Serial.println("Failed to set duty cycle.");
    }
    delay(100);
  }
}
```

code/timers/pwm_pour_esp32.cpp

Le nombre de PWMs que la carte peut piloter en parallèle avec LED Control est :

μC	ESP32	EPS32-S2/S3	ESP32-C3/C6/H2
nb de pwm	16	8	6

Consultez la disposition et la disponibilité des broches de votre microcontrolleur à la page A FAIRE : . On rapelle que les broches acceptant une PWM sont représenté par un ~.

(On rapelle que les cartes ESP32-WROOM-32E, ESP32-WROOM-32UE, ESP32-WROVER-E, ESP32-WROVER-IE, ESP32-WROOM-32D, ESP32-WROOM-32U, ESP32-WROOM-DA, ,ESP32-SOLO-1 et sont des cartes ESP32)

Les cartes les plus récentes, (ESP32, ESP32-C6, ESP32-S3, ESP32-P4 et ESP32-H4) possèdent des générateurs de PWM dédiées tout spécialement au contrôle des moteurs : modules MCPWM pour ESP32-S3.

(version longue)

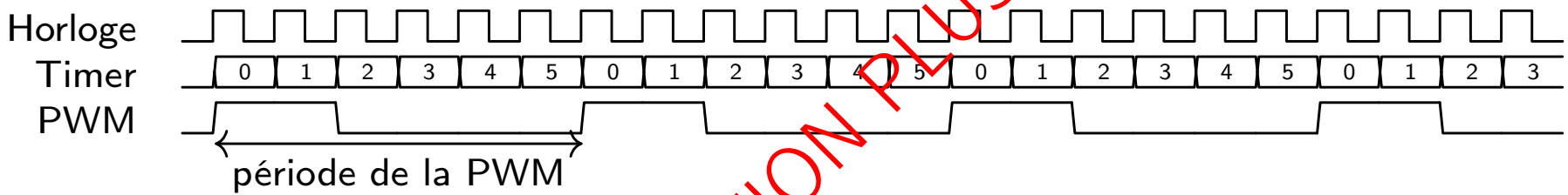
Comprendre comment sont générés les PWMs – 1/2

A partir des timers, on définit des PWMs à l'aide d'un entier e de comparaison. Cet entier permet de régler le rapport cyclique (duty cycle) de la pwm car l'état du signal digital est défini ainsi :

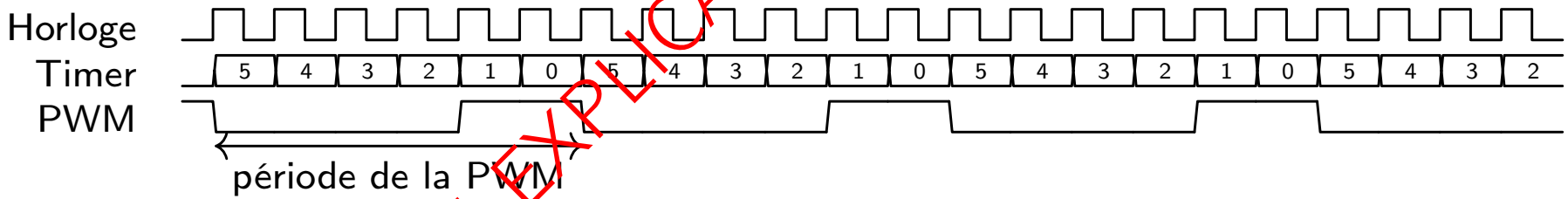
$$pwm = \begin{cases} 1 & \text{si timer} < e; \\ 0 & \textit{sinon}. \end{cases}$$

Il y a trois mode de PWM qui découlent des trois modes de timers précédent :

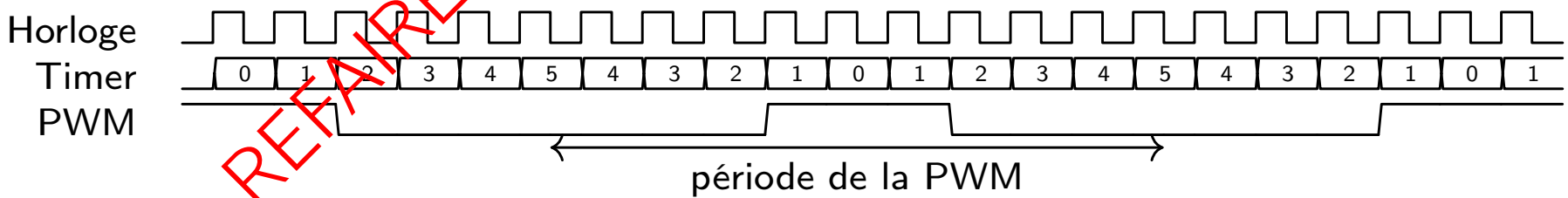
– La PWM alignée à gauche (timer Up) : (entier de comparaison = 2, préscalaire = 5)



– La PWM alignée à droite (timer Down) :(entier de comparaison = 2, préscalaire = 5)



– La PWM alignée centrée (timer Up-Down): (entier de comparaison = 2, préscalaire = 5)



A REFAIRE ! EXPLICATION PLUS COMPLIQUÉE !

(version longue)

Comprendre comment sont générés les PWMs – 1/2

Les mic Uno R3rocontroleurs propose aussi des PWMs définis à l'aide de deux entiers de comparaisons, $l \leq h$. Cela permet de constuire des PWM déphasés vis à vis du timer. On définit la pwm ainsi :

$$pwm = \begin{cases} 1 & \text{si } l < \text{timer} < e; \\ 0 & \text{sinon.} \end{cases}$$

A FAIRE : Finir les dessins.

Plan

- Les timers logiciels/software (non concurrentiel)
- Les timers matériels/hardware
- Les interruptions périodiques
- Les interruptions externes
- Les PWMs
- Calculs des fréquences et rapports cyclique des timers et PWMs

Fréquences et rapports cyclique des timers et des PWMs

La fréquence et le rapport cyclique des pwms se déduisent en fonction de leurs modes :

– Les timers Up ou down et les PWM alignée à gauche ou à droite :

$$f_{pwm} = f_{timer} = \frac{f_{clk}}{p+1} \quad \text{et} \quad \begin{array}{c}\text{raport}\\ \text{cyclique}\end{array} = \begin{cases} \dfrac{e}{p+1} & \text{si } 0 \le e \le p+1; \\ 1 & \text{si } e > p+1. \end{cases}$$

– Le timer Up-Down ou la PWM alignée centrée :

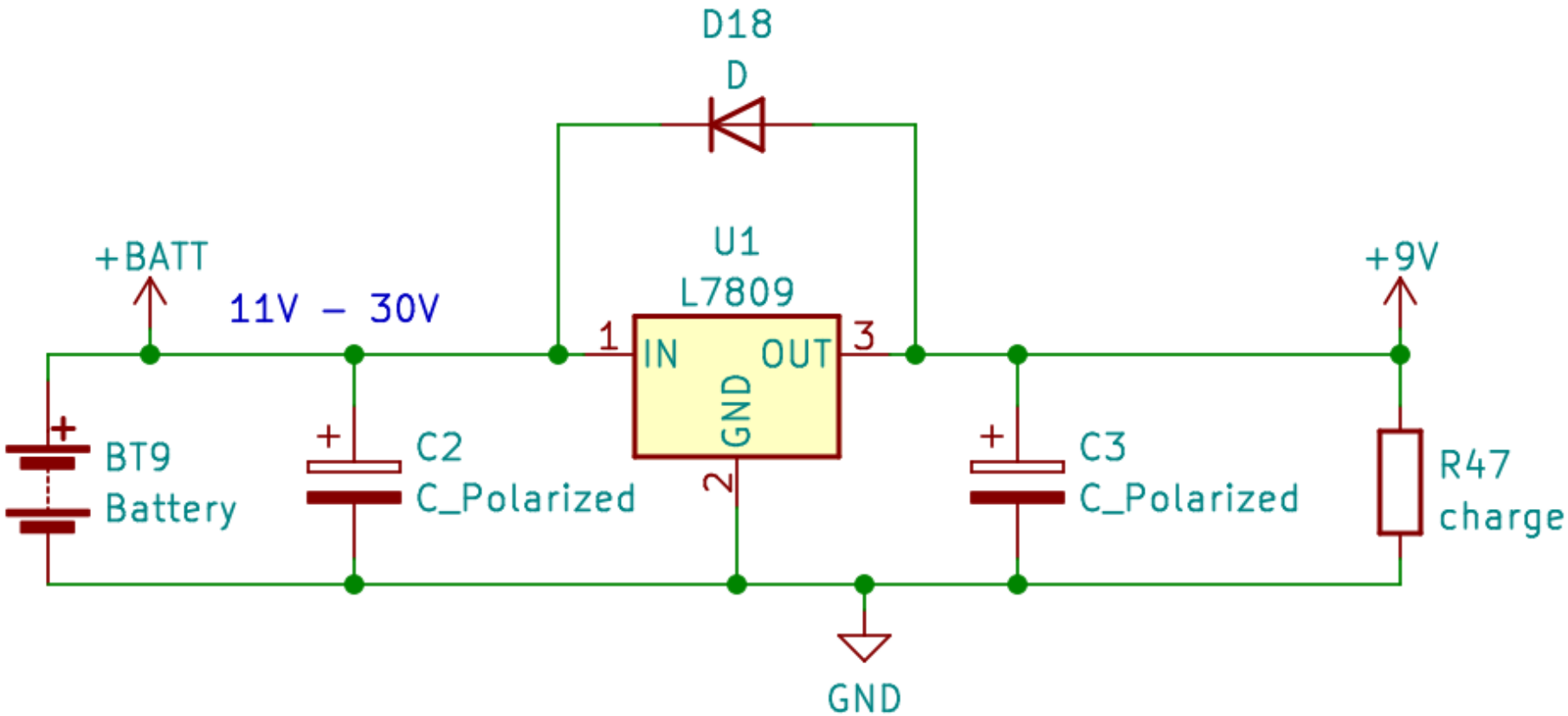
$$f_{pwm} = f_{timer} = \frac{f_{clk}}{2\cdot p+1} \quad \text{et} \quad \begin{array}{c}\text{raport}\\ \text{cyclique}\end{array} = \begin{cases} \dfrac{2\cdot e}{2\cdot p+1} & \text{si } 0 \le e < p; \\ 1 & \text{si } e \ge p+1. \end{cases}$$

Plan

(présent dans la version courte)

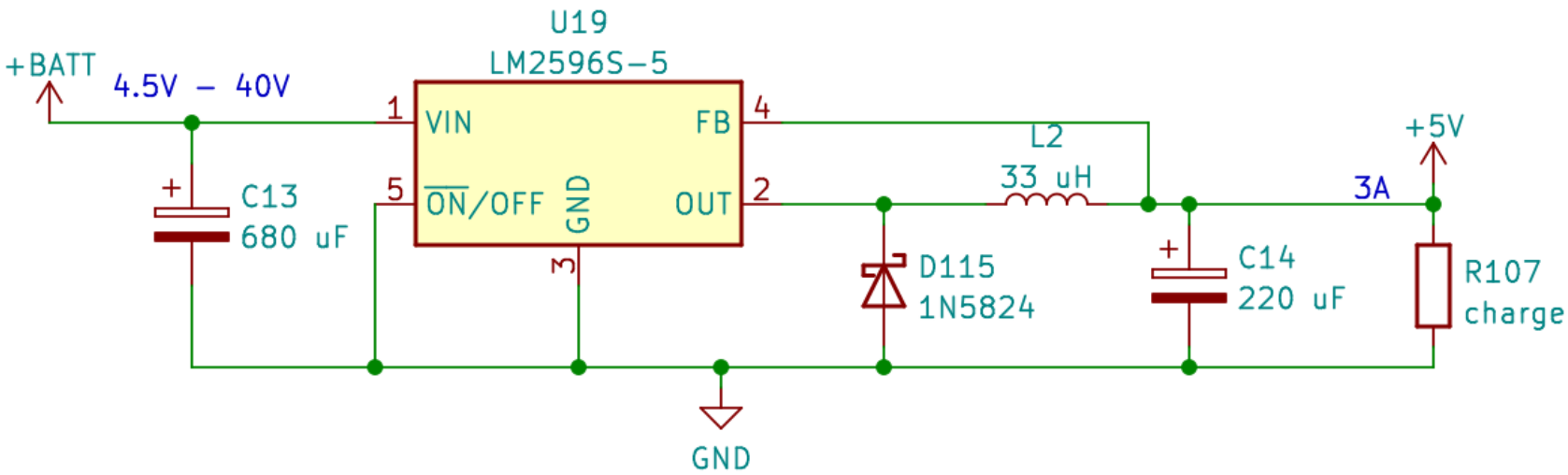
Régulateur Linéaire : la série 78XX

78XX = tension régulée de XX V

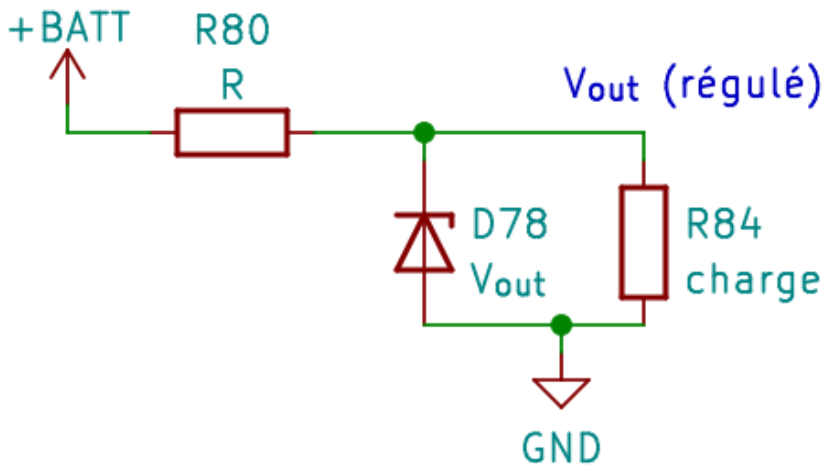


(présent dans la version courte)

Régulateur Buck



Montage simple avec une diode Zener

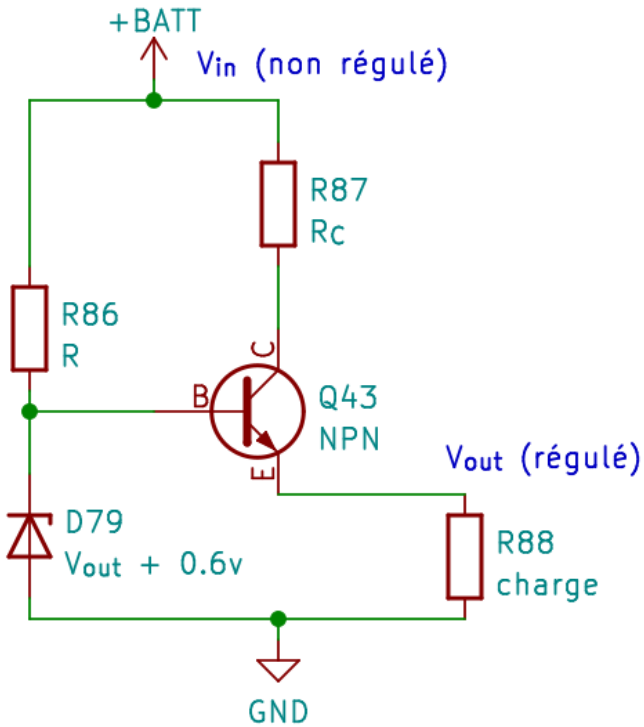


Lorsque $+BATT > V_{out}$, la tension d'avalanche de la diode zener D78, la tension au borne de R84 est égale à V_{out}.

La résistance R est déterminée afin afin ne pas détériorer la diode Zener : $R = \frac{+BATT - Vout}{I_{\text{diode, max}} + V_{out}/R_{\text{charge}}}$.

Ce montage est donc peut utilisé et est remplacer par celui de la page suivante.

Montage avec une diode Zener

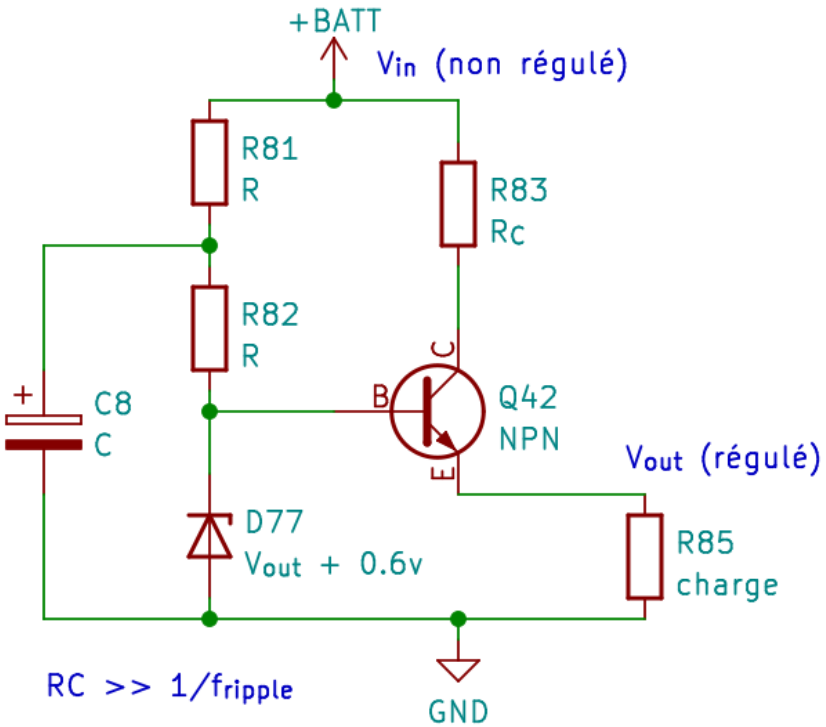


La diode zener D79 vient polariser la base du transistor NPN à la tension $V_{out} + 0.6V$.

Il faut choisir une diode zener de tension $V_{out} + 0.6V$ et une résistance $R \geq \frac{+BATT - Vout}{I_{\text{diode, max}}}$.

La résistance R_c sert à limiter le courant circulant dans le transistor Q43. Cette résistance peut être retirée si le transistor est suffisamment refroidit.

Montage avec une diode Zener



La diode zener D79 vient polariser la base du transistor NPN à la tension $V_{out} + 0.6V$.

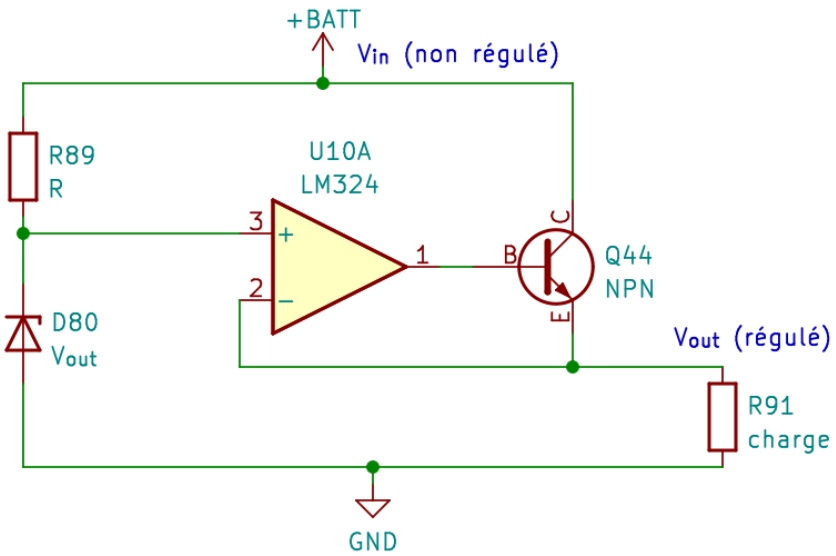
Il faut choisir une diode zener de tension $V_{out} + 0.6V$ et une résistance $R \geq \frac{+BATT - Vout}{I_{\text{diode, max}} \times 2}$.

La résistance R_c sert à limiter le courant circulant dans le transistor Q43. Cette résistance peut être retirée si le transistor est suffisamment refroidit.

On ajoute un filtre RC pour stabiliser la tension vis à vis des variations de l'alimentation.

(version longue)

Montage avec un Amplificateur Opérationnel – 1/2



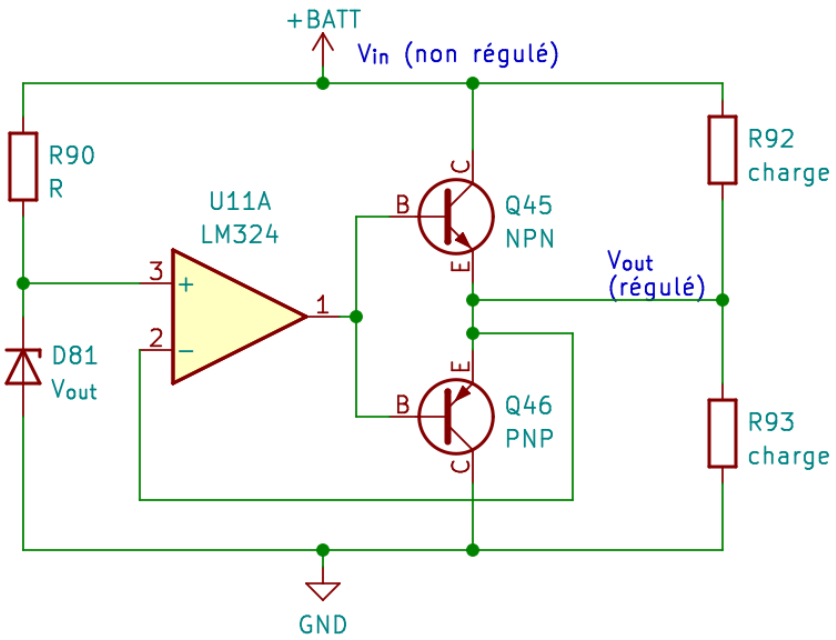
Ce montage donne une tension régulée plus stable que le précédent.

L'amplificateur opérationnel est monté avec une rétroaction négative (V_- est pilotée par sa sortie), donc $V_+ = V_-$ et la tension régulée est égale à V_{out}.

On détermine R comme dans le circuit de la page précédente.

On peut aussi ajouter un filtre RC comme à la page précédente.

Montage avec un Amplificateur Opérationnel – 2/2



C'est le même schéma que la page précédente, mais avec deux charges, une dite en pull up, connectée à l'alimentation +BATT et l'autre en pull down, connectée à la masse GND.

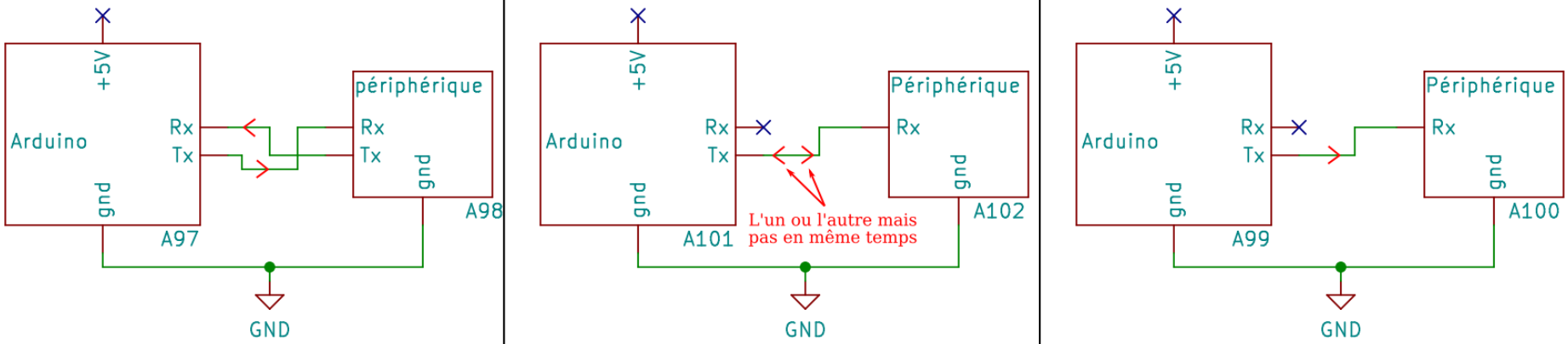
Plan

Plan

- Le protocole UART (Universal Asynchronous Receiver Transmitter)
- Le protocole I2C (Inter-Integrated Circuit)
- L'interface SPI (Serial Peripheral Interface)
- Autres protocoles
- Lire et déverminer un signal série

(présent dans la version courte)

Le protocole UART (Universal Asynchronous Receiver Transmitter)



Rx : Broche de Reception. Tx: Broche de transmition.

Il faut donc connecter la broche Rx à la broche Tx.

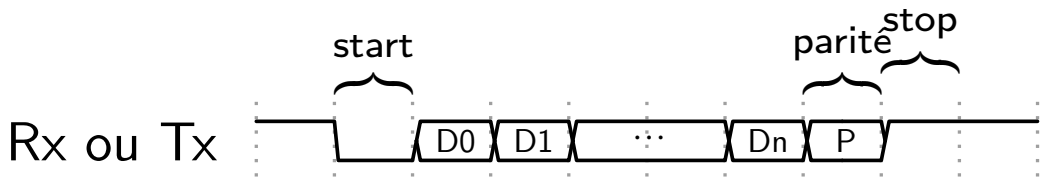
Ce protocole est notament utilisé pour la comunication série de l'Arduino.

Les taux de transferts sont des valeurs fixes. Les plus utilisées sont 9600, 19200, 57600 et 115200 bauds (nombre de symboles/seconde = nb de bits/seconde pour la comunication série arduino). Pour Arduino, $baud \approx Hz$.

(présent dans la version courte)

Le protocole UART (Universal Asynchronous Receiver Transmitter)

La comunication est dite asynchrone, car il n'y a pas d'horloge pour cadencer la comunication.



La donnée D0 à Dn est consitutée de 5 à 9 bits.

Le bit de parité P est optionnel et sert à detecter une erreur de transmition. Il se calcule de la manière suivante.

En parité paire :

$$P = \begin{cases} 1 & \text{si } D_0 + \cdots + D_n \text{ est impaire;} \\ 0 & \text{sinon.} \end{cases}$$

En parité impaire :

$$P = \begin{cases} 1 & \text{si } D_0 + \cdots + D_n \text{ est paire;} \\ 0 & \text{sinon.} \end{cases}$$

(présent dans la version courte)

Les ports UART de chaque microcontrolleur

microcontrolleur		Spéc. Hardware		Programmation		brochage	
marque	nom	protocole	port	nom	USB	RX	TX
Arduino	Uno R3 / Nano	UART		Serial	✓	0 (RX)	1 (TX)
	Leonardo / Zero	UART		Serial	✓		
				Serial1		0 (RX)	1 (TX)
	Uno R4	UART		Serial	✓		
				Serial1		0 (RX0)	1 (TX0)
	Giga R1	UART	?	Serial	✓		
		UART		Serial1		0 (RX)	1 (TX)
		UART		Serial2		19 (RX1)	18 (TX1)
		UART		Serial3		17 (RX2)	16 (TX2)
		UART		Serial4		15 (RX3)	14 (TX3)
ESP32 (devKitC/M)	Wroom-32 D/U	UART	0	Serial	✓	1 *1	3 *1
		UART	1	Serial1		~~9~~ *2	~~10~~ *2
		UART	2	Serial2		16 *1*4	17 *1*4
	C3-Wroom	UART	0	Serial	✓	20 *3 ?	21 *3 ?
		UART	1	Serial1		*1	*1
	S3-Wroom	UART	0	Serial	✓	44*3	43*3
		UART	1	Serial1		18 *1	17*1
		UART	2	Serial2		*1	*1
teensy	4.1	UART	0	Serial	✓		
		UART	6	Serial1		0 (RX1)	1 (TX1)
		UART	3	Serial2		7 (RX2)	8 (TX2)
		UART	2 (MM→4)	Serial3		15 (RX3)	14 (TX3)
		UART	4 (M→2)	Serial4		16 (RX4)	17 (TX4)
		UART	8	Serial5		21 (RX5)	20 (TX5)
		UART	1	Serial6		25 (RX6)	24 (TX6)
		UART	7	Serial7		28 (RX7)	29 (TX7)
		UART	5	Serial8		34 (RX8)	35 (TX8)

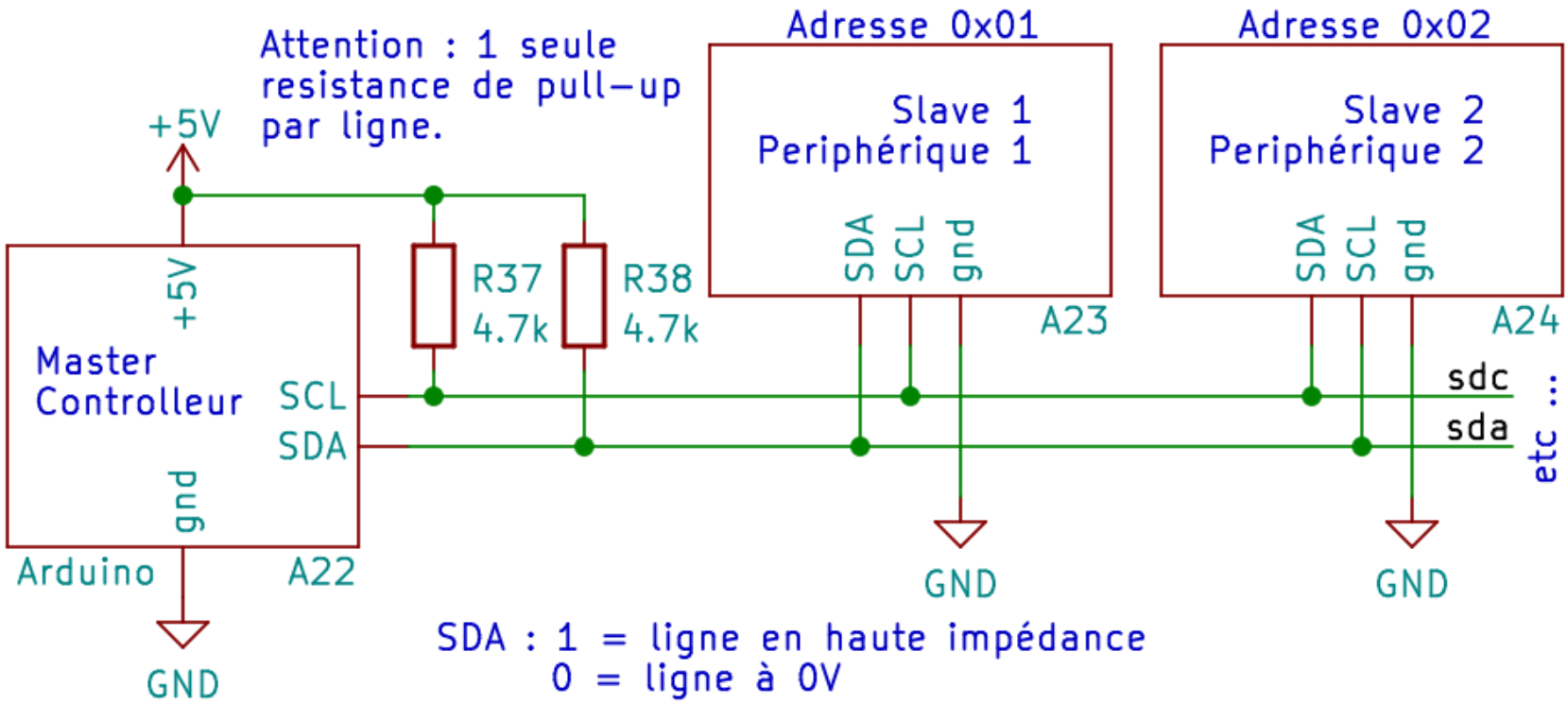
*1 On peut configurer une autre broche. *2 Le brochage DOIT être réassigné car déjà utilisé. *3 Le brochage ne peut être réassigné. *4 Le brochage doit être configuré car il ne l'est pas par default.

(version longue)

Plan

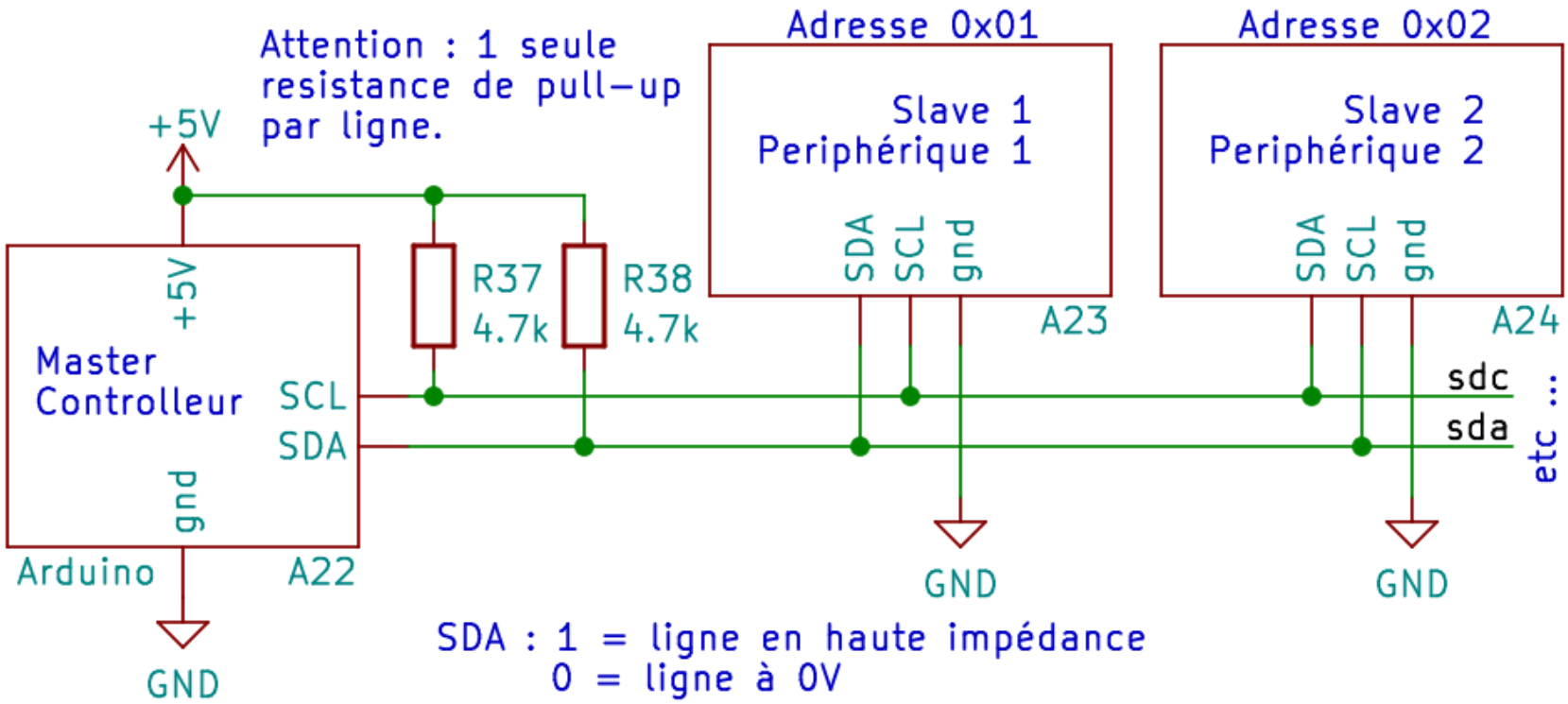
(présent dans la version courte)

L'I2C (Inter-Integrated Circuit)



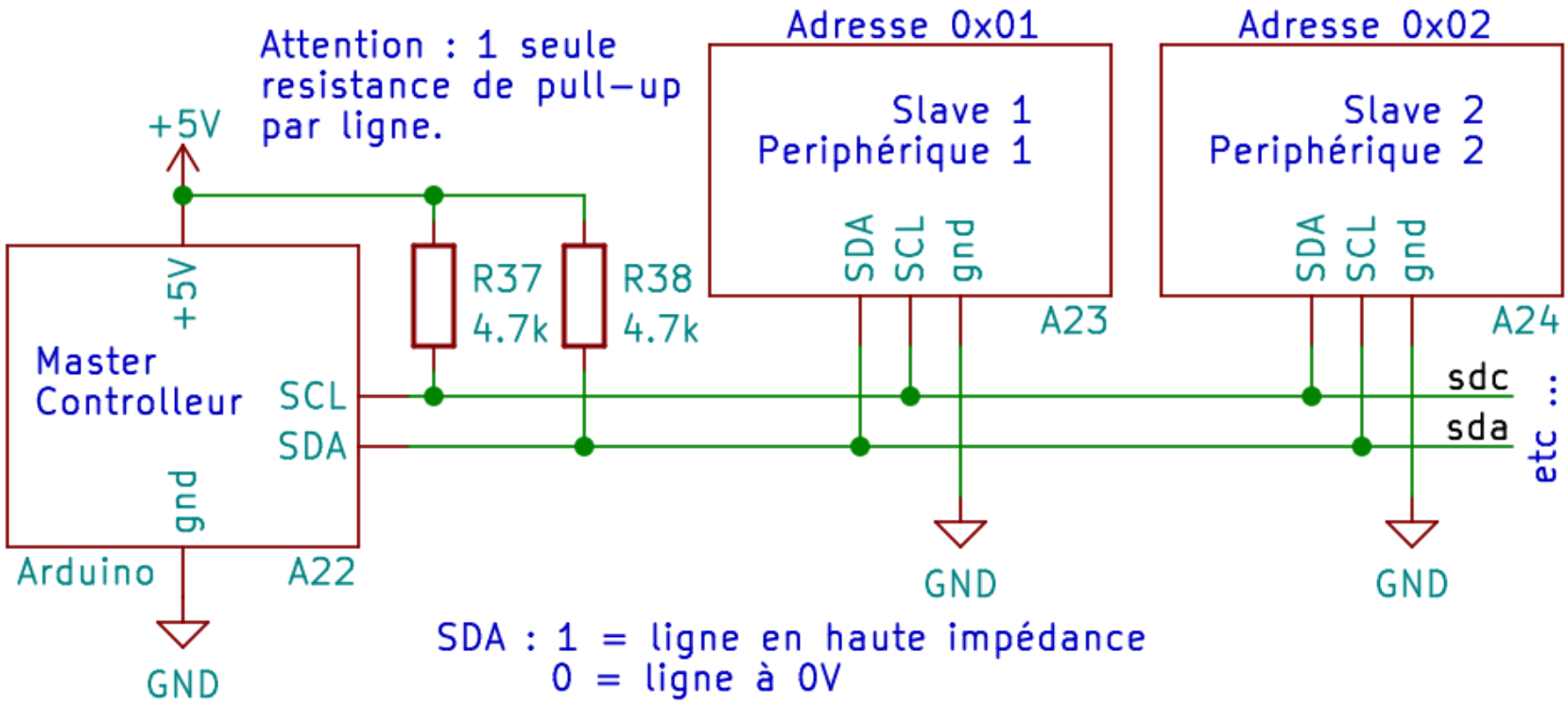
SCL = clock, SDA : data

L'I2C (Inter-Integrated Circuit)

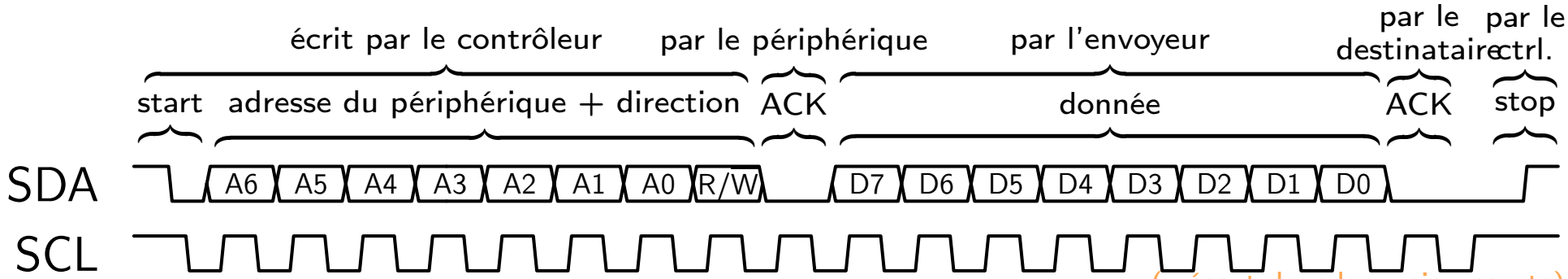


SCL = clock, SDA : data
état bas/haut horloge ⇒ écriture/lecture (sauf pour start et stop)

L'I2C (Inter-Integrated Circuit)



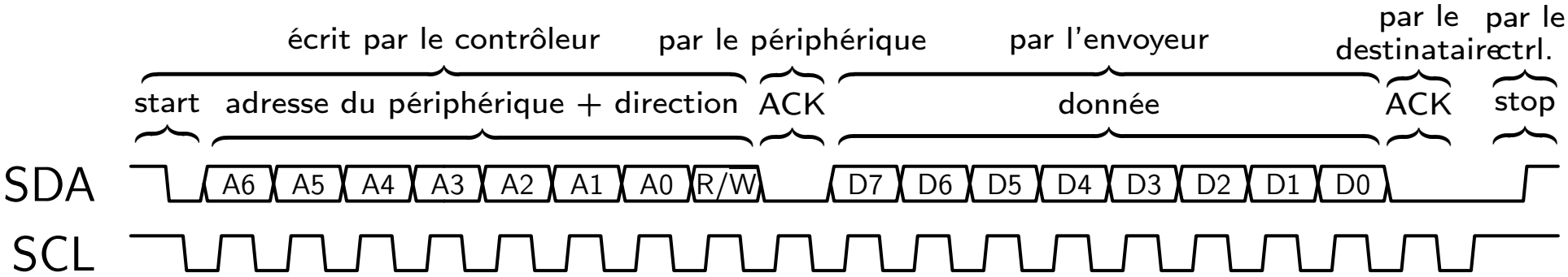
SCL = clock, SDA : data ACK = acknowledgment
état bas/haut horloge ⇒ écriture/lecture (sauf pour start et stop)



(présent dans la version courte)

Le protocole I2C (Inter-Integrated Circuit)

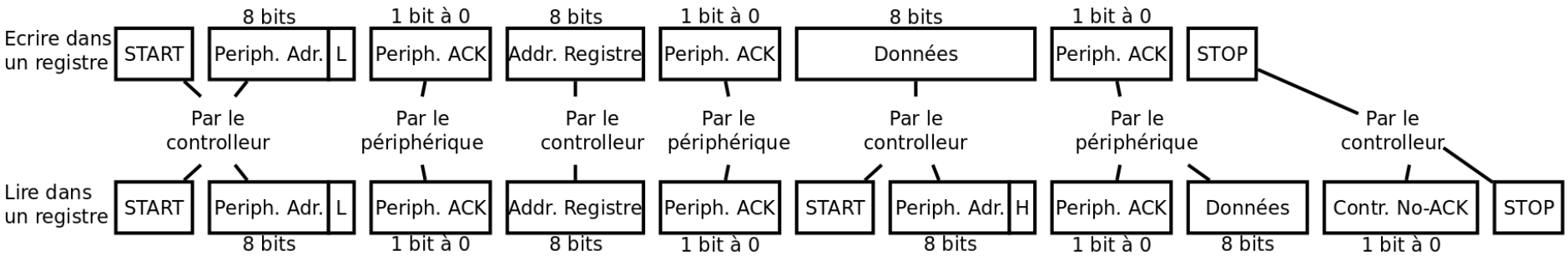
C'est toujours le controlleur qui initie la comunication. pour une lecture (driection = R) ou une écritrue (direction=W). Voici le chronograme des bits échangés pour les deux premiers paquets. Le premier consiste à écrire l'adresse du périphérique avec lequel le controlleur souhaite échanger des informations. Seul le périphérique appellé à le droit d'utiliser la ligne de comunication (la mettre à 0V), les autres doivent laisser la ligne en haute impédance :



Le controlleur peut réaliser deux tâches distinctes :

- Écrire dans une registre : envoyer une donnée à un périphérique concernant un registre donnée;
- Lire dans un registre : récupérer une information du périphérique correspondant à un regitre donné.

Les échanges de paquets à réaliser pour ces deux tâches sont donc les suivantes :



Schémas reproduit à partir du livre The Art of Electronics, Paul Horowitz et Winfield Hill, 3th Edition, 2015, Cambridge Press, page 1034.

(version longue)

Les fréquences du protocole I2C (Inter-Integrated Circuit)

Par défault, l'arduino configure l'I2C à 100 kHz.

Si les puces le permettent, l'I2C peut être configuré en :

- Mode standard (standard mode – Sm): ≤ 100 kHz;
- Mode rapide (fast mode – Fm): ≤ 400 kHz;
- Mode rapide+ (fast mode plus – Fm+) : $\leq 1 MHz$;
- Mode haute vitesse (high-speed mode – Hs-mode) : $\leq 3.4 MHz$;
- Mode ultra rapide (Ultra-fast mode – UFm) : $\leq 5 MHz$, unidirectionnel seulement.

Le protocole I2C est très sensible au bruit, il est donc conseillé d'utiliser le mode standard et de l'utiliser pour des comunications de puce à puce en rappocahnt les puces entre elles.

A FAIRE : Mettre un exemple de code ou on change la frequence de l'I2C.

Pour plus d'information, consultez l'API officielle d'Arduino concernant le protocole I2C.

Les bus I2C des microcontrolleurs et leurs brochages

microcontrolleur		spécification constructeur		Programmation		brochage	
marque	nom	protocole	bus	nom	default*4	SDA[3]	SCL*3
Arduino	Uno R3	I2C		Wire	✓	A4	A5
	Zero	I2C	?	Wire	✓	20	21
	Leonardo	I2C	?	Wire	✓	D2	D3
	Nano	I2C	?	Wire	✓	A4	A5
	Uno R4	I2C	?	Wire	✓	A4	A5
		I2C	?	Wire1		D27	D26
	Giga R1	I2C	?	Wire	✓	D20	D21
		I2C	?	Wire1		D102 (SDA1)	D101 (SCL1)
		I2C	?	Wire2		D9 (SDA2)	D8 (SCL2)
ESP32 (devKitC/M*5)	Wroom-32 D/U	I2C	0	Wire	✓	21*1	22*1
		I2C	1	Wire1		*1 *2	*1 *2
	C3-Wroom	I2C	0	Wire	✓	8*1	9*1
	C6-Wroom	I2C	0	Wire	✓	23*1	22*1
	h2-MINI	I2C	0	Wire	✓	12*1	22*1
		I2C	1	Wire1		*1 *2	*1 *2
	S2/S3-Wroom	I2C	0	Wire	✓	8*1	9*1
		I2C	1	Wire1		*1 *2	*1 *2
teensy	4.1	I2C	0	Wire	✓	18	19
		I2C	1	Wire1		17	16
		I2C	1	Wire1		44*2	45*2
		I2C	2	Wire2		25	24

*1 Il s'agit des broches par default. N'importe quelle autre broche peut être programmée.
*2 Le brochage n'est pas défini par default et doit être configuré manuellement au setup.
*3 C'est aussi le nom de la macro C dans le code source.
*4 Le bus I2C qui est utilisé par défaut par de nombreuses bibliothèque.
*5 Le brochage I2C correspond à celui des cartes de developpement "devKitC" et "devKitM" officielles de espressif.

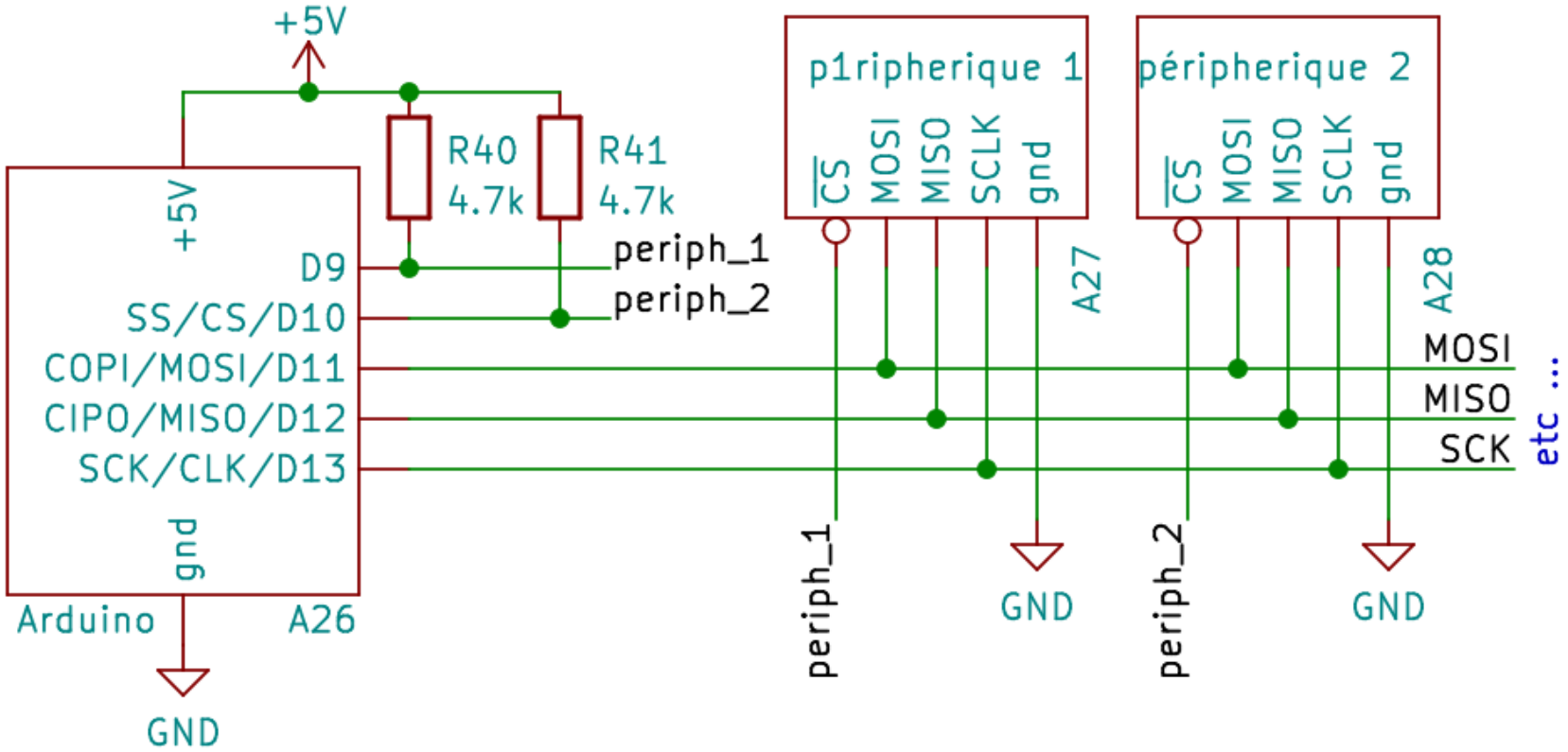
(version longue)

Plan

- Le protocole UART (Universal Asynchronous Receiver Transmitter)
- Le protocole I2C (Inter-Integrated Circuit)
- L'interface SPI (Serial Peripheral Interface)
- Autres protocoles
- Lire et déverminer un signal série

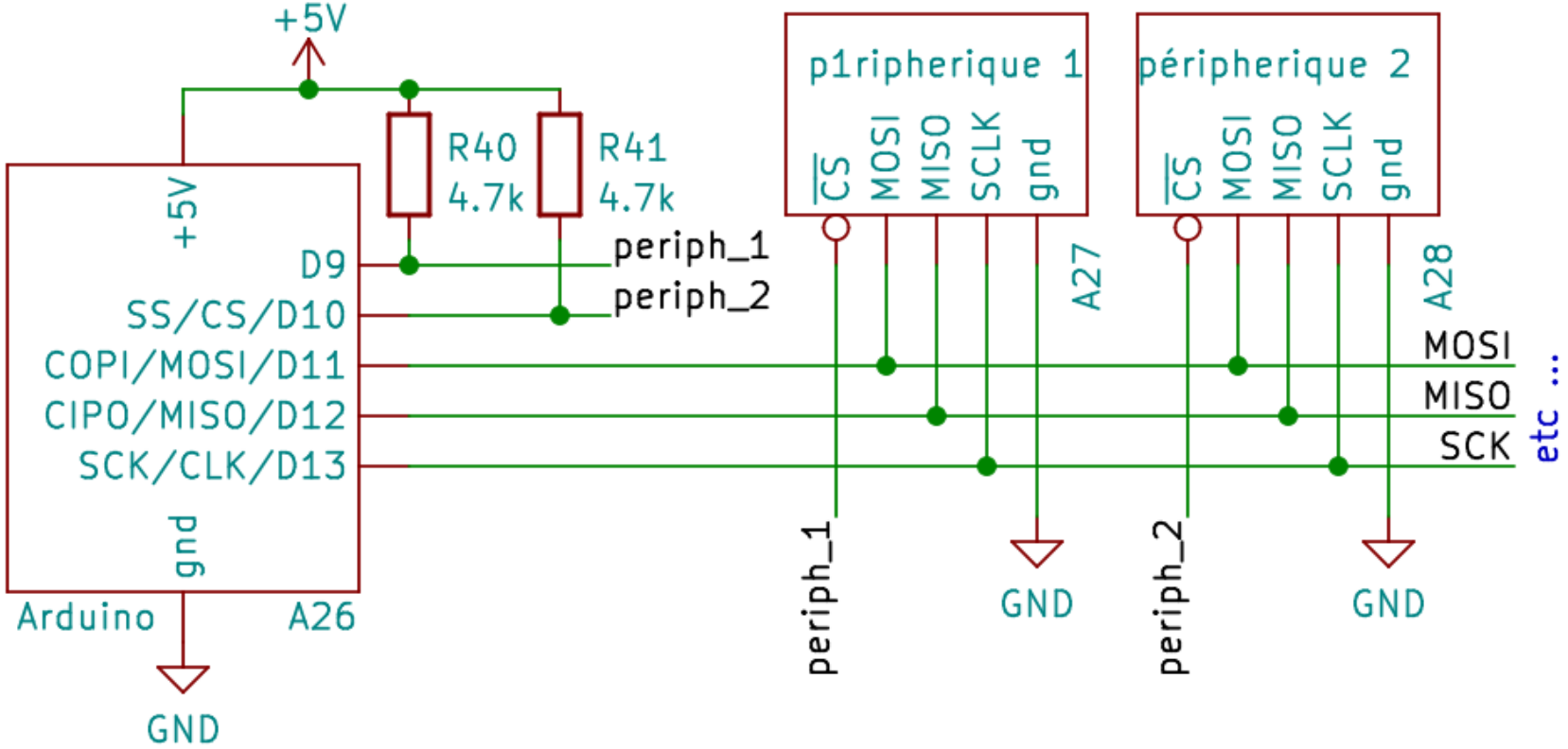
(présent dans la version courte)

L'interface SPI (Serial Peripheral Interface)



(présent dans la version courte)

L'interface SPI (Serial Peripheral Interface)

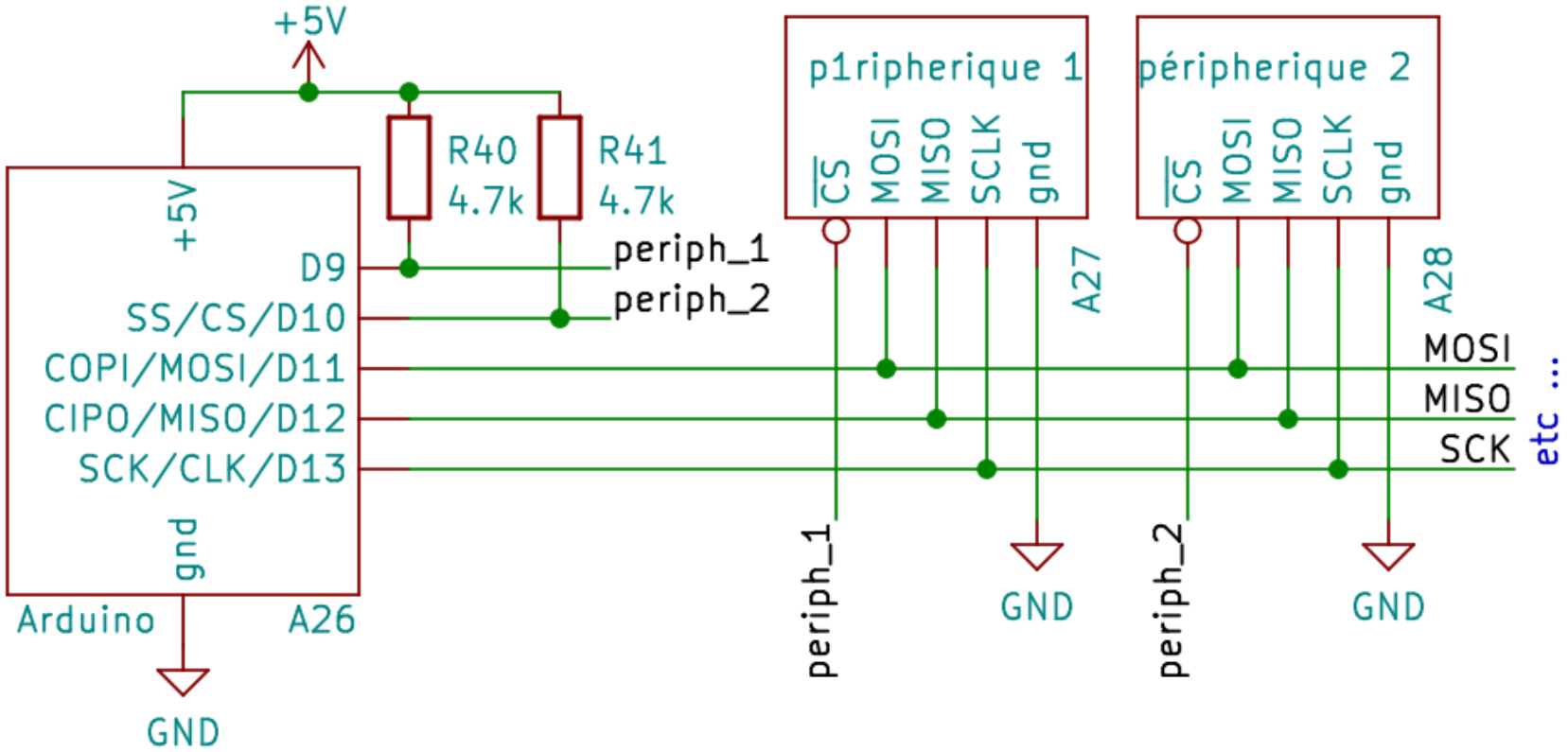


CS : Chip Select, SCLK : clock, MOSI : Master Out Slave In, MISO : Master In Slave Out.

front montant/descendant horloge ⇒ lecture/écriture

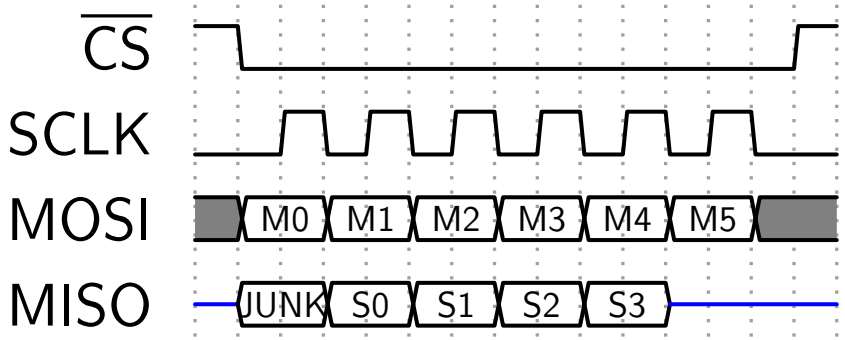
(présent dans la version courte)

L'interface SPI (Serial Peripheral Interface)



CS : Chip Select, SCLK : clock, MOSI : Master Out Slave In, MISO : Master In Slave Out.

front montant/descendant horloge ⇒ lecture/écriture



(présent dans la version courte)

Les fréquences du bus SPI

La fréquence maximal des bus SPI dépend des microcontrolleurs et des périphériques.

Par exemple, la fréquence par défault du SPI de l'arduino UNO est 4 MHz.

A FAIRE : Mettre un exemple de code pour modifier la fréquence du bus SPI.

Pour plus d'information, consultez
l'API officielle d'Arduino concernant l'interface SPI.

Les interfaces SPI des microcontrolleurs et leurs brochages

microcontrolleur		spécification constructeur		Programmation			brochage			
marque	nom	protocole	bus	instance	bus	default*4	SS*3	MOSI*3	MISO*3	SCK*3
Arduino	Uno R3/R4 Zero Leonardo Nano	SPI		SPI		✓	10	11	12	13
	Giga R1	SPI	?	SPI		✓	10	90	89	91
		SPI	?	SPI1			10 (SS1)	11 (MOSI1)	12 (MISO1)	13 (SCK1)
ESP32 (devKitC/M*5)	Wroom-32 D/U	SPI	2	SPI	HSPI	✓	5*1	23*1	19*1	18*1
		SPI	3	*7	VSPI		*2			
	C2/C3-Wroom	SPI	2	SPI	FSPI	✓	7*1	6*1	5*1	4*1
	C6-Wroom	SPI	2	SPI	FSPI	✓	18*1	19*1	20*1	21*1
	h2-MINI	SPI	2	SPI	FSPI	✓	0*1	25*1	11*1	10*1
	S2-Wroom	SPI	2	SPI	FSPI	✓	34*1	35*1	37*1	36*1
		SPI	3	*7	HSPI		*2			
	S3-Wroom	SPI	2	SPI	FSPI	✓	10*1	11*1	12*1	13*1
		SPI	3	*7	HSPI		*2			
teensy	4.1	SPI	4	SPI		✓	10	11	12	13
		SPI	3	SPI1			0*6	26*6	1*6	27*6
		SPI	1	SPI2			44*6	43*6	42*6	45*6

*1 Il s'agit des broches par default. N'importe quelle autre broche GPIO peut être programmée à la place.
*2 Non définies par default et doivent être configurées manuellement au setup. Toutes les broches GPIO peuvent être utilisées.
*3 C'est aussi le nom de la macro/variable C dans le code source.
*4 Le bus SPI qui est utilisé par défault par de nombreuses bibliothèques.
*5 Le brochage SPI correspond à celui des cartes de developpement "devKitC" et "devKitM" officielles de espressif.
*6 Aucune macro/variable C n'est définie pour ces broches dans le framework Arduinoteensy.
*7 L'instance SPI1 associée à l'interface SPI n'existe pas par default. Il faut la créer manuellement pour pouvoir l'utiliser :

(version longue)

Plan

(présent dans la version courte)

Autres protocoles

A FAIRE : traiter le RS-232, RS-422, RS-485, le bus Can, le bus etherCAT et l'USB

A FAIRE : Expliquer l'interêt des paires différentielles pour se prémunir du bruit.

A FAIRE : Faire un tableau comparatif des différents protocoles, de leurs avantages et inconvénients.

Plan

- Le protocole UART (Universal Asynchronous Receiver Transmitter)
- Le protocole I2C (Inter-Integrated Circuit)
- L'interface SPI (Serial Peripheral Interface)
- Autres protocoles
- Lire et déverminer un signal série

Lire et déverminer un signal série - L'analyseur logique

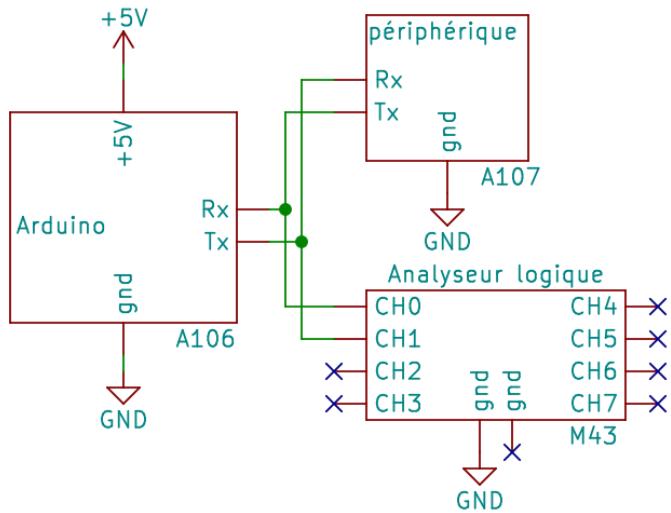
Pour étudier l'évolution de plusieurs signaux logiques, on utilise un analyseur logique. Certains oscilloscopes proposent des analyseurs logiques intégrés. Par exemple, l'oscilloscope suivant possède 32 canaux logiques :

A FAIRE : Mettre une image d'un oscilloscope avec un analyseur logique.

Bien que cela ne remplace pas un analyseur logique proffessionel, il est aussi possible de se procurer l'analyseur logique bon marché de sparkfun ou une de ses copies (entre 5 et 20 euros) qui est présentés ci-dessous.

Cet appareil peut échantillonner un signal à une fréquence de 24MHz.
Il peut donc être utilisé pour lire l'I2C, l'UART, mais aussi les SPI si la fréquence de l'horloge est inférieure à 6 MHz (théoriquement strictement inférieur à 12 MHz).

L'image du bas montre l'analyseur logique. Celle de droite montre son utilisation pour épier les échanges entre un Arduino et un périphérique.



(version longue)

Lire et déverminer un signal série - Pulseview

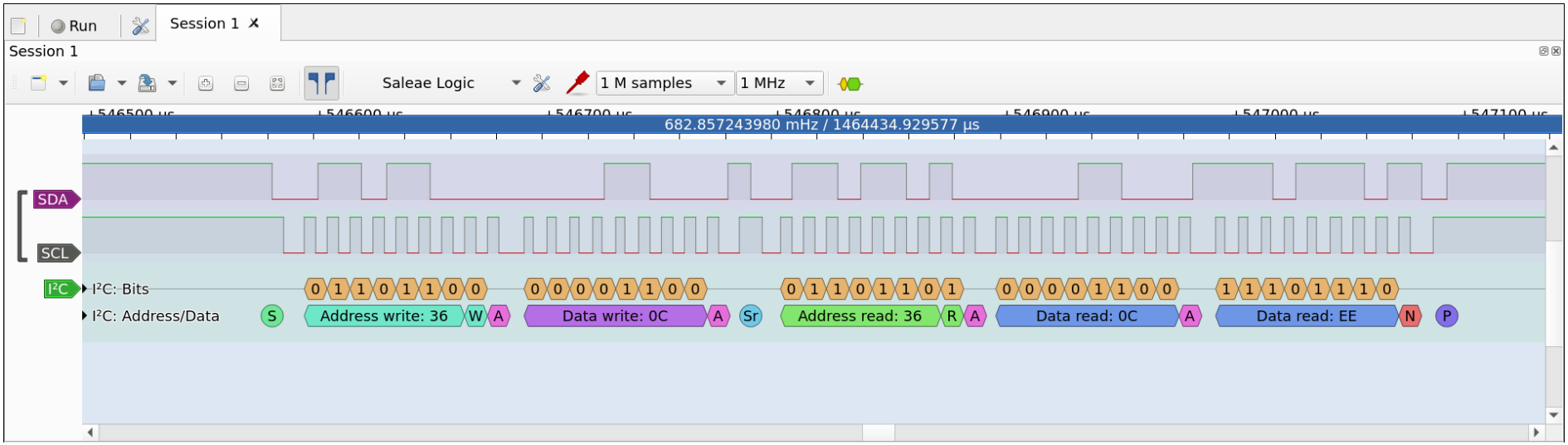
Pour utiliser l'analyseur logique de spakfun, vous pouvez utiliser le logiciel Pulseview - Sigrok.

Pour installer, puis exécuter pulseview, il vous suffit de taper, sous linux, les commandes ci-contre.

```
sudo apt install pulseview sigrok-firmware-fx2lafw
pulseview
```

Pour utiliser correctement pulseview, vous devez configurer la fréquence à laquelle vous aller échantilloner les données. D'après le théorème de Shanon, cette fréquence doit être strictement supérieure à 2 fois à la fréquence des signaux à mesurer. On utilise un facteur allant de 4 à 10.

Voici un exemple d'utilisation de pulseview pour décoder une comunication série I2C.



Comme la fréquence par défault de l'I2C de l'arduino est de 100 kHz, la fréquence d'échantillonnage choisie est de 1 MHz $> 2 \times 100$kHz.

(version longue)

Plan

(présent dans la version courte)

Les modules prêts à l'emploi

Souvent, il est plus rapide, fiable et moins onéreux d'utiliser des modules prêts à l'emploi. Voici 4 sites/références où trouver ces modules :

1. les composants du starter kit Arduino étendu ;
2. les capteurs des modules Adaftuits ;
3. les modules grove (communiquent en I2C) ;
4. les modules du site waveshare .

Accédé le 12/02/2024.

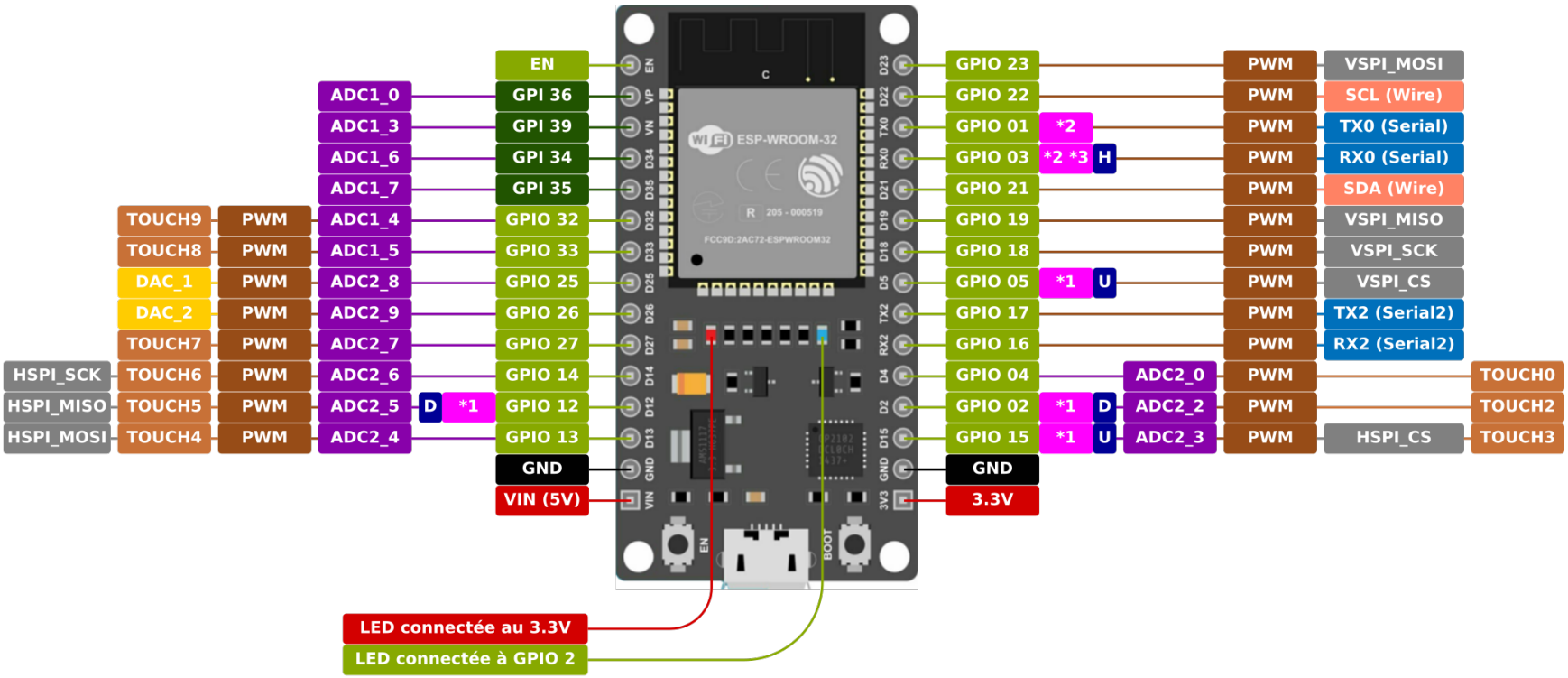
Plan

(version longue)

Plan

- Présentation de l'ESP32
- Communiquer par bluetooth
- Utiliser le wifi : créer une application web

La carte de développement DOIT-ESP32-devkit-v1



DOIT ESP32 devkit v1 (kit à 30 broches)

Cette carte n'est pas la carte de développement officielle de Espressif, mais c'est la plus répandue.

Son microncontrolleur est de la famille ESP32 dont voici quelques puces trouvables sur ces cartes : ESP32-WROOM-32, ESP32-WROOM-32U, ..., ESP32-WROVER-E, ... et ESP32-SOLO-1.

Courant par GPIO : 40 mA sortant et 28 mA entrant.
Courant maximal total sortant : 1.2A si alimenté par VIN, 500mA si alimenté par l'USB de l'ordinateur.
Tension minimale et maximale admissible des GPIO : -0.3V et 3.6V.
Niveaux logiques des GPIO : Vil = 0.825V. Vih = 2.47V, Vol = 0.33V, Voh = 2.64V.
Communication USB↔UART0 : drivers CP102 ------- Régulateur interne : AMS1117-3.3

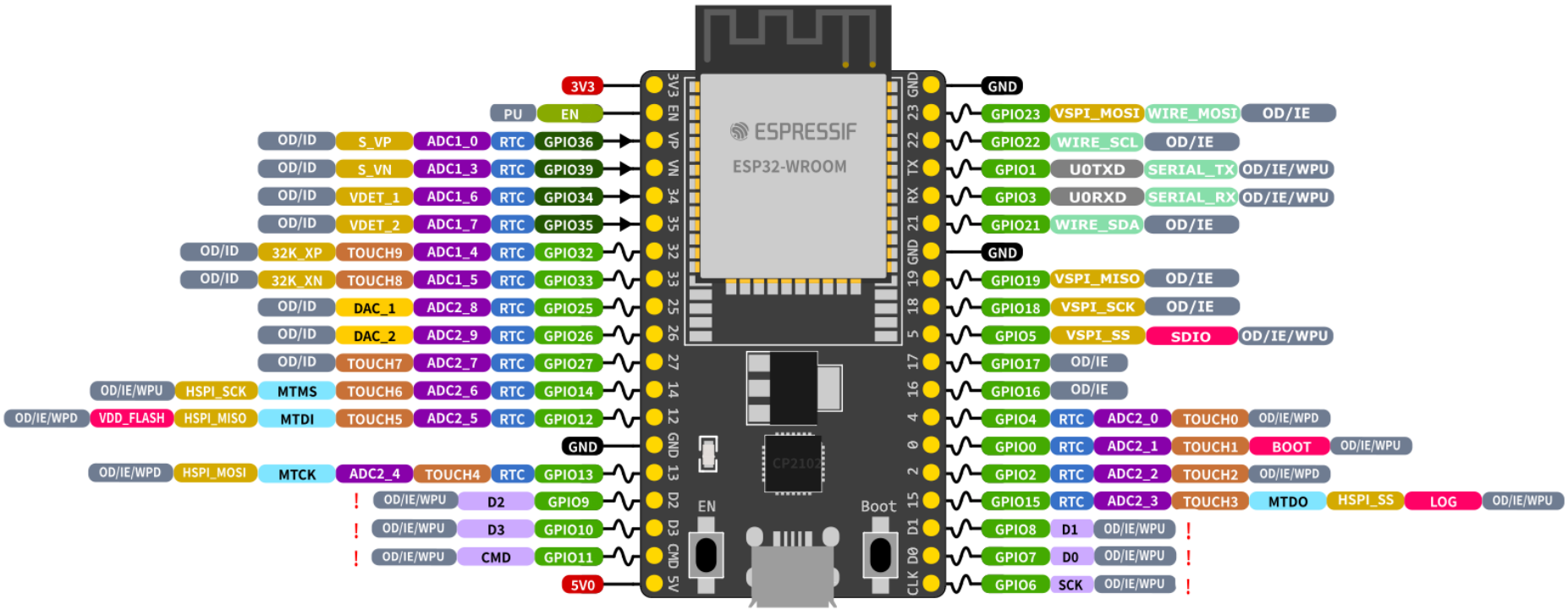
*1 : les strapping pin 2, 5, 12 et 15 servent à configurer la carte dans différents modes au demarrage. Au demarrage, des résistances de pull-up/down sont mises pour configurer la carte par default.
*2 : les broches 1 (TX0) et 3 (RX0) sont aussi utilisées lors de la communication en USB entre le PC et la carte. Elles sont utilisées pour mettre à jour le firmware et pour la comunication série.
*3 : la broche 3 est mise à HIGH au démarrage.

- entrée/sortie digitale
- entrée digitale
- PWM (16 configurables)
- 2 ADC: ADC1, ADC2
- 2 DAC: DAC1, DAC2
- 2 UART: UART0, UART2
- I2C
- 2 SPI : VSPI, HSPI
- Touches capacitives
- Alimentation (power)
- Masse (Ground)
- broche spéciale
- au boot : H=HIGH, L=LOW, U=pull-up, D=pull-down

(version longue)

La carte de développement ESP32 officielle (38 broches)

ESP32-DevKitC



ESP32 Specs

32-bit Xtensa® dual-core @240 MHz
Wi-Fi IEEE 802.11b/g/n 2.4 GHz
Bluetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet

PWM Capable Pin
GPIOX GPIO Input Only
GPIOX GPIO Input and Output
DAC_X Digital-to-Analog Converter
DEBUG JTAG for Debugging
FLASH External Flash Memory (SPI)
ADCX_CH Analog-to-Digital Converter
TOUCHX Touch Sensor Input Channel
OTHER Other Related Functions
SERIAL Serial for Debug/Programming
ARDUINO Arduino Related Functions
STRAP Strapping Pin Functions

RTC RTC Power Domain (VDD3P3_RTC)
GND Ground
PWD Power Rails (3V3 and 5V)
! Pin Shared with the Flash Memory Can't be used as regular GPIO

GPIO STATE
WPU: Weak Pull-up (Internal)
WPD: Weak Pull-down (Internal)
PU: Pull-up (External)
IE: Input Enable (After Reset)
ID: Input Disabled (After Reset)
OE: Output Enable (After Reset)
OD: Output Disabled (After Reset)

(version longue)

Les différentes familles de microcontrolleur de Espressif

Microcontrôleur	pinouts devkit	datasheet	freq. (MHz)	nb de coeurs	SRAM (KB)	ROM (KB)	nb GPIO	nb PWM	nb Touch	nb ADC 12-bit	nb DAC 8-bit	nb SPI	nb UART	nb I2C	nb I2S	USB OTG	Wi-fi 802.11 2.4 GHz	802.15.4	Bluetooth
ESP32	➡	➡	240	2	520	448	34	16	10	2	2	4	3	2	2		b/g/n		4.2 BR/ EDR LE
ESP32-C2 ESP8684	➡	➡	120	1	272	576	14	6	0	1	0	3	2	1	0		b/g/n		LE 5.3
ESP32-C3	➡	➡	160	1	400	384	22	6	0	2	0	3	2	1	1		b/g/n		LE 5 mesh
ESP32-C5	➡	?	240	1	384	320	29	6	0	1	0	2	2	1	0		ax 2.4GHz 5GHz	✓	LE 5
ESP32-C61	➡	?	160	1	320	256	25	6	0	1	0	2	3	1	1		ax		LE 5
ESP32-C6	➡	➡	160	1	512	320	30	6	0	1	0	1+2	1+1	1+1	1		b/g/n ax	✓	LE 5.3 mesh
ESP32-H2	➡	➡	96	1	320	128	19	?	0	1	0	1+2	2	2	1			✓	LE 5.3 mesh
ESP32-S2	➡	➡	240	2	320	128	43	8	14	2	2	4	2	2	1	✓	b/g/n		∅
ESP32-S3	➡	➡	240	2	512	384	45	8	14	2	0	2+2	3	2	2	✓	b/g/n		LE 5 mesh

Pour télécharger les documents techniques de ces microcontrolleurs, rendez-vous ici : Page de téléchargement des documentations techniques des ESP32.

(version longue)

Présentation de la carte ESP32

Les cartes de dev des Esp32 sont OpenHardware

Prix : 5 euros.

Microcontrôleur : Esp32 Wroom

Types gérés par la carte :

- Entier 32 bits : natifs ($[-2147483647, 2147483647]$ ou $[0, 4294967295]$);
- Réel 32 bits : natif;
- Entier 64 bits : émulés;
- Réel 64 bits : émulés.

Fréquence du microcontrôleur : 240 Mhz.

Table des types pour l'esp32

type	natif	entier				réel		signé	intervale
		nombre de bits							
		8	16	32	64	32	64		
char int8_t	✓	✓						✓	$[-127,127]$
unsigned char uint8_t	✓	✓							$[0,255]$
short int16_t	✓		✓					✓	$[-32767,32767]$
unsigned short uint16_t	✓		✓						$[0,65535]$
int int32_t				✓				✓	$[-2147483647,$ $2147483647]$
unsigned int uint32_t				✓					$[0,4294967295]$
long int int64_t					✓			✓	$[-2^{63}+1,2^{63}-1]$
unsigned long int uint64_t					✓				$[0,2^{64}-1]$
float	✓					✓		✓	
double							✓	✓	

Les types char, int, long int et long long int et leurs versions signées dépendent de l'architecture. Il vaut mieux utiliser leurs versions explicites : int8_t, int16_t, int32_t, int64_t. uint8_t, uint16_t, uint32_t et uint64_t.

(version longue)

Installer une carte de développement ESP32 sous Arduino IDE

Pour compiler et uploader le firmware, vous devez installer la chaîne de compilation de la carte ESP32.

Dans l'IDE Arduino, allez dans l'onglet :
Tools ⟶ Board ⟶ Board manager

Ensuite, dans la barre de recherche tapez : esp32.
Installez la bibliothèque "ESP32" de "Espressif System".

Vous pouvez développer, compiler et uploader votre premier programme sous ESP32. Pour une carte de développement utilisant le brochage de uPesy, vous devez choisir la carte "uPesy ESP32 Wroom DevKit".

Si vous n'arrivez pas à uploader un firmware dans une carte esp32, consultez la page 614 d'aide dédiée au téléversement des firmwares.

Les pièges à éviter avec une carte ESP32 !

- Les broches 6, 7, 8, 9, 10 et 11 ne peuvent pas être utilisées car elle sont déjà utilisées par la mémoire flash intégrée de votre carte.
- Il vaut mieux éviter d'utiliser les broches 0, 1, 2, 3, 5, 12 et 15. En effet :
  - la broche 3 est à l'état haut au demarrage. De plus, les broches 1 et 3 sont les broches séries Rx/Tx de l'UART 0 et sont utilisées pour téléverser le code. Ces broches doivent être flottantes pendant le téléversement. En dehors de la phase de démarrage et de téléversement, ces broches peuvent être utilisées. Utilisez plutôt les broches 16/17 – Rx2/Tx2 (UART2)
  - les broches 0, 2, 5, 12 et 15 sont des "strapping pin". Elles servent à configurer la carte dans différents modes. Il faut donc faire attention à ce que votre circuit électronique, au démarrage ou au reset, laisse ces broches flottantes pour configurer la carte en mode par default.
  - Si la broche 0 est mise à LOW au demarrage, la carte passe en mode "reprogramation" (flashing mode). Il faut donc laisser la broche 0 flottante ou à HIGH.
- Les broches 34, 35, 36 et 39 sont des entrées seulement. De plus elles **ne possèdent pas de pull-up internes**.

(version longue)

Configuration des ESP32 par les "Strapping Pin"

Si les broches 0, 2, 5, 12 et 15 sont flottantes au demarrage, la carte est en mode par default.
Voici les tables expliquant comment configurer la carte au démarrage :

Tension interne LDO (VDD_SDIO)

broche	default	3.3V	1.8V
MTDI GPIO 12	Pull-down	0	1

Mode de reprogrammation (booting mode)

broche	default	SPI Boot	Download Boot
GPIO 0	Pull-up	1	0
GPIO 2	Pull-down	/	0

Activation du log sur la sortie TX0 (UART0) durant le mode de reprogrammation

broche	default	RX0 Toggling (UART0)	RX0 Silent (UART0)
MTDO GPIO 15	Pull-up	1	0

Timing of SDIO Slave

broche	default	Falling-edge input Falling-edge ouput	Falling-edge input Rising-edge ouput	Rising-edge input Falling-edge ouput	Rising-edge input Rising-edge ouput
MTDO GPIO 15	Pull-up	0	0	1	1
GPIO 5	Pull-up	0	1	0	1

Consultez la section 2, "Boot Configurations" de document de spécification des ESP32 :
"ESP32 Series Datasheet Version 4.8" de Espressif.

(version longue)

Plan

Transformer l'ESP32 en un périphérique Bluetooth

Le bluetooth est natif dans les ESP32, il n'est pas nécessaire d'installer de bibliothèques supplémentaires. Cet exemple envoie Hello world.

```
#include "BluetoothSerial.h"

#if !defined(CONFIG_BT_SPP_ENABLED)
  #error Serial Bluetooth not enabled. It is only available for the ESP32 chip.
#endif

const char *pin = "1234"; // The pin secure number asked when pairing
String device_name = "ESP32-Bluetooth";
BluetoothSerial SerialBT;

void setup() {
  Serial.begin(115200);
  SerialBT.begin(device_name);
  Serial.printf("Name: %s\n, Mac Add.:", device_name.c_str(), SerialBT.getBtAddressString());
  // SerialBT.setPin(pin); Serial.println("Using PIN"); // Uncomment if you need to use Pin Number
}

void loop() {
  String msg("Hello !\n");
  SerialBT.write((const uint8_t*) msg.c_str(), msg.length());
  delay(1000);
}
```

code/esp32_bluetooth/esp32_bluetooth_hello_world.cpp

Transformer l'ESP32 en un périphérique Bluetooth

Ce programme renvoie tous les caractères envoyés à l'esp32 vio bluetooth.

```
#include "BluetoothSerial.h"

#if !defined(CONFIG_BT_SPP_ENABLED)
  #error Serial Bluetooth not enabled. It is only available for the ESP32 chip.
#endif

const char *pin = "1234"; // The pin secure number asked when pairing
String device_name = "ESP32-Bluetooth";
BluetoothSerial SerialBT;

void setup() {
  Serial.begin(115200);
  SerialBT.begin(device_name);
  Serial.printf("Name: %s\n, Mac Add.:", device_name.c_str(), SerialBT.getBtAddressString());
  // SerialBT.setPin(pin); Serial.println("Using PIN"); // Uncomment if you need to use Pin Number
}

void loop() {
  while( SerialBT.available() ){
    uint8_t incoming_Byte = SerialBT.read();
    SerialBT.write(incoming_Byte);
  }
}
```

code/esp32_bluetooth/esp32_bluetooth_copy.cpp

Communiquer en bluetooth avec l'ESP32 depuis un ordinateur sous Linux

Vous devez installer le logiciel blueman :

```
sudo apt-get install blueman
```

Ensuite démarrez le logiciel blueman-manager et connectez-vous au port série bluetooth de l'ESP32 qui devrait avoir le nom suivant : "ESP32-Bluetooth".

Reperez le nom du port série dans le logiciel blueman, il devrait avoir pour nom "rfcomm0". Connectez-vous sur le port série en ligne de commande, sur un autre terminal :

```
cu -l /dev/rfcomm0
```

Dans le programme cu, si vous tapez du texte, le texte est envoyée à l'ESP32.

Vous pouvez aussi envoyer du texte avec le logiciel cat en ligne de commande :

```
cat "This is some text" > /dev/rfcomm0
```

Pour envoyer des donnée sur le port série en python, consultez la page 113.

Pour Quitter le programme cu, tapez le caractère ~ suivi du caractère . ce qui done : "~.".

(version longue)

Transformer votre esp32 en souris/clavier/gamepad bluetooth.

Si vous voulez transformer la carte en périphérique bluetooth clavier, souris et gamepad, consultez les sites :

- ESP32 BLE Mouse library.
- ESP32 BLE Keyboard library.
- ESP32 BLE Gamepad library.
- ESP32 BLE Absolute Mouse library.

(sites consultés le 13/04/2024)

Attention ! Vos ESP32 deviendront des périphériques bluetooth UNIQUEMENT !

Vous ne pouvez pas transformer vos ESP32 en souris/clavier/gamepad USB. Seules les ESP32-S2 et ESP32-S3 peuvent le faire car ils ont la fonctionnalité USB OTG. Vous pouvez consulter la page **??** pour plus de détails.

Transformer votre esp32-S3 en souris/clavier/gamepad USB

Seules les catrtes EPS32-S2 et ESP32-S3 peuvent être transformer en souris/claviers/gamepad USB car elle implémentent l'USB OTG.

Les autres ESP ne peuvent pas devenir des souris/claviers/gamepads USB. Vous pouvez par contre les transformer en clavier/souris/gamepad bluetooth.

Les cartes des kits de developpements ESP32-S2-devkitC1 et ESP32-S3-devkitC1 possèdent 2 connecteurs USB :

- le port ayant pour nom COM qui sert uniquement à la comunication et la programmation série (comme pour toutes les autres cartes ESP).
- le port ayant pour nom USB qui sert pour connecter un autre périphérique USB ou pour transformer l'ESP32-S3 en périphérique USB comme les claviers, souris et gamepads.

La slide suivante donne un exemple de code pour transformer l'ESP32 en souris/clavier/gamepad que l'on peut connecter à un ordinateur.

Transformer votre esp32-S3 en souris/clavier/gamepad USB

Seules les EPS32-S2 et ESP32-S3 supportent les souris/claviers/gamepads USB !

code/usb_mouse_keyboard_gamepad_device/esp32_S3/main.cpp

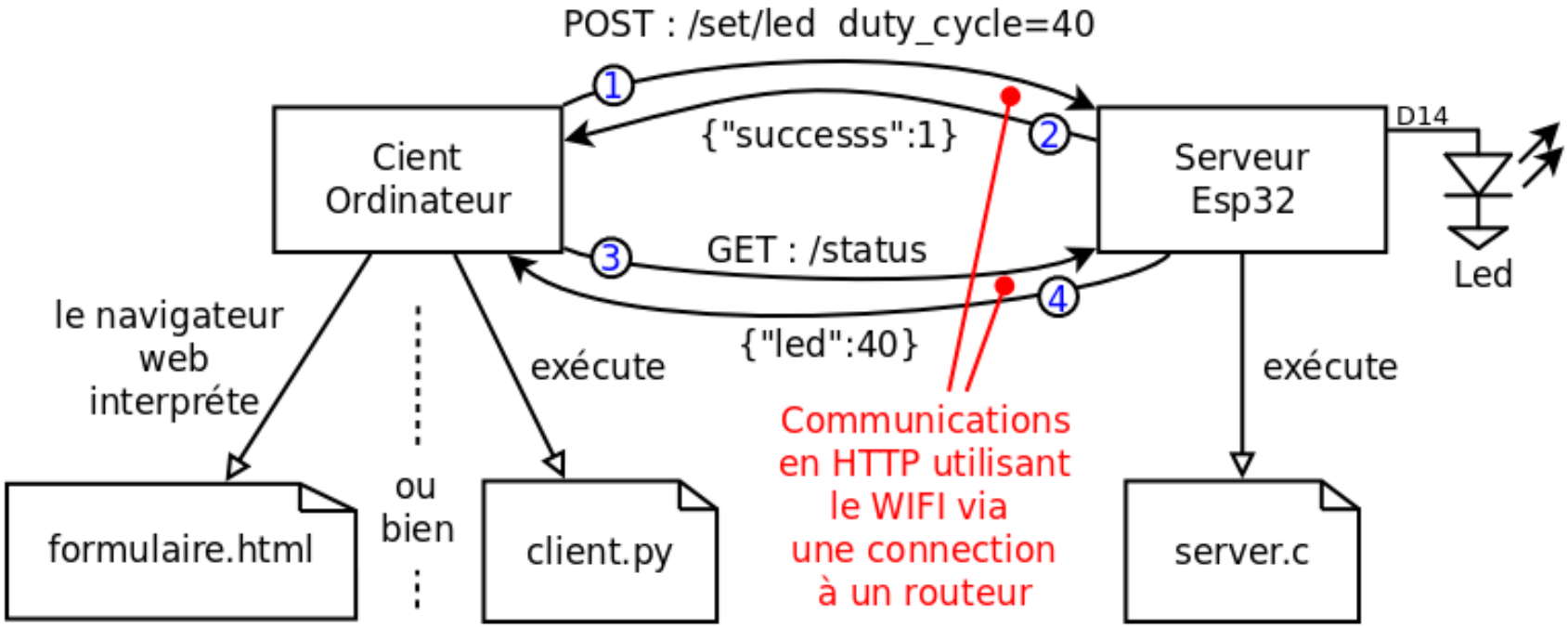
```
#include "USB.h"
#include "USBHIDMouse.h"
#include "USBHIDKeyboard.h"
#include "USBHIDGamepad.h"
USBHIDMouse mouse;
USBHIDKeyboard keyboard;
SBHIDGamepad gamepad;

void setup() {
  mouse.begin();
  keyboard.begin();
  gamepad.begin();
  USB.begin();
}
void loop() {
  int8_t delta_x = 5, delta_y = 0, delta_wheel = 0;

  // See https://github.com/espressif/arduino-esp32/blob/master/libraries/USB/src/USBHIDMouse.h
  mouse.move(delta_x, delta_y, delta_wheel);
  mouse.click(MOUSE_LEFT);

  // See https://github.com/espressif/arduino-esp32/blob/master/libraries/USB/src/USBHIDKeyboard.h
  keyboard.write('u');
  keyboard.press(KEY_LEFT_ARROW);
  keyboard.release(KEY_RIGHT_ARROW);

  // See https://github.com/espressif/arduino-esp32/blob/master/libraries/USB/src/USBHIDGamepad.h
  gamepad.leftStick(delta_x, delta_y);
  gamepad.pressButton(BUTTON_A);
  gamepad.releaseButton(BUTTON_A);
  delay(5000); // 5 seconds
}
```

(version longue)

Plan

- Présentation de l'ESP32
- Communiquer par bluetooth
- Utiliser le wifi : créer une application web

Communiquer en wifi avec l'esp32 par application web



Conventions du protocole HTTP :

- GET : demande au serveur une information à renvoyer;
- POST : demande au serveur d'effectuer une action qui le modifie.

Nous n'allons pas faire un site web. Cependant, dans un site Web, la même route (par exemple /set/led) est utilisée deux fois. Une fois en GET pour récupérer le formulaire à remplir dans un format html. Et une autre en POST pour soumettre le formulaire rempli.

Le code du serveur web: server.cpp - 1/4

```
#include <WiFi.h>
#include <WebServer.h>

const char* wifi_ssid = "TO_FILL";
const char* password = "TO_FILL";
WebServer server(80); // HTTP:port 80

void setup_wifi(){
  WiFi.begin(wifi_ssid, password);
  do {
    delay(1000); Serial.print(".");
  } while(WiFi.status()!=WL_CONNECTED);
  Serial.print(" IP Address: ");
  Serial.println(WiFi.localIP());
}

void setup_server(){
  server.on(
    "/status", HTTP_GET, get_status);
  server.on(
    "/set/led", HTTP_POST, post_led);
  server.onNotFound(handle_not_found);
  server.begin();
}
```

code/esp32_api_rest/server.cpp

La classe WiFi s'occupe de gérer la transmission des paquets par le protocole TCP/IP via le réseau WIFI. L'initialisation du réseau WiFi est réalisée avec la fonction setup_wifi(). On notera par W.X.Y.Z l'adresse IP assignée par le routeur.

WebServer s'occupe de transmettre les messages à l'aide du protocole HTTP. L'initialisation du serveur web est réalisée par la fonction setup_server().

On y déclare un service GET pour lire la led. Ce service sera accessible via l'URL : http://W.X.Y.Z/status il appellera la fonction get_status().

On y déclare un service POST pour modifier le rapport cyclique de la led. Ce service sera accessible via l'URL : http://W.X.Y.Z/set/led avec (on verra cela plus tard) pour donnée POST un entier duty_cycle. Il appellera la fonction get_status().

(version longue)

Le code du serveur web: server.cpp - 2/4

```
// Functionalties
const int LED_PIN = 14;
int duty_cycle = 0;

void set_led(int value){
  duty_cycle = max(0, min(100, value));
  int pwm = map(
    duty_cycle, 0, 100, 0, 255
  );
  analogWrite(LED_PIN, duty_cycle);
}

String status(){
  String res = "{";
    res += "\"led\" : ";
    res += String(duty_cycle).c_str();
  res += "}";
  return res;
}
```

code/esp32_api_rest/server.cpp

On déclare les fonctions permettant de modifier la pwm de la led et d'obtenir le status dans le format JSON.

Le code du serveur web: server.cpp - 3/4

```
void get_status() {
  server.send(200, "application/json",
    status());
}

void post_led() {
  if(server.hasArg("duty_cycle")){
    String txt = server.arg("duty_cycle");
    set_led(txt.toInt());
    server.send(200, "application/json",
      "{\"success\":1}");
  }else{
    server.send(400, "text/plain",
      "Invalid parameter");
  }
}

void handle_not_found(){
  server.send(
    404, "text/plain", "Not found");
}
```

code/esp32_api_rest/server.cpp

On implémente ici l'API Web.

Les fonctions get_status() et post_led() renvoient des données à l'aide de la fonction server.send(...).

Les codes 200, 400, 404 sont des codes du protocole HTTP, qui signifient :

- 200 : OK;
- 400 : Bad Request;
- 404 : Not Found.

Le type des données envoyés par le serveur est aussi normalisé. Nous en utilisons 2 : "application/json" et "text/plain".
Pour une page au format HTML, vous devez utiliser : "text/html" (cf. la page 501).

La fonction post_led() associée à la route /set/led récupère le paramètre duty_cycle envoyé avec la requête HTTP.

(version longue)

Le code du serveur web: server.cpp - 4/4

```
void setup() {
  analogWrite(LED_PIN, 0);
  Serial.begin(115200);

  setup_wifi();
  setup_server();
}

void loop() {
  server.handleClient();
}
```

code/esp32_api_rest/server.cpp

Vient enfin l'initialisation de la carte.

La mis à jour du serveur web est réalisée dans la boucle, introduisant une latence quand un client se connecte.

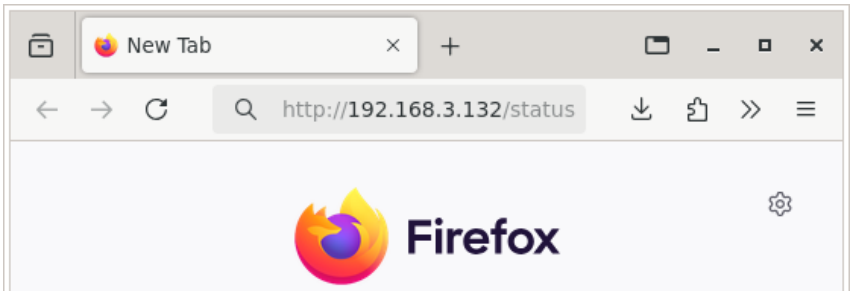
Si votre applications nécessite de réaliser une tâche à une fréquence fixe et rapide, vous devez utiliser des interruptions déclenchées périodiquement avec une priorité d'exécution plus importante que celle de la fonction loop().

Consultez la section 16 à la page 410 pour utiliser des interruptions.

Le client en utilisant un navigateur Web – 1/2

Pour accéder aux informations de type GET, il suffit d'utiliser votre navigateur WEB et de rentrer la route correspondant à la donnée recherchéé.

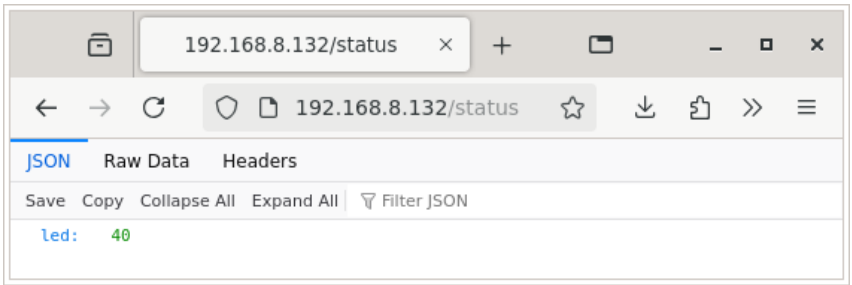
Par exemple, pour accéder à la routre */status* vous devez accéder, à partir de votre navigateur Web, à l'URL http://W.X.Y.Z/status où W.X.Y.Z est l'adresse IP donnée par votre carte ESP32 à l'initialiation. Attention: il est important de ne pas omettre le prefix "http://" à l'URL (et de na pas mettre "https://" qui est le protocole HTTP avec une couche de chiffrement TLS).



A gauche une copie d'écran de l'adresse à mettre dans le navigateur web firefox. En bas, le message envoyé par le port série de l'ESP32 à l'initialisation.

```
... IP Address: 192.168.8.132
```

On obtient, du serveur web, en retour de notre requête de type GET, la réponse suivante :



Il s'agit d'un texte au format JSON qui est le suivant :

```
{"led" : 40}
```

Le client en utilisant un navigateur Web – 2/2

```
<!DOCTYPE html><html lang="en">
<head>
  <title>LED Control</title>
  <meta charset="UTF-8">
</head>
<body>
  <h1>Form to manipulate the led</h1>
  <form
    action="http://W.X.Y.Z/set/led"
    method="POST"
  >
    <input name="duty_cycle" value=5>
    <button type="submit" value="Submit">
      Post the led value
    </button>
  </form>
</body>
</html>
```

code/esp32_api_rest/formulaire.html

Pour faire des requêtes POST avec un navigateur, il faut passer par un formulaire HTML. Créez un fichier *formulaire.html* contenant le code ci contre, Ouvrez le fichier avec votre navigateur web. Il ne reste plus qu'à cliquer pour faire la requête POST présente dans le formulaire.

Vous pouvez ajoutez dans la même page d'autres formulaires, avec des requêtes GET aussi, en laissant la ligne "input" si la requête GET nécessite des arguments.

Un exemple de formulaire plus complet, avec des sliders et des boutons est disponnible dans le code source du cours code.zip dans le fichier code/esp32_api_rest/formulaire_complet.html.

Vous pouvez aussi faire en sorte que votre serveur renvoie ce formulaire avec une requête GET. Ainsi, il n'est plus nécessaire de créer ce fichier pour faire les requêtes POST. Pour ce faire, consultez la page 501. Par contre, cela alourdit le code du microcontolleur.

(version longue)

Le client en ligne de commande

Vous pouvez communiquer avec votre serveur en utilisant le terminal en utilisant curl :

```
curl http://W.X.Y.Z/status
curl -d duty_cycle=60 http://W.X.Y.Z/set/led
```

ou bien en utilisant wget

```
wget -qO- http://W.X.Y.Z/status
wget -qO- --post-data duty_cycle=60 http://W.X.Y.Z/set/led
```

Le code du client web en python : client.py

Voici un exemple de client écrit en python :

```python
import requests

IP = "192.168.0.108" # Set the IP adress of the server

URL_set_led = f"http://{IP}/set/led"
PARAMS_set_led = {"duty_cycle": 50}

r = requests.post(url = URL_set_led, params= PARAMS_set_led)
if r.status_code != 200 :
    print(f"erreur - {r.status_code} : {r.content}")
    quit()
data = r.json()
print(data)

URL_get_status = f"http://{IP}/status"
PARAMS_get_status = None

r = requests.get(url = URL_get_status, params = PARAMS_get_status)
data = r.json()
if r.status_code != 200 :
    print(f"erreur - {r.status_code} : {r.content}")
    quit()
print(data)
```

code/esp32_api_rest/client.py

(version longue)

Ajouter un formulaire en format HTML

Pour ajouter un formulaire HTML à la route /set/led, modifiez la fonction setup_server() ainsi :

```
void setup_server(){
  server.on("/status", HTTP_GET, get_status);
  server.on("/set/led", HTTP_GET, get_led_form);
  server.on("/set/led", HTTP_POST, post_led);
  server.onNotFound(handle_not_found);
  server.begin();
}
```

et ajoutez la fonction get_led_form() suivante :

```
void get_led_form(){
  String res="<!DOCTYPE html><html lang=\"en\"><head>";
  res+="<title>LED Control</title><meta charset=\"UTF-8\"></head>\n";
  res+="<body><h1>Form to change the Led Power</h1>\n";
  res+="<form action=\"/set/led\" method=\"POST\">\n";
  // Remove type=\"range\" to have a text box instead of a slider.
  res+="<input name=\"duty_cycle\" type=\"range\" min=\"0\" ";
  res+=String("max=\"100\" value=\"") + String(duty_cycle).c_str();
  res+="\" class=\"slider\">\n";
  res+="<button type=\"submit\" value=\"Submit\">Submit</button>\n";
  res+="</form>\n";
  res+="</body></html>\n";
  server.send(200, "text/html", res);
}
```

code/esp32_api_rest/server_web.cpp

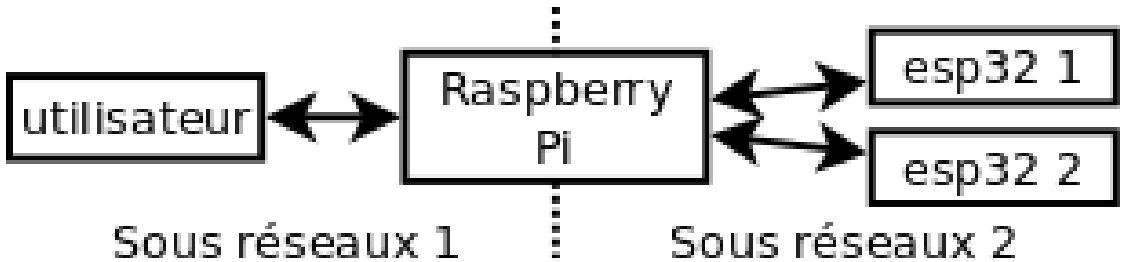
(version longue)

Ajouter un formulaire en format HTML - 2/2

Maintenant, en allant, avec un navigateur web, à l'adresse http://W.X.Y.Z/set/led, vous obtiendrez un formulaire.

Sachez cependant, qu'il faut stocker en mémoire toutes les pages Web correspondantes, et que générer la page HTML prend du temps. Tout cela n'est pas souhaitable dans un microcontrôleur.

Généralement, on utilise le micro-ordinateur Raspeberry Pi comme serveur Web dédié aux interactions Homme-Machine. Il s'agit d'un petit ordinateur, peu onéreux (20 à 50 euros) contenant un GNU/linux. On y installe un serveur web qui se charge de piloter par WiFi tous les microcontrôleur esp32 via leurs API web.

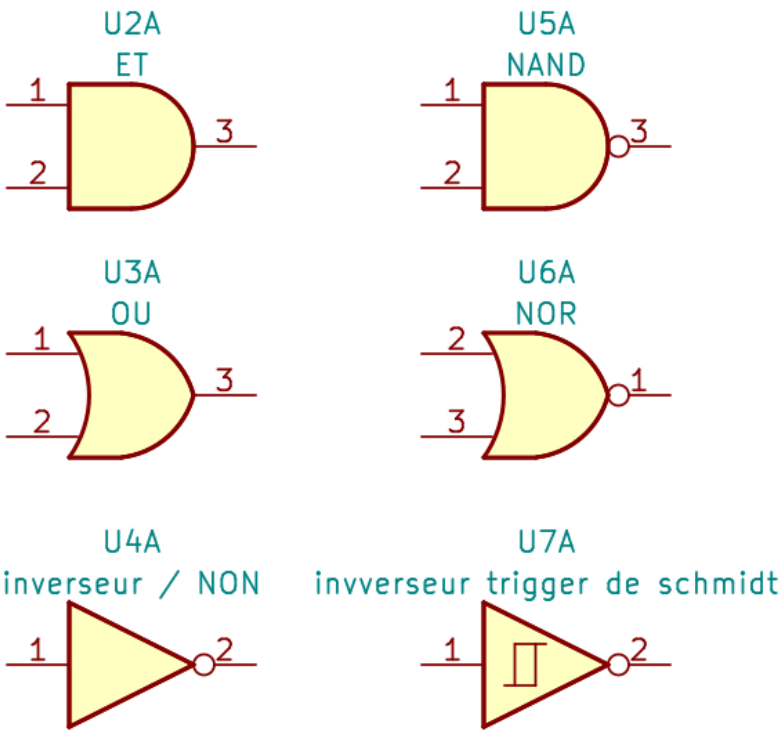


Cette dernière solution permet de cloisonner les différents réseaux et séparer celui des utilisateurs de celui des objets connectés. Il permet aussi de développer des Interfaces Homme-Machine bien plus évoluées.

Plan

(version longue)

Les portes logiques



OU	0	1
0	0	1
1	1	1

ET	0	1
0	0	0
1	0	1

	0	1
non	1	0

a nor b = non (a ou b)

a nand b = non (a et b)

La série des 74XXXX

Liste détaillée : List of 7400-series integrated circuits (Wikipedia).

	AND	NAND	OR	NOR	XOR	XNOR
Quad 2-input	74x08	74x00	74x32	74x02	74x86	74x7266
Triple 3-input	74x11	74x10	74x4075	74x27	/	/
Dual 4-input	74x21	74x20	74x4072	74x29	/	/

	Schmidt-trigger	
	Buffer	Inverter
Hex 1-input	74x7014	74x14

	série vers parallèle	parallèlle vers série
8 bits	74x595	74x165

Les niveaux logiques

CMOS = technologie avec des mosfet
TTL = technologie avec des transistors bipolaires

Tensions d'entrée

booléen	CMOS $3V3$	CMOS $5V$	TTL	Arduino Uno	esp32
0	$<0.8V$	$<1.5V$	$<0.8V$	$<1.5V$	$<0.825V$
1	$>2V$	$>3.5V$	$>2V$	$>3V$	$>2.48V$

Tensions de sortie

booléen	CMOS $3V3$	CMOS $5V$	TTL	Arduino Uno	esp32
0	$<0.2V$	$<0.1V$	$<0.4V$	$<0.8V$	$<0.33V$
1	$>3.1V$	$>4.9V$	$>2.4V$	$>4.1V$	$>2.64V$

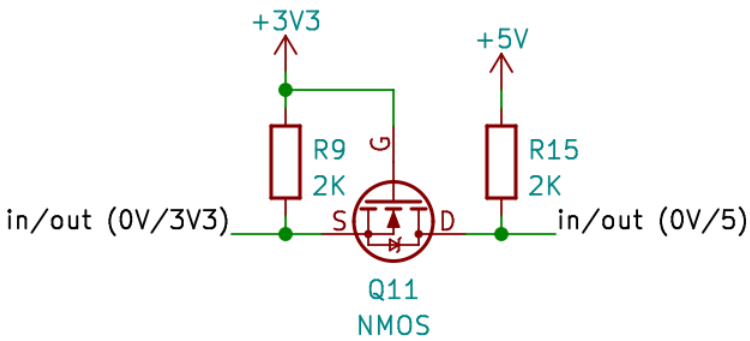
TTL : 74LSXX (compatible Arduino Uno et Esp32)
Input TTL + Output CMOS 5V : 74HCTXX
CMOS 5V : 74HCXX, arduino Uno
CMOS 3V3 : 74LVCXX, Esp32

(version longue)

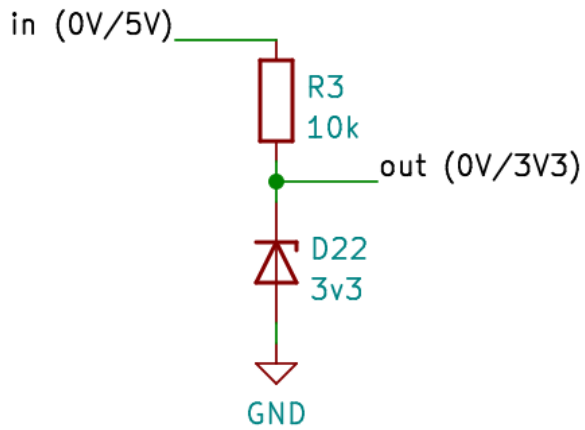
Conversion TTL (+5V) – CMOS (+3V3) – 1/2

Ajouter le TXS0108E – YF08E. Ajouter le 74AHCT125 et le 74HCT245

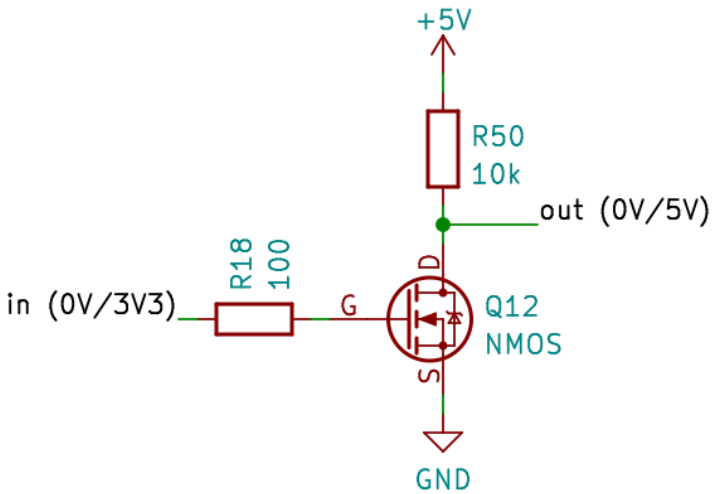
Unidirectionnel



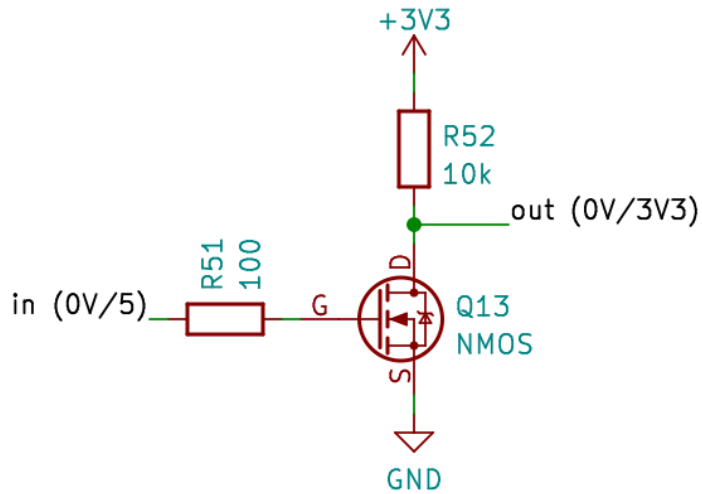
5V vers 3V3



3V3 vers 5V

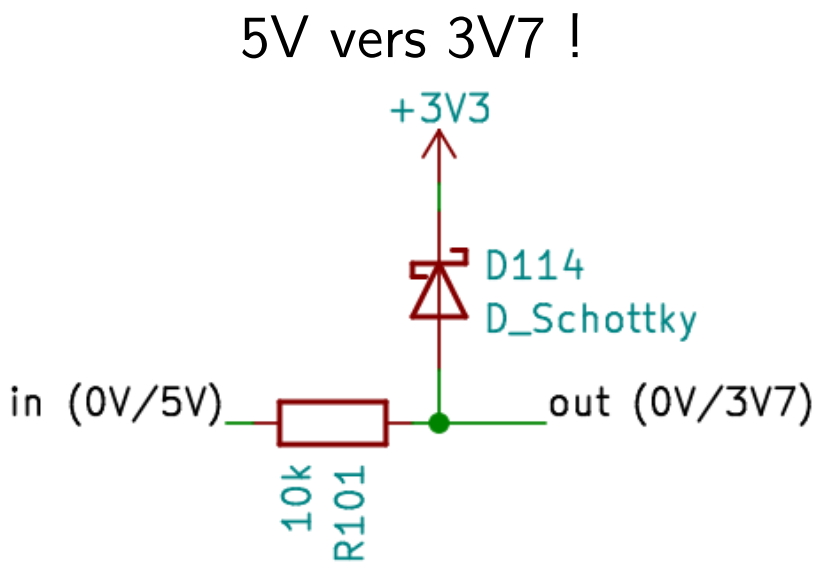


5V vers 3V3



(version longue)

Conversion TTL (+5V) – CMOS (+3V3) – 2/2

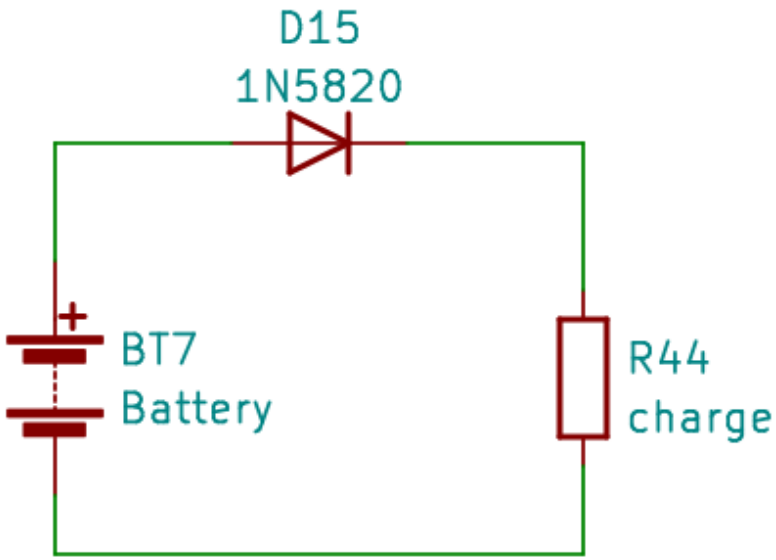


D'autres astuces peuvent être trouvées dans le document : 3V Tips'n Tricks , DS41285A, 2006, Microchip.

Plan

(présent dans la version courte)

Protéger contre une inversion de polarité



Ce circuit est tiré du livre : The Art of Electronics: The X-Chapters, de Paul Horowitz et Winfield Hill, 2020, page 396.

Dans ce montage, la diode empêche tout courant inverse.

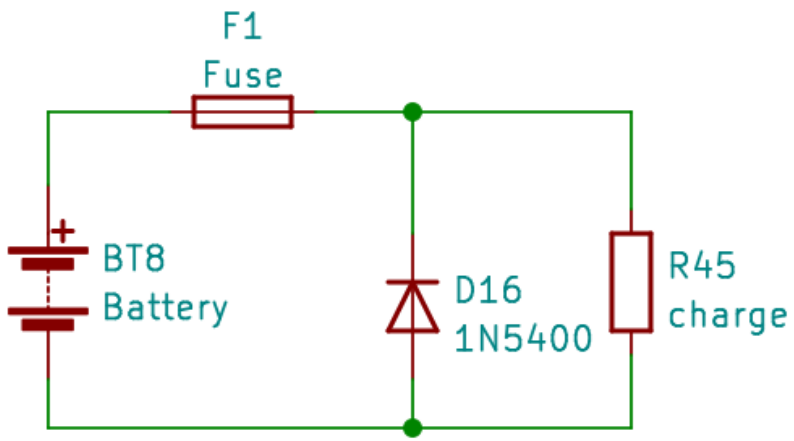
Si on connecte la batterie à l'envers, aucun courant ne circule et le circuit est sauvé.

Il faut prévoir une chute de tension au borne de la diode. Pour des diodes générales (famille 1N400X) cette chute varie entre 0.6V et 1V. Pour un diode Schottky (famille 1N151X et 1N152X), elle est d'environ 0.4V. Voir la table des diodes page 645

(présent dans la version courte)

Protéger contre une inversion de polarité - et une surtension

A utiliser avec une diode TVS quand on crée un circuit avec des moteurs qui produisent des surtensions.

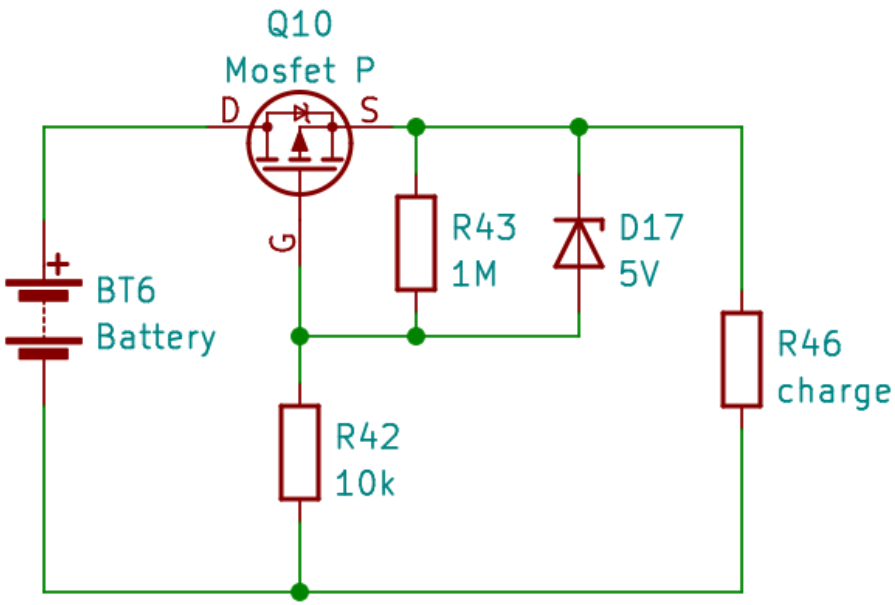


Ce circuit est tiré du livre : The Art of Electronics: The X-Chapters, de Paul Horowitz et Winfield Hill, 2020, page 396.

Si la batterie est connectée à l'envers, le fusible brûle ou se désarme, protégeant le circuit.

Pour protéger le circuit des surtensions, remplacez la diode 1N5400, par une diode TVS (1.5KEXXA) d'une tension légèrement supérieure à celle de la batterie. Cette diode deviendra passante dès que la tension dépasse celle de la batterie, protégeant le circuit. Si la diode TVS ne survit pas, elle se transforme en fil. Le fusible brûle et finit de protéger le circuit. La réparation consistera alors à remplacer/réarmer le fusible et remplacer la diode TVS. Consultez la table des diodes [page 645 pour choisir les diodes.

Protéger contre une inversion de polarité - suite



Ce circuit est tiré et adapté du livre : The Art of Electronics: The X-Chapters, de Paul Horowitz et Winfield Hill, 2020, page 396.

Il faut remarquer qu'ici le Mosfet P est monté à l'envers : la source est orientée vers la masse.

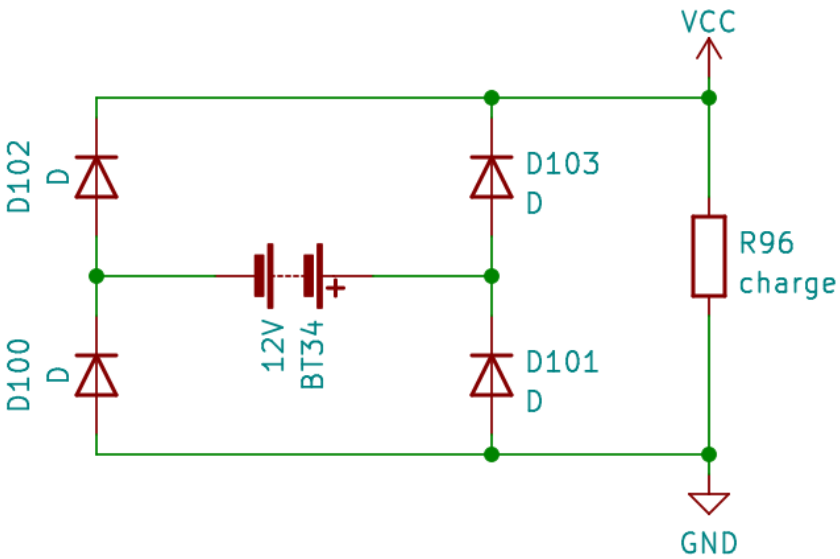
Si la batterie est montée correctement, la tension $V_g - V_s = V_{\text{zener}}$ (car la diode interne du mosfet laisse passer le courant) et le transistor est passant.

Si la batterie est montée à l'envers, grâce à la résistance R43, $V_g - V_s = 0$ et le transistor est ouvert.

La diode zener sert à protéger le mosfet, la tension $V_g - V_s$ ne doit pas dépasser une tension critique sous peine d'être détruite, la résistance R42 sert à limiter le courant passant dans la diode zener.

(version longue)

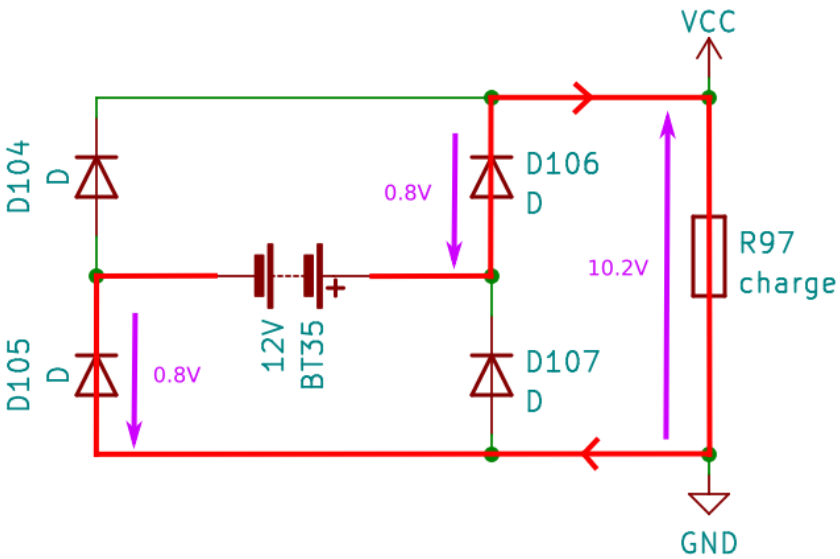
Permettre de connecter la batterie dans n'importe quel sens.



Le pont de diodes, constitué des 4 diodes, permet de redresser la tension afin qu'elle soit toujours positive au borne du circuit représenté par la résistance de charge R97.

Il faut prévoir une chute de tension qui dépend des diodes choisies. Pour des diodes générales (famille 1N400X) cette tension varie entre 0.6V et 0.8V par diode. Pour les diode Schottky (famille 1N151X), cette chute est d'environ 0.4V par diode.

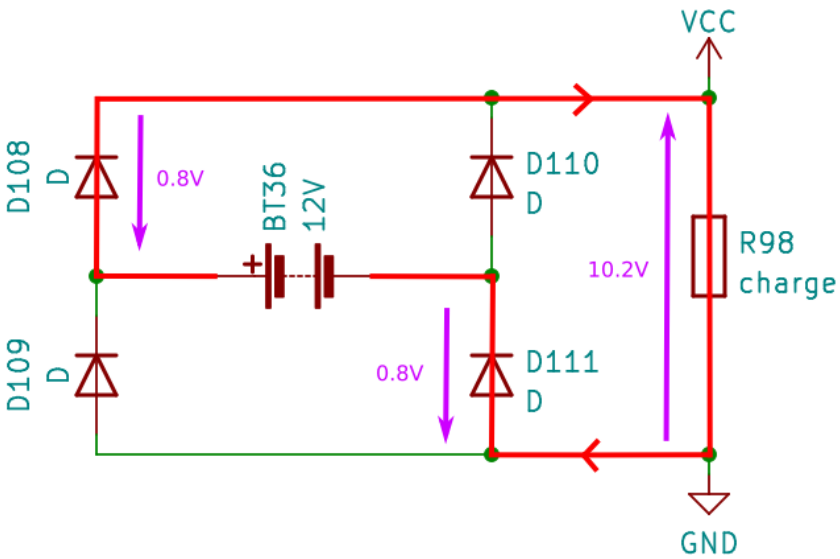
Permettre de connecter la batterie dans n'importe quel sens.



Le pont de diodes, constitué des 4 diodes, permet de redresser la tension afin qu'elle soit toujours positive au borne du circuit représenté par la résistance de charge R97.

Il faut prévoir une chute de tension qui dépend des diodes choisies. Pour des diodes générales (famille 1N400X) cette tension varie entre 0.6V et 0.8V par diode. Pour les diode Schottky (famille 1N151X), cette chute est d'environ 0.4V par diode.

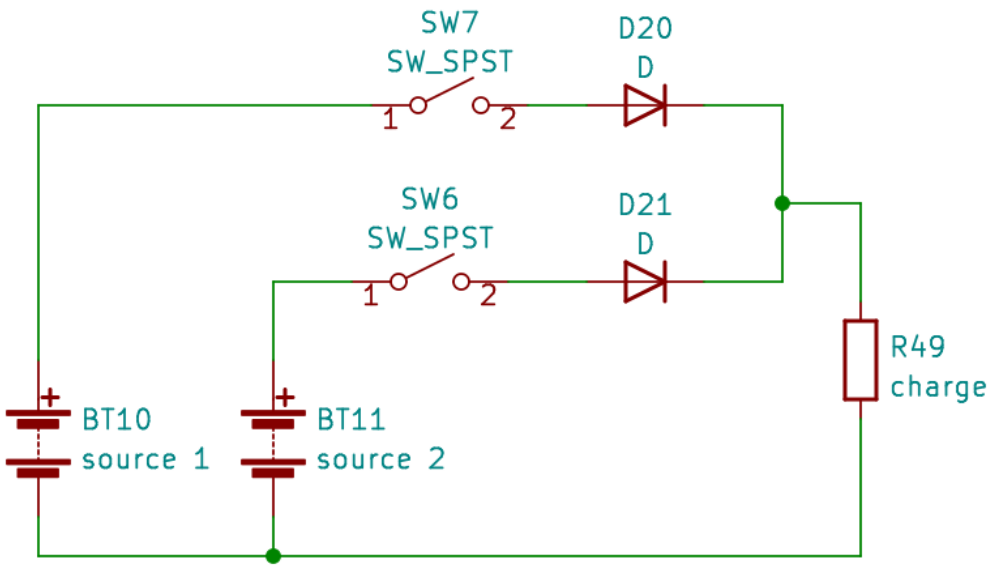
Permettre de connecter la batterie dans n'importe quel sens.



Le pont de diodes, constitué des 4 diodes, permet de redresser la tension afin qu'elle soit toujours positive au borne du circuit représenté par la résistance de charge R97.

Il faut prévoir une chute de tension qui dépend des diodes choisies. Pour des diodes générales (famille 1N400X) cette tension varie entre 0.6V et 0.8V par diode. Pour les diode Schottky (famille 1N151X), cette chute est d'environ 0.4V par diode.

Protéger ses sources d'alimentations



Dans ce montage, les diodes empêchent tout courant de revenir dans la batterie.

Si un montage accepte la présence de plusieurs sources (pas exemple, USB 5V et batterie 12V), ces diodes les empêche d'être en court circuit.

On protège les différentes sources d'alimentation ainsi que l'inversion de polarité d'une batterie.

Les mesures de protections contre les surtensions - protéger le port USB

Une mesure permettant d'empêcher une surtension sur le port USB est l'isolation galvanique.

Par exemple, si le driver d'un moteur n'est pas isolé galvaniquement, une surtention peut endommager le port USB. Pour éviter ce problème, à defaut d'isoler galvaniquement le driver, vous pouvez isoler le port USB du reste du circuit avec une clef USB utilisant le circuit ADMU3160. On trouve ces clefs sous le nom de "USB Digital Isolator".

(version longue)

Plan

(version longue)

Mise en garde !

Ce document est en cours d'écriture et de relecture !

Bien que l'auteur essaye d'être le plus rigoureux possible et essaye de justifier les schémas et affirmations donnés, ses domaines de spécialité sont les mathématiques et l'informatique.

Vous devez donc être très critique en lisant ce document. Vous devez comprendre le fonctionnement des circuits et multiplier et croiser les sources d'informations.

Ce document est conçu pour aider des ingénieurs à développer et concevoir de nouveaux objets. Les circuits doivent donc être validés par le lecteur avant d'être construit et utilisés. Ils doivent être testés intensivement avant d'être diffusés au grand public. L'auteur ne peut pas être tenu pour responsable des dégâts occasionnés par les circuits conçus, produits ou reproduits par le lecteur.

Si vous rencontrez des erreurs à la lecture de ce document, n'hésitez pas à contacter l'auteur pour qu'il le corrige et l'améliore.

(version longue)

Plan

- Présentation des piles et batteries
- Le cas des batteries Lithium
- La gestion de la charge et la décharge des batteries : les BMS

Piles et batteries

La capactié C_a des batteries correspond à la quantité totale d'électrons qu'elles peuvent fournir. Elle se mesure en ampère heure : A.h (ampères × heures).

Comme la tension est "relativement" constante, cette quantité est proportionnele à la quantité d'energie que contient la batterie et vaut approximativement : $E_{\text{batterie}} \approx C_a \times V_{\text{batt,moyen}}$.

Le temps d'autonomie (en heure) d'un circuit T_{aut} se calcule à partir de sa consommation I_{conso} et de la capacité totale C_a de la batterire par la formule :

$$T_{\text{aut}} = \frac{C_a}{I_{\text{conso}}}.$$

Piles ne nécessitant pas de protection (sans danger)

	désignation	matiere	assemblage nb de cellule	tension max	tension typique	tension déchargé	capacité (mAh)	rechargeable	prix (euro)
pile 1.5 V	LR8D425 AAAA	Alcaline ZNMnO2	1 × 1.5	1.5	1.3	0.9	250-650		
	LR03 AAA	Alcaline	1 × 1.5	1.5	1.1-1.3	0.9	350-1200		
	LR6 – AA	Alcaline	1 × 1.5	1.5	1.1-1.3	0.9	1200-2870		
	LR14 – C	Alcaline	1 × 1.5	1.5	1.1-1.3	0.9	5000-8200		
	LR20 – D	Alcaline	1 × 1.5	1.5	1.1-1.3	0.9	9000-16000		
	HR03 AAA	NiMH	1 × 1.2	1.4	1.2	1.0	800-1000	✓	
	HR6 – AA	NiMH	1 × 1.2	1.4	1.2	1.0	1900-2300	✓	
	HR14 – C	NiMH	1 × 1.2	1.4	1.2	1.0	2500-3000	✓	
	HR20 – D	NiMH	1 × 1.2	1.4	1.2	1.0	2500-3000	✓	
pile 4.5V	3LR12	Alcaline	3 × 1.5	4.5	3.4-3.9	2.7	1900-2500		
	3R12	carbone zinc	3 × 1.5			2.7	1000-2000		
pile 9V	6LR61	Alcaline	6 × 1.5	9.5	7-8	5.4	260-600		
	HR22	NIMH	6 × 1.2	9.5	8.4	7.0	175	✓	
	6F22	Carbon Zinc		9.5	6-8	5.4	270-370		

Ces chiffres sont tirés d'un mix des spécifications des piles VARTA et ENERGIZER.

Un exemple de dimensionnement de batteries

On souhaite alimenter une circut consommant 100 mA avec un régulateur 5V pendant 8 heures. On cherche une batterie qui pourrait convenir.

Le régulateur néecessite au moins 7 V pour fonctionner et la capacité de la batterie doit être de $100\text{mA} \times 8\text{h} = 800$ mA.h. D'après la table de la page 522, 7 piles AA – LR6 feront l'affaire.

Pour passer à 6/7 piles AA/AAA, il faut consulter les courbes de décharge des documents techniques.

Avec 6 piles LR03 – AAA, la tension risque de descendre au dessous de $1.2V \approx 7\text{V}/6$ piles. Et à 7 piles, l'autonomie pour une tension supérieur à 1.0V est de 7-8 heures.

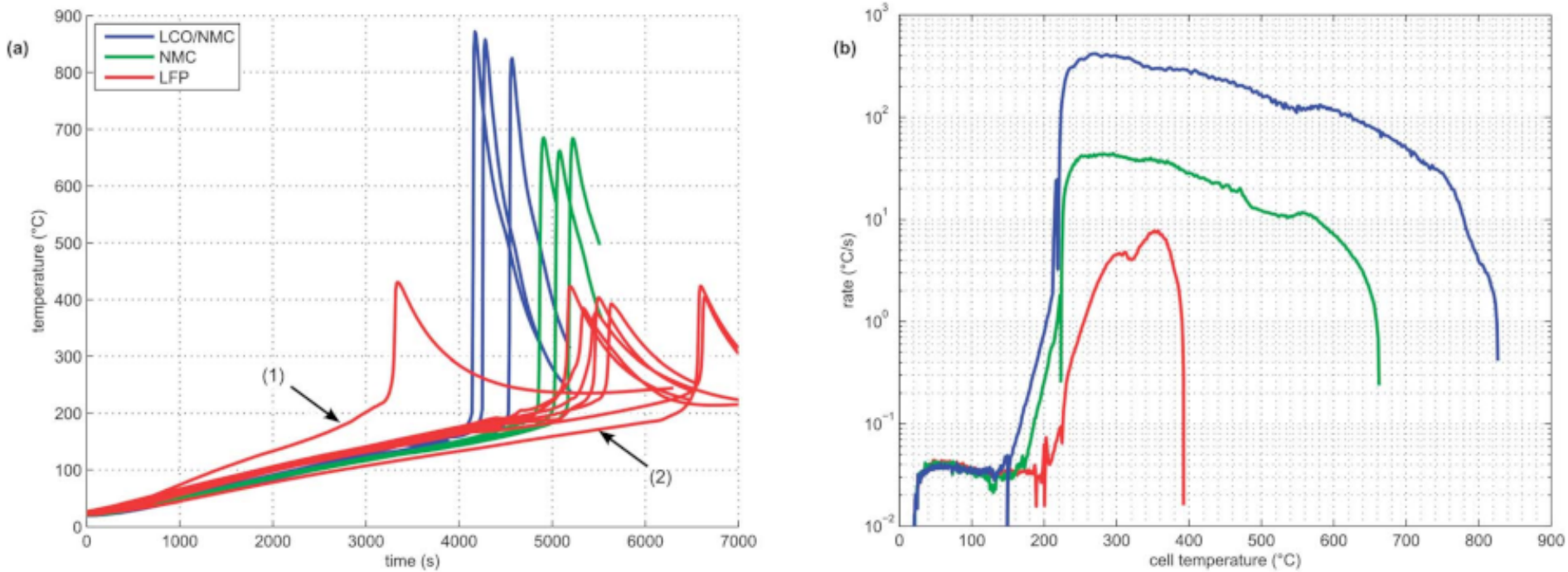
Avec 6 piles AA alcaline LR6, la tension reste au dessus de 1.2 V après 14 heures de foctionnement.

On choisira donc 6 piles LR6 – AA.

Plan

Dangerosité des piles au lithium – 1/2

Le terme “batteries lithium” regroupe les batteries contenant du lithium.
Ces batteries doivent être utilisées avec précaution ! Lorsqu'elles se mettent trop à chauffer, un emballement thermique a lieu. Cet emballement est accompagné d'un dégazage intense de carburant et comburant à la fois. Cela provoque des incendies et des explosions. La figure ci dessous montre que l'emballement thermique peut atteindre des températures allant de 400 à 900 degrés celsus, pour des accumulateurs de 40 grammes environ !



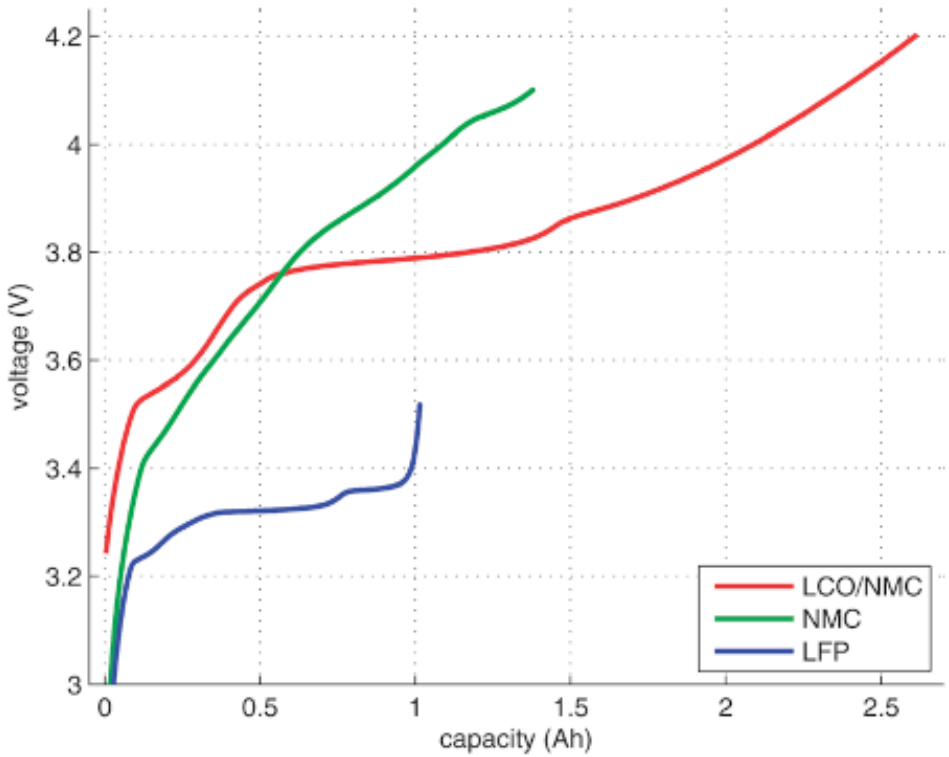
LFP : Lithium iron phosphate – LiFePO4, NMC : nickel, manganese and cobalt, LCO : lithium cobalt oxide

Source: Thermal-runaway experiments on consumer Li-ionbatteries with metal-oxide and olivin-type cathodes, Andrey W. Golubkov et al., 2014

(version longue)

Dangerosité des piles au lithium – 2/2

Des incendies peuvent survenir en cas de décharge profonde, de surcharge, ou de court-circuit de la batterie. Ces incendies sont aussi provoquée par des chocs mécaniques.
Pendant toute la durée de vie de la batterie, les tensions maximales et minimales à surveiller sont donnés à la figure suivantes :



LFP : Lithium iron phosphate – LiFePO4, NMC : nickel, manganese and cobalt, LCO : lithium cobalt oxide

Source: Thermal-runaway experiments on consumer Li-ionbatteries with metal-oxide and olivin-type cathodes, Andrey W. Golubkov et al., 2014

(version longue)

Batteries Lithium - recommandations.

Voici quelques recommandations et précautions d'usages concernant l'utilisation des batteries lithium :

- attendre que la batterie refroidisse avant de la recharger/décharger;
- ne pas mettre une batterie en décharge profonde ou en sous-tension;
- ne pas mettre la batterie en surtension;
- ne jamais mettre la batterie en court-circuit;
- ne pas jetter la batterie à la poubelle et suivre impérativement la procédure de recyclage;
- ne pas l'exposer au feu et à la chaleur;
- ne pas effectuer de brasure à l'étain sur ses broches, il faut toujours souder par points;
- ne jamais essayer de l'ouvrir;
- ne pas la percer, ni l'écraser;
- ne pas mettre les batteries en parallèle (utilisez plutôt le montage page 516 en dimensionnant correctement les diodes).

Piles au lithium - 1 seule cellule : 1S

La liste de piles et batteries qui suivent sont au lithium. Ces piles sont dangereuses . Lisez la page 525 avant de choisir ces piles.

Les piles les plus sures sont les piles avec protections intégrés L91 et L533 et peuvent être utilisées telle quel (sans ajout de circuit de protection supplémentaire). Ensuite viennent les piles LiFePo4 car leurs températures d'emballement thermique sont les plus hautes (cf. figure page 525) mais il faut ajouter un circuit de protection présenté à la page 532.

Si vous mettez plusieurs cellules en séries, il faut utiliser des BMS spécifiques pour piloter et vérifier chaque cellule indépendamant les une des autres lors de la charge et la décharge !

désignation		matiere	assembl. nb de cellule	tension max	tension typique	tension déchargé	capacité (mAh)	recharg-eable	protection integré	prix (euro)
pile 1.5 V	L91 AA	LiFeS2	1 × 1.5	1.8	1.4-1.5	1.2	3500		✓	
pile 3.2V	LiFePo4 18650	LeFePo4	1 × 3.2	3.65	3.2	2.6	1400-1500	✓		
pile 3.2V	LiFePo4 26650	LiFePo4	1 × 3.2	3.65	3.2	2.6	1900-2300	✓		
pile 3.7V	Li-ion 18650	Li-ion	1 × 3.7	4.2	3.4-3.7	3.0	2350-2600	✓		
pile 3.7V	LiPo	LiPo	1 × 3.7	4.2		3.0	1000-2000*1	✓	*2	
pile 9V	L522	LiMnO2	1 × 9	9.0	7-8.2	5.4	750–1200		✓	

*1 Les tailles des cellules Lithium Polymer ne sont pas normalisées. Leurs capacité dépendent des constructeurs.
*2 Selon les constructeurs, ces cellules peuvent contenir des protections intégrés, mais jamais de circuits de charges et décharges.

(version longue)

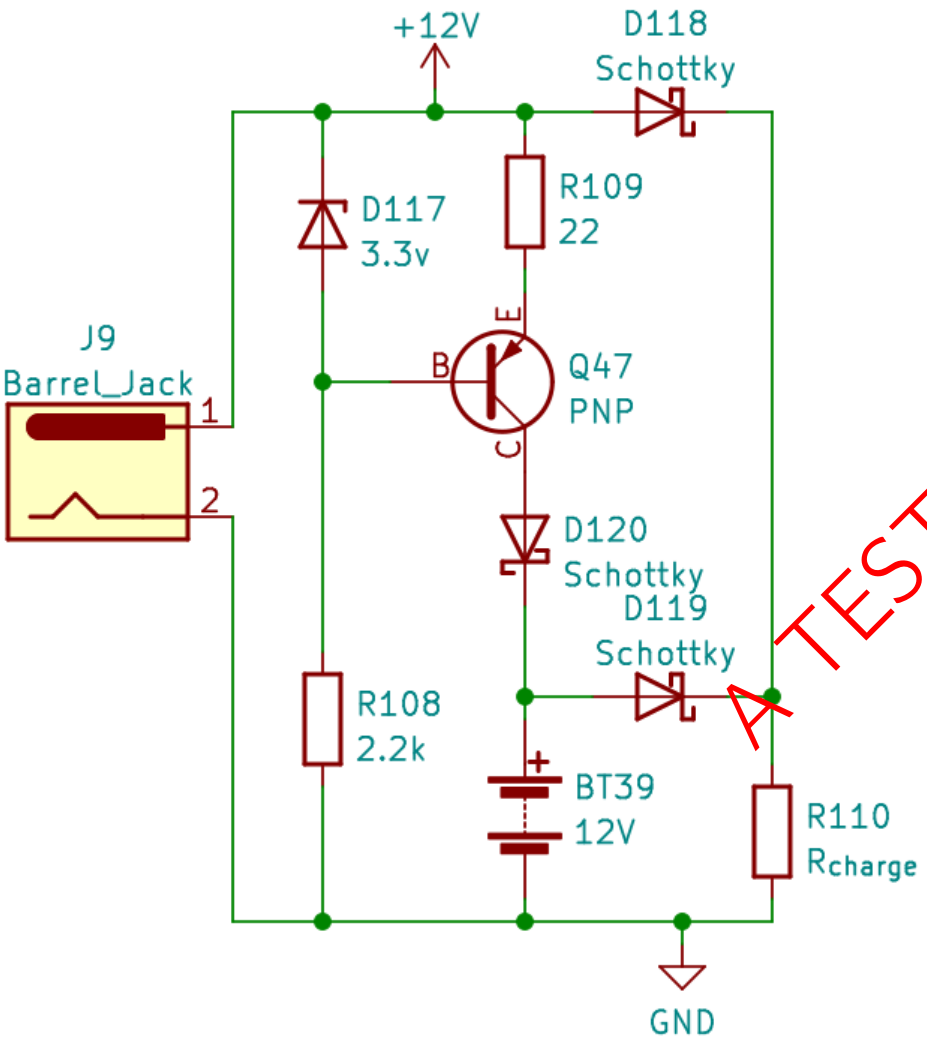
Piles au lithium - assembl. de plusieurs cellules : 2S,3S,$\cdots$

A FAIRE : Finir la table.

Plan

- Présentation des piles et batteries
- Le cas des batteries Lithium
- La gestion de la charge et la décharge des batteries : les BMS

Un système de gestion des batterie pour piles NiMH



Quand la prise Jack est alimentée, la diode zener assure une polarisation du transistor PNP. La tension au borne de la résistance R109 est de $3.3V - 0.6V = 2.4V$ et le transistor devient un générateur de courant de $110mA = 2.4V/22\Omega$ qui charge la batterie NiMH . De surcroit, le circuit (R_{charge}) est aliménté via la diode D118.

Quand la prise Jack est débranchée, la batterie prend le relai et alimente le circuit via la diode D119. La diode D120 empiêche la fuite de courant via le transistor Q47 si le circuit R110 est éteint.

N'utilisez ce circuit qu'avec les batteries NiMH !

Ce circuit est une adaptation du chargeur proposé par le site sonelec-musiqe.com.

Cette modification a été testée uniquement en simulation : simulation du chargeur.

(version longue)

Les systèmes de gestion des batteries (BMS) des piles lithium à 1 cellule

A FAIRE : Donner une référence de circuit de protection pour une seule cellule (BMS1S).
A FAIRE : Présenter l'Alarme sonore de basse tension, testeur d'indicateur de tension Lipo A FAIRE : Présenter le TP4046 pour les batteries Lithium-Ion
A FAIRE : Présenter le TP5400 qui gère les Lithium-Ion et LiFePO4
A FAIRE : Présenter le TP5000 qui a une sortie 5V en plus !

Les systèmes de gestion des batteries (BMS) des piles lithium à plusieurs cellules

A FAIRE :

Plan

(présent dans la version courte)

Les outils pour souder

A FAIRE :

(présent dans la version courte)

Apprendre à bien souder

A FAIRE :

(présent dans la version courte)

Raccorder des fils

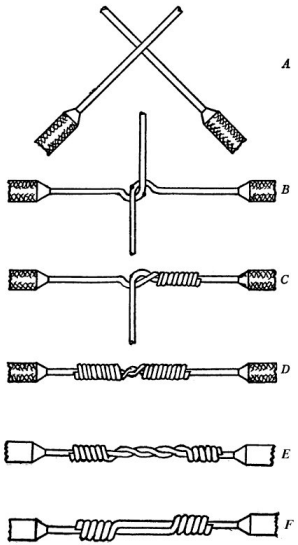
A FAIRE :

Références :

– Workmanship standard for crimping, interconnecting cables, harnesses, and wiring, NASA thechnical standard, NASA-STD 8739.4A, 2016.

– High reliability assembly for surface mount and through hole connections, European cooperation dor space standardization, ECSS-Q-ST-70-61C, 8 avril 2022

L'épissure de Western Union



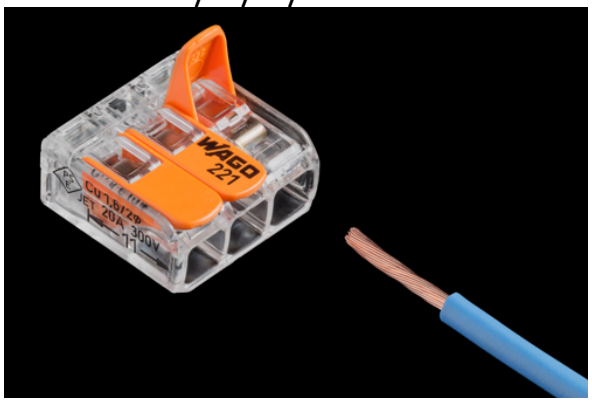
*1

On soude ensuite les 2 fils !

Testé par la NASA et jugé très robuste !

Interdit pour les cables du secteur.

Utiliser un connecteur Wago 2/3/4/... fils



Ici 3 fils

Adapté pour les cables d'alimentation du secteur

A FAIRE : Présenter les manchons de raccordements étanches à souder et à sertir.

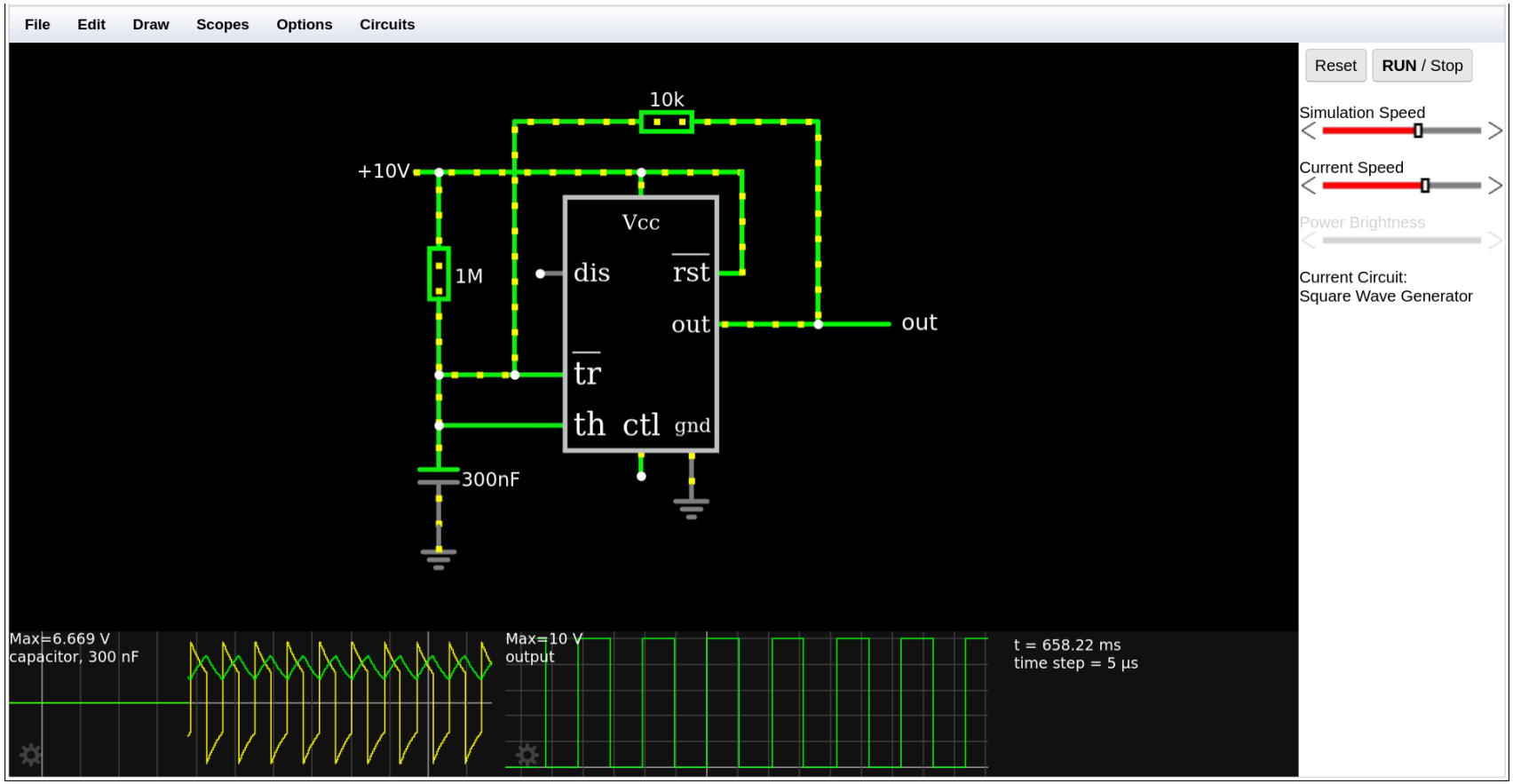
A FAIRE : Présenter d'autres méthodes de raccordement de l'ECSS ou de la NASA.

*1: tiré du livre Practical Electric Wiring, John M. Sharp, D. Appleton and company New York London, 1916.

(présent dans la version courte)

Concevoir un circuit avec un simulateur

Le Simulateur, sous licence libre, de circuit en ligne de Paul Falstad et Iain Sharp : www.falstad.com/circuit.



(version longue)

Plan

(version longue)

Plan

- Amplificateurs operationnels (AOP)

Référence bibliographique sur les AOP

Livres :

1. Électronique, Fondements et applications, J.-P. Pérez, C. Lagoute, J.-Y. Fourniols et S. Bouhours, 2ieme Édition, 2012, Dunod – Chapitre 8 : Amplificateur opérationnel : montages de base – Chapitre 9 : Amplificateur operationnel : compléments.
2. The Art of Electronics, Paul Horowitz et Winfield Hill, 3th Edition, 2015, Cambridge Press – Chapitre 4 : Operationnal Amplifiers.

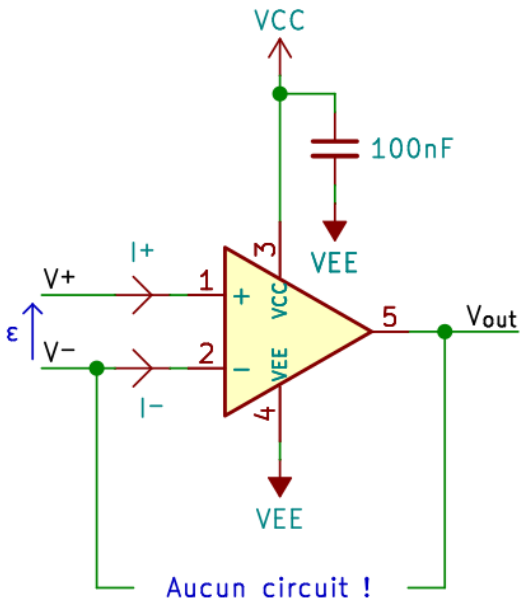
Notes techniques d'application :

3. AN-20 An Applications Guide for Op Amps, Texas Instrument, SNOA621C, February 1969, Revised May 2013.
4. AN-31 amplifier circuit collection, Texas Instrument, SNLA140D,march 2019/revised october 2020.
5. Analog Engineer's Circuit Cookbook: Amplifiers, Tim Green, Pete Semig and Collin Wells, Texas Instrument, SLYY137, match 2019, Second edition,

Modèle de l'AOP idéal – Mode non linéaire – 1/2

L'amplificateur opérationnel possède deux modes de fonctionnement différents.

Quand il n'y a pas de circuit entre V^- et V_{out} l'AOP est en boucle ouverte. On dit qu'il est en mode non linéaire ou en mode saturée.



Le modèle mathématique de l'AOP en boucle ouverte est alors le suivant :

$$I^+ = 0A, \quad I^- = 0A,$$

et

$$V_{out} = \begin{cases} V_{sat,max} & \text{si } \epsilon > 0; \\ V_{sat,min} & \text{si } \epsilon < 0; \end{cases}$$

avec $\epsilon = V^+ - V^-$.

Les tensions $V_{sat,min}$ et $V_{sat,max}$ sont les tensions de saturation, données par le constructeur, que les AOPs ne peuvent pas dépasser.

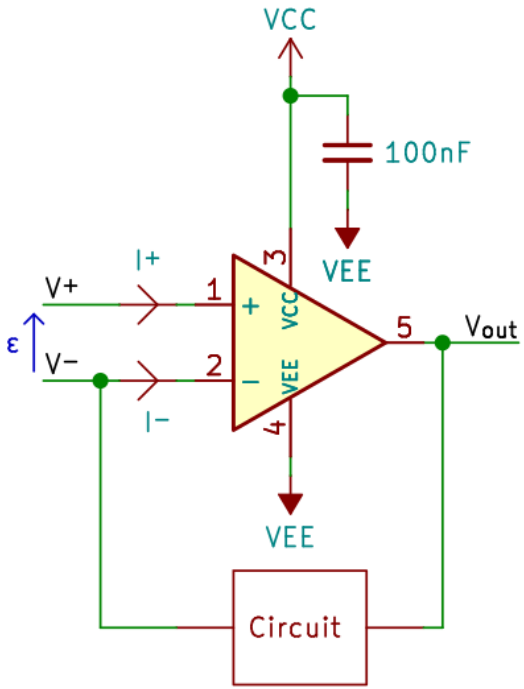
Par exemple, pour les LM358 et LM324, $V_{sat,min} = V_{EE} + 0.1V$ et $V_{sat,max} = V_{CC} - 1.3V$.

Un amplificateur dont $V_{sat,min} = V_{EE}$ et $V_{sat,max} = V_{CC}$ est un amplificateur dit "rail à rail". En mode non linéaire ils sont pratiques car le signal de sorties donne V_{EE} ou V_{CC} sans atténuation.

Modèle de l'AOP idéal – Mode linéaire – 2/2

L'amplificateur opérationnel possède deux modes de fonctionnement différents.

Quand il y a un circuit entre V^- et V_{out} l'AOP est en boucle fermée. On dit qu'il est en mode linéaire.



Le modèle mathématique de l'AOP en boucle fermée est alors le suivant :

$$I^+ = 0A, \quad I^- = 0A,$$

$$\epsilon = v^+ - v^- = 0V.$$

et

$$V_{out} \in [V_{sat,min}, V_{sat,max}].$$

Les tensions $V_{sat,min}$ et $V_{sat,max}$ sont les tensions de saturation, données par le constructeur, que les AOPs ne peuvent pas dépasser.

Par exemple, pour les LM358 et LM324, $V_{sat,min} = V_{EE} + 0.1V$ et $V_{sat,max} = V_{CC} - 1.3V$.

Un amplificateur dont $V_{sat,min} = V_{EE}$ et $V_{sat,max} = V_{CC}$ est un amplificateur dit "rail à rail". Il sont pratiques mais le signal de sortie présente des distortions quand sa tension s'approche des rails d'alimentation ce qui peut dégrader la qualité du signal final.

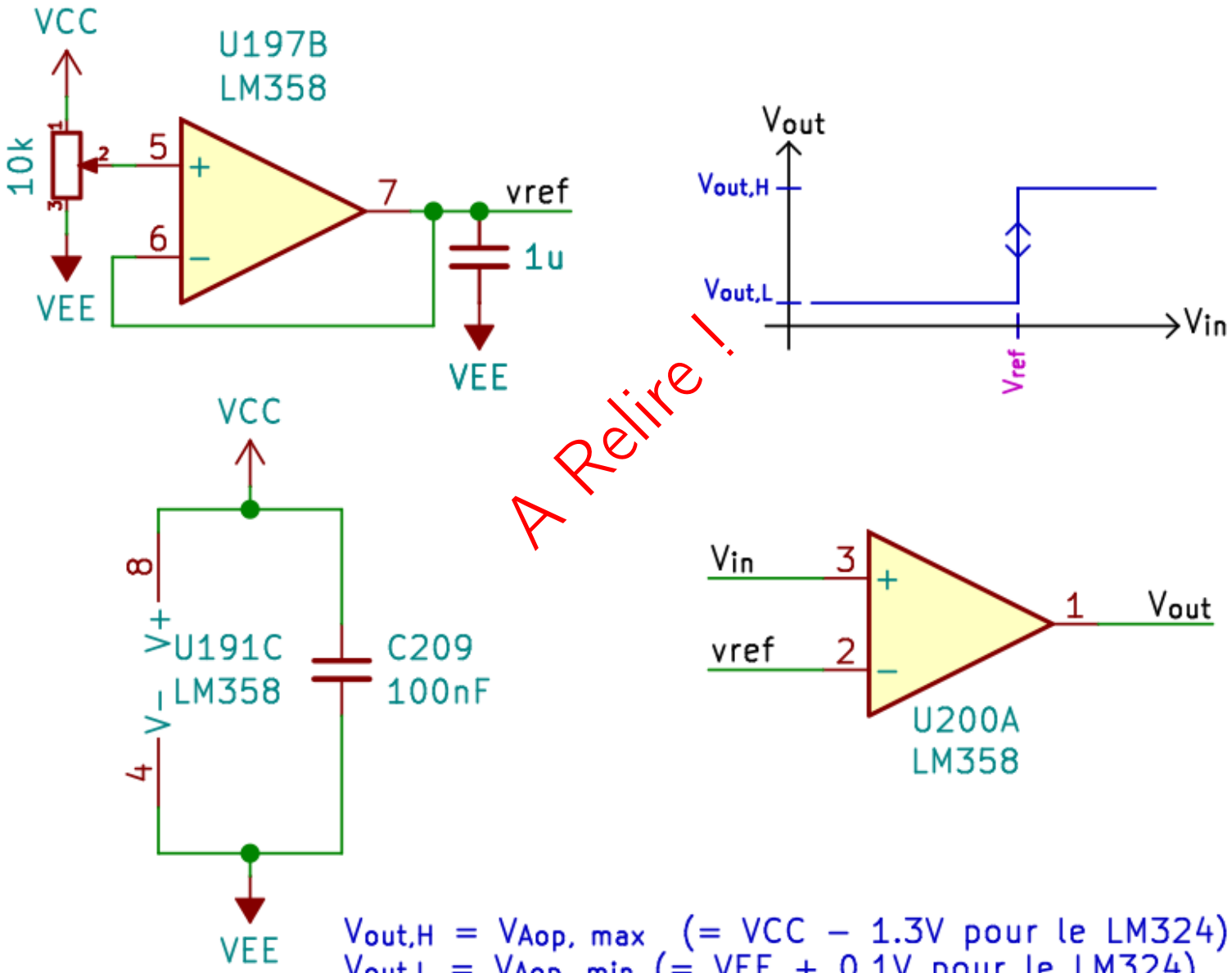
(version longue)

Table de AOP classiques

A FAIRE :

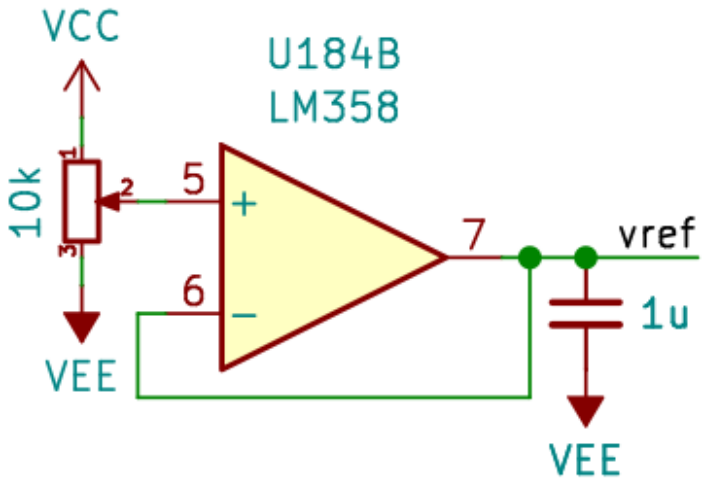
Sommaire des montages classiques utilisant des AOP

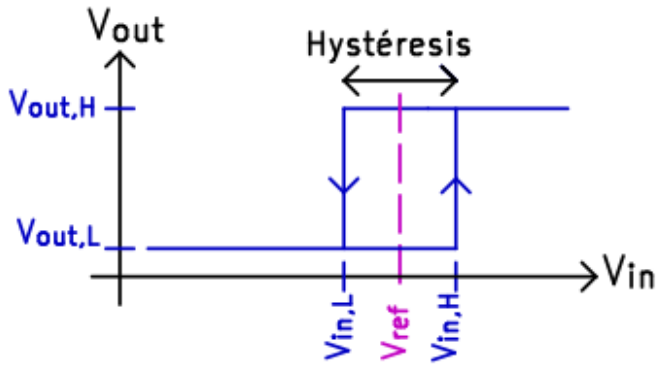
Comparateur simple

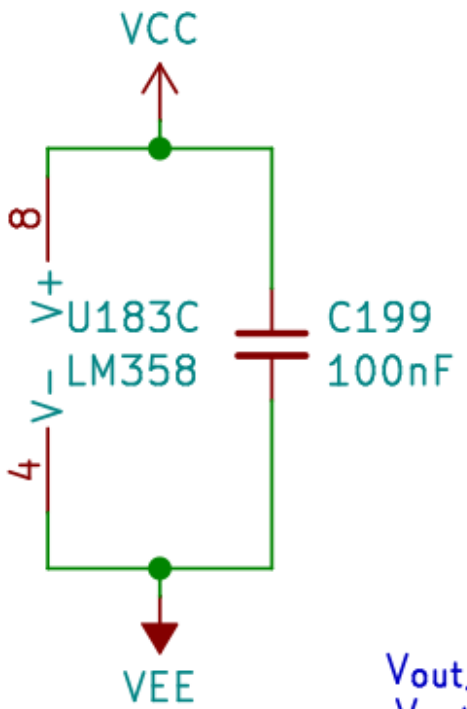


ersion longue)

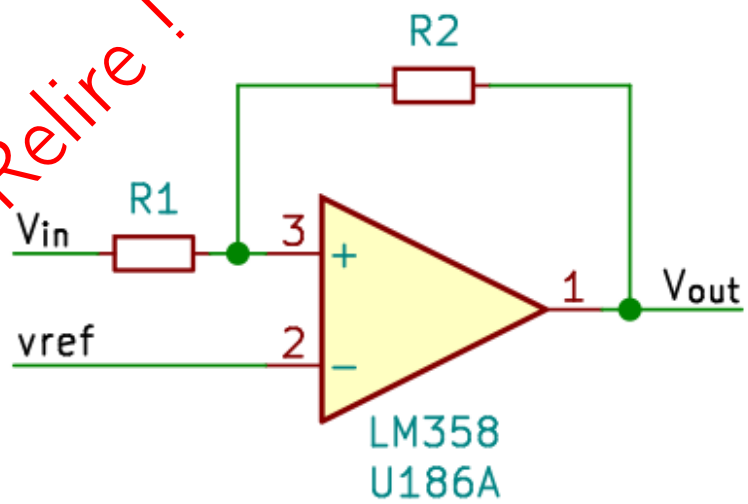
Comparateur à hystéresis (bi-stable/trigger de Schmitt) non inverseur







A Relire !

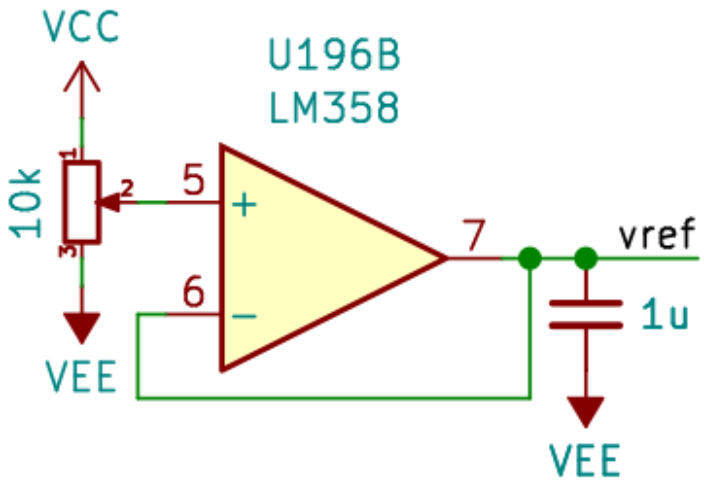


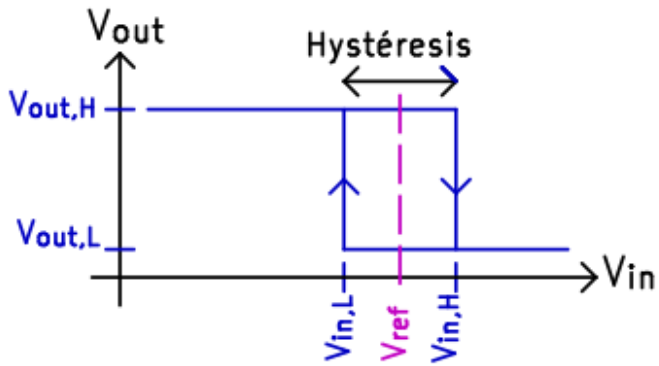
$V_{in,L} = R1/(R1+R2)\ (V_{out,L} - V_{ref}) + V_{ref}$
$V_{in,H} = R1/(R1+R2)\ (V_{out,H} - V_{ref}) + V_{ref}$
$Hysteresis = R1/(R1+R2)\ (V_{out,H} - V_{out,L})$

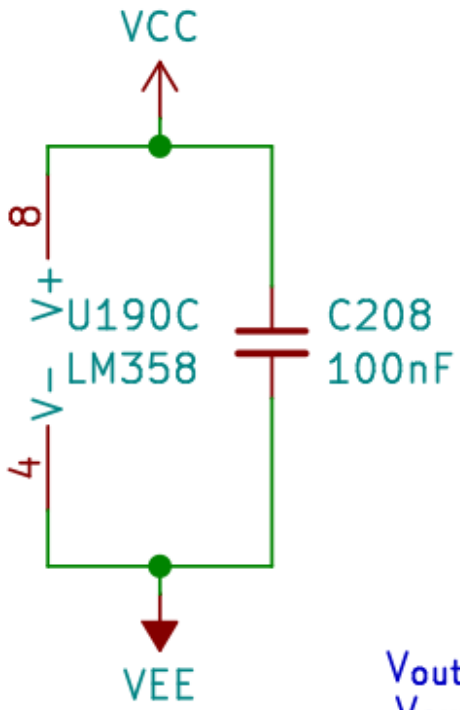
$V_{out,H} = V_{Aop,\ max}$ (= VCC − 1.3V pour le LM324)
$V_{out,L} = V_{Aop,\ min}$ (= VEE + 0.1V pour le LM324)

ersion longue)

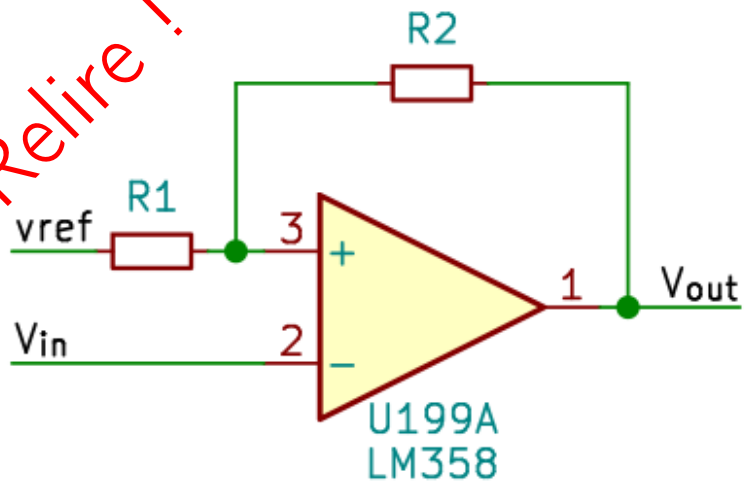
Comparateur à hysteresis (bi-stable/trigger de Schmitt) inverseur







A Relire !

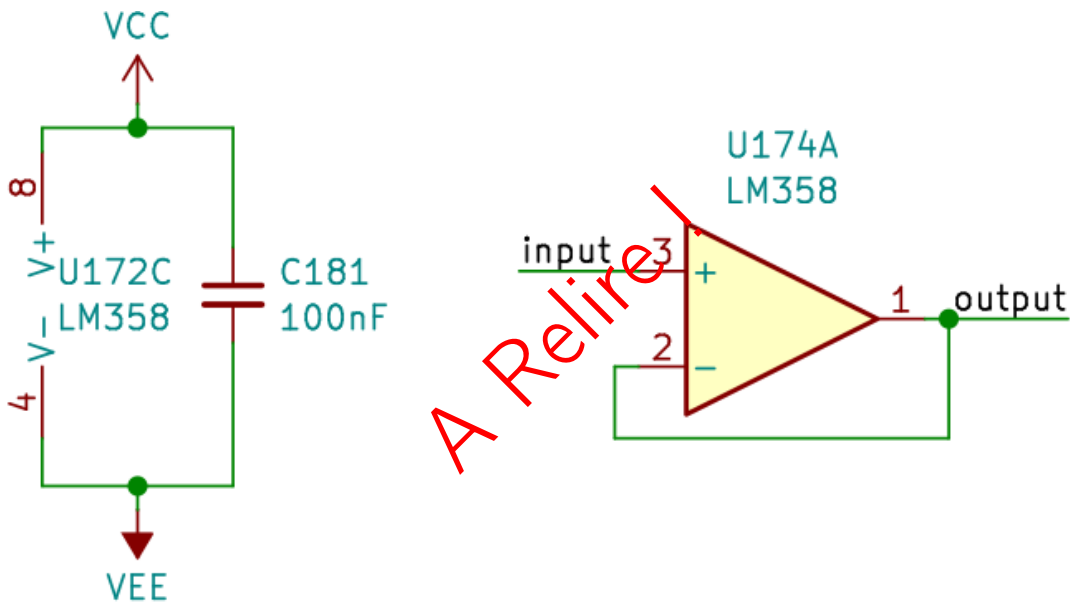


$V_{in,L} = R1/(R1+R2)\ (V_{out,L} - V_{ref}) + V_{ref}$
$V_{in,H} = R1/(R1+R2)\ (V_{out,H} - V_{ref}) + V_{ref}$
$Hysteresis = R1/(R1+R2)\ (V_{out,H} - V_{out,L})$

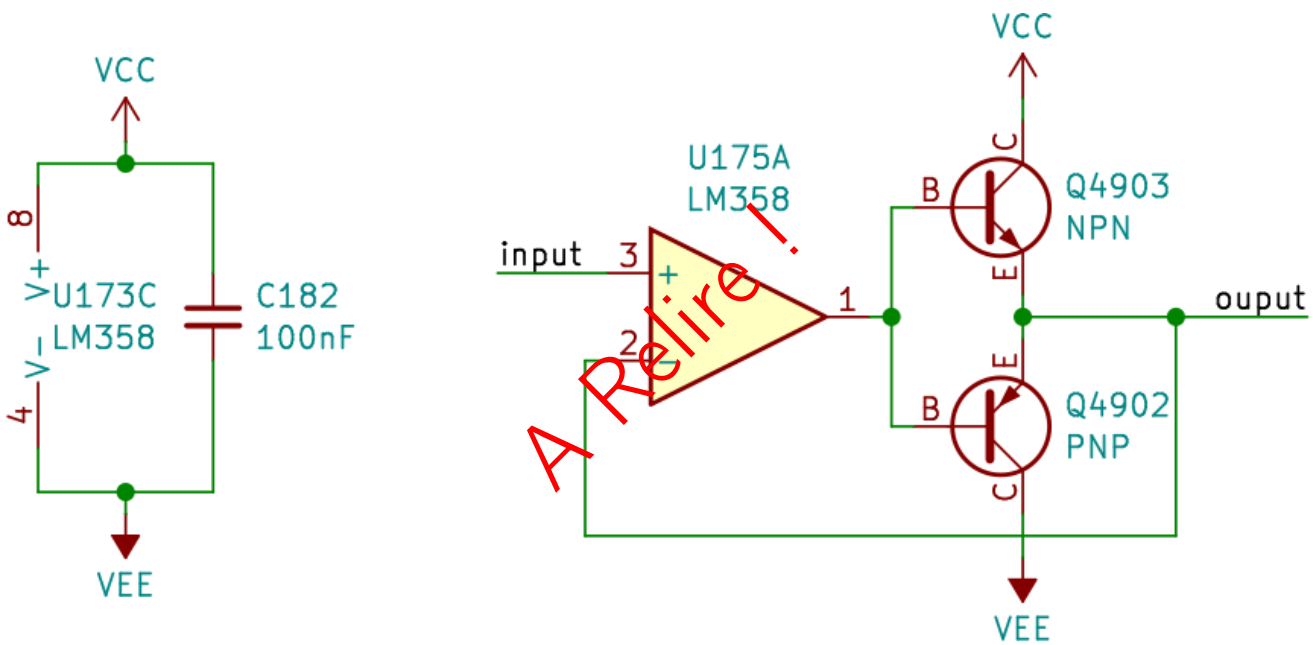
$V_{out,H} = V_{Aop,\ max}$ (= VCC − 1.3V pour le LM324)
$V_{out,L} = V_{Aop,\ min}$ (= VEE + 0.1V pour le LM324)

ersion longue)

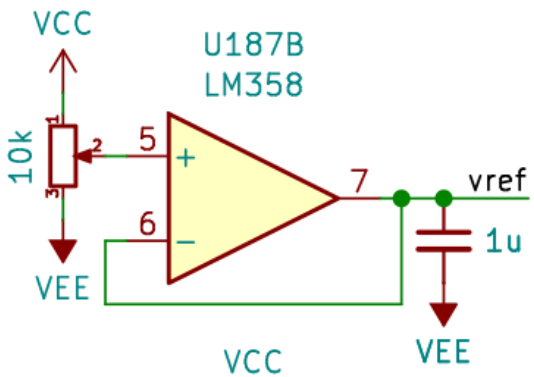
Montage suiveur ou buffer

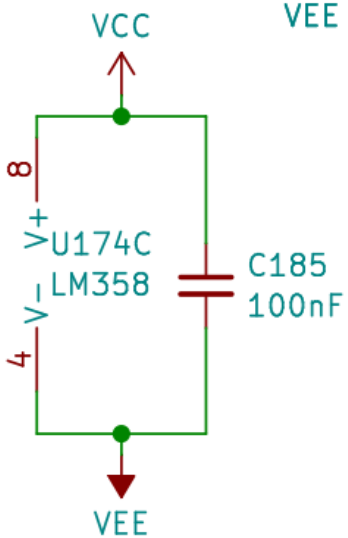


Montage suiveur ou buffer avec étage push-pull

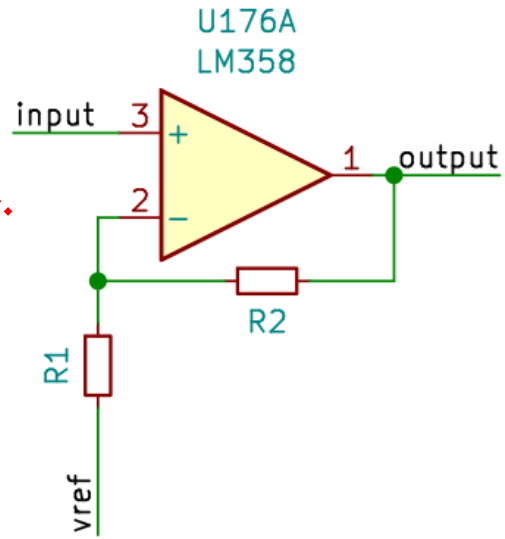


Amplificateur



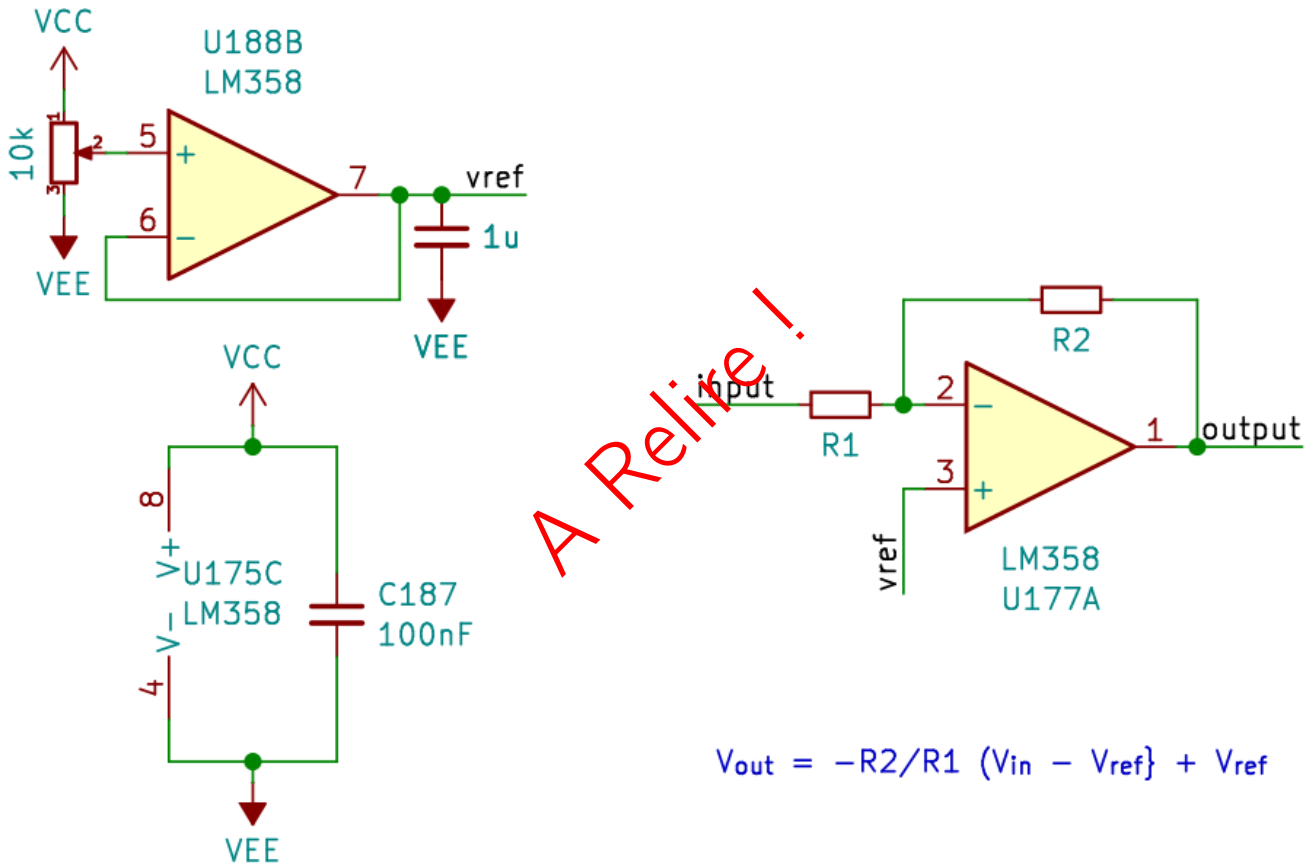


A Relire !

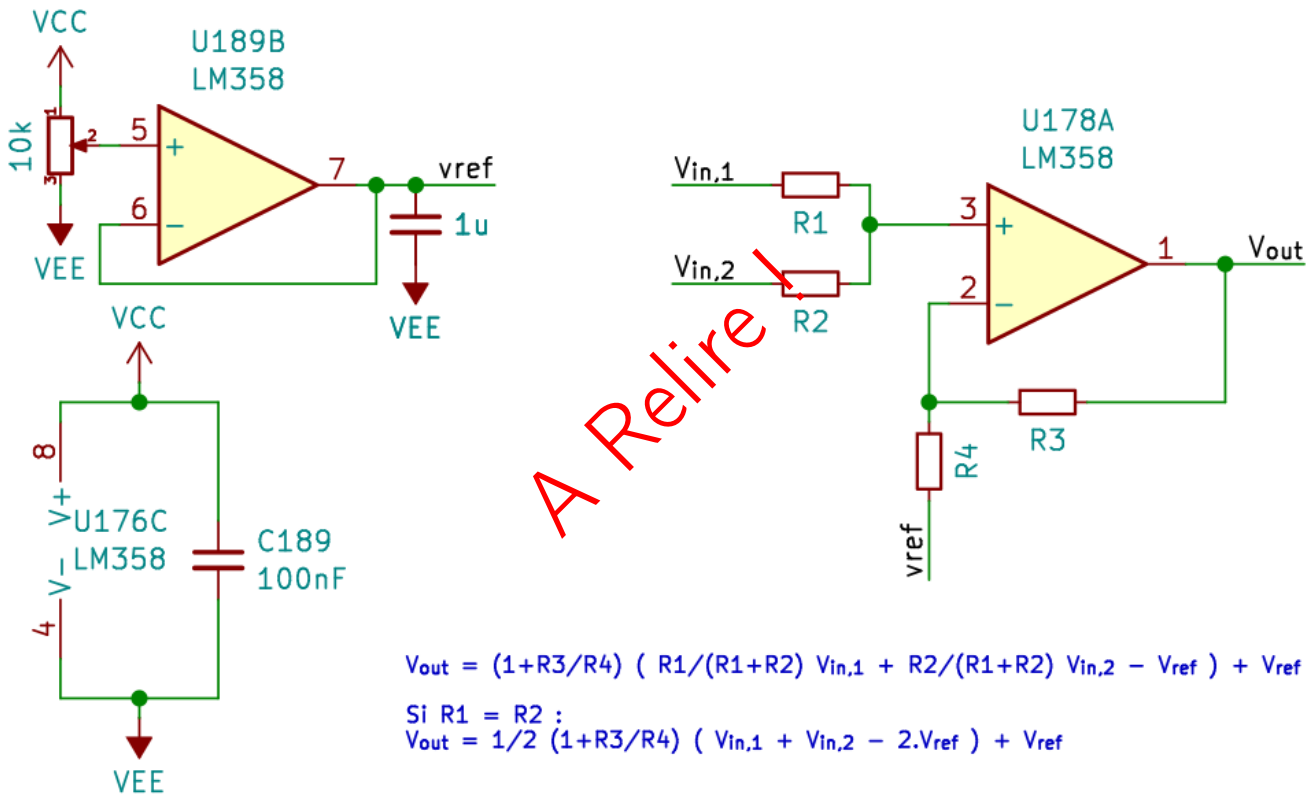


$$V_{out} = (1 + R2/R1)(V_{in} - V_{ref}) + V_{ref}$$

Inverseur


VCC
U188B
LM358
10k
5
6
7
vref
1u
VEE
VEE
VCC
8
V+
U175C
LM358
V-
4
C187
100nF
VEE
A Relire !
R2
input
R1
2
3
1
output
vref
LM358
U177A
Vout = −R2/R1 (Vin − Vref) + Vref

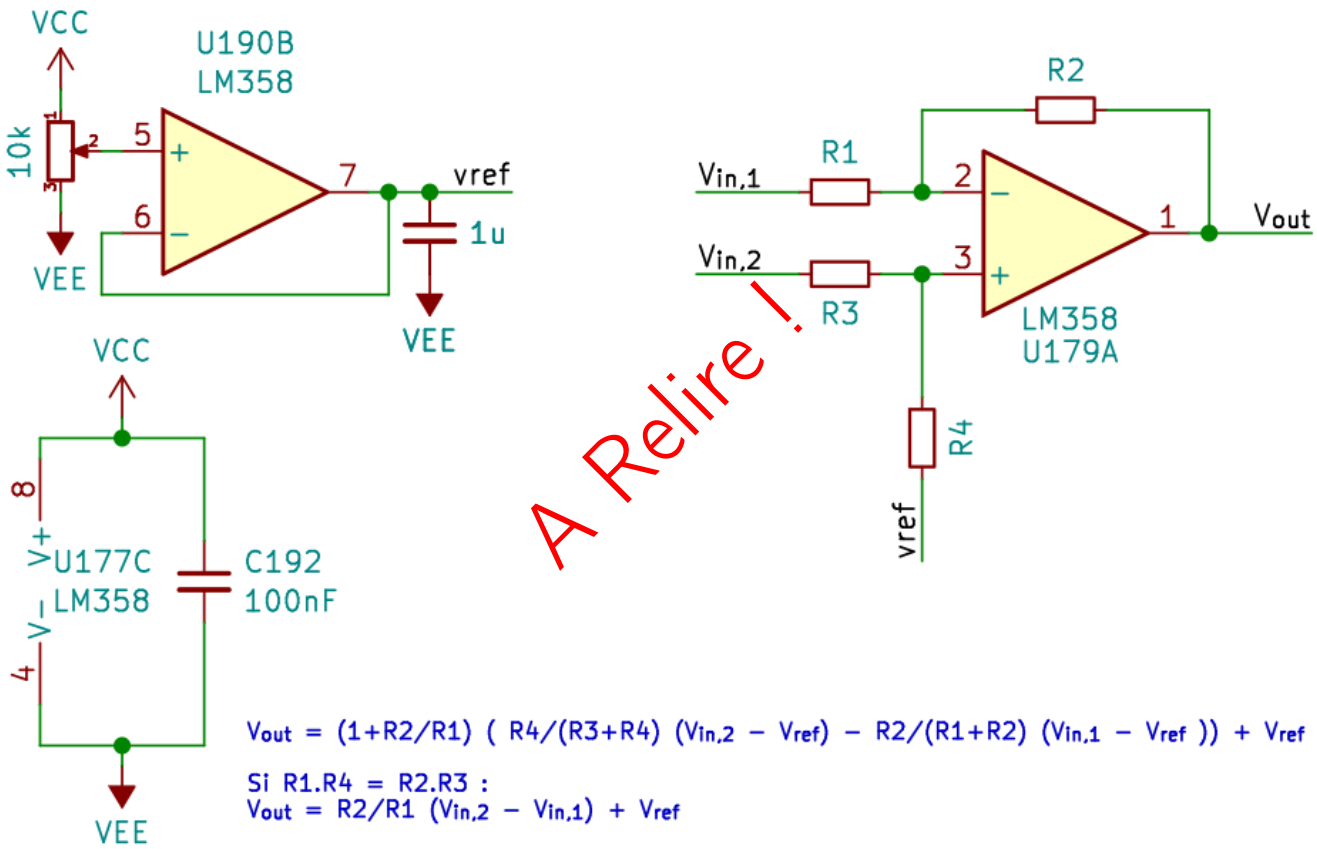
Sommateur



$V_{out} = (1+R3/R4)\ (\ R1/(R1+R2)\ V_{in,1} + R2/(R1+R2)\ V_{in,2} - V_{ref}\ ) + V_{ref}$

Si R1 = R2 :
$V_{out} = 1/2\ (1+R3/R4)\ (\ V_{in,1} + V_{in,2} - 2.V_{ref}\ ) + V_{ref}$

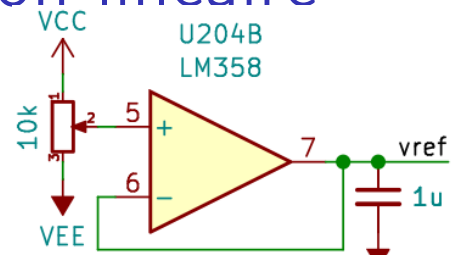
Soustracteur

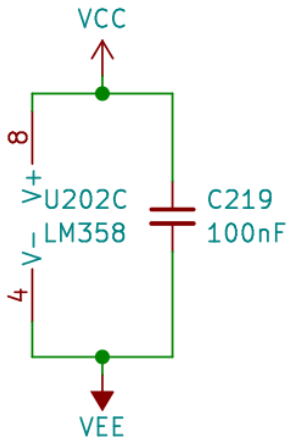


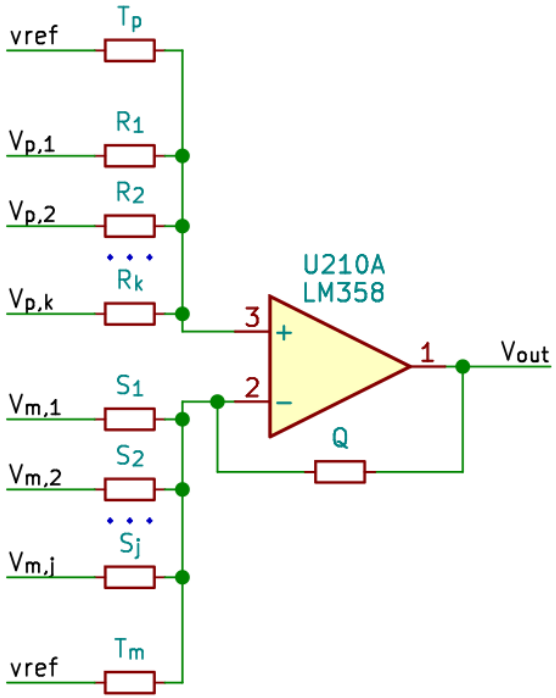
$V_{out} = (1+R2/R1)\ (\ R4/(R3+R4)\ (V_{in,2} - V_{ref}) - R2/(R1+R2)\ (V_{in,1} - V_{ref}\ ))\ + V_{ref}$

Si R1.R4 = R2.R3 :
$V_{out} = R2/R1\ (V_{in,2} - V_{in,1}) + V_{ref}$

Combinaison linéaire







$$V_{out} - V_{ref} = \sum_{i=1}^{k} \alpha_i \cdot (V_{p,i} - V_{ref}) - \sum_{i=1}^{j} \beta_i \cdot (V_{m,i} - V_{ref});$$

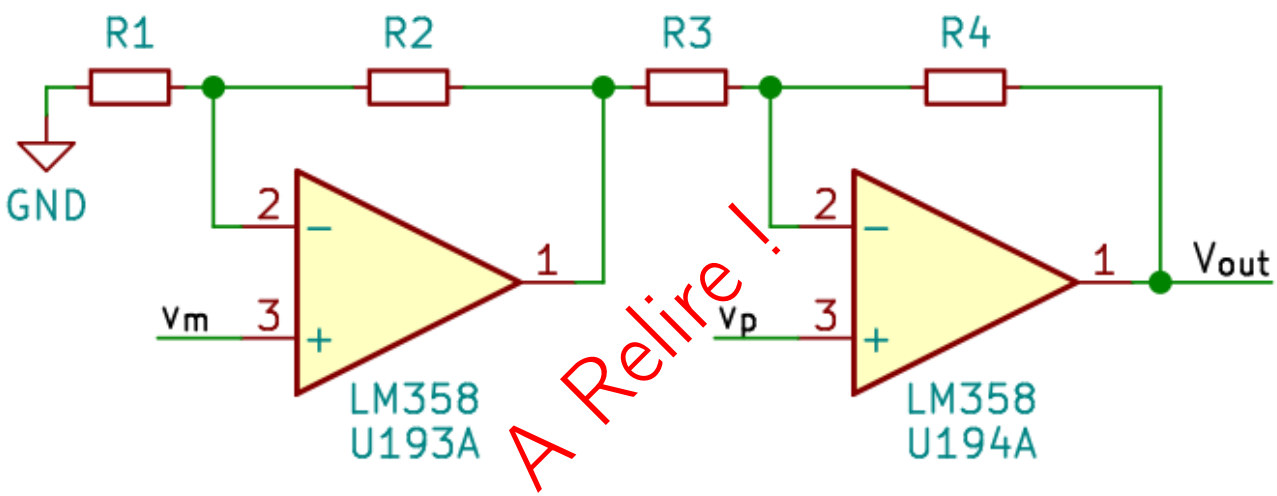
$$\begin{cases} \gamma = 1 - \sum_{i=1}^{k} \alpha_i + \sum_{i=1}^{j} \beta_i; \\ T_p,\ T_m = \text{les valeurs de votre choix (éventuellement } +\infty); \\ Q = \gamma \cdot (T_p \| - T_m) = \gamma \cdot \dfrac{1}{1/T_p - 1/T_m}; \\ R_i = \dfrac{Q}{\alpha_i} \quad \text{pour tout } i \in [\![1,k]\!]; \\ S_i = \dfrac{Q}{\beta_i} \quad \text{pour tout } i \in [\![1,j]\!]. \end{cases}$$

Si $\gamma = 0$, amplifiez une entrée (avec un étage d'amplification supplémentaire) pour que γ soit différent de 0.

Simulation du montage Combinaison linéaire.

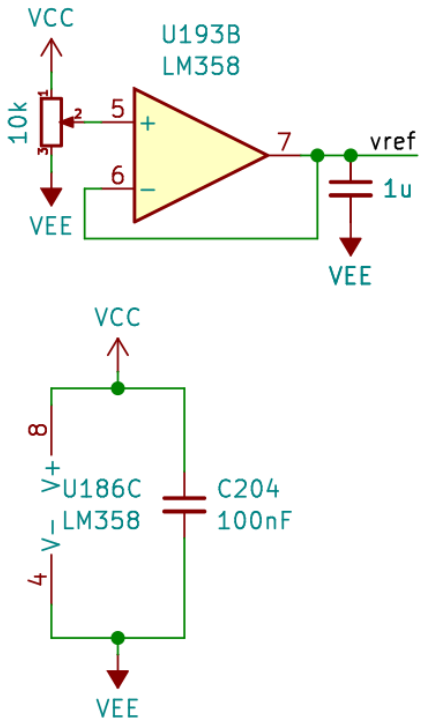
(version longue)

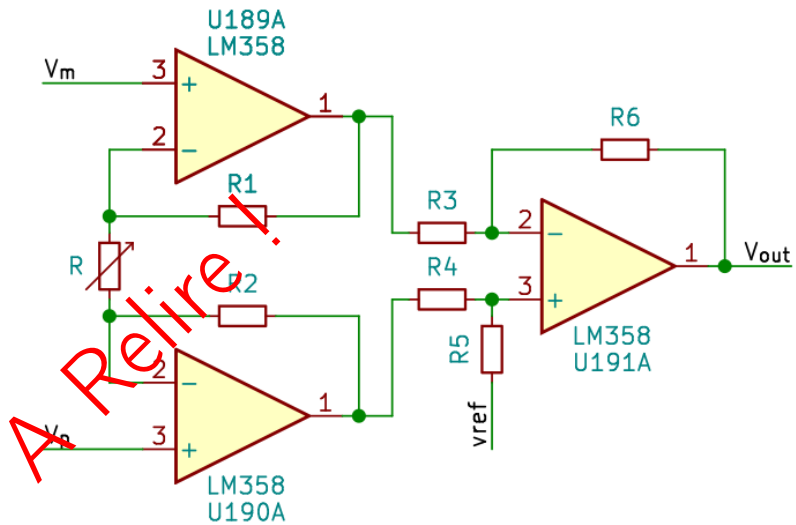
Amplification d'instrumentation à 2 AOP



Si R1=R4 et R2 = R3 alors
$V_{out} = (1 + R1/R2)\ (v_p - v_p)$

Amplification d'instrumentation à 3 AOP



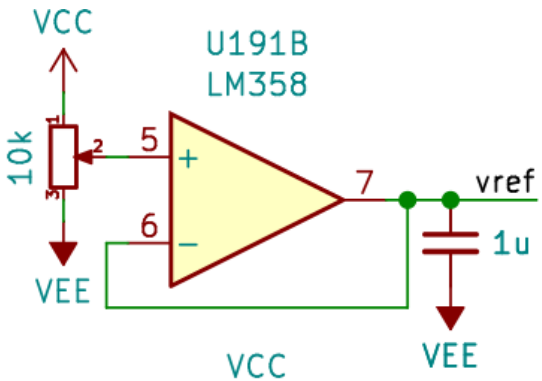


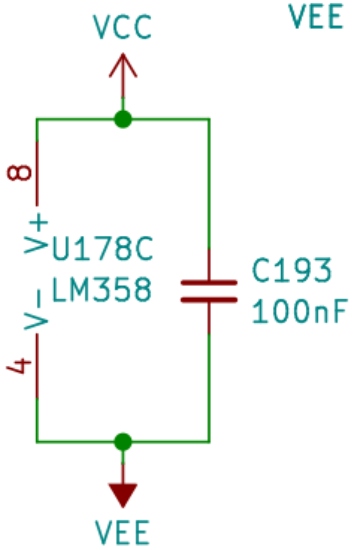
Si R3.R5 = R4.R6 alors

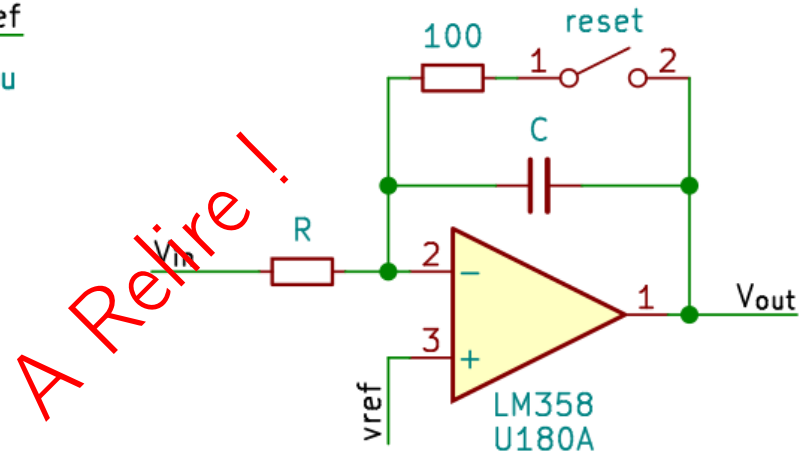
Us = R6/43 . (1 + (R1 + R2)/R) . (Vp − Vm) + Vref

(version longue)

Intégrateur



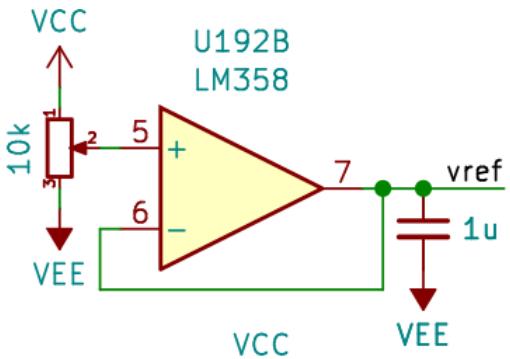


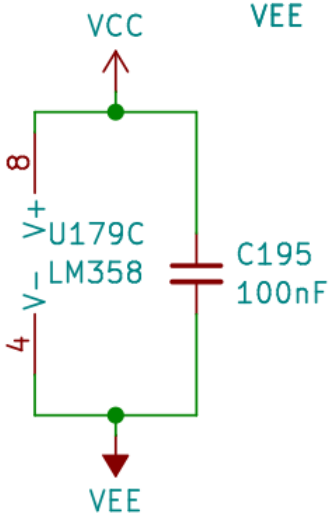


$$\partial V_{out} = -1/(R.C)\ (V_{in} - V_{ref})$$

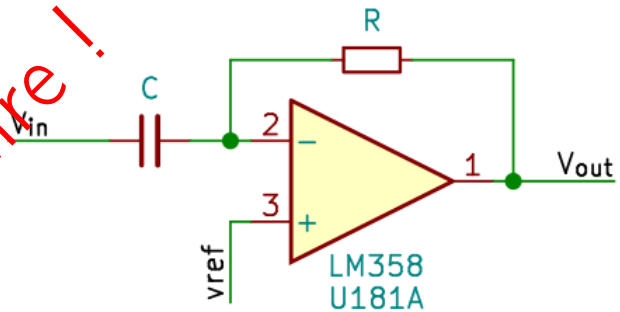
$$V_{out} = -1/(R.C) \int (V_{in} - V_{ref}) + V_{out,\ t=0}$$

Dérivateur



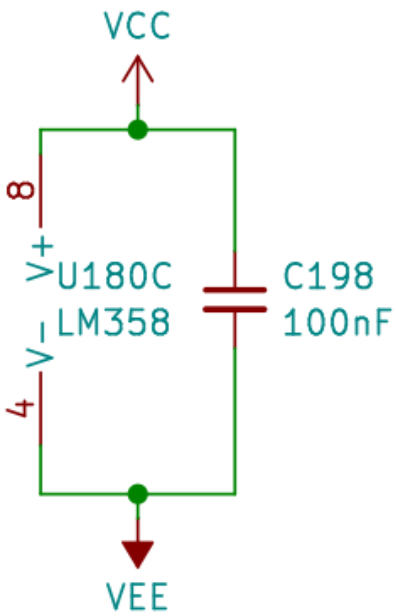


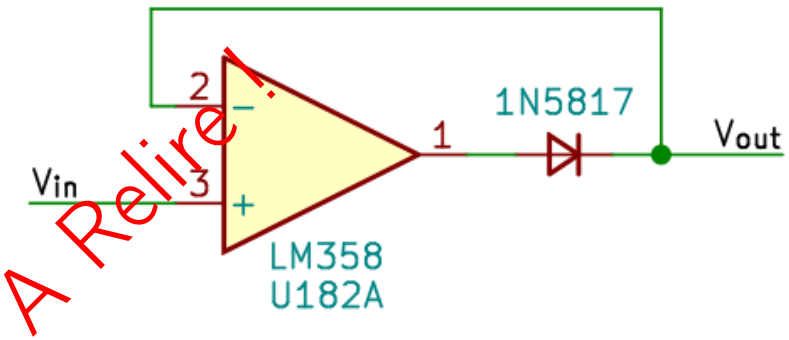
A Relire !



$V_{out} = - R.C . \partial V_{in} + V_{ref}$

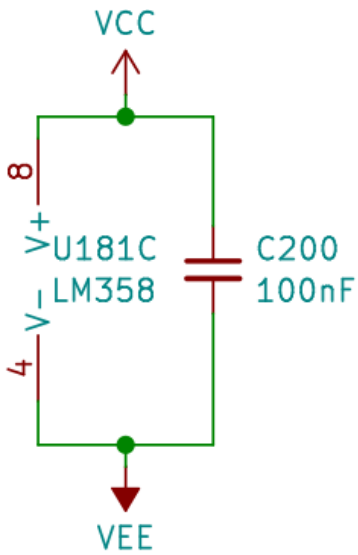
Diode parfaite

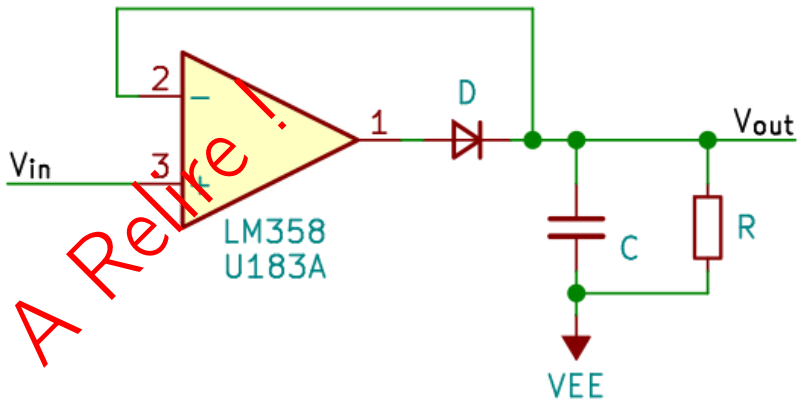




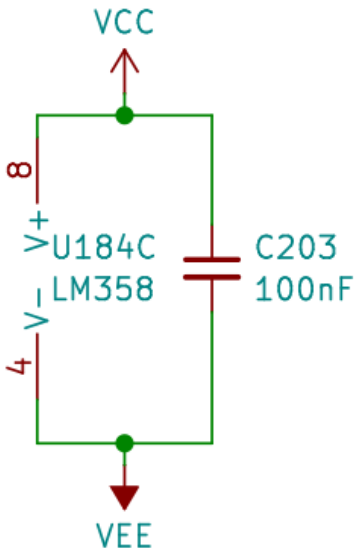
$V_{out} - V_{in} = 0$ Si I > 0 (c.a.d Vout > Vref)
$V_{out} - V_{in} > 0$ Si I = 0 (c.a.d Vout < Vref)

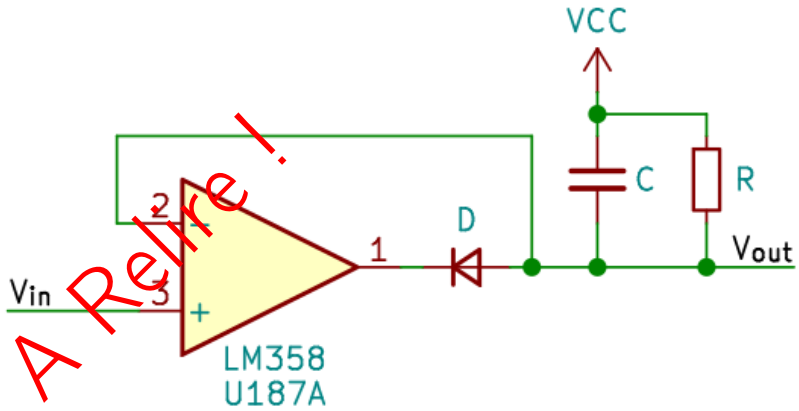
Detecteur de crête ou de pique





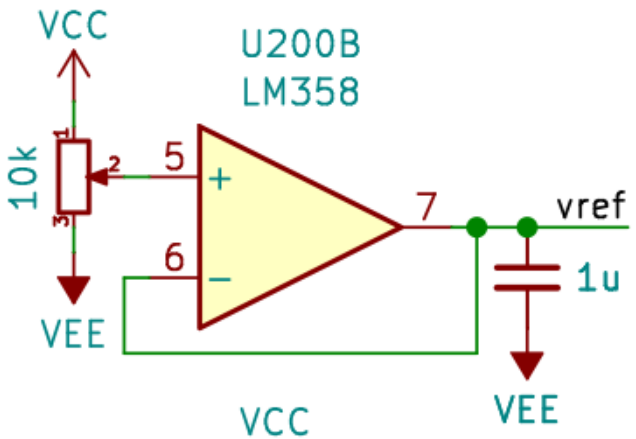
Detecteur de vallée

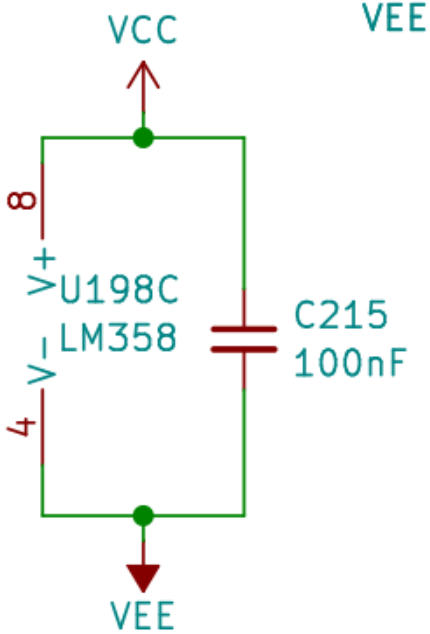


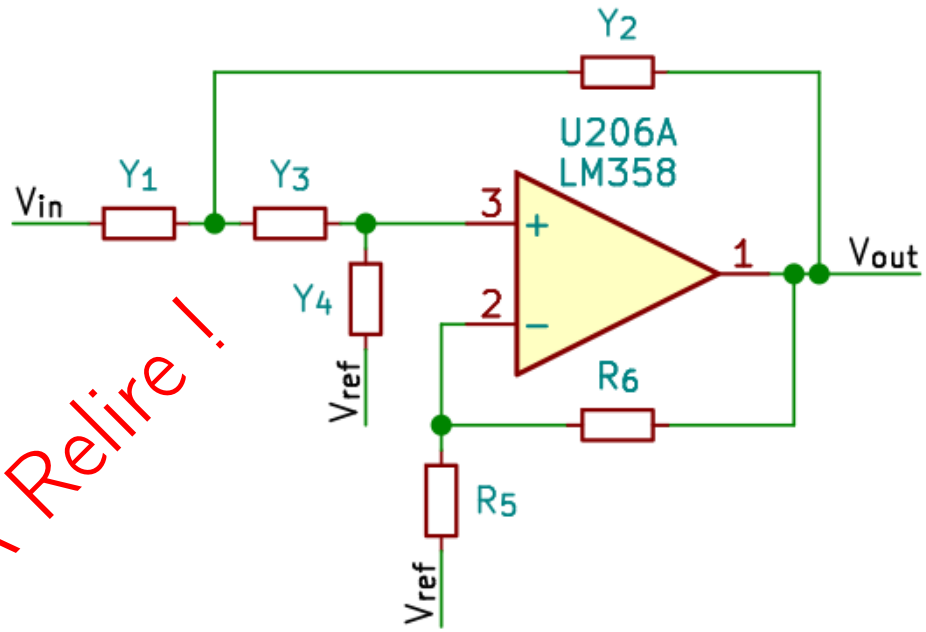


(version longue)

Cellule de Sallen-Key







A Relire !

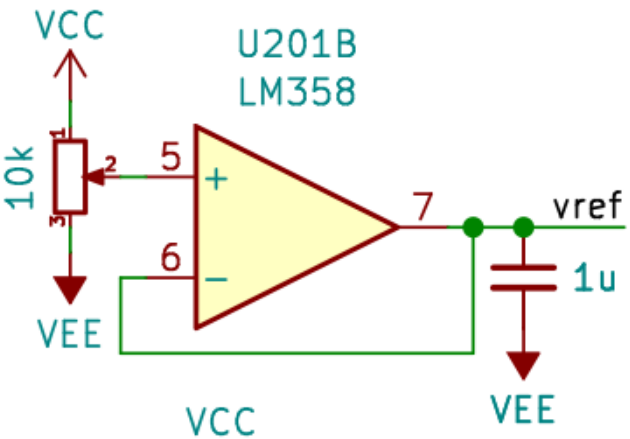
Y_1, Y_2, Y_3, Y_4 sont des admittances (inverse des impédances).
R_5 et R_6 sont des résistances.

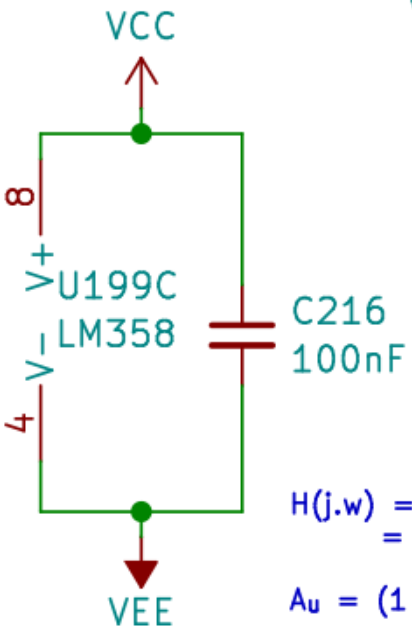
$$H(j.w) = FFT(V_{out})(w)/FFT(V_{in})(w)$$
$$= A_u.Y_1.Y_3 /((Y_1+Y_2).(Y_3+Y_4) + Y_3.(Y_4 - A_u.Y_2))$$

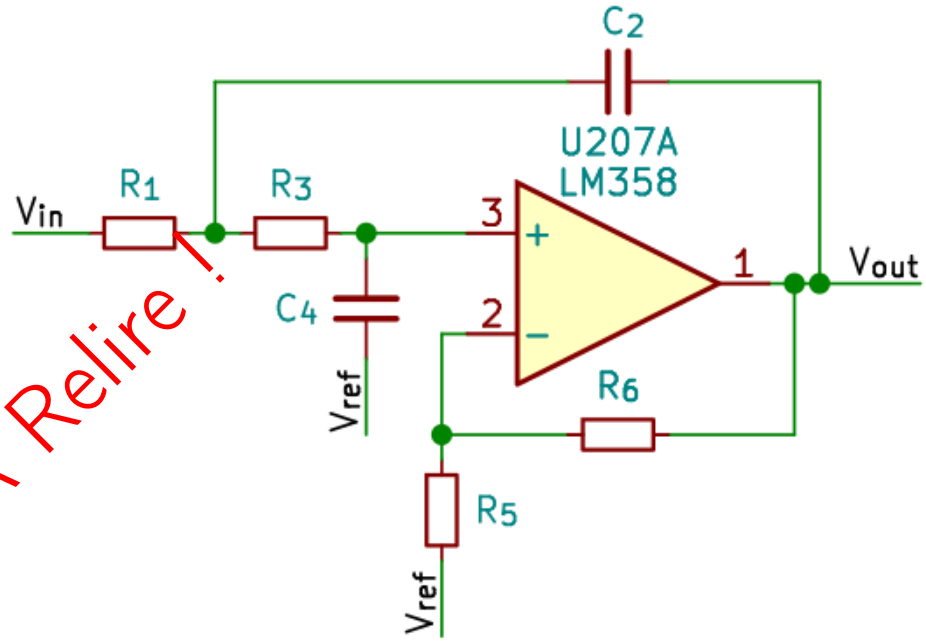
$$A_u = (1 + R_6/R_5)$$

(version longue)

Cellule de Sallen-Key - filtre passe bas







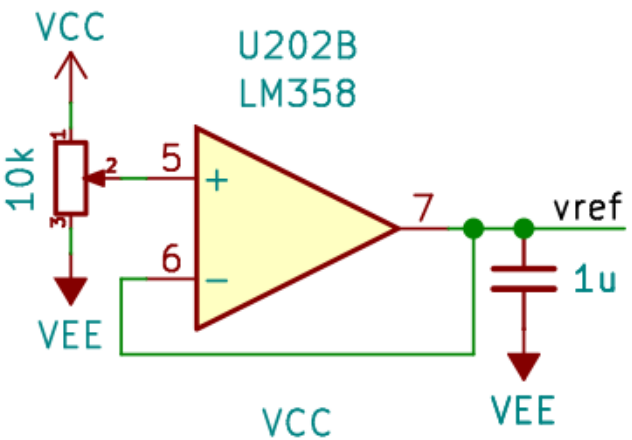
A Relire !.

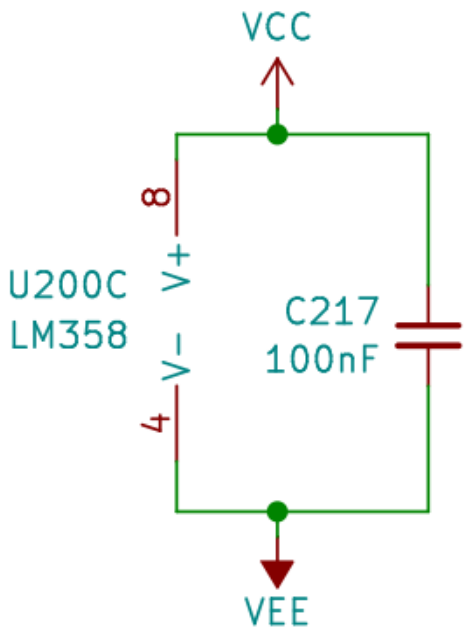
$$H(j.w) = FT(V_{out})(w)/FT(V_{in})(w) = A_u / (1 + ((R_1.C_2 + R_3.C_4 + R_1.C_4 - R_1.C_2.A_u) . j.w + R_1.C_2.R_3.C_4 . (jw)^2)$$

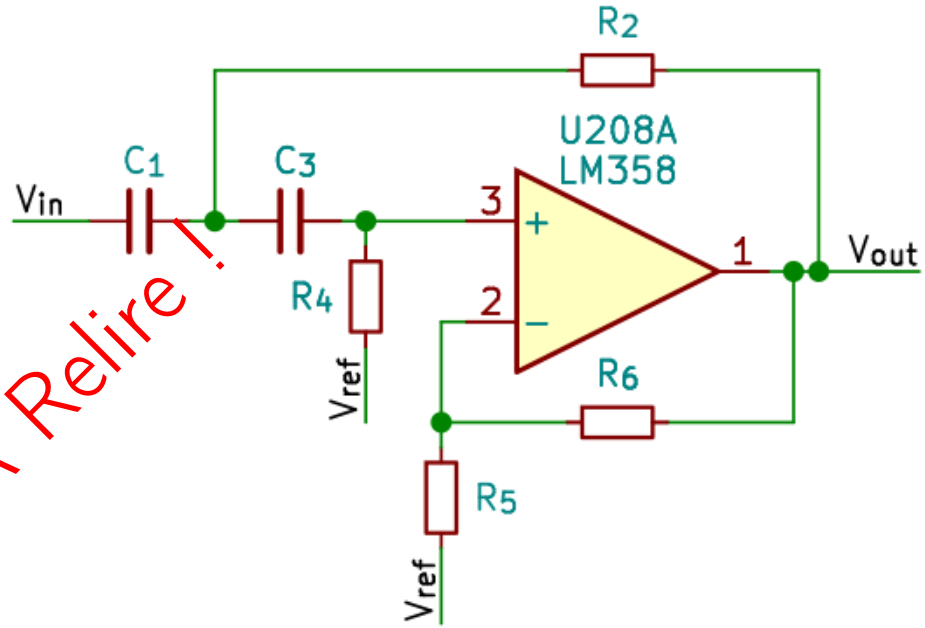
$A_u = (1 + R_6/R_5)$ Pulsation de coupure : $2.Pi.f_c = w_c = (R_1.C_2.R_3.C_4)^{-1}$

(version longue)

Cellule de Sallen-Key - filtre passe haut







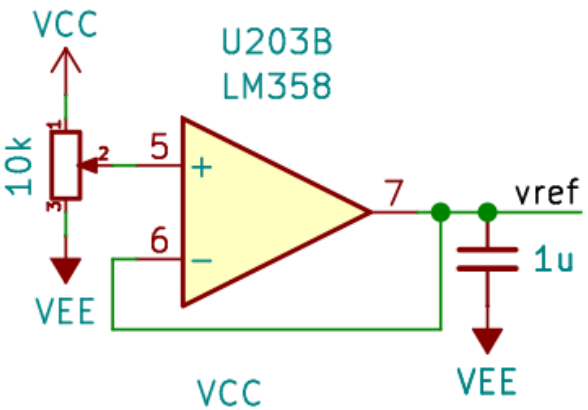
A Relire !

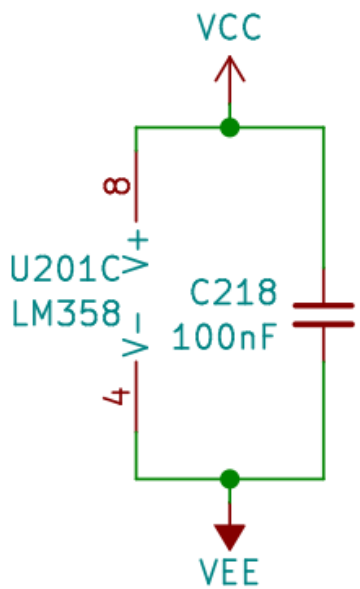
$$H(j.w) = FT(V_{out})(w)/FT(V_{in})(w)$$
$$= A_u \,/\, \{ 1 + [(R_2.C_1)^{-1} + (R_4.C_3)^{-1} + (R_4.C_1)^{-1} - (R_2.C_1)^{-1}.A_u) \,.\, 1/(j.w) + (R_2.C_1.R_4.C_3)^{-1} \,.\, 1/(jw)^2 \}$$

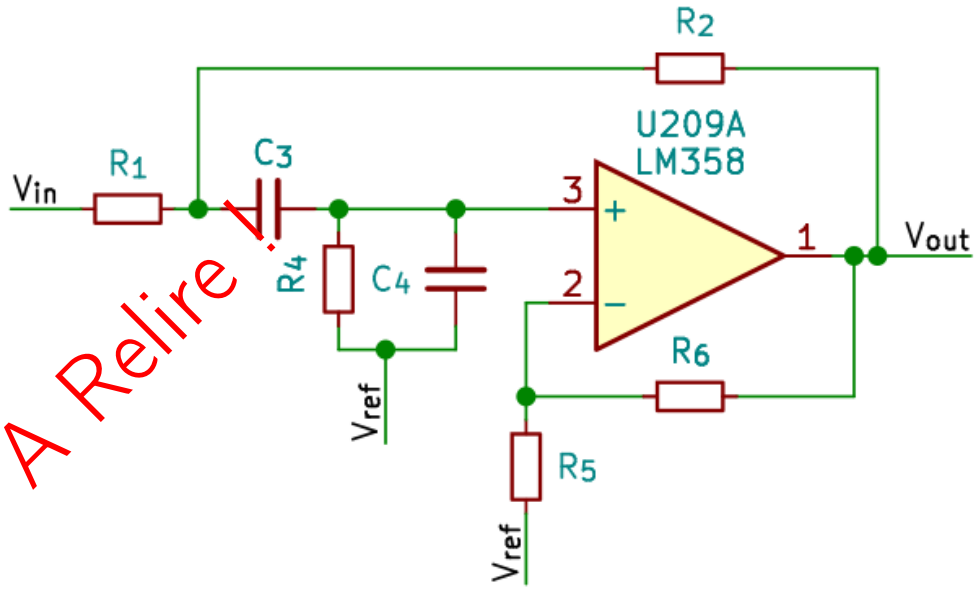
$A_u = (1 + R_6/R_5)$ Pulsation de coupure : $2.Pi.f_c = w_c = (R_2.C_1.R_4.C_3)^{-1}$

(version longue)

Cellule de Sallen-Key - filtre passe bande







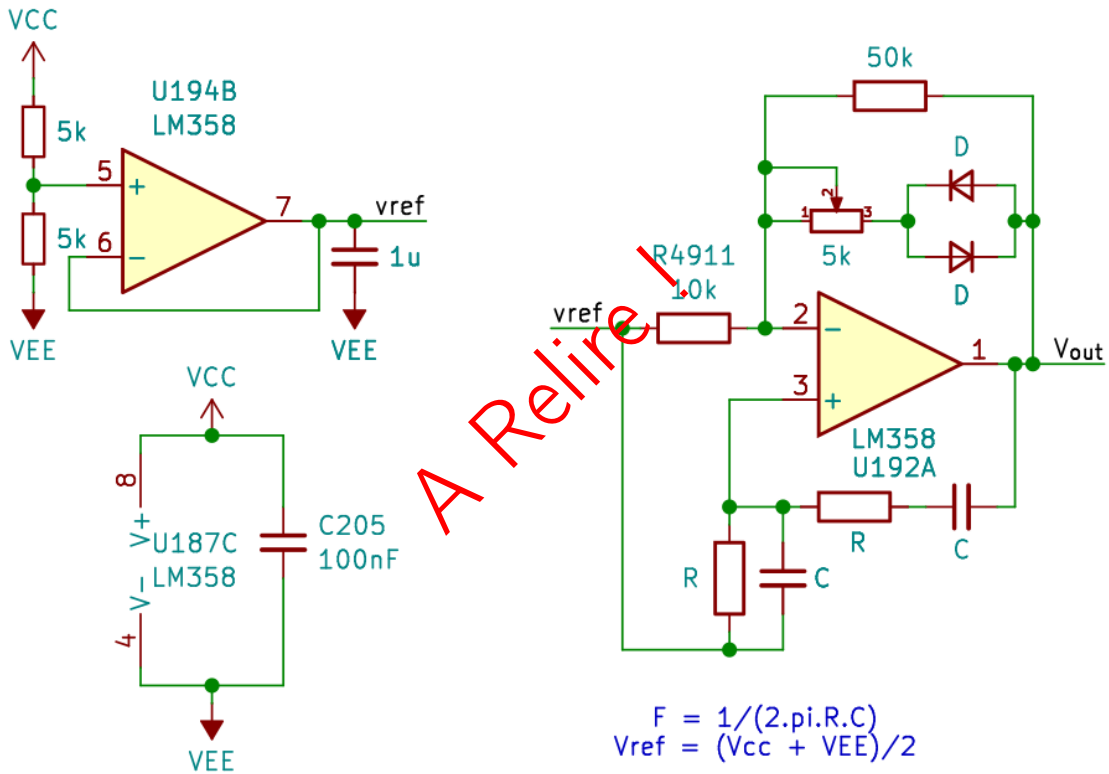
A Relire !!!

$$H(j.w) = FT(V_{out})(w)/FT(V_{in})(w) = A_u.(R_2/(R_1+R_2)).R_4.C_3 \; . \; jw \; / \; (\; 1 + ((C_3+C_4).R_4) + (R_2 - A_u.R_4).(R_1/(R_1+R_2)).C_3) \; . \; jw + (R_1/(R_1+R_2)).R_2.R_4.C_3.C_4 \; . \; (jw)^2 \;)$$

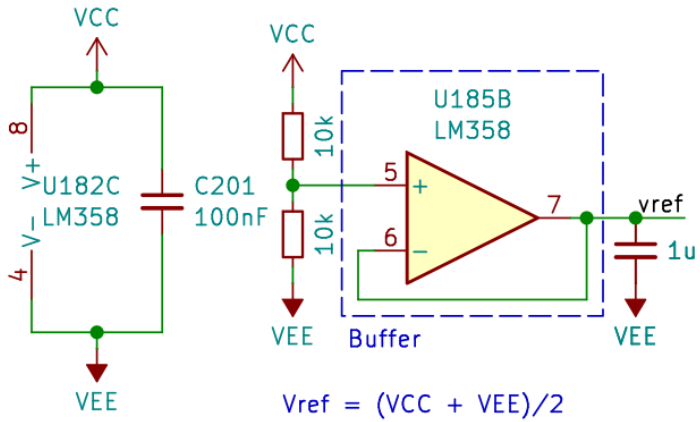
$A_u = (1 + R_6/R_5)$ Pulsation de coupure : $2.Pi.f_c = w_c = (1 + R_2/R_1) \; . \; (R_2.R_4.C_3.C_4)^{-1}$

(version longue)

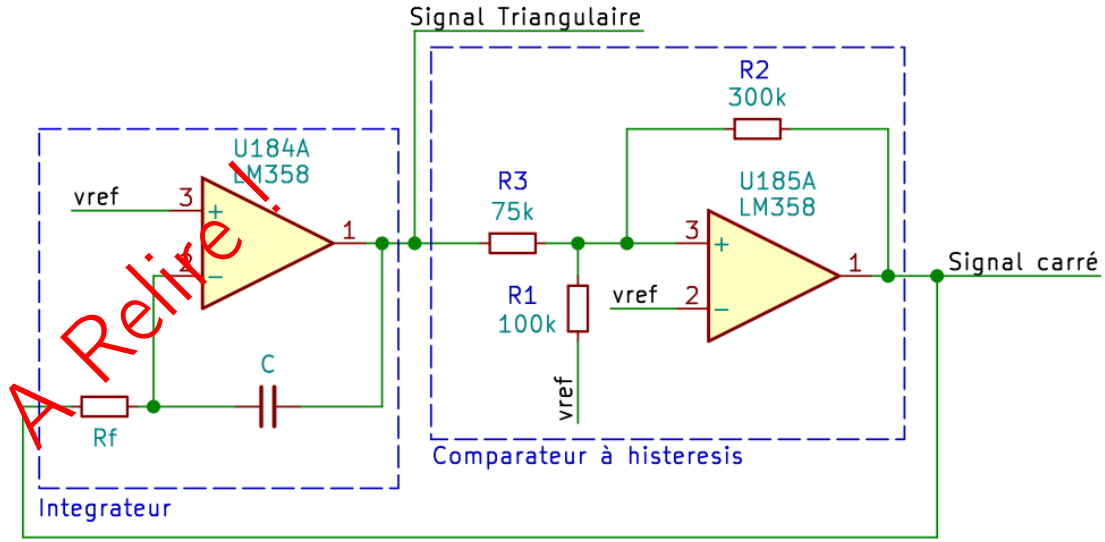
Oscillateur de Wien



Générateur de fonction triangulaire et carrée



Vref = (VCC + VEE)/2



Si R3 = R2 R1 / (R2 + R1) alors
freq = (R1+Rc)/(4 C Rf R1)

(version longue)

Logarithme

A FAIRE :

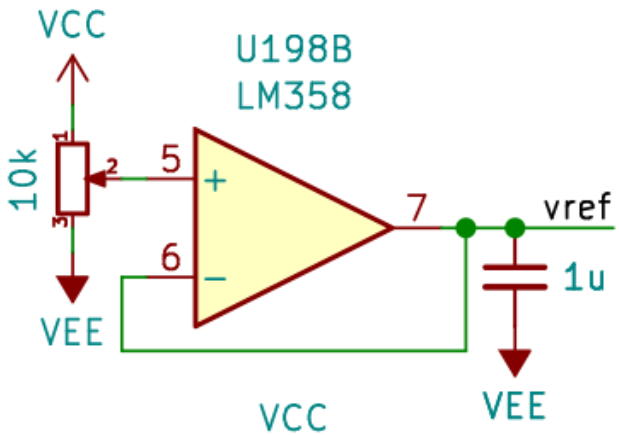
Exponnentielle

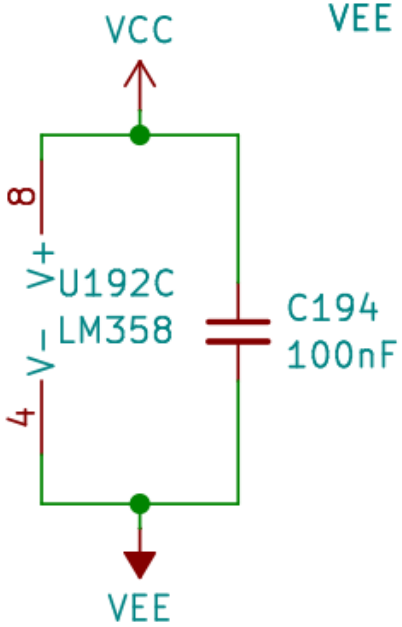
A FAIRE :

Multiplieur

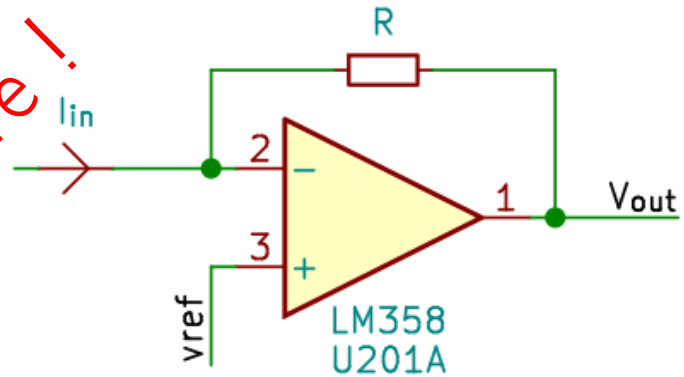
A FAIRE :

Convertisseur courant-tension



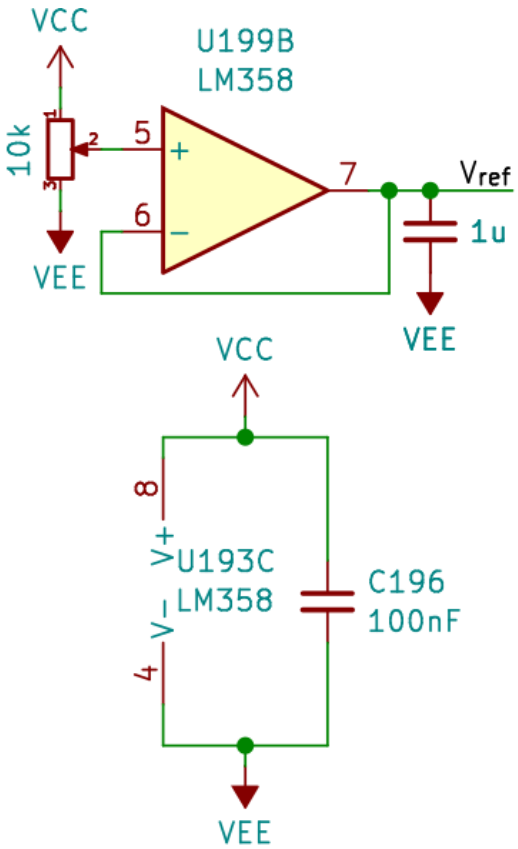


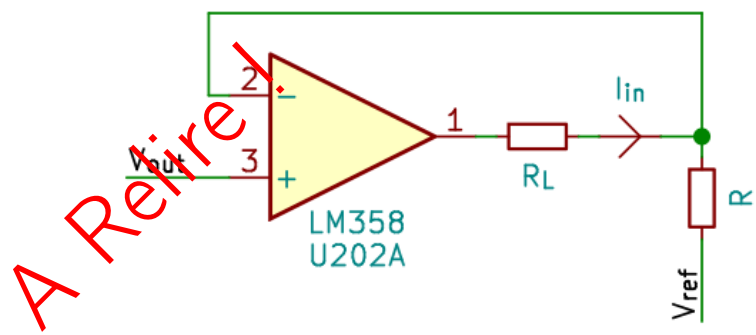
A Relire !



$V_{out} = - R \cdot I_{in} + V_{ref}$

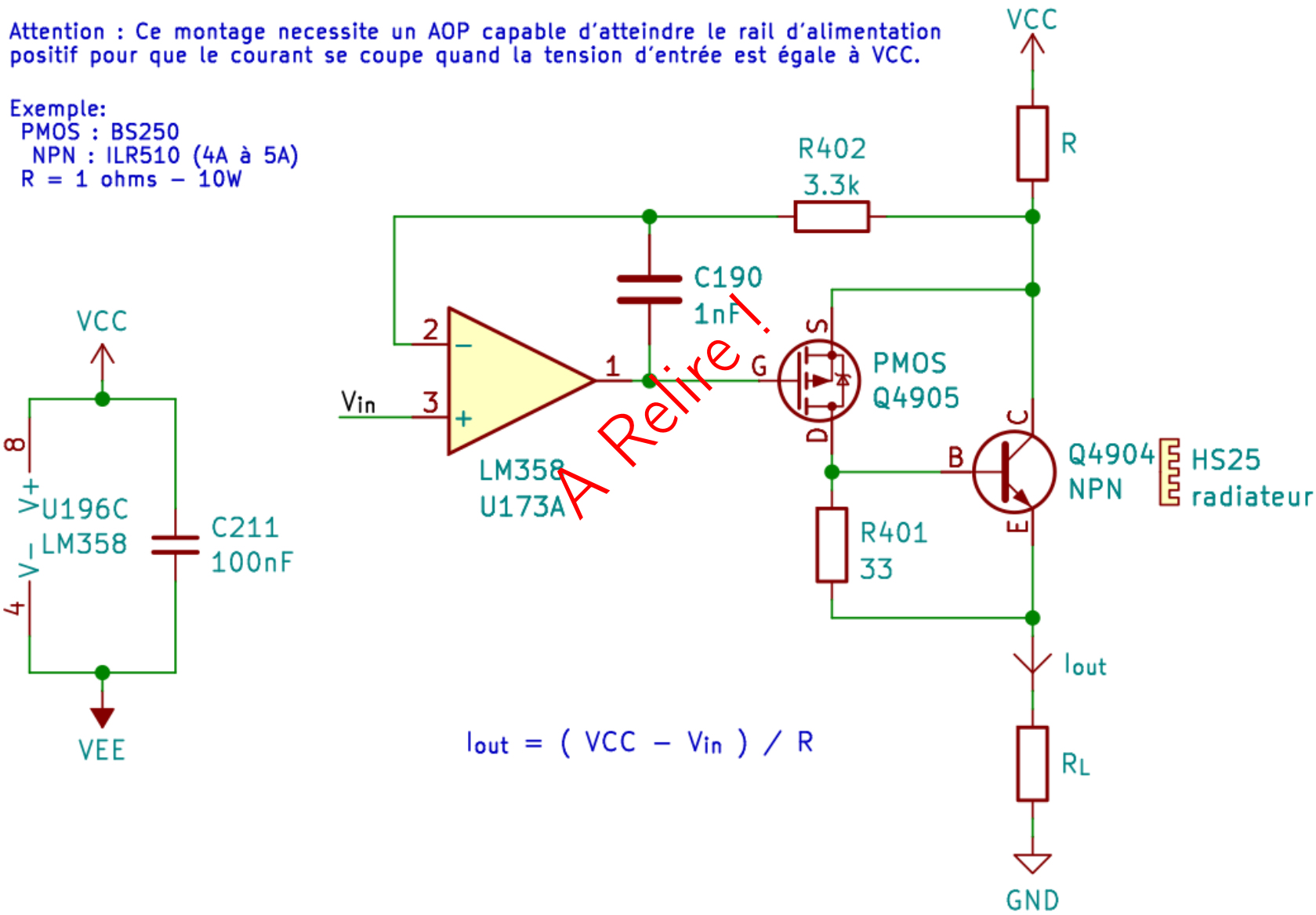
Convertisseur tension-courant





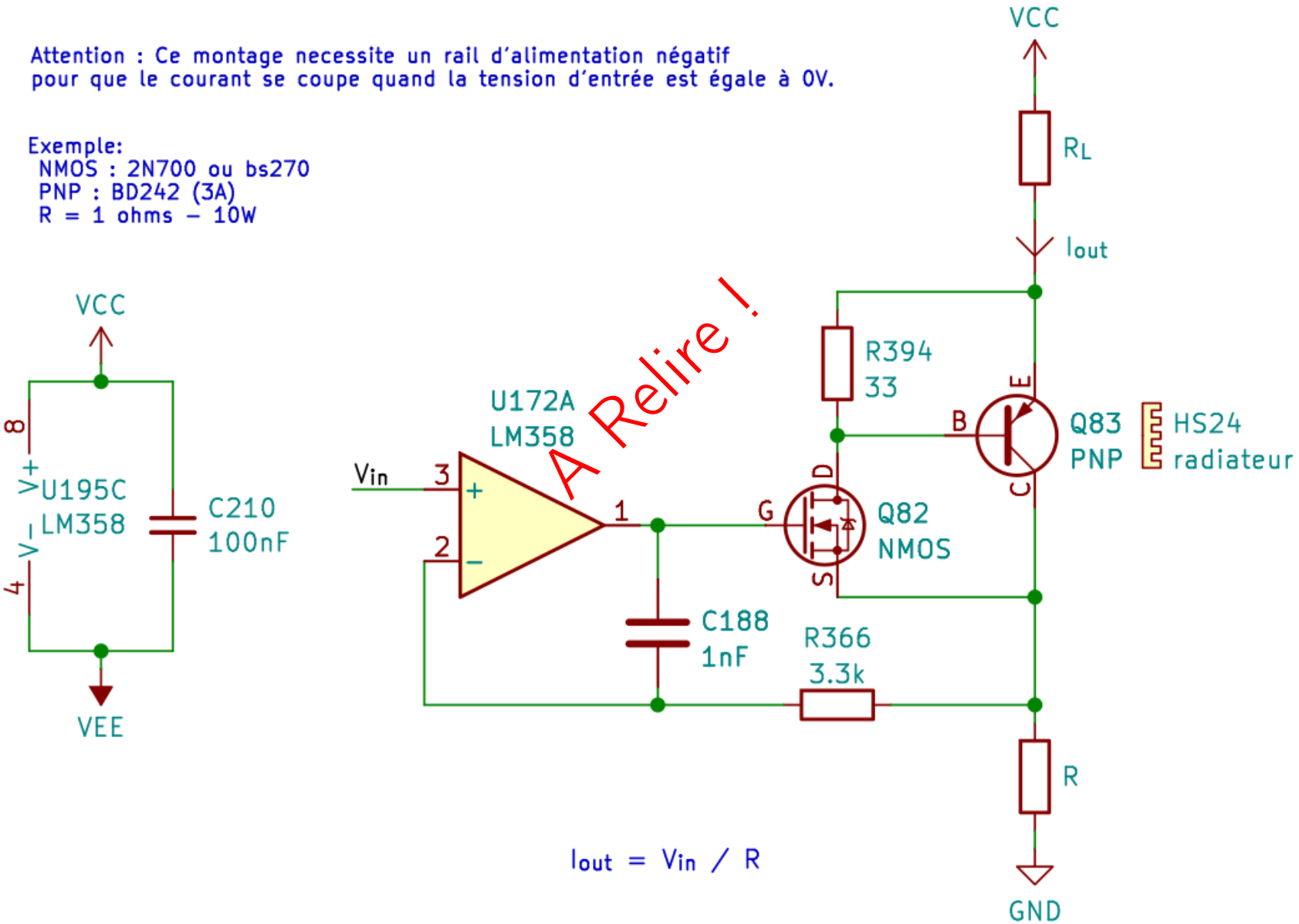
$I_{in} = (V_{out} - V_{ref}) / R$

Convertisseur tension-courant de puissance – montage NPN



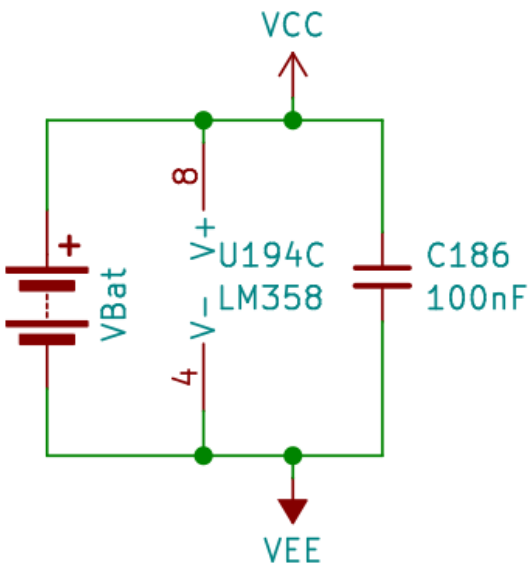
(version longue)

Convertisseur tension-courant de puissance – montage PNP

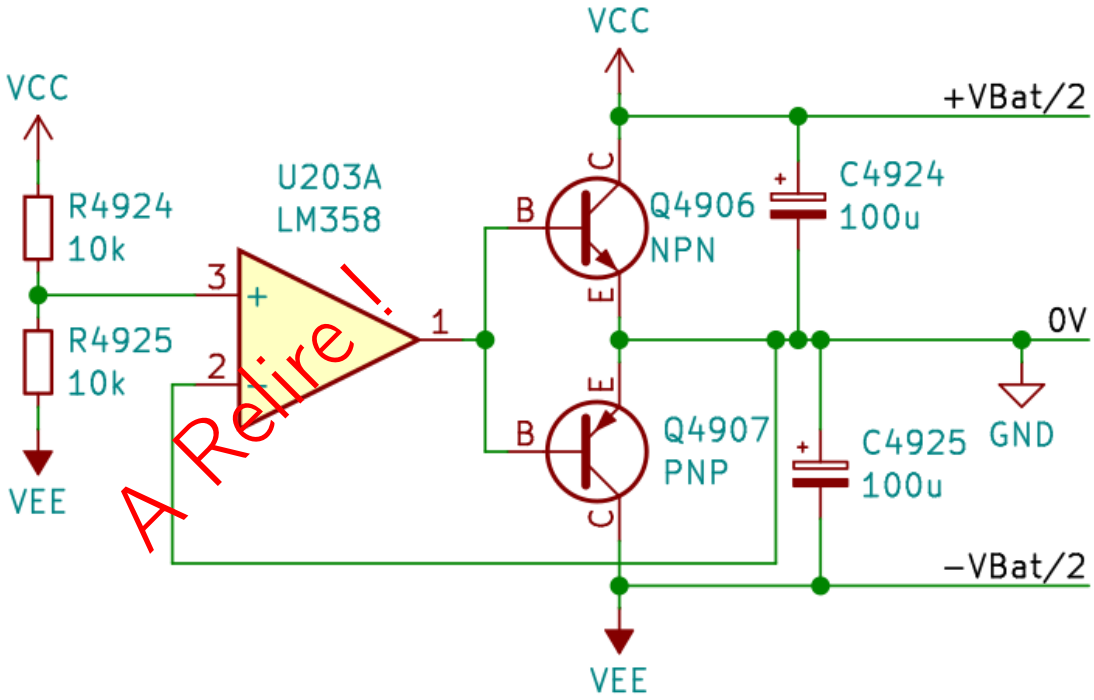


longue)

Créer deux rails d'alimentations à partir d'une alimentation – 1/3



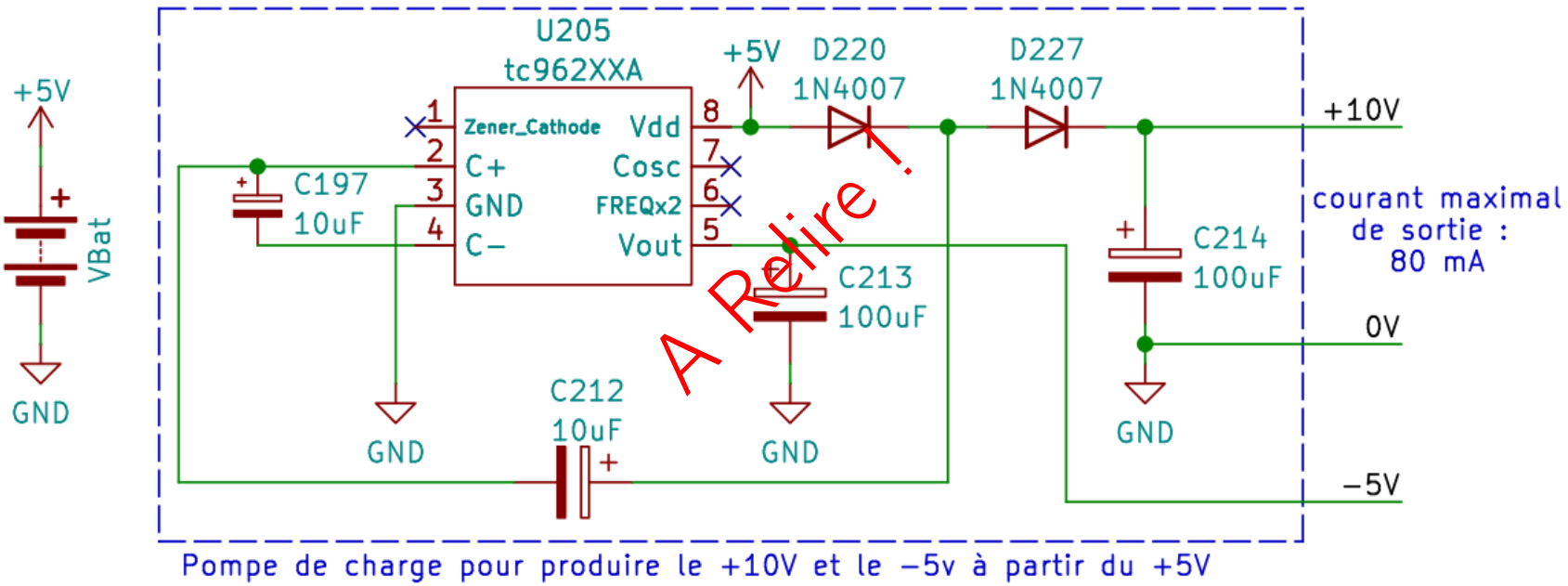
Exemples 1:
(courants faibles)
NPN : 2N2222 (600 ma)
PNP : 2N2907 (600 ma)



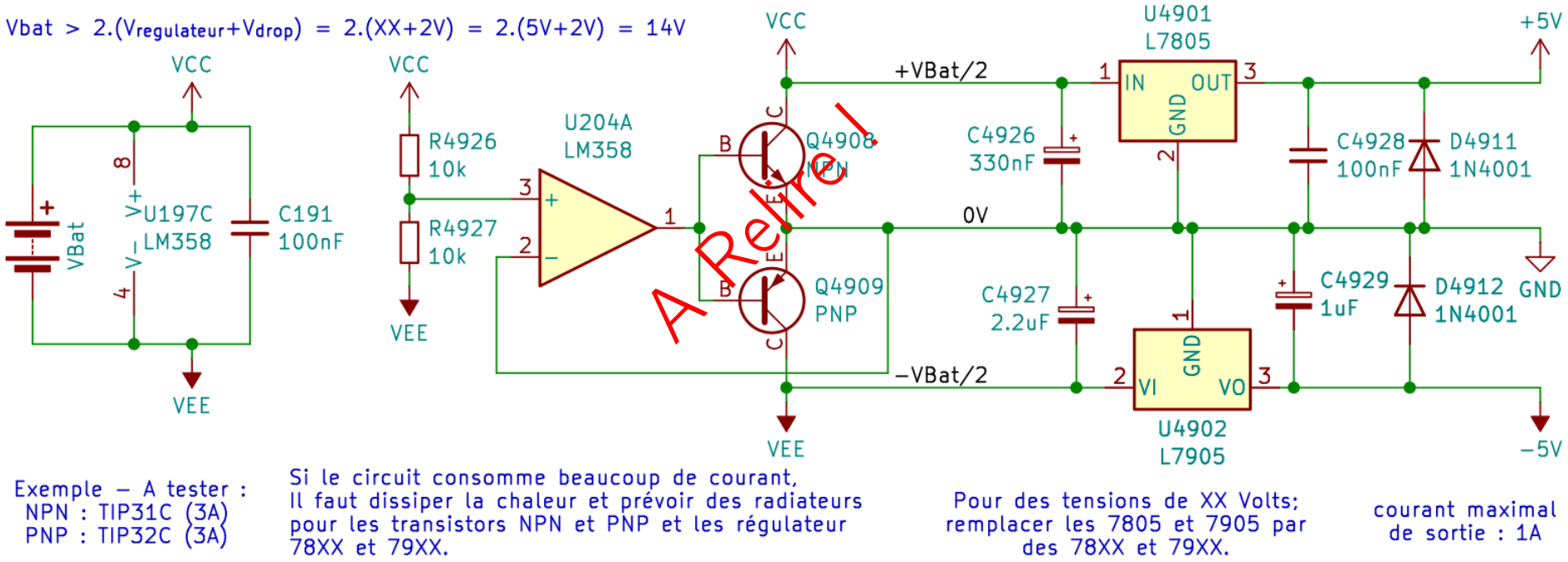
Exemples 2 -- A tester ! :
(courants moyens)
NPN : TIP31C (3A)
PNP : TIP32C (3A)

Exemples 2 -- A tester ! :
(courants forts)
NPN : MJL21193 (16A)
PNP : MJL21194 (16A)

Créer deux rails +10/-5V d'alimentations à partir de 5V – 2/3



Créer deux rails d'alimentation stabel à partir d'une batterie (en utilisant avec des régulateurs) – 3/3



Exemple – A tester :
NPN : TIP31C (3A)
PNP : TIP32C (3A)

Si le circuit consomme beaucoup de courant,
Il faut dissiper la chaleur et prévoir des radiateurs pour les transistors NPN et PNP et les régulateur 78XX et 79XX.

Pour des tensions de XX Volts; remplacer les 7805 et 7905 par des 78XX et 79XX.

courant maximal de sortie : 1A

(version longue)

Plan

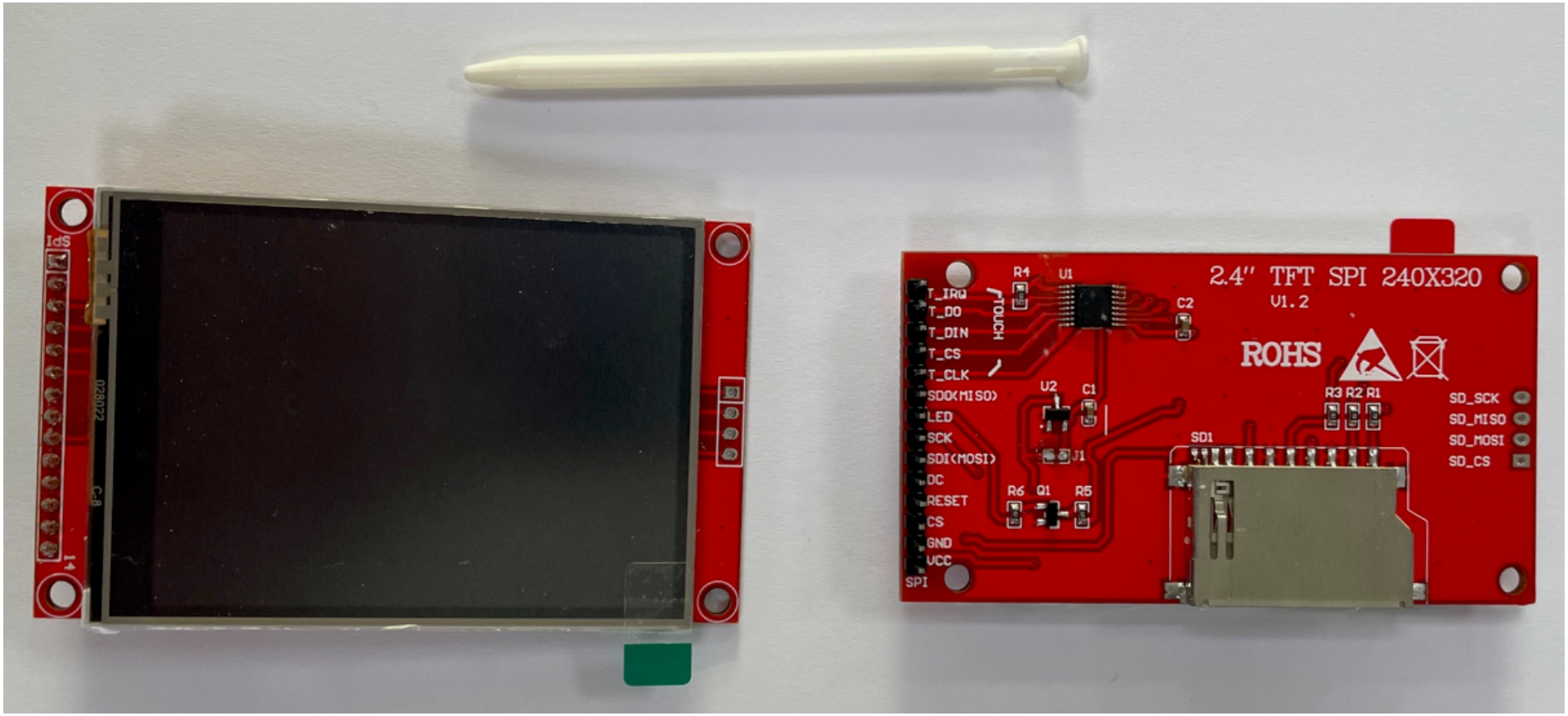
(version longue)

Plan

- Les Écran LCD
- Les rubans leds adressables
- Les rubans leds non adressable
- Module peletier

Présentation des écrans LCD – 1/2

Voici un exemple d'écran LCD. Il s'agit de l'écran LCD 240x320 – ST7789 – XPT2046 – SPI possédant un pad sensitif :



(version longue)

Présentation des écrans LCD – 2/2

Ces écrans contiennent généralement 3 périphériques internes indépendants :

- un écran LCD (piloté par les drivers ST7789, ILI9341, ...);
- un pad sensitif (piloté par un drivers compatible XPT2046);
- un connecteur pour carte SD (pour stocker les images à afficher).

qui comuniquent avec la carte à l'aide du protocole SPI.

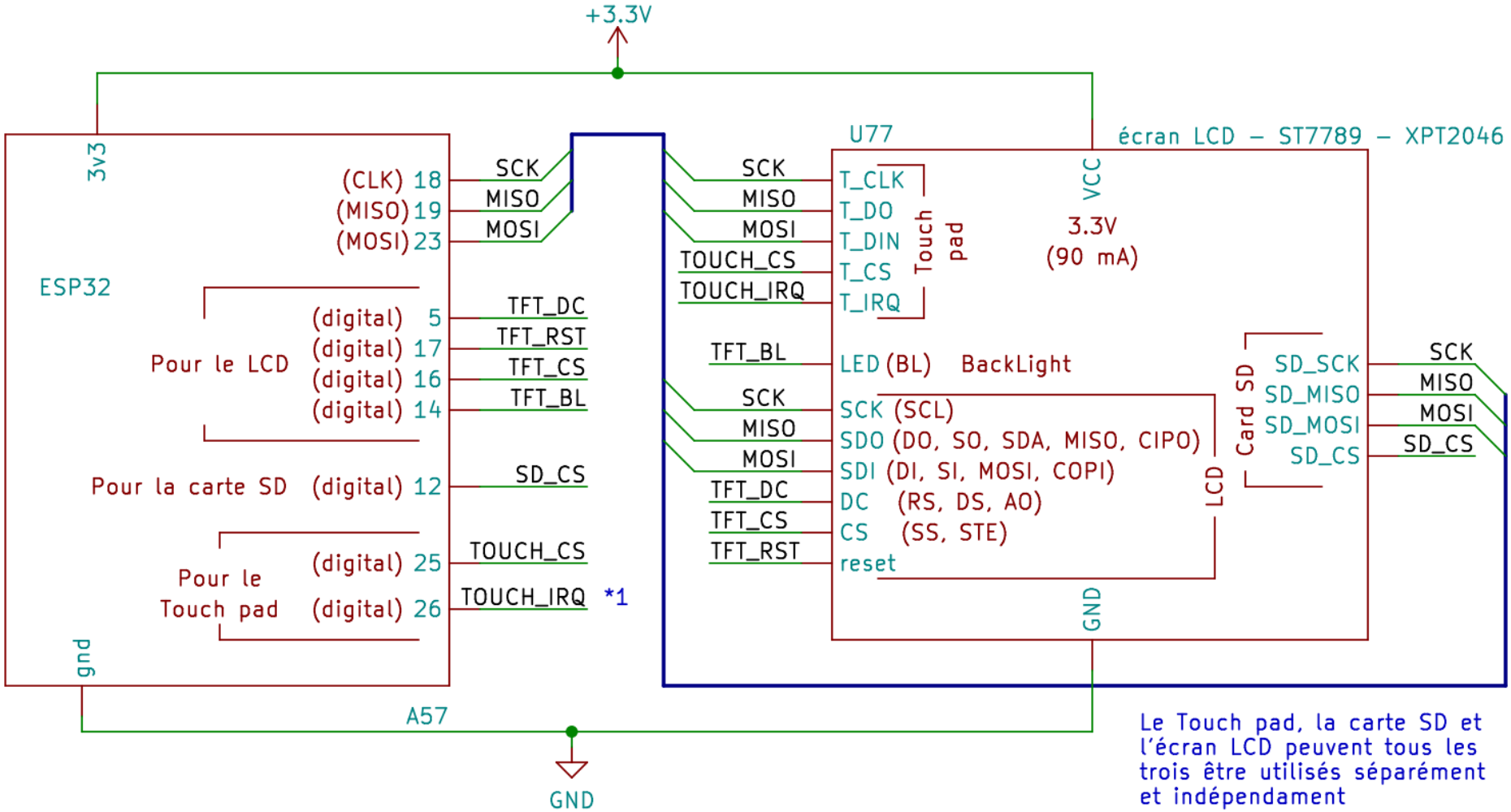
Il existe de nombreuses versions d'écran LCD. Le site suivant les rpértorie et propose des schémas et programmes pour les piloter:

Wiki sur le fonctionnement des cartes LCD.,

Il existe de nombreuses bibliothèques pour les piloter.

- Pour les ESP32, il y a la bibliothèque bodmer/TFT_eSPI. Elle gérer à la fois l'écrans LCD et le pad sensitif.
- Pour les cartes arduinos, il faut utiliser une combinaison de deux bibliothèques.
  - Pour l'écran LCD, il faut utiliser la bibiothèque AdaFruit_YYYY (où YYYY est le nom du driver LCD).
  - Pour le pad sensitif, il faut utiliser la bibliothèque paulstoffregen/XPT2046_Touchscreen.

Schéma ESP32 – LCD – ST7789 – XPT2046 – SPI



On relie les 3 broches SPI des trois compsants sur le même port SPI de l'ESP32, celui configuré par défault par les bibliothèque de l'EPS32 (l'ESP32 contient 2 ports SPI). La ligne bleue symbolise un bus qui contient 3 fils séparés, un fil qui relie toutes les broches MISO ensenble, un autre qui relie tous les broches MOSI ensemble et un dernier pour CLK. Ne reliez donc surtout pas MISO, MOSI et SCK ensemble !

(version longue)

Programme pour esp32 pour l'écran LCD – ST7789 – 1/2

code/lcd_touch/platformio.ini

```
[env:upesy_wroom]
platform = espressif32
board = upesy_wroom
framework = arduino
lib_deps =
  bodmer/TFT_eSPI
build_flags =
  -Os
  -DCORE_DEBUG_LEVEL=ARDUHAL_LOG_LEVEL_DEBUG
  -DUSER_SETUP_LOADED=1
  -DST7789_DRIVER=1
  -DTFT_MISO=19
  -DTFT_MOSI=23
  -DTFT_SCLK=18
  -DTFT_CS=16
  -DTFT_DC=5
  -DTFT_RST=17
  -DTFT_BL=14
  -DTOUCH_CS=25
  -DLOAD_GLCD=1
  -DLOAD_FONT2=1
  -DLOAD_FONT4=1
  -DLOAD_FONT6=1
  -DLOAD_FONT7=1
  -DLOAD_FONT8=1
  -DLOAD_GFXFF=1
  -DSMOOTH_FONT=1
  -DSPI_FREQUENCY=27000000
```

Nous allons uiliser la bibliothèque bodmer/TFT_eSPI avec la chaîne de compilation Palftom.io.

Pour installer et utiliser Platform.io, consultez la page 535.

Le fichier platform.ini du projet est le suivant. Il permet de configurer les broches de l'écran LCD, les polices de caractère et la fréquence de l'horologe du protocole série SPI.

Programme pour esp32 pour l'écran LCD – ST7789 – 2/2

```cpp
#include "SPI.h"
#include "TFT_eSPI.h"

TFT_eSPI tft = TFT_eSPI();
String msg("Hello world !");
int bg_color = TFT_BLACK, fg_color = TFT_WHITE;

void setup() {
  tft.init();
  tft.setRotation(0);
  tft.fillScreen(bg_color);
  tft.setTextColor(fg_color);
  tft.setTextSize(3);
  tft.setCursor(0, 0);
  tft.println(msg.c_str());
  int top_left_x = 50, top_left_y = 60;
  int width = tft.width()/2, height = tft.height()/2;
  tft.drawRect(top_left_x, top_left_y, width, height, TFT_GREEN);
}

void loop(void) {
  uint16_t x, y;
  if (tft.getTouch(&x, &y)){
    tft.setCursor(0, 0);
    tft.setTextColor(bg_color);
    tft.println(msg.c_str()); // On efface l'ancien texte

    msg = String("X : ") + String(x) + String(", Y : ") + String(y);
    tft.setCursor(0, 0);
    tft.setTextColor(fg_color);
    tft.println(msg.c_str()); // On affiche le nouveau texte
  }
}
```

code/lcd_touch/lcd_touche_with_TFT_eSPI.cpp

Quand on touche l'écran, la position de l'impact est écrit sur l'écran.

La modification de l'affichage se fait pixel par pixel. Ainsi, il est plus rapide, pour effacer du texte ou un dessin, de réécrire par dessus le même texte ou dessin avec la couleur de fond.

(version longue)

Trouver d'autres programme pour esp32 pour l'écran LCD

Consulter les exemples de la bibliothèque TFT_eSPI pour connaître tous les effets graphiques qu'il est possible de faire:

Le dépot git de la bibliothèque TFT_eSPI,
Les exemples de cette bibliothèque,

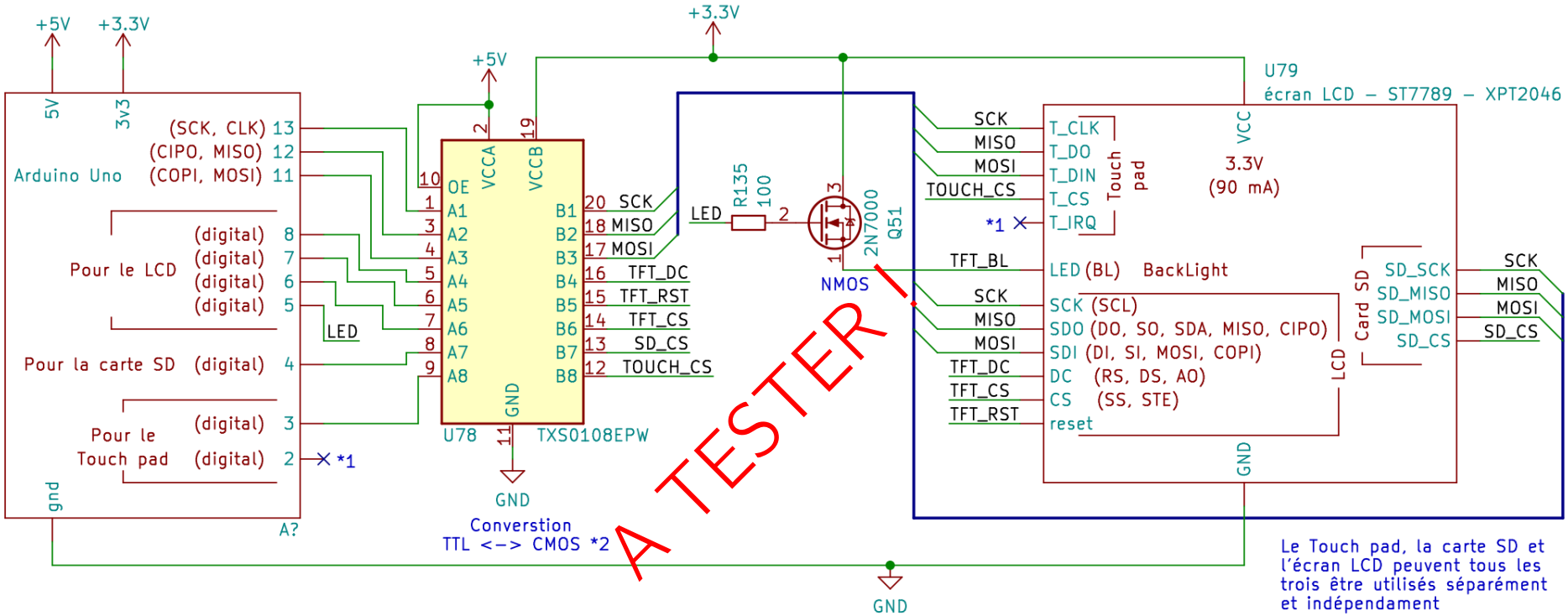
Pour afficher une image issue de la carte SD, vous pouvez utilisez l'exemple :

A FAIRE : Mettre le liens qui permet d'afficher une image

Pour afficher une démonstration complète des possibilités graphiques, vous pouvez tester l'exemple :

Une démo complète des posibilité graphique de la biblithèque.,

Schéma Arduino – LCD – ST7789 – XPT2046 – SPI



On rapelle que la ligne bleue symbolise un bus qui contient 3 fils séparés.

*1 : Si vous avez besoin de l'IRQ de l'écran sensitif, vous devez aussi le convertir en 3V3 et le relié à la broche 2.

*2 : Comme les sorties de l'Arduino UNO sont en 5V, il faut convertir les signaux de 6V en 3V3. Vous pouvez utilisez des modules pret à l'emploi à base de TXS0108E. Sinon, consultez la page 507 concernant les conversion TTL/CMOS.

(version longue)

Programme pour Arduino UNO pour l'écran LCD – ST7789

Ce programme utilise la bibliothèque .

A FAIRE :

A FAIRE :

Trouver d'autres programme pour Arduino UNI pour l'écran LCD

Consulter les exemples de la bibliothèque A FAIRE : pour connaître tous les effets graphiques qu'il est possible de faire:

A FAIRE : mettre le liens de la doc sur github A FAIRE : mettre le liens des exemples sur github

Pour afficher une image issue de la carte SD, vous pouvez utilisez l'exemple :

A FAIRE : Mettre le liens qui permet d'afficher une image

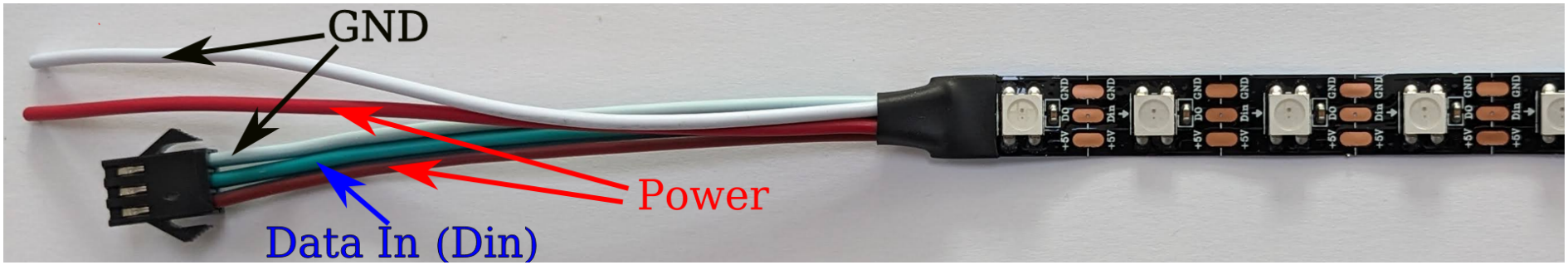
Pour afficher une démonstration complète des possibilités graphiques, vous pouvez tester l'exemple :

A FAIRE : Mettre le liens de TFT_graphictest

Plan

Présentation des rubans leds adressables

Les couleurs de chacune des leds d'un ruban adressable peuvent être configurées séparément en comunicant à l'aide d'un protocole série sur le fil Din. Attention ! Ces rubans ont un sens et l'arduino doit être connecté du côté Din, mais pas du côté Dout.

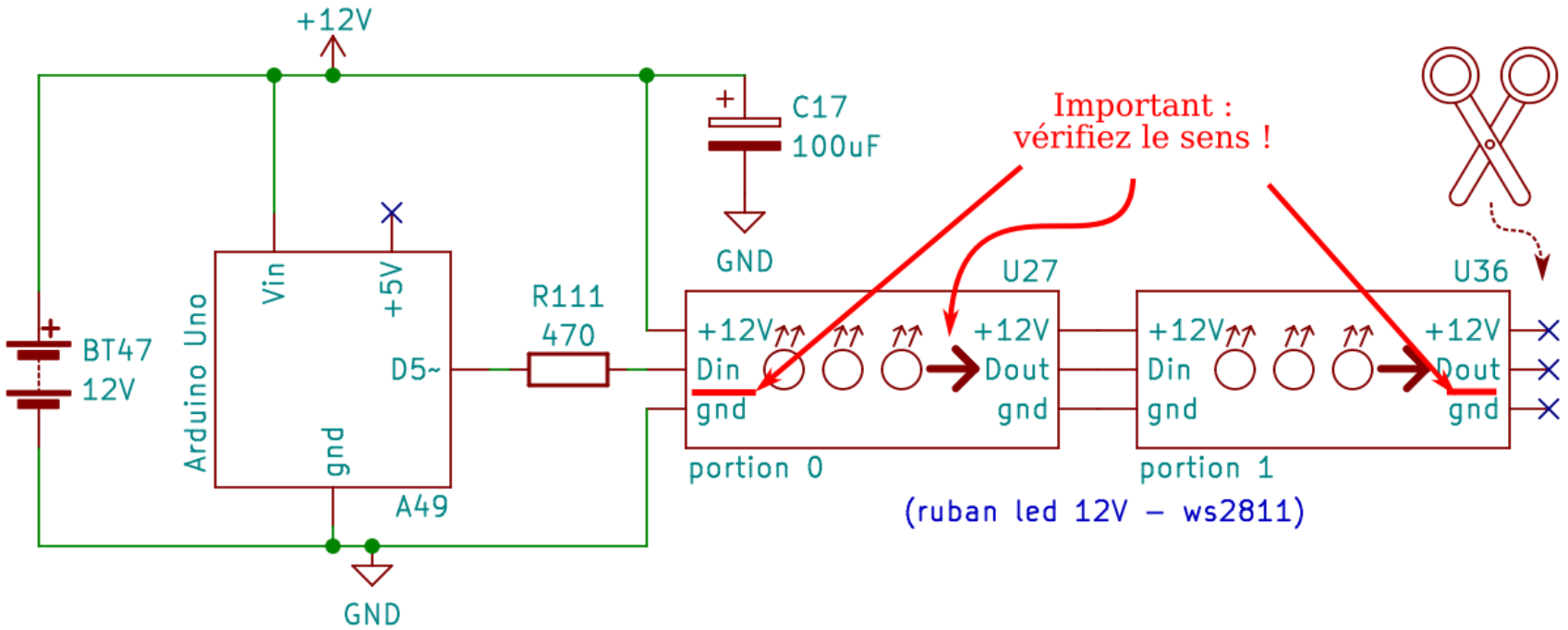


Ruban Led 5V ws2812E

Une portion d'un ruban Led 12V ws2811

(version longue)

Montage 1/7 pour un ruban LED 12V et un Arduino Uno



Le condensateur C17 sert à stabiliser l'alimentation et éviter les chutes de tension pour le microcontrolleur comme pour les leds (les leds peuvent se mettre à scintiller lorsqu'elles sont éteintes). Souvent ce condensateur n'est pas nécessaire.

La résistance R111 sert à stabiliser le signal de transmission limitant les pics de tensions et les oscillations indésirables dues aux effets capacitifs et inductifs des câbles et pistes. Dans la plupart des applications cette résistance n'est pas nécessaire.

(version longue)

Programmer une carte pour piloter un ruban led adressable

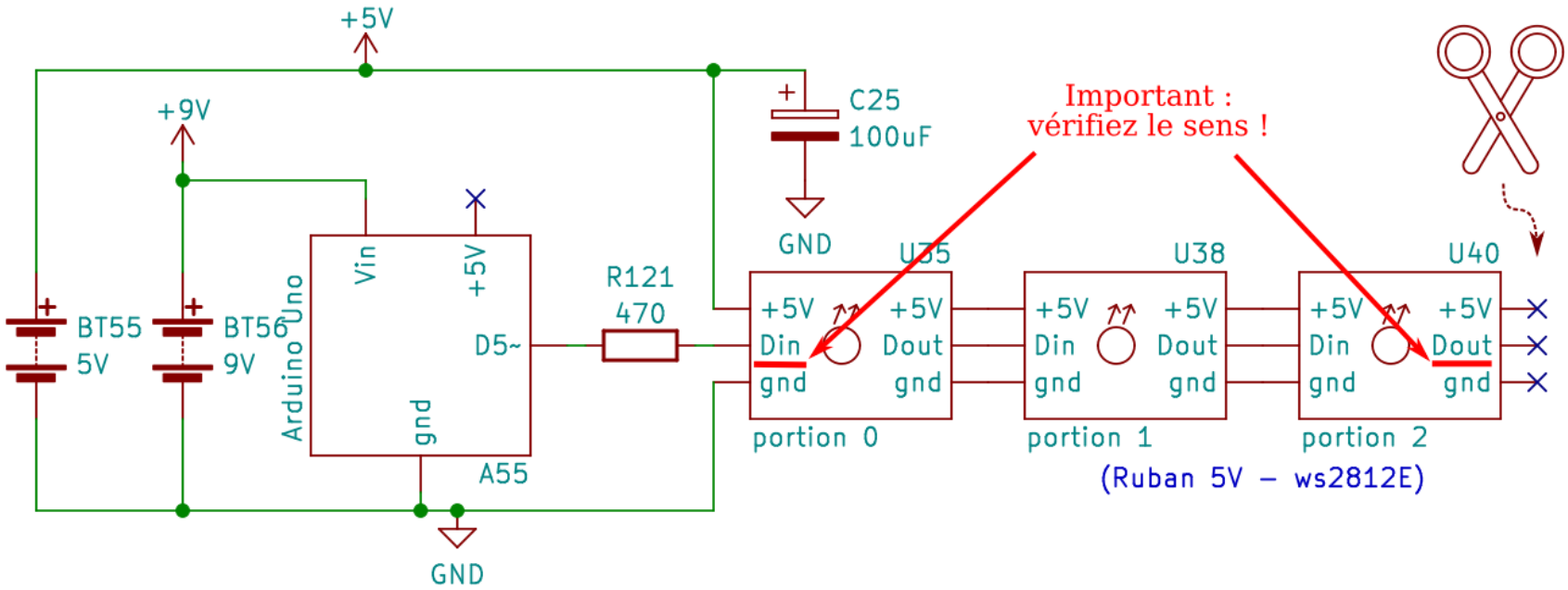
Vous devez installer la bilbiothèque Adafruit NeoPixel. Changer l'ordre R, G et B dans NEO_RGB pour afficher les couleurs correctement.

```cpp
#include <Adafruit_NeoPixel.h>

const int LED_STRIP_PIN = 5;
const int LED_NUMBER = 32;
// NEO_BRG : l'ordre des couleurs (R-rouge, G-vert, B-bleu). NEO_KHZ800 : la frequence de mise a jour.
Adafruit_NeoPixel led_strip(LED_NUMBER, LED_STRIP_PIN, NEO_RGB + NEO_KHZ800);

void setup() {
  led_strip.begin();
  int brightness = 100; // un entier entre 0 et 255 inclus. 0 = off, 255 = on
  led_strip.setBrightness(brightness); // Ne pas utiliser au milieu d'une animation.
}

void loop() {
  // Eteind le ruban de led et met les trois premieres protions aux coleurs R, G et B.
  led_strip.clear();
  led_strip.setPixelColor(0, led_strip.Color(255, 0, 0)); // Rouge
  led_strip.setPixelColor(1, led_strip.Color(0, 255, 0)); // Vert
  led_strip.setPixelColor(2, led_strip.Color(0, 0, 255)); // Bleu
  led_strip.show(); delay(1000);

  // Allume le ruban de led..
  for(int i=0; i<LED_NUMBER; i++) {
    // Couleur = entier entre 0 et 255 inclus
    //int red=245; int green=244; int blue=238; // Blanc naturel
    int red=255; int green=210; int blue=70; // Blanc plus chaud
    led_strip.setPixelColor(i, led_strip.Color(red, green, blue));
  }
  led_strip.show(); delay(1000);
}
```

code/led_strip.cpp

(version longue)

Montage 2/7 pour un ruban LED 5V et un Arduino Uno



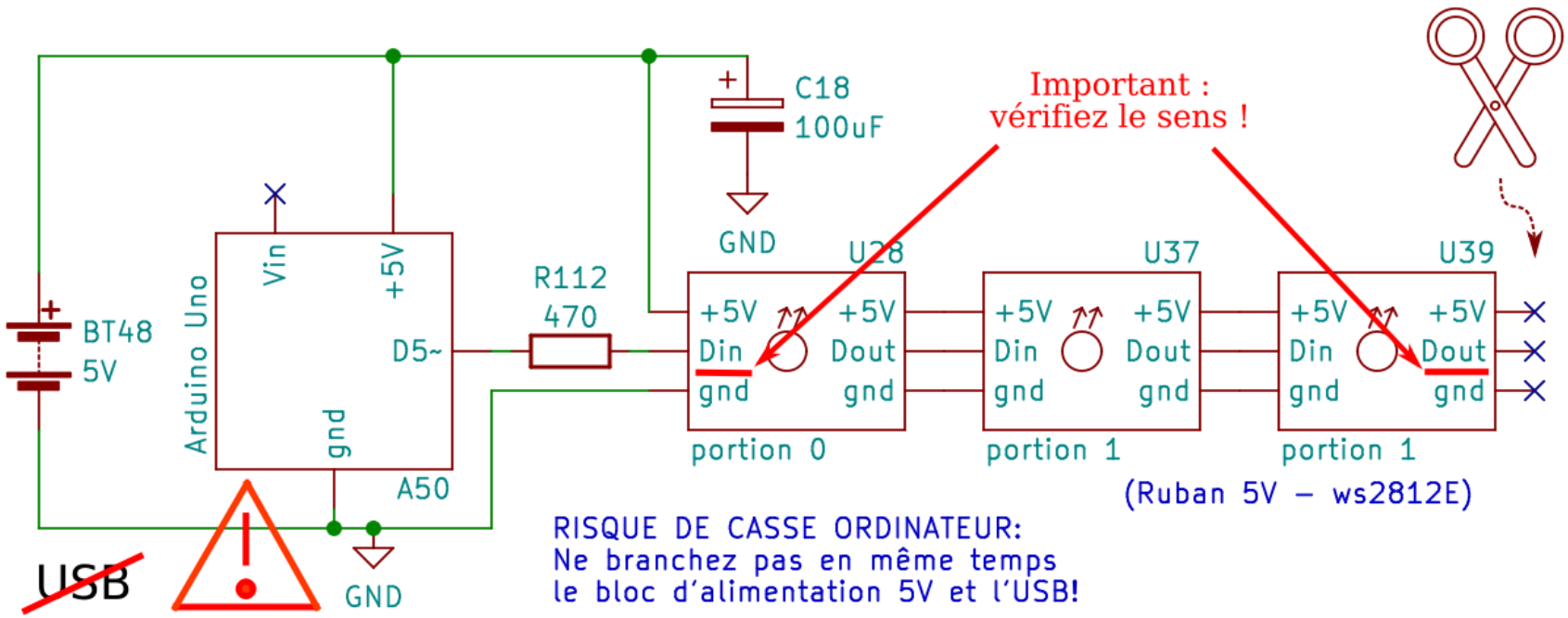
Pour choisir l'alimentation, il faut determiner celle qui doit fournir le plus de puissance. On met une batterie ou un bloc d'alimentation pour cette alimentation et un régulateur (buck, si on doit élever la tension, et boost, si on doit abaisser la tension) pour l'autre.

Par exemple, imaginons que la batterie BT55 doit fournir une puissance 4W et que la batterie BT56 doit fournir 1W. Alors, on choit, pour BT55 un bloc d'alimetation 4W + 1W pour alimenter à la fois le ruban et le régulateur boost 9V 1W qui sera BT56 et fournira la tention de 9V.

Pour faire le circuit boost ou buck consulter la page 442.

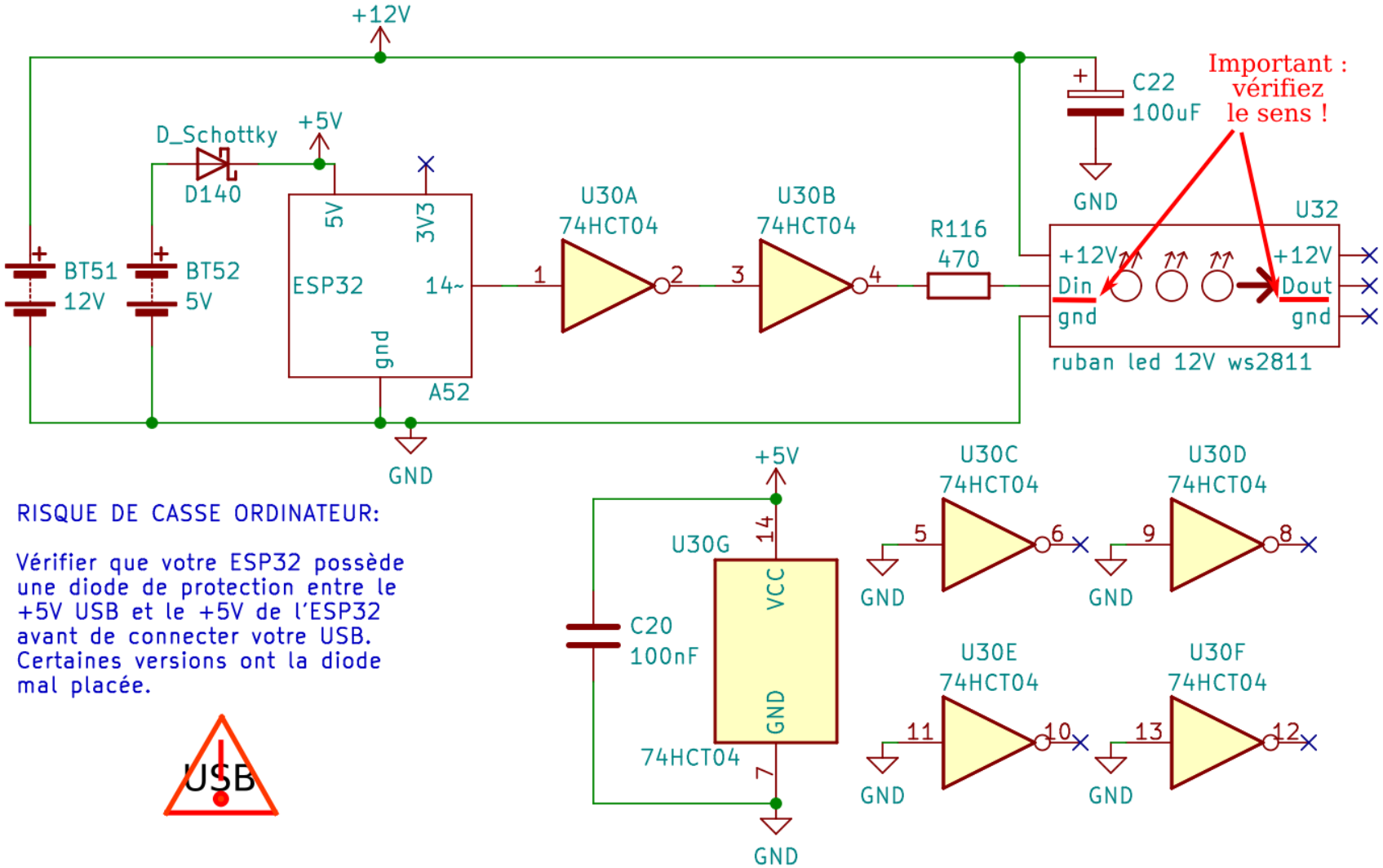
(version longue)

Montage 3/7 pour un ruban LED 5V et un Arduino Uno



Consulez la page 628 pour comprendre pourquoi il ne faut pas connecter l'USB dans ce cas.

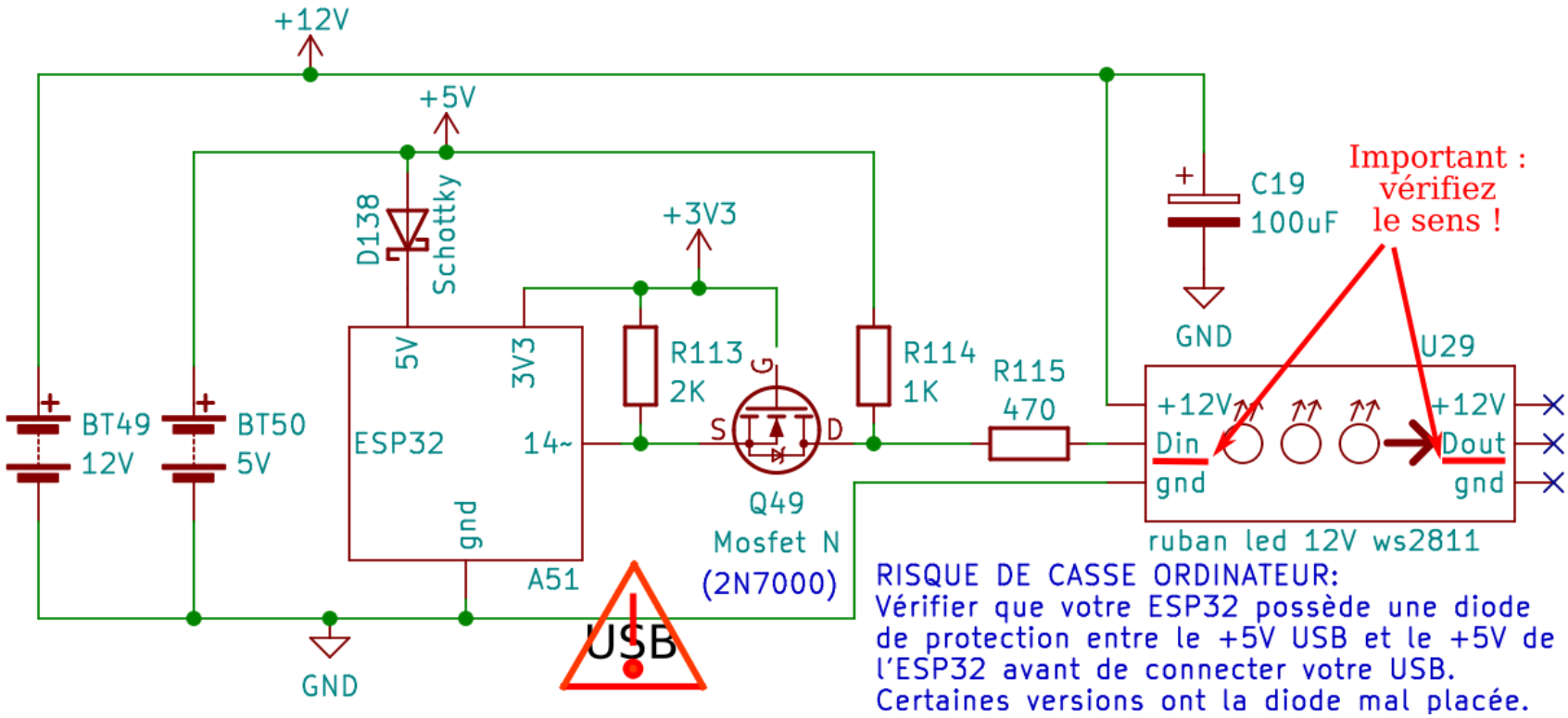
Montage 4/7 pour un ruban LED 12V et un ESP32



Pour choisir les alimentations consulter la page 512.

Consulez les page 634 et 638 pour comprendre quand on peut connecter l'USB.

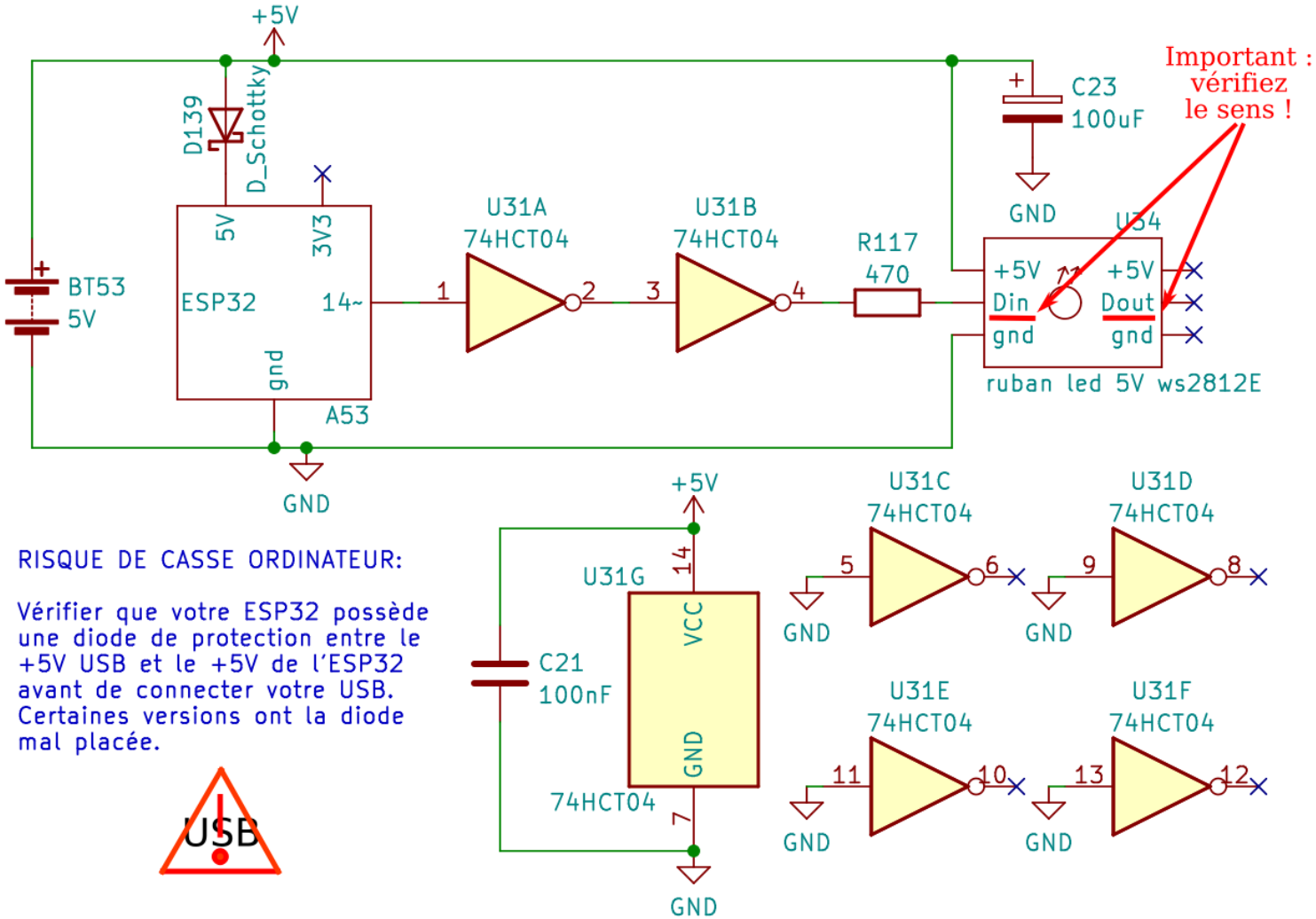
Montage 5/7 pour un ruban LED 12V et un ESP32



Pour choisir les alimentations consulter la page 512.

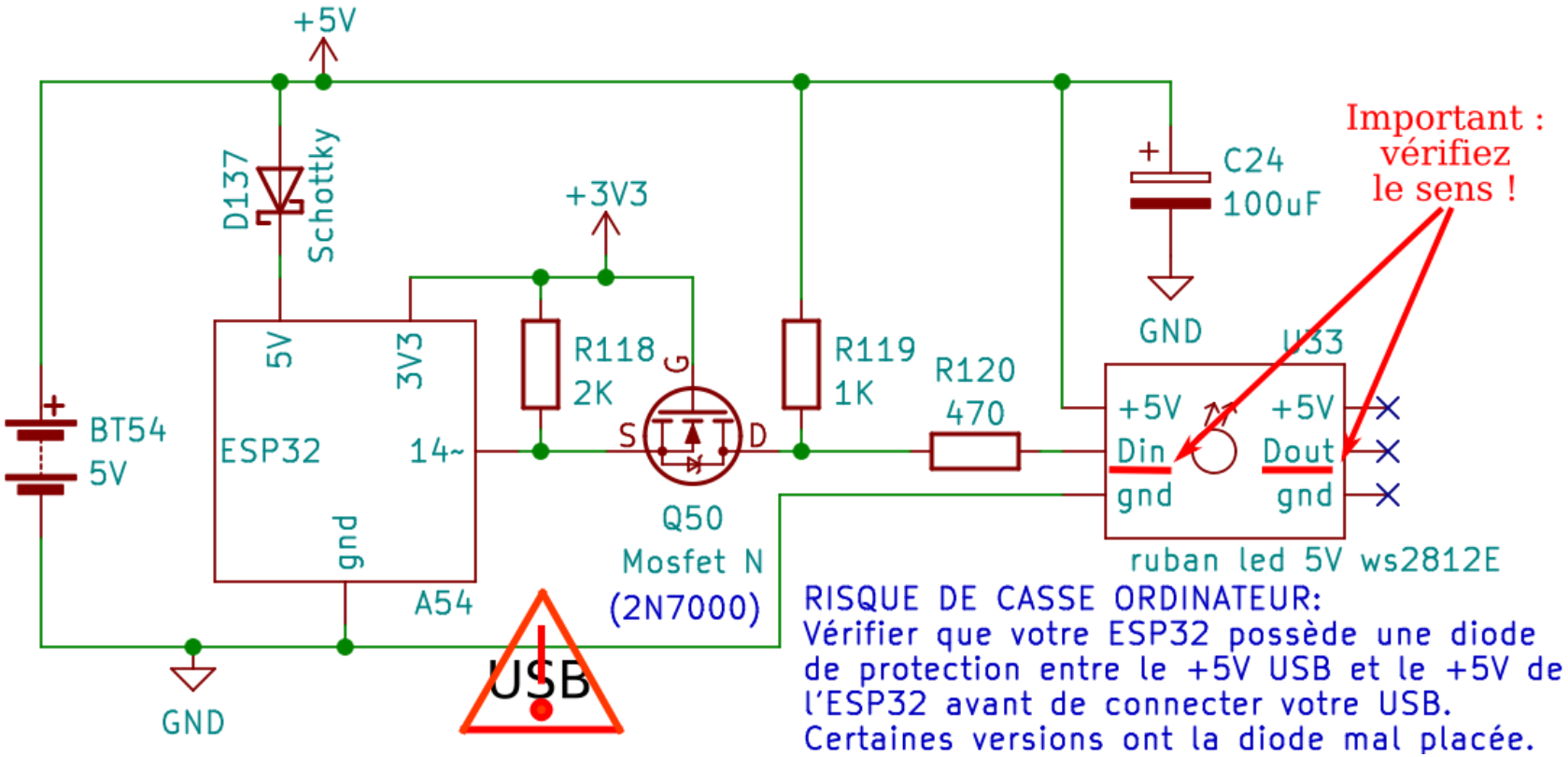
Consulez les page 634 et 638 pour comprendre quand on peut connecter l'USB.

Montage 6/7 pour un ruban LED 5V et un ESP32



Consulez les page 634 et 638 pour comprendre quand on peut connecter l'USB.

Montage 7/7 pour un ruban LED 5V et un ESP32

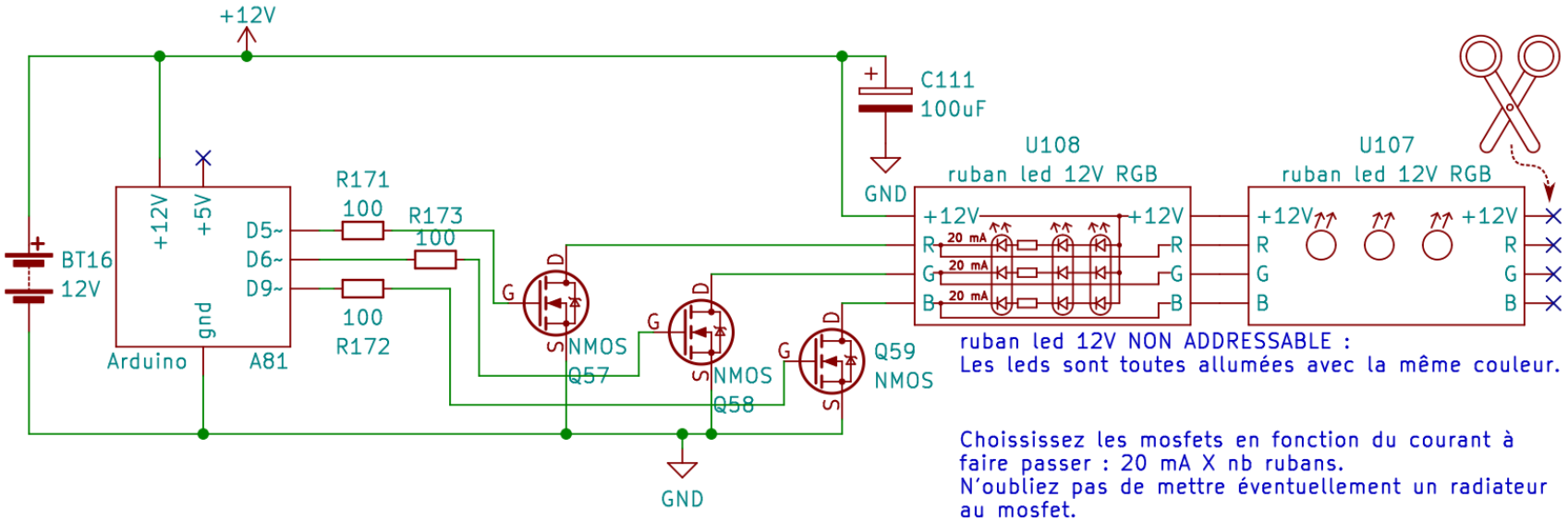


Consulez les page 634 et 638 pour comprendre quand on peut connecter l'USB.

Plan

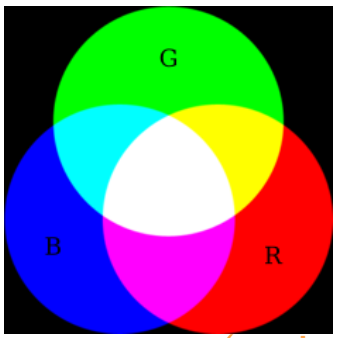
- Les Écran LCD
- Les rubans leds adressables
- Les rubans leds non adressable
- Module peletier

Montage 1/2 pour un ruban LED non adressable 12V et un arduino



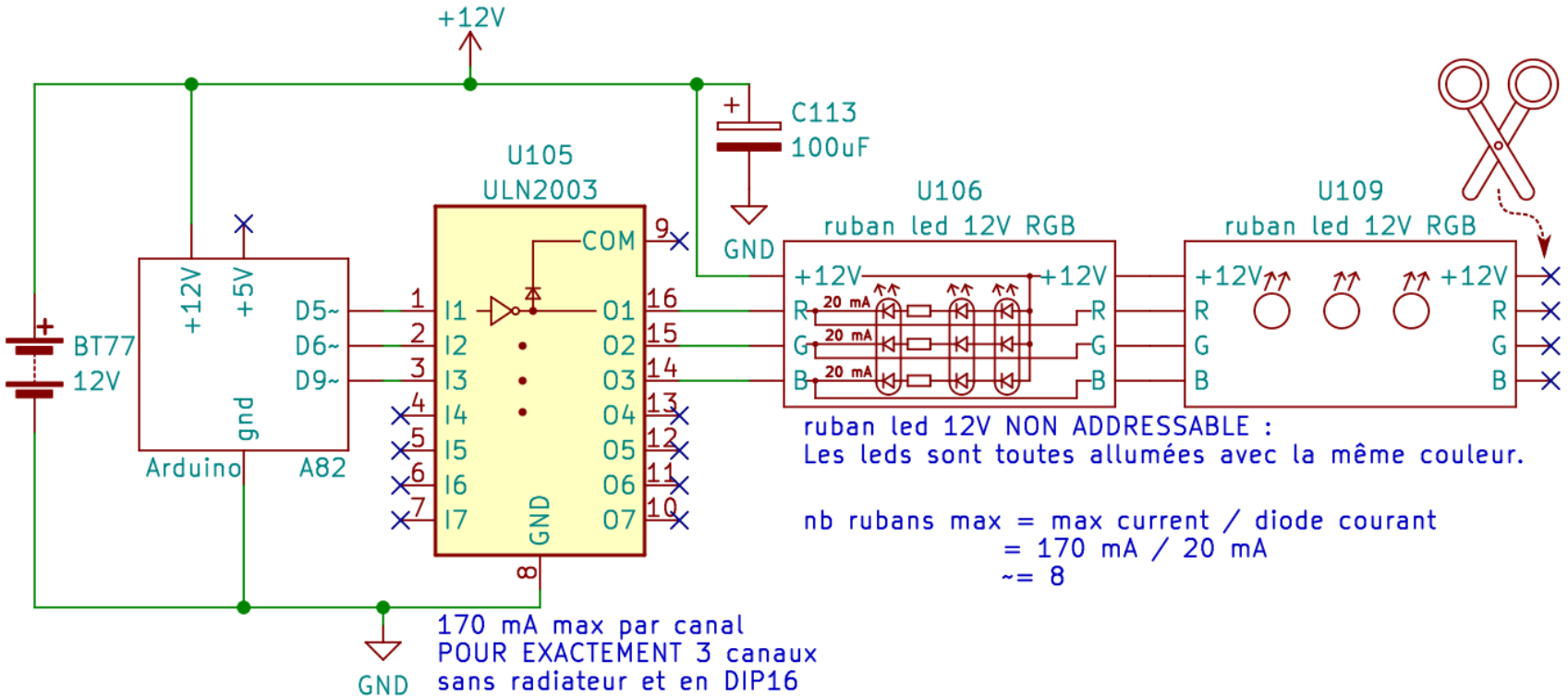
La lumisoité de chaque canal de couleur est proportionnel au courant qui passe dans les diodes. Elle se règle à l'aide d'une PWM. Par exemple, pour choisir la couleur brun (de code RGB : (165,42,42)) on configure les PWM ainsi :

- Canal rouge (broche D5) : $(165/256) * 100 = 64\%$ PWM;
- Canal vert (broche D6) : $(42/256) * 100 = 16\%$ PWM;
- Canal vert, (broche D9) : $(42/256) * 100 = 16\%$ PWM.



(version longue)

Montage 2/2 pour un ruban non adressable et un arduino



Le driver ULN2003 peut être utilisé pour piloter des leds. Ce composant permet de piloter un nombre assez limité de rubans. Il faut consulter les spécifications du composant pour détermier par le calcul ce nombre.

$$P_{max} = 170\ \text{mA} \times 1.24\ \text{V} \times 3\ \text{canaux} = 0.64\ \text{W}$$

$$T_{max} = T_{\text{ambiant}} + P_{max} \times R_{jonction-ambiant} = 35^{\circ}\text{C} + 0.64\ \text{W} \times 70^{\circ}\text{C/W} = 79.8^{\circ}C$$

Comme pour le schéma précédent, chaque canal se pilote à l'aide d'une PWM.

(version longue)

Plan

- Les Écran LCD
- Les rubans leds adressables
- Les rubans leds non adressable
- Module peletier

Présentation du module Peletier

A FAIRE : Mettre une image

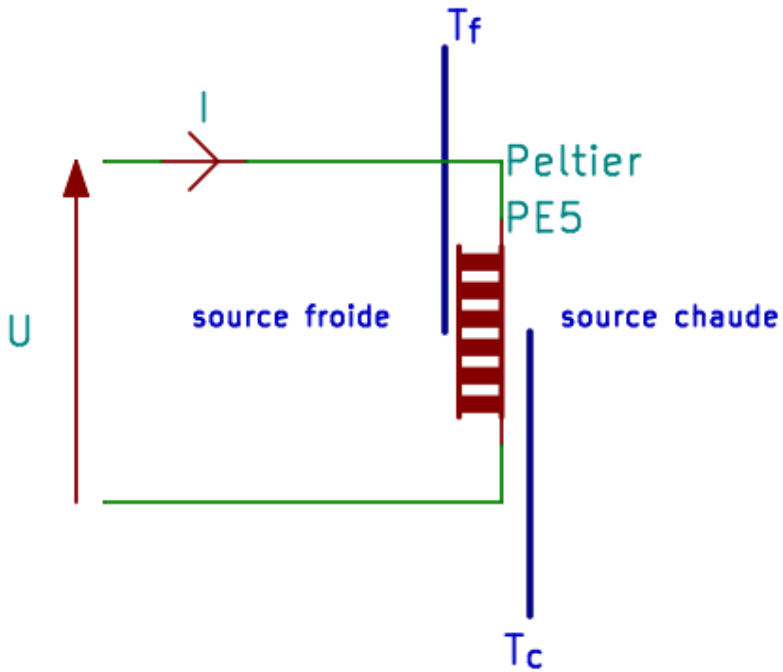
Un module peletier sert à refroidir ou réchauffer un milieu en le mettant en contact sur une surface du module.

Lorsque l'on applique une tension au borne du module peletier un transfert thermique apparait entre les deux plaques de céramiques du module. cela pemet de transdérer de la chaleur d'un mlieu à un autre, reffroisdissant l'un et réchauffant l'autre.

Si l'on inverse la tension au borne du module peletier, on inverse le sens du transfert thermique.

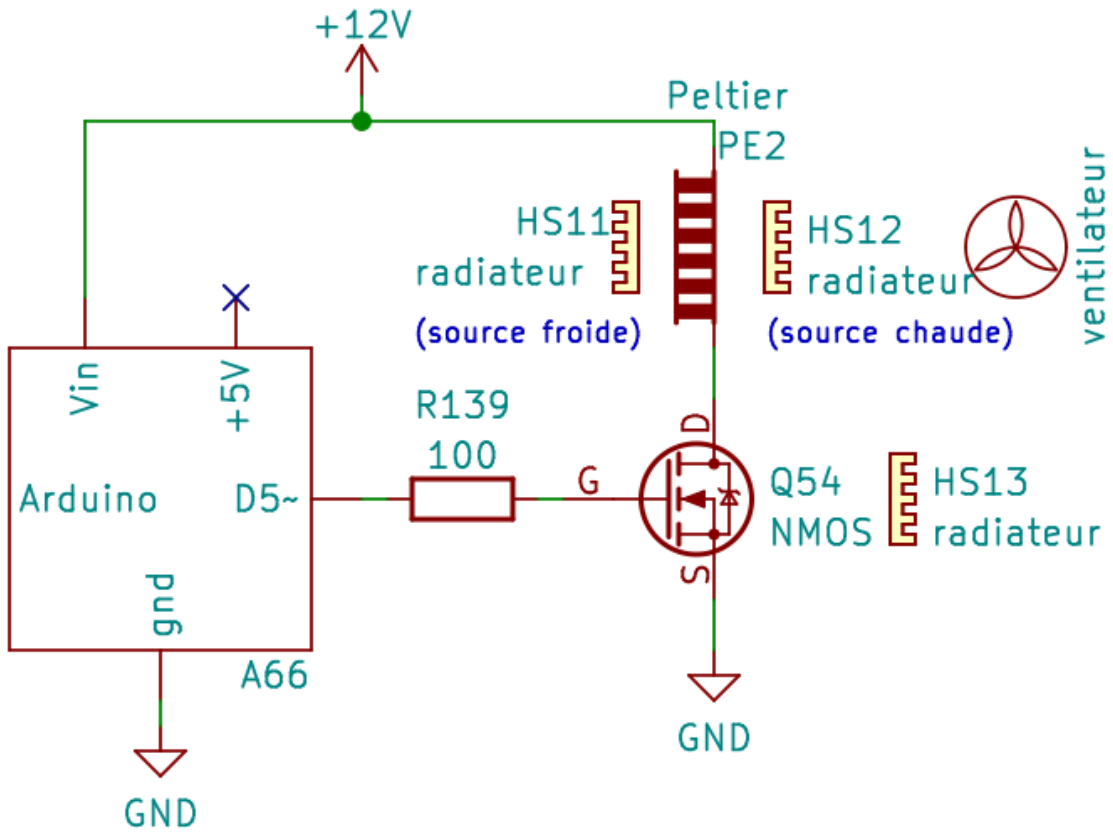
Référence : Modeling and Analysis of Thermoelectric Modules, Simon Lineykin and Shmuel Ben-Yaakov, IEEE Transactions on industry applications, volume 43, Number 2, march/april 2007.

Schéma du module peletier



U : Tension au borne du module peletier (Volt, V),
I : Courant traversant le module peletier (Ampère, A),
T_f : température à la surface de la source froide (degrés Celsus, °K),
T_c : température à la surface de la source chaude (degré Celsus, °K),

Contrôle simple, en tension, d'un module peletier



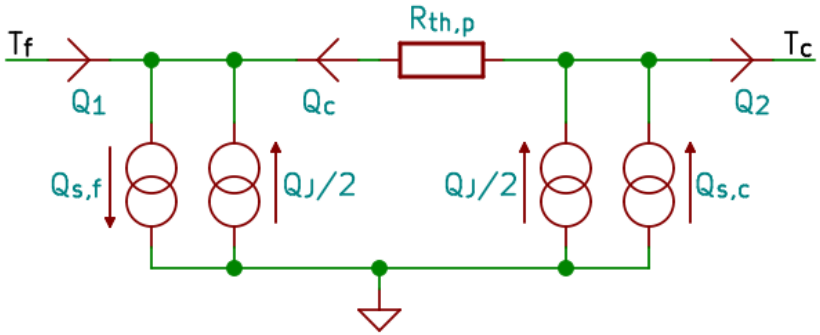
Faites attention, vous devez toujours faire en sorte que la différence de température entre source chaude et source froide ne dépasse pas une certaine valeur !

Il faut mettre des radiateurs sur les sources chaudes et froides. On met aussi un ventilateur sur la source en contact avec l'air ambiant. Donc, lorsque l'on utilise le module peletier pour refroidir quelque chose, on met un ventilateur sur la sourche chaude. Si on l'utilise pour chauffer, on met le ventilateur sur la source froide.

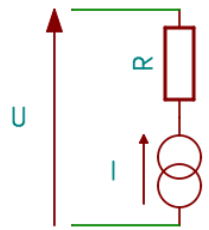
(version longue)

Modèle thermique et électrique en régime permanent

Modèle thermique :



Modèle électrique :



$$
\begin{aligned}
Q_1 &= Q_{s,f} - \frac{1}{2} Q_J - Q_c \\
Q_2 &= Q_{s,c} + \frac{1}{2} Q_J - Q_c
\end{aligned}
$$

$Q_{s,f}$	=	$S \times T_f \times I$	Effet de Seebeck
$Q_{s,c}$	=	$S \times T_c \times I$	Effet de Seebeck
Q_J	=	$R \times I^2$	Effet Joule
Q_c	=	$\frac{T_c - T_f}{R_{th,p}}$	Conduction source chaude/froide

$$U = R \times I + S \times (T_c - T_f) \quad \text{Modèle électrique}$$

S : Coefficient de Seebeck (V/°K)

R : Résidtance électrique (Ω)

$R_{th,p}$: Résistance thermique (°K/W)

(version longue)

Calculer les coefficients à partir des spécifications constructeurs

Les constructeurs donnent les informations suivantes :

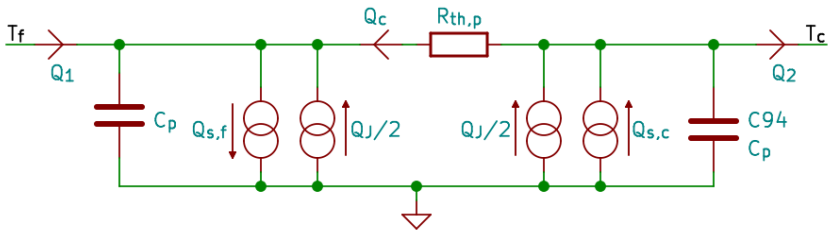
- T_h : température de la source chaude (°K)
- ΔT_{max} : écart de température maximal sourche chaude/froide (°K)
- U_{max} : Tension maximal (V)
- I_{max} : Courant maximal (A)

Vous pouvez déterminer les coefficients S, R et $R_{th,p}$ ainsi :

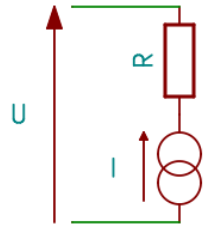
$$\begin{aligned} S &= \frac{U_{max}}{T_h} \\ R_{th,p} &= \frac{\Delta_{max}}{I_{max} U_{max}} \frac{2T_h}{T_h - \Delta_{max}} \\ R &= \frac{U_{max}}{I_{max}} \frac{T_h - \Delta_{max}}{T_h} \end{aligned}$$

Référence : Modeling and Analysis of Thermoelectric Modules, Simon Lineykin and Shmuel Ben-Yaakov, IEEE Transactions on industry applications, volume 43, Number 2, march/april 2007.

Modèle thermique et électrique en régime dynamique



Modèle électrique :



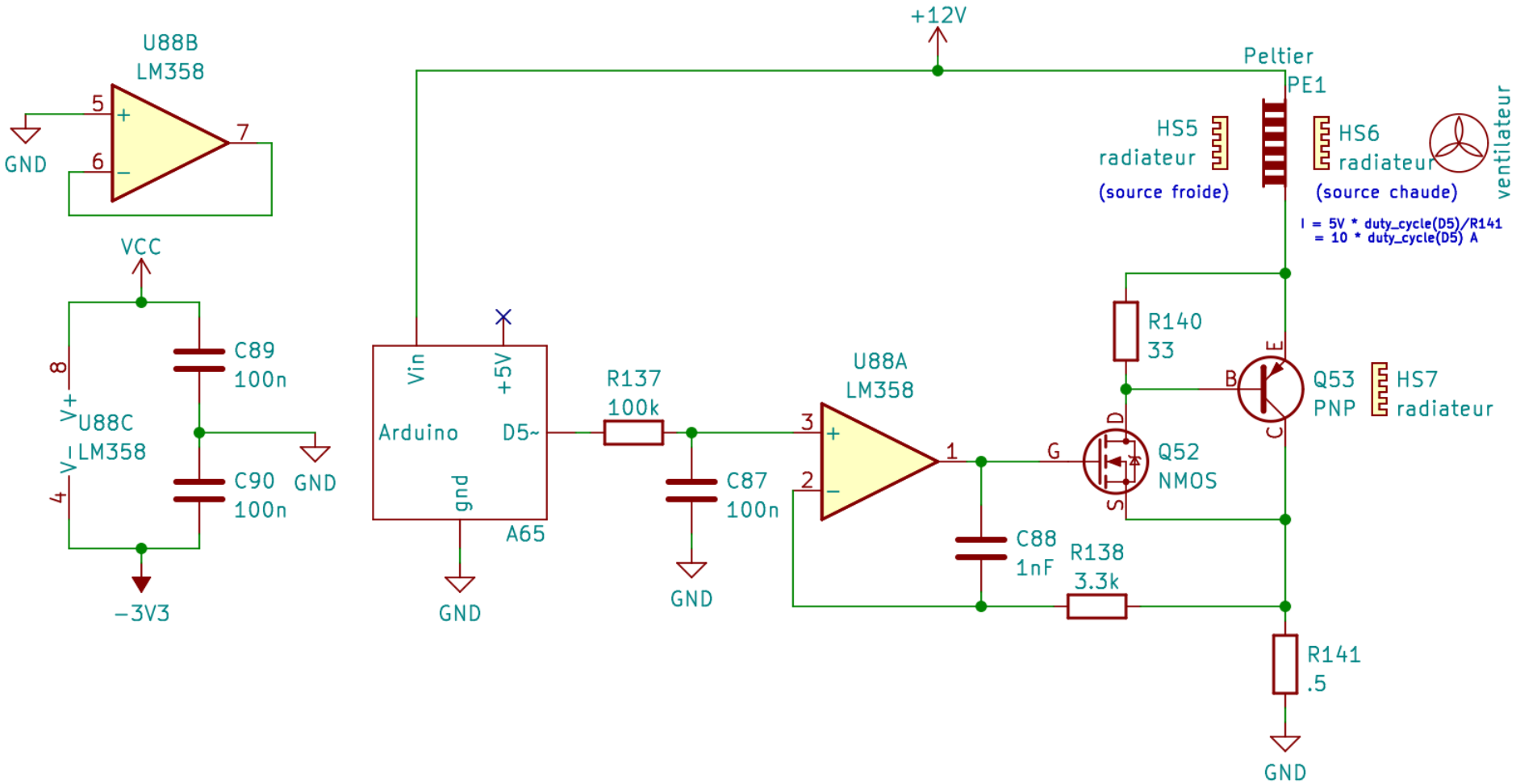
$$
\begin{aligned}
Q_1 &= Q_{s,f} - \frac{1}{2} Q_J - Q_c + C_p \frac{dT_f}{dt} \\
Q_2 &= Q_{s,c} + \frac{1}{2} Q_J - Q_c - C_p \frac{dT_c}{dt}
\end{aligned}
$$

$Q_{s,f}$	=	$S \times T_f \times I$	Effet de Seebeck
$Q_{s,c}$	=	$S \times T_c \times I$	Effet de Seebeck
Q_J	=	$R \times I^2$	Effet Joule
Q_c	=	$\frac{T_c - T_f}{R_{th,p}}$	Conduction source chaude/froide
U	=	$R \times I + S \times (T_c - T_f)$	Modèle électrique

S : Coefficient de Seebeck (J/°C/A)
R : Résidtance électrique (Ω)
$R_{th,p}$: Résistance thermique (°C/W)
C_p : Inertie thermique des plaques de céramique (J/°C)

(version longue)

Contrôle en courant d'un module peletier



(version longue)

Plan

(présent dans la version courte)

Références

Livres :

1. The Art of Electronics, Paul Horowitz et Winfield Hill, 3th Edition, 2015, Cambridge Press.
2. The Art of Electronics: The X-Chapters, Paul Horowitz et Winfield Hill, 2020, Cambridge Press.
3. Électronique, Fondements et applications, J.-P. Pérez, C. Lagoute, J.-Y. Fourniols et S. Bouhours, 2ieme Édition, 2012, Dunod.
4. Guide du Technicien en électronique, C. Cimelli et R. Bourgeron, 2007, Hachette technique.
5. Moteurs électriques pour la robotique, Pierre Mayé, 4eme Édition, 2023, Dunod.
6. Le grand livre de l'électricité, Thierry Gallauziaux, David Fedullo, 2021, Eyrolles.

Sites et vidéos (accédé le 12/02/2024) :

1. sonelec-musique.com, Rémy Mallard, Rubriques incontournables : Bases, Théorie; Réalisations et conceptions.
2. Circuit Simulator Falstad, Paul Falstad et Iain Sharp, version 2.8.2js, 2014. https://www.falstad.com/circuit;
3. Électro-Bidouilleur, chaîne youtube d'électronique, site web, Bertrand.

Base de données de projets (accédé le 12/02/2024):

1. HACKADAY.IO, site communautaire de développement matériel.
2. AUTODESK Instructables, site communautaire de projets DIY (Do It Yourself).

(présent dans la version courte)

Plan

(présent dans la version courte)

Solutions pour uploader le microcode Sous Linux - 1/2

1. Mettre à jour votre Arduino IDE
2. Fermer toutes les fenêtres arduino sauf celle du votre programme (pour fermer tous les ports série déjà ouvert).
3. Dans le fichier /etc/group, ajoutez votre login dans les groupes dialout et plugdev en vous inspirant de l'exemple suivant :

```
dialout:x:20:YOUR_LOGIN
plugdev:x:46:YOUR_LOGIN
```

   Ensuite, déloguez-vous et reloguez-vous à votre session.
4. Dans Arduino IDE, dans l'onglet Tools/Port, choisissez le bon port. Le port peut se lire en tapant la commande :

```
sudo dmesg
```

5. Pour les esp32 et cartes à base de STM32 : Mettre la carte en mode upload avant tout upload, vous utiliserez les boutons reset et boot en faisant la séquence :
   Appuyer et maintenir reset, appuyer et maintenir boot, relâcher reset, relâcher boot. Vous pouvez uploader votre firmware.

(présent dans la version courte)

Solutions pour uploader le firmware Sous Linux - 2/2

6. Pour les cartes Arduino Uno R4 et les cartes à base de STM32 : Installer l'outil dfu-util :

```
sudo apt install dfu-util
```

Trouvez les numéros idVendor et idProduct de votre carte en tapant au choix :

```
sudo dmesg
dfu-util -l
```

Adapter et ajouter les règles udev spécifiques à votre carte en créant le fichier /etc/udev/rules.d/60-arduino-board.rules :

```
SUBSYSTEMS=="usb", ATTRS{idVendor}=="2341", MODE:="0666"
```

7. Le paquet modemmanager est installé d'office et vient lire et monopoliser les lignes séries quand elles sont crées. Il vaut mieux désinstaller ce paquet :

```
sudo apt remove modemmanager
```

8. Carte BlackPill : Relier le blindage de la prise USB à la prise de terre. Cela réduit le bruit et aide l'ordinateur à égrainer l'USB.

Plan

(version longue)

Présentation de Platform.io

Arduino IDE est un vrai clickodrome difficile à utiliser dans des gros projets. Il est donc nécessaire d'utiliser des outils plus efficace, en ligne de commande.

Platform.io permet de gérer plusieurs cartes, de les compiler et d'uploader les firmware en ligne de commande.

Pour installer platform.io, consultez la documentation officielle : installer Platform io ;

Pour créer un projet vous pouvez consulter la ressource documentaire suivante : documentation pour démarrer rapidement. .

Nous allons maintenant voir les commandes les plus utiles.

Créer un projet pour Platform.io

Pour créer un projet vide :

```
mkdir my_project
cd my_project
pio boards esp32 # To get the avalaible platforms for esp32
pio project init --board uno --board upesy_wroom
```

Ajouter dans le dossier src le code source de votre application. On ajoutera le fichier main.cpp suivant :

```
#include <Arduino.h>
void setup(){ Serial.begin(9600); }
void loop(){ Serial.println("Hello !"); delay(1000); }
```

Pour compiler et uploader le firmware pour la carte esp32 faites :

```
pio run -e upesy_wroom -t upload --upload-port /dev/ttyUSB0
```

Si vous avez déclaré qu'une seule carte et que le port USB se detecte automatiquement :

```
pio run -t upload
```

Se connecter sur un port avec un projet Platform.io

Pour vous connecter sur le port série en ligne de commande, consultez la page 112.

Pour vous connecter sur le port série avec un programme Python, consultez la page 113.

Utiliser et installer une bibliothèque externe sous Platform.io

Pour installer une bibliothèque, cherchez le nom et la version de la bibliothèque que vous voulez utiliser en utilisant la commande suivante :

```
pio pkg search NOM_BIBLIOTHEQUE
```

Modifier le fichier platformio.ini en vous inspirant de l'exemple suivant:

```
[env:VOTRE_ENVIRONNEMENT]
lib_deps =
  ; Depend on the main SPI and SERVO library
  SPI
  SERVO
  ; Depend on the latest 7.x stable version of ArduinoJson.
  ; The minimum required version is 7.0.3.
  ; New functionality (backward-compatible) and bug-fixed are allowed
  bblanchon/ArduinoJson @ ^7.0.3
```

Vous pouvez compiler votre projet de nouveau. Platform.io installera automatiquement la bibliothèque.

Plan

(version longue)

Mise en garde !

Ce document est en cours d'écriture et de relecture !

Bien que l'auteur essaye d'être le plus rigoureux possible et essaye de justifier les schémas et affirmations donnés, ses domaines de spécialité sont les mathématiques et l'informatique.

Vous devez donc être très critique en lisant ce document. Vous devez comprendre le fonctionnement des circuits et multiplier et croiser les sources d'informations.

Ce document est conçu pour aider des ingénieurs à développer et concevoir de nouveaux objets. Les circuits doivent donc être validés par le lecteur avant d'être construit et utilisés. Ils doivent être testés intensivement avant d'être diffusés au grand public. L'auteur ne peut pas être tenu pour responsable des dégâts occasionnés par les circuits conçus, produits ou reproduits par le lecteur.

Si vous rencontrez des erreurs à la lecture de ce document, n'hésitez pas à contacter l'auteur pour qu'il le corrige et l'améliore.

(version longue)

Plan

- Outils pour protéger le port USB de l'ordinateur
- Le stage de puissance des Arduino Uno R3 et Leonardo
- Le stage de puissance des Arduino Uno R4, Nano V3 et Every
- Le stage d'alimentation des cartes ESP32 bien conçues, des Arduino Nano Ble et Arduino Nano ESP32
- Le stage d'alimentation des cartes ESP32 MAL conçues

Outils pour protéger le port USB de l'ordinateur

Si vous avez un doute sur un circuit ou un materiel defectueux, protégez le port USB de l'ordinateur en l'isolant galvaniquement du circuit à l'aide d'un isolateur USB 1500V basé sur le composant ADUM3160.

Ce composant coûte entre 5 et 10 euros pièces et se trouve facilement sur le marché.

Ce périphérique transmet le signal USB et peut juste alimenter la carte du microcontrôleur. Vous devez donc alimenter votre circuit avec une batterie ou un bloc d'alimentation externe.

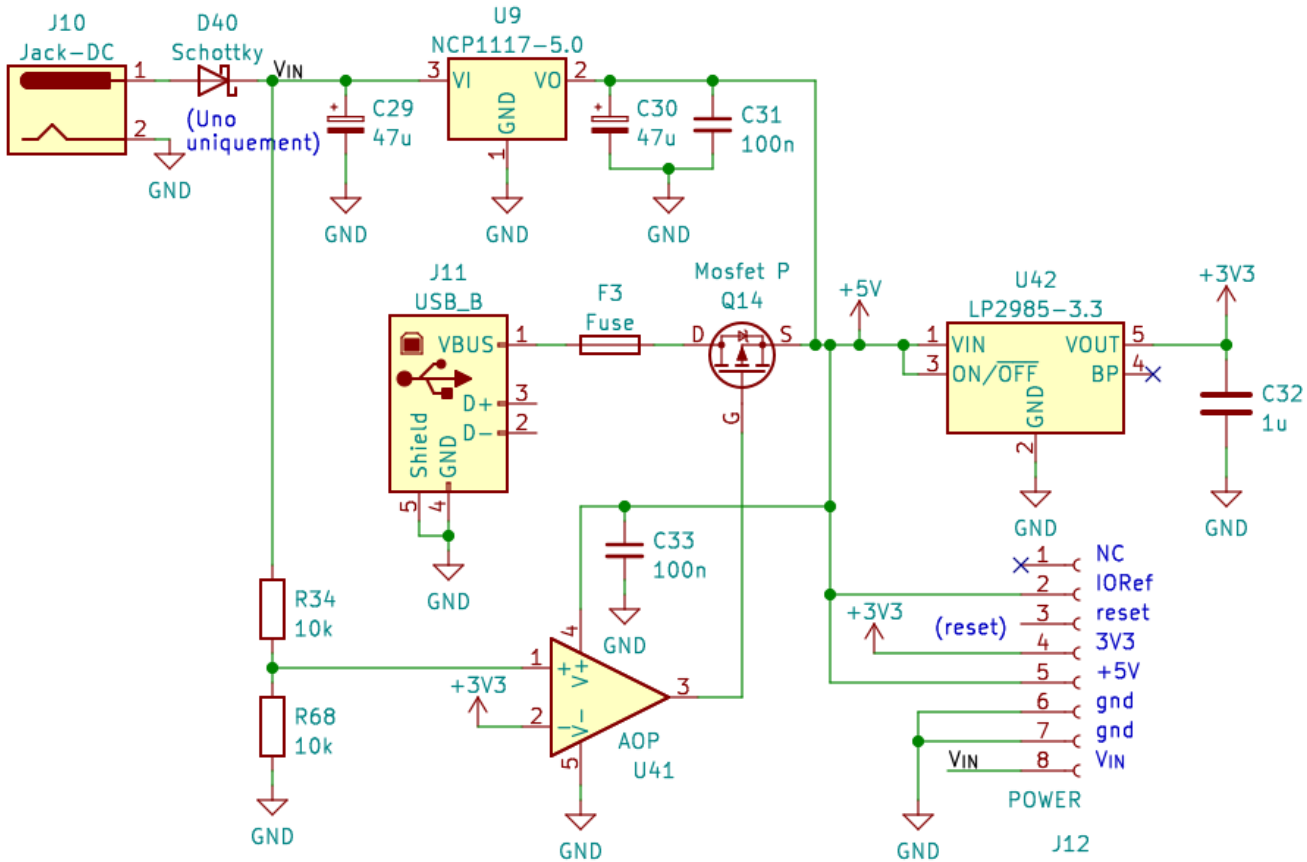
Si la seule source d'alimentation de votre circuit est celle de votre ordinateur, cet outils n'est généralement pas nécessaire.

Gardez toujours en tête qu'un moteur qui tourne devient une source d'alimentation. C'est pourquoi les blocs de puissance des moteurs possèdent souvent une alimentation séparée, que les lignes de signal sont isolés avec des optocoupleurs et que l'on utilise une diode TVS (avec son fusible) en parallèle à l'alimentation des moteurs pour absorber les surtensions. Utiliser l'isolateur USB permet de protéger le port USB quand les protections ci-dessus ne sont pas présentes.

Plan

Le stage de puissance de l'Arduino Uno R3 et Leonardo

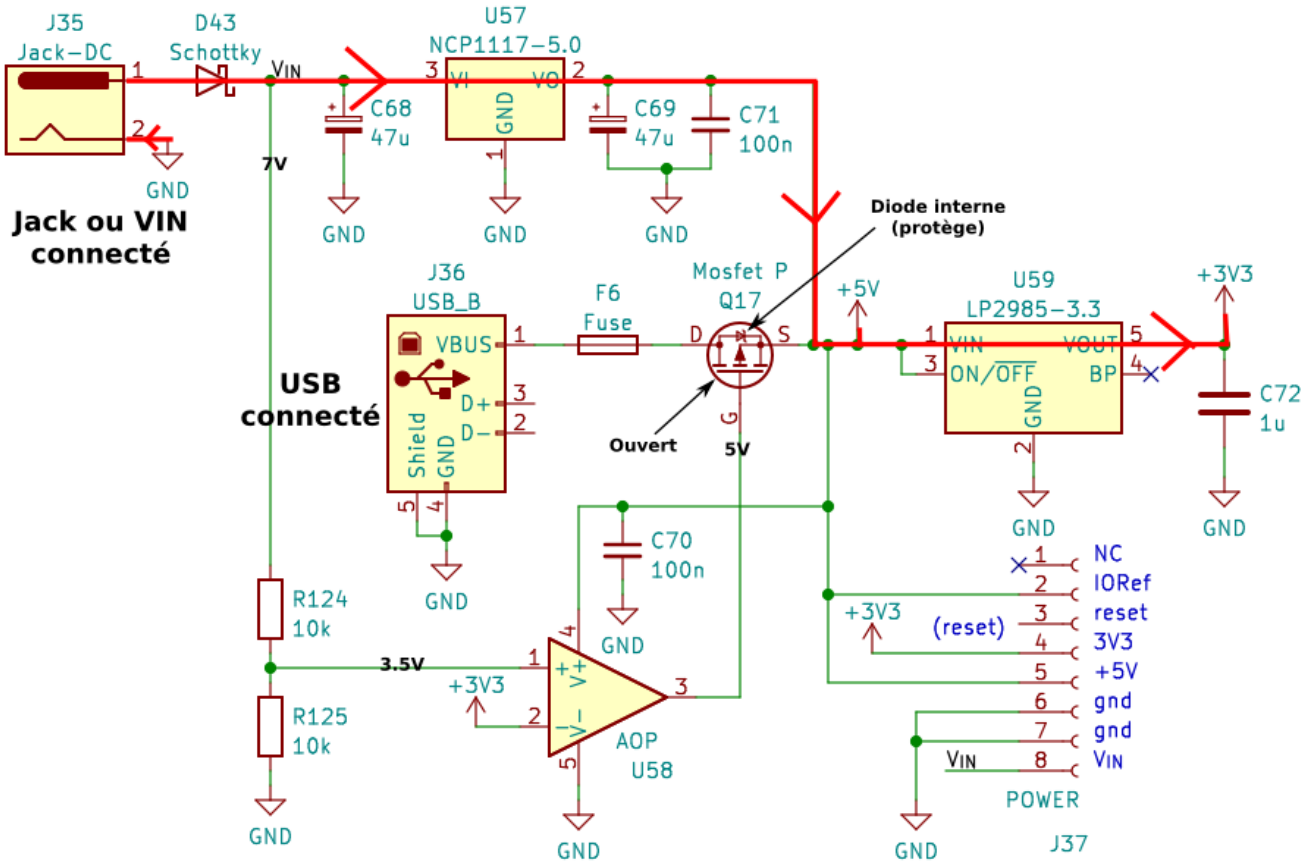
Le système de gestion de puissance des Arduino Uno 1, 2, et 3 et Leonardo est le suivant:



La bonne façon d'alimenter une carte est d'utiliser la prise Jack ou la broche Vin, avec un bloc d'alimentation de 7V à 12V On peut alors toujours se connecter en USB sur la carte.

Le stage de puissance de l'Arduino Uno R3 et Leonardo

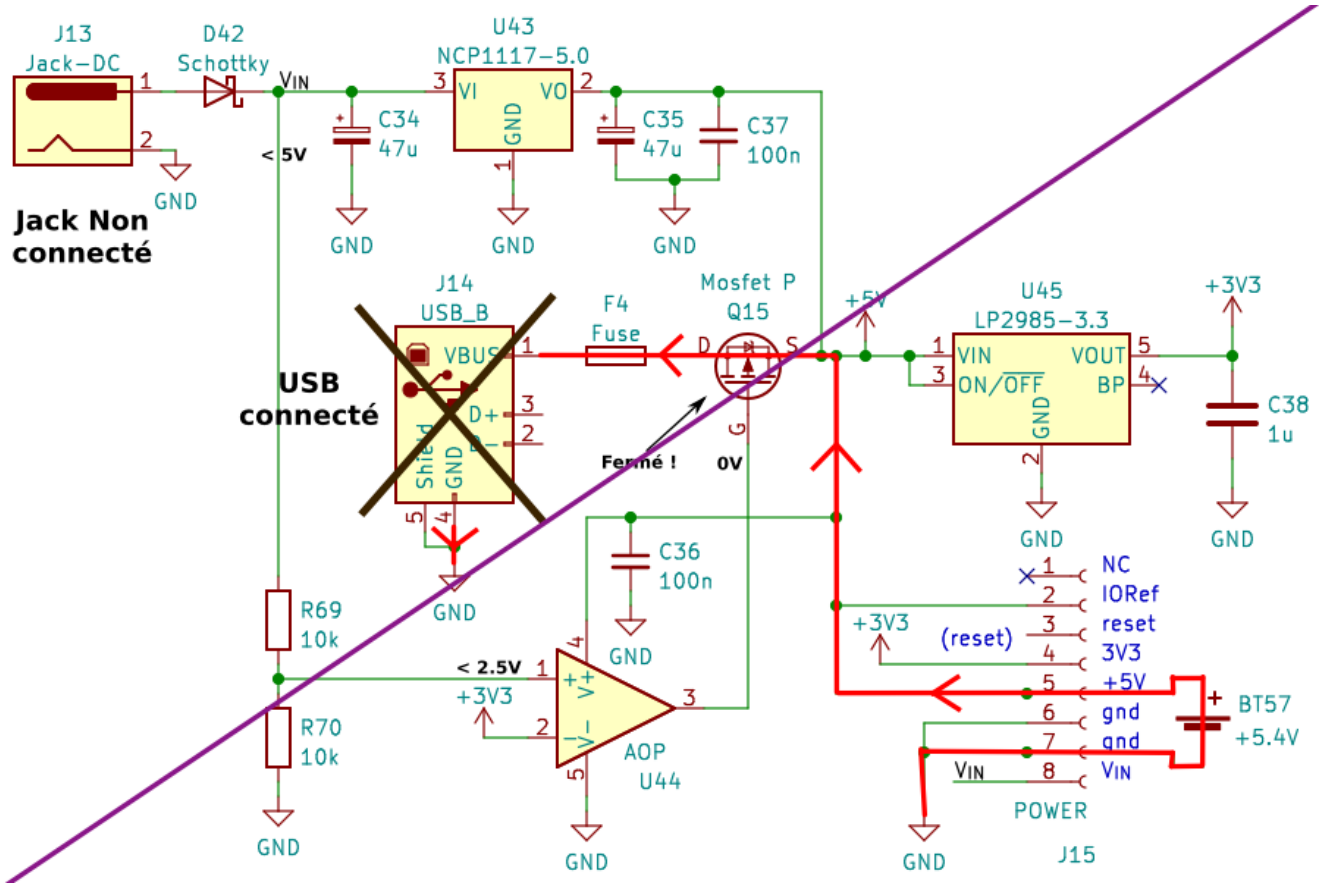
Le système de gestion de puissance des Arduino Uno 1, 2, et 3 et Leonardo est le suivant:



La bonne façon d'alimenter une carte est d'utiliser la prise Jack ou la broche Vin, avec un bloc d'alimentation de 7V à 12V On peut alors toujours se connecter en USB sur la carte.

Protection 5V USB/Vin/+5V pour l'Arduino Uno R3 et Leonardo

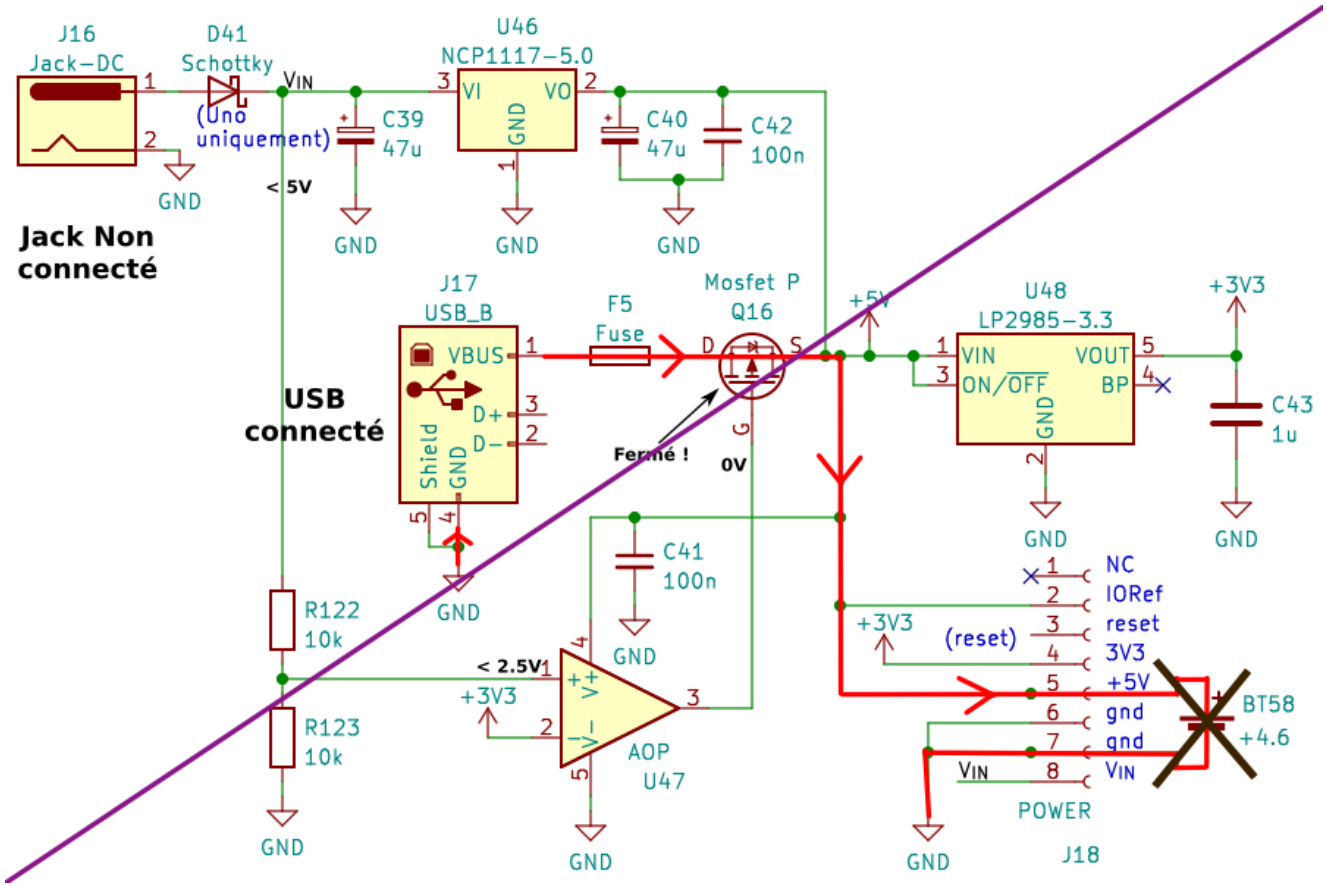
Vous pouvez connecter une alimentation 5V sur le +5V UNIQUEMENT si vous n'utilisez pas le Vin et l'USB ! Voici ce qui se passe si vous connectez la batterie au +5V et aussi l'USB :



simulation du branchement d'une batterie sur le 5V d'un carte Arduino Uno.

Protection 5V USB/Vin/+5V pour l'Arduino Uno R3 et Leonardo

Vous pouvez connecter une alimentation 5V sur le +5V UNIQUEMENT si vous n'utilisez pas le Vin et l'USB ! Voici ce qui se passe si vous connectez la batterie au +5V et aussi l'USB :



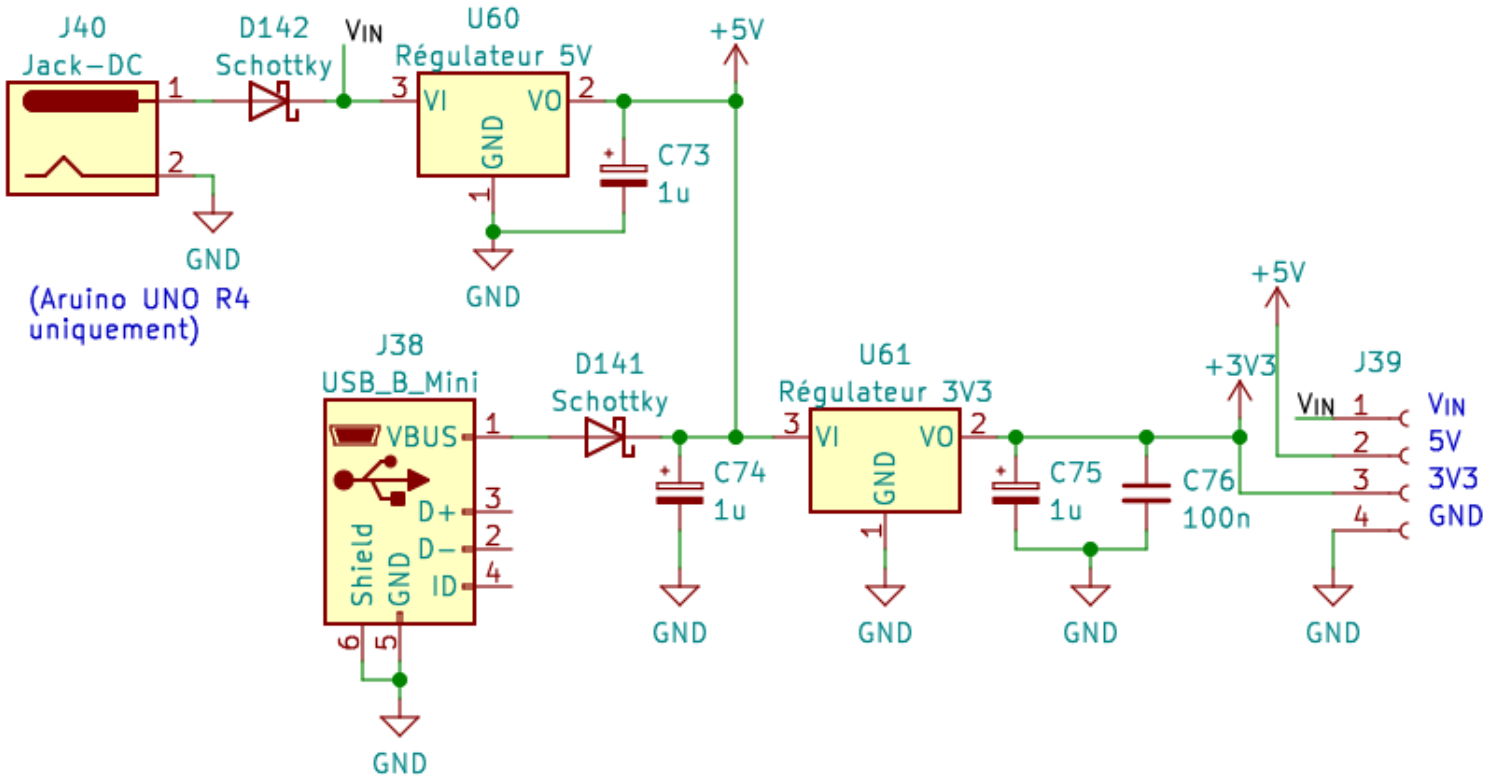
simulation du branchement d'une batterie sur le 5V d'un carte Arduino Uno.

(version longue)

Plan

Le stage de puissance des Arduino Uno R4, Nano V3 et Every

Dans ces cartes, l'USB de l'ordinateur est protégé.
La meilleur façon d'alimenter ces cartes est d'utiliser la prise Jack ou la broche Vin, avec un bloc d'alimentation de 7V à 12V.



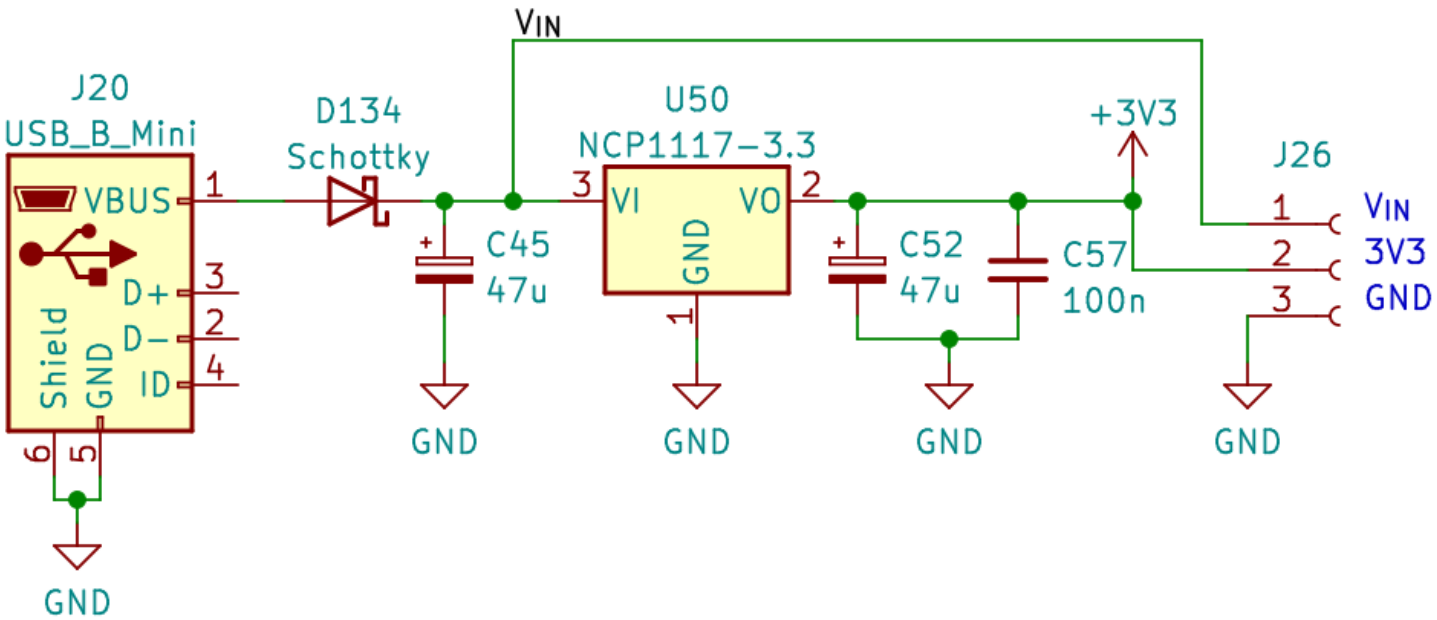
Vous pouvez cependant connecter une batterie sur la broche +5V en suivant les recommandations pour les cartes ESP32 bien conçue de la page 634.

(version longue)

Plan

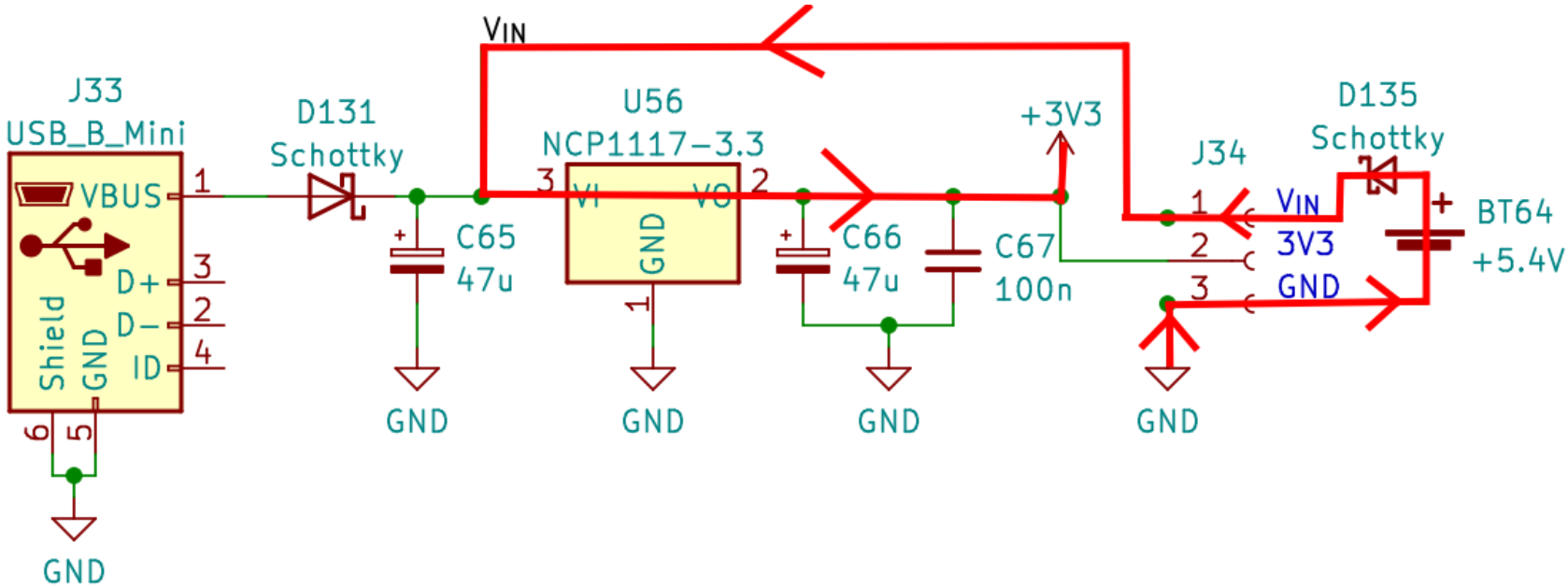
Le stage d'alimentation des cartes ESP32 bien conçues, des Arduino Nano Ble et Arduino Nano ESP32

Dans cette carte, l'USB de l'ordinateur est protégé.



Protection 5V USB/Vin/+5V pour les cartes ESP32 bien conçues

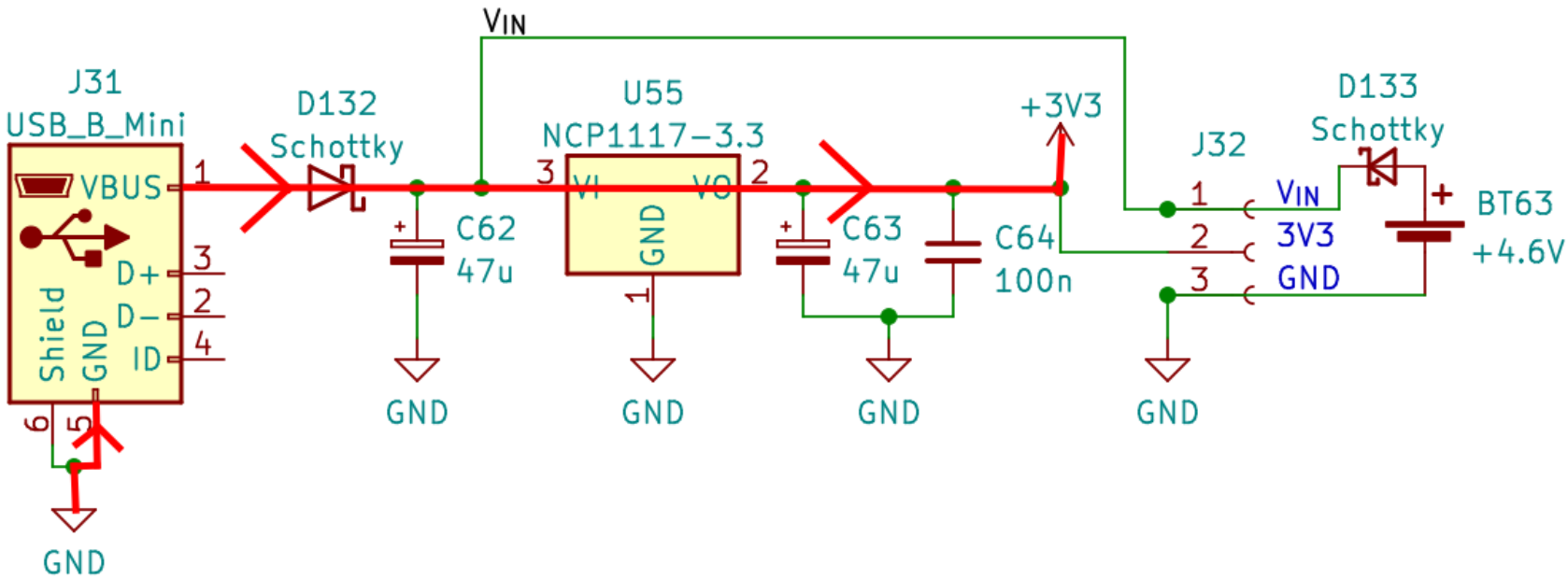
Si vous connectez un bloc d'alimentation de 5V au Vin, n'oubliez pas de mettre une diode de protection pour protéger l'alim de l'USB (qui est déjà protégée par la diode D132):



Simulation du branchement d'une batterie sur une carte ESP32 bien conçue.

Protection 5V USB/Vin/+5V pour les cartes ESP32 bien conçues

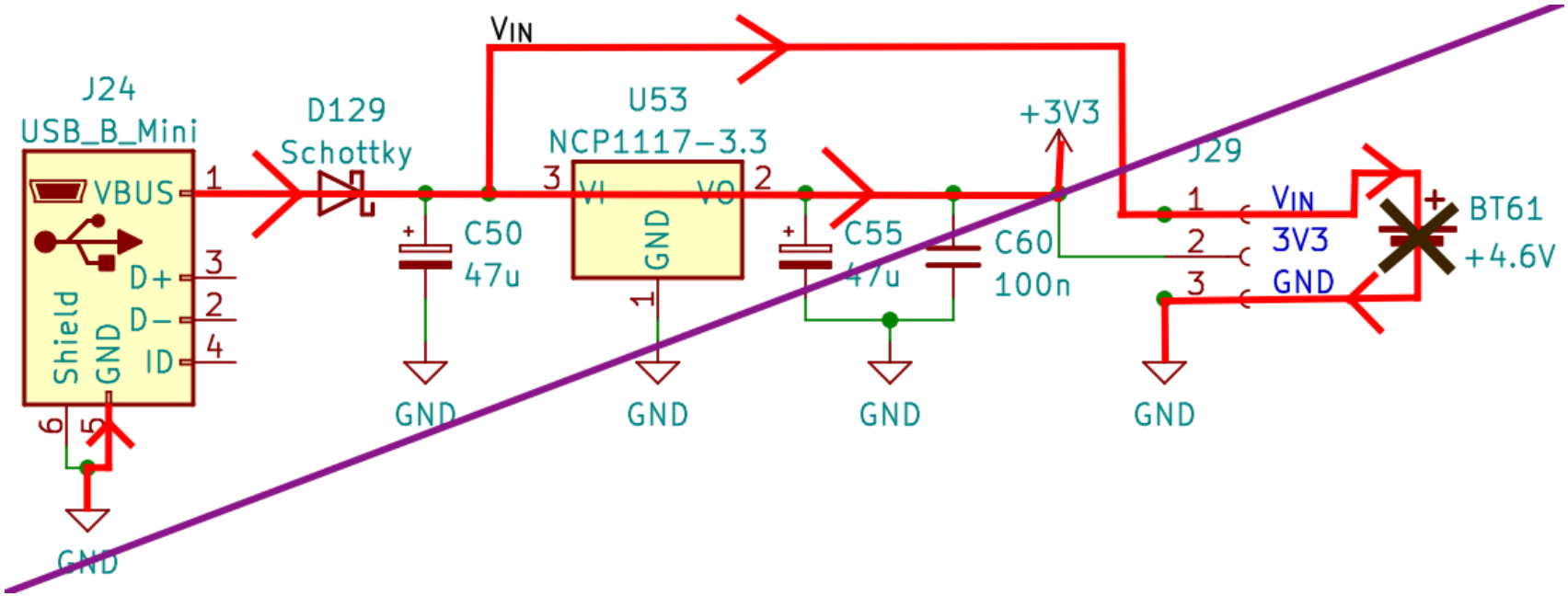
Si vous connectez un bloc d'alimentation de 5V au Vin, n'oubliez pas de mettre une diode de protection pour protéger l'alim de l'USB (qui est déjà protégée par la diode D132):



Simulation du branchement d'une batterie sur une carte ESP32 bien conçue.

Protection 5V USB/Vin/+5V pour les cartes ESP32 bien conçues

Si vous connectez un bloc d'alimentation de 5V au Vin, n'oubliez pas de mettre une diode de protection pour protéger l'alim de l'USB (qui est déjà protégée par la diode D132):



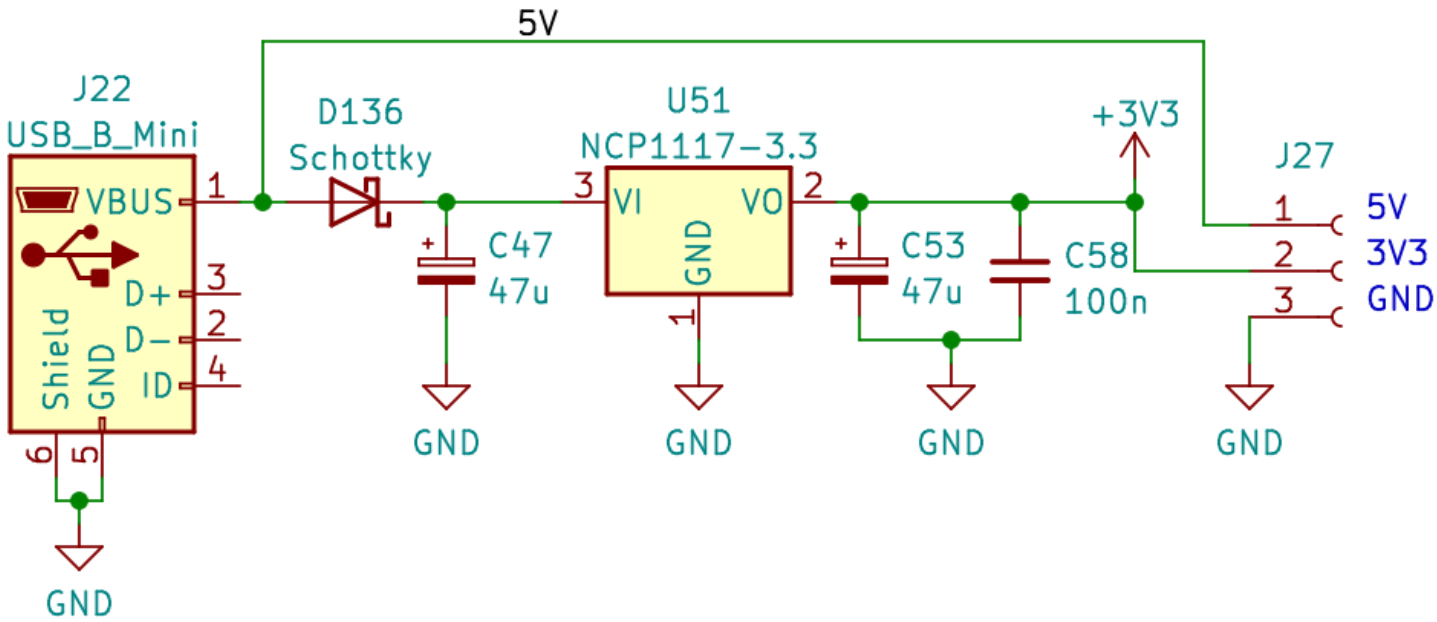
Simulation du branchement d'une batterie sur une carte ESP32 bien conçue.

(version longue)

Plan

Stage d'alimentation d'une carte ESP32 MAL conçue

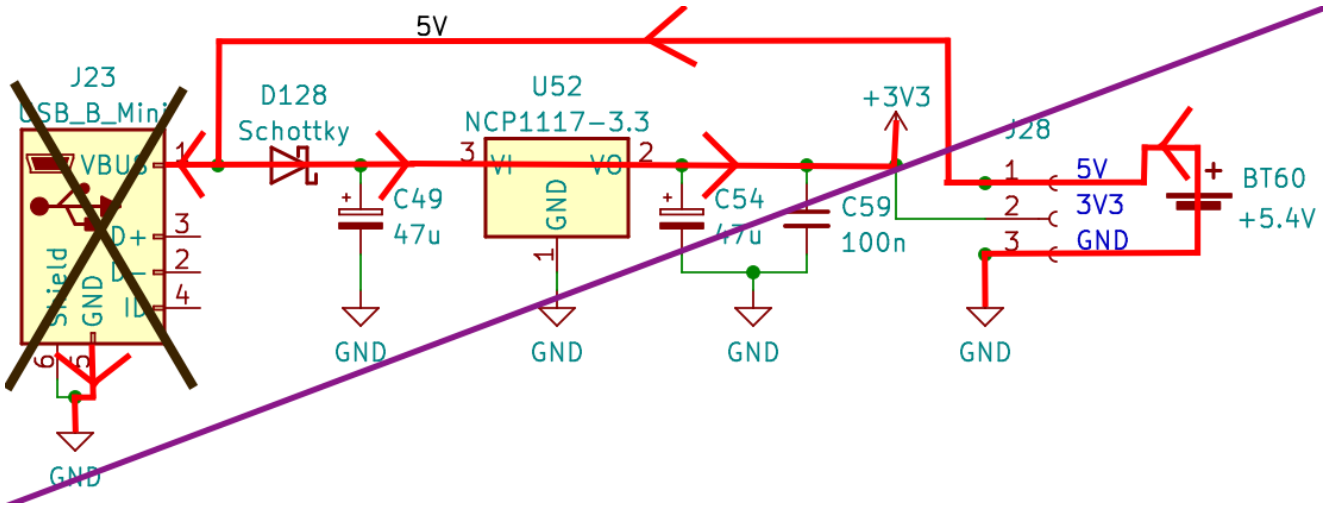
Certaines cartes ESP32 que l'on peut acheter sont MAL conçues ! En voici un exemple : le 5V est repiqué du mauvais côté de la diode.



Si vous branchez une alimentation sur le 5V de la carte, NE branchez surtout pas l'USB de votre ordinateur où son port USB sera détruit.

Mauvaise conception de certaines cartes ESP32

Voici ce qu'il se passe si vous branchez un bloc d'alimentation 5V sur le Vin en même temps que l'USB de votre ordinateur :

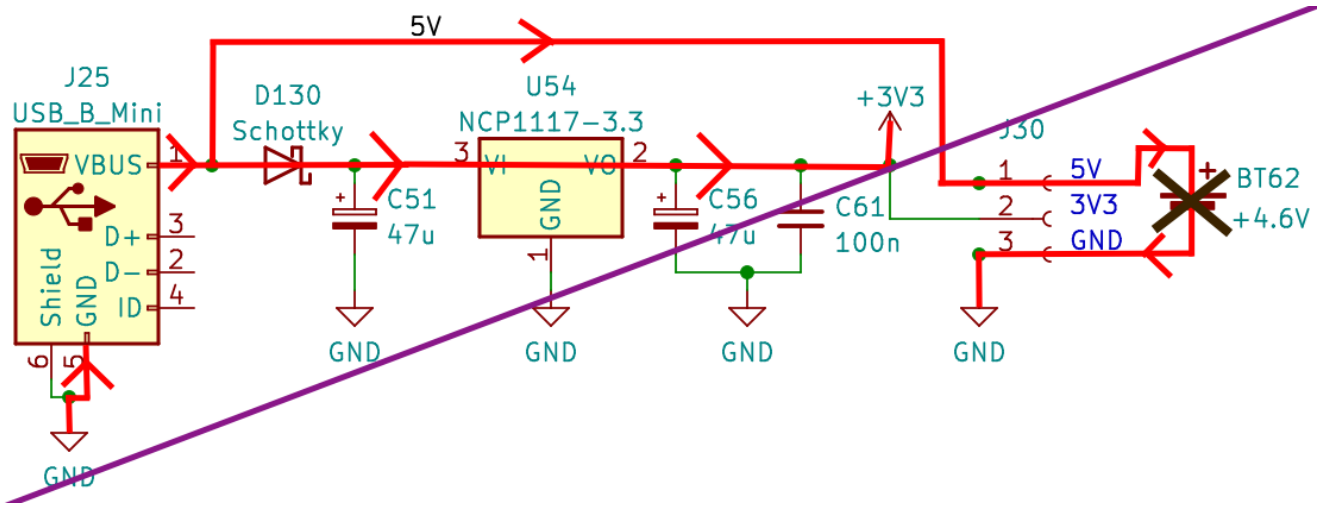


simulation du branchement d'une batterie sur une carte ESP32 mal conçue .

(version longue)

Mauvaise conception de certaines cartes ESP32

Voici ce qu'il se passe si vous branchez un bloc d'alimentation 5V sur le Vin en même temps que l'USB de votre ordinateur :



simulation du branchement d'une batterie sur une carte ESP32 mal conçue .

Plan

(version longue)

Sommaire des tables utiles

Wire Size & Current Rating (A) Guide

Current carrying capacity is defined as the amperage (A) of which a conductor can carry before melting either the conductor or the insulation. Heat caused by an electrical current flowing through the conductor will determine the amount of current a wire will handle.

Theoretically, the amount of current that can be passed through a single bare copper wire can be increased until the heat generated reaches the melting temperature of the copper. However, there are many factors that will limit the amount of current that can be passed through a wire, of which the key ones are detailed below;

· **Conductor Size:**
The larger the circular mil area, the greater the current carrying capacity. The amount of heat generated should never exceed the maximum temperature rating of the insulation.

· **Ambient Temperature:**
The higher the ambient temperature, the less heat required to reach the maximum temperature rating of the insulation.

· **Conductor Number:**
Heat dissipation is lessened as the number of individually insulated conductors, bundled together, is increased.

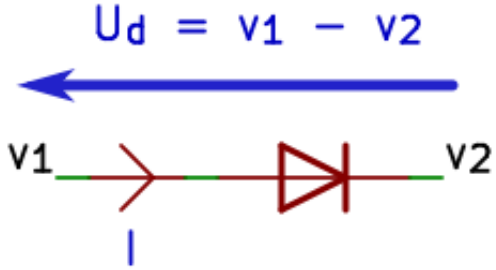
· **Installation Conductors:**
Restricting the heat dissipation by installing the conductors in conduit, duct, trays or raceways lessens the current carrying capacity. This restriction can be alleviated somewhat by using proper ventilation methods, forced air cooling, etc.

Taking into account all the variables involved, no simple chart of current ratings can be developed and used as the final word when designing a system where amperage ratings can become critical.

Conductor Size This chart shows the current requiredto raise the temperatures of stranded insulated (Rubber/Vinyl) conductors in free air (30°C ambient).					Current Rating
A.W.G.	C.M.A.	Diameter (mm)	mm^2	Size	
#32	63	0.20	0.03		0.3A
#30	101	0.26	0.05		0.5A
#28	160	0.32	0.08		0.7A
#26	254	0.41	0.13		1.0A
#24	404	0.51	0.20		2.0A
#22	643	0.64	0.33		3.0A
#20	1,020	0.81	0.52		5.0A
#18	1,624	1.02	0.82		7.0A
#16	2,583	1.29	1.31		10.0A
#14	4,106	1.63	2.08		20.0A
#12	6,530	2.05	3.31		30.0A
#10	10,384	2.59	5.26		50.0A

JST
The Quality Connection
J.S.T.(U.K.) Ltd.
Blyth Road, Halesworth,
Suffolk, IP19 8EW, England
Phone: +44 1986 874131
Fax: +44 1986 874276
www.jst.co.uk

Tensions directes et longueurs d'onde de différentes diodes

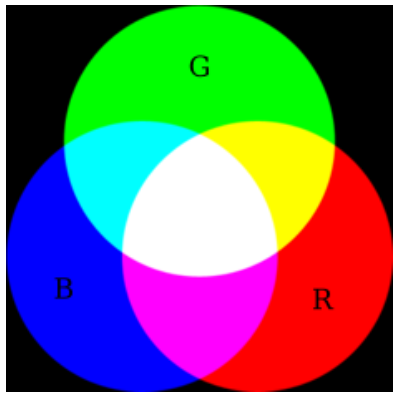


Les tensions directes U_d des différentes diodes sont données pour de faibles courants ($I \leq 200mA$).

	diodes		diode électroluminescente (led)						
	Schottky	Silicium	Infra rouge	Rouge	Jaune	Vert (GaP)	Vert clair (GaN)	Bleue	Ultra violet
tension directe (V)	0.4	0.6-0.8	1.3	2	2	2	3	3.5	3.7
longueur d'onde (nm)	/	/	> 760	615 à 650	574 à 582	500 à 570		466 à 490	< 400

Comment fonctionne les led :
https://www.youtube.com/watch?v=AF8d72mA41M
https://www.youtube.com/watch?v=l2y-w9aS98k

(consultés le 21/03/2025)



(version longue)

Diodes usuelles

désignation		tension directe (V)	tension inverse (V)	tension inv. max (V)	tension de claquage (V)	courant direct maximal (A)	courant inverse maximal (A)	puissance (W)	temps de recouvr. (ns)	se transforme en fil en brûlant *2	symbole
redress.	1N400X X=1..7	**1.1**		$50..10^3$		**1**					
	1N540X X=0..8	**1.2**		$50..10^3$		**3**					
signaux com. rapide	1N415X X=0..1	**0.540-1.0**		50		0.3 0.15*6 4.. 2 *7			**2-4**		
Schottky	1N581X X=7..9	**0.45..0.5**		20..40		**1**					
	1N582X X=0..2	**0.47..0.52**		20..40		**3**					
	1N582X X=3..5	**0.47..0.52**		20..40		**5**					
Zener	BZX55 Y*1 V	1.5	**Y = 2.4 .. 75**			0.2	**0.5/Y**	**0.5**			
TVS	1.5KEYA *3	3.5-5	cf. tens. de claq.		**Y=6.8 ..540**	200 *5	**143 ..2.03**	**1500** *4		✓	

*1 : dans cette ligne Y désigne les tensions directes; *2 : les cases vides de cette colonne signifient "ne sait pas"; *3 : dans cette ligne Y désigne la tension de claquage (breakdown voltage); *4 : pulse de 10/1000 μs; *5 : pulse de 3.8 ms; *5 : en moyenne. *6 : courant de crête de surtension (pulse de 1μs).

(version longue)

Choix de quelques associations de led IR et de photodiodes

Voici quelques exemples d'associations entre une photodiode et une led IR compatibles :

photodiode	diode
BPV22F BPV23F SFH 213 FA SFH 203 FA SFH 205 FA SFH 203 PFA	TSALXXXX SFH 454X
BPV22NF BPV23NF	TSFFXXXX TSHFXXXX

Voici quelques associations entre un émetteur IR avec demodulation et une led IR compatibles :

émetteur IR avec démodulation	diode
TSOPXXXXX HS0038B3VM VS1838B	TSALXXXX SFH 454X

Remplacez les X par les numéros qui correspondent à vos besoins. Consultez les tables des pages suivantes pour selectionner les diodes et photodiodes qui vous arrangent.

Consultez le catalogue de Vishay pour plus de choix (consulté le 29/09/2024).

Photodiodes Infrarouges

Tension directe : 1.3V

nom	marque *1	dimensions (mm)	angle (deg.)	longueur d'onde (nm)		photo-courant (μA)	temps de monté/ descente (ns)	aire sensitive (mm^2)	capacité (pF)	réact-ance (A/W)
SFH 229 FA	a	3	±15	900	740···1100	11	6000	0.31	12	
SFH 213 FA	a	5	±10	900	750···1100	42	5	1	11	0.65
SFH 203 FA	a	5	±20	900	750···1100	25	5	1	11	0.62
SFH 205 FA	a	5	±60	900	740···1100	56	20	7.02	72	0.62
SFH 235 FA	a	3	±65	900	740···1120	22	20	7.02	72	0.65
SFH 203 PFA	a	5	±75	900	750···1100	3	5	1	11	0.62
BPV10NF	v	5	±20	940	780···1050	60	80		11	0.55
BPV22NF	v	4.5x5x6	±60	940	790···1050	85	100	7.5	70	0.6
BPV23NF	v	4.5x5x6	±60	940	790···1050	65	70	4.4	48	0.6
BPW82	v	5x4x6.8	±65	950	790···1050	38	100	7.5	70	
BPW83	v	5x3x6.4	±65	950	790···1050	38	100	7.5	70	
BPV09NF	v	5	±22	940	780···1050	55	80/60		11	
BPV22F	v	4.5x5x6	±60	950	870···1050	80	100	7.5	70	0.6
BPV23F	v	4.5x5x6	±60	950	870···1050	60	70	4.4	48	0.6
BPW41N	v	5x4x6.8	±65	950	870···1050	38	100	7.5	70	
SFH 229	a	3	±15	860	380···1100	14	1000	0.31	12	
BPV10	v	5	±20	920	380···1100	65	80/60		11	0.55
SFH 213	a	5	±10	850	400···1100	125	5	1	11	0.65
SFH 203	a	5	±20	850	400···1100	80	5	1	11	0.62
SFH 203P	a	5	±75	850	400···1100	9.3	5	1	11	0.62
SFH 206 K	a	5	±60	920	420···1120	80	20	1	72	0.62
BPW46	v	5x3x6.4	±65	900	430···1100	47	100	7.5	70	
BPW24R	v	4.7	±12	940	610···1040	55	80/60	0.88	11	

*1: "a" pour ams-OSRAM, "v" pour Vishay.

(version longue)

Led Infrarouge

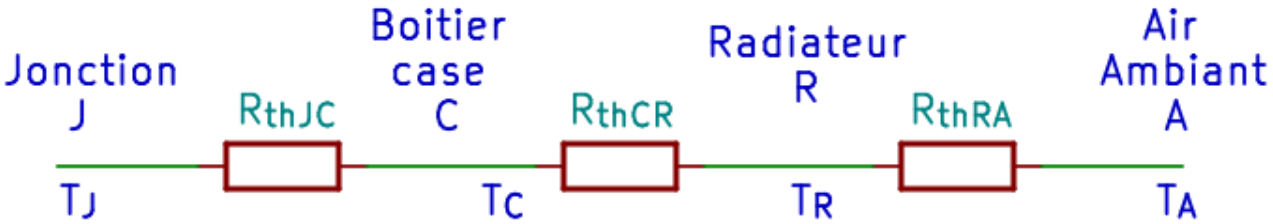
nom	marque *1	diamètre (mm)	longueur d'onde (nm)	angle (deg)	Intensité angulaire pour 1A (mW/sr)	temps de montée/ descente (ns)	tension directe (V)	tension directe à 1A (V)	courant direct (mA)	courant pulsé (A)	puissance (mW)
SFH 4550	a	5	860	±3	8500	12	1.5-1.7	2.4-2.9	100	1	180
SFH 4554	a	5	860	±10	2250	12	1.7-1.9	3.6-4.5	100	1	200
SFH 4555	a	5	860	±5	4400	12	1.5-1.7	2.4-2.9	100	1	180
SFH 4556	a	5	860	±20	1150	12	1.5-1.7	2.4-2.9	100	1	180
SFH 4557	a	5	860	±30	450	12	1.5-1.7	2.4-2.9	100	1	180
TSFF5210*2	v	5	870	±10	1800	15	1.5-1.8	2.3-3	100	1	180
TSFF5410*2	v	5	870	±22	700	15	1.5-1.8	2.3-3	100	1	180
TSFF6210*2	v	5	870	±10	1800	15	1.5-1.8	2.3-3	100	1	180
TSFF6410*2	v	5	870	±22	700	15	1.5-1.8	2.3-3	100	1	180
TSHF5210	v	5	890	±8	2700	10	1.5-1.7	3	100	1	170
TSHF5410	v	5	890	±27	528	10	1.5-1.7	3	100	1	170
TSHF6210	v	5	890	±8	2700	10	1.5-1.7	3	100	1	170
TSHF6410	v	5	890	±27	528	10	1.5-1.7	3	100	1	170
TSAL6100	v	5	940	±10	1450	15	1.35-1.6	2.2-3	100	1	160
TSAL6200	v	5	940	±17	600	15	1.35-1.6	2.2-3	100	1	160
TSAL6400	v	5	940	±25	420	15	1.35-1.6	2.2-3	100	1	160
TSAL4400	v	3	940	±25	290	15	1.35-1.6	2.2-3	100	1	160
SFH 4544	a	5	950	±10	2300	12	1.6-1.8	3.6-4.5	100	1	200
SFH 4545	a	5	950	±5	4200	12	1.5-1.7	2.3-2.9	100	1	180
SFH 4546	a	5	950	±20	1000	12	1.5-1.7	2.3-2.9	100	1	180
SFH 4547	a	5	950	±30	375	12	1.5-1.7	2.3-2.9	100	1	180

*1: "a" pour ams-OSRAM, "v" pour Vishay; *2: ce composant n'est plus produit.

(version longue)

Dissipation thermique

T_j, T_C, T_R et T_A : température de jonction, boîtier, radiateur et ambiante



$$(T_A - T_j) = P_{\text{dissipé}} \times (R_{thJC} + R_{thCR} + R_{thRA})$$

R_{thJC}, R_{thCR}, R_{thRA} : résistances thermiques en $°C/W$.

Température ambiante max : $T_A = 40°C$.

Résistance thermique boîtier - air ambiant, sans radiateur :

Boîtier	TO 3	TO 5	TO 61	TO 63	TO 66	TO 126	TO 220
$R_{thCA} = R_{thCR} + R_{tcRA}$ $(°C/W)$	30	180	45	30	45	80	60

Chiffres tirés du livre Guide du technicien en électronique de C. Cimelli, 2007-2008, page 99.

(version longue)

Les niveaux logiques

CMOS = technologie avec des mosfet
TTL = technologie avec des transistors bipolaires

Tensions d'entrée

booléen	CMOS $3V3$	CMOS $5V$	TTL	Arduino Uno	esp32
0	$<0.8V$	$<1.5V$	$<0.8V$	$<1.5V$	$<0.825V$
1	$>2V$	$>3.5V$	$>2V$	$>3V$	$>2.48V$

Tensions de sortie

booléen	CMOS $3V3$	CMOS $5V$	TTL	Arduino Uno	esp32
0	$<0.2V$	$<0.1V$	$<0.4V$	$<0.8V$	$<0.33V$
1	$>3.1V$	$>4.9V$	$>2.4V$	$>4.1V$	$>2.64V$

TTL : 74LSXX (compatible Arduino Uno et Esp32)
Input TTL + Output CMOS 5V : 74HCTXX
CMOS 5V : 74HCXX, arduino Uno
CMOS 3V3 : 74LVCXX, Esp32

(version longue)

La série des 74XXXX

Liste détaillée : List of 7400-series integrated circuits (Wikipedia).

	AND	NAND	OR	NOR	XOR	XNOR
Quad 2-input	74x08	74x00	74x32	74x02	74x86	74x7266
Triple 3-input	74x11	74x10	74x4075	74x27	/	/
Dual 4-input	74x21	74x20	74x4072	74x29	/	/

	Schmidt-trigger	
	Buffer	Inverter
Hex 1-input	74x7014	74x14

	série vers parallèle	parallèlle vers série
8 bits	74x595	74x165

Les ports UART de chaque microcontrolleur

microcontrolleur		Spéc. Hardware		Programmation		brochage	
marque	nom	protocole	port	nom	USB	RX	TX
Arduino	Uno R3 / Nano	UART		Serial	✓	0 (RX)	1 (TX)
	Leonardo / Zero	UART		Serial	✓		
				Serial1		0 (RX)	1 (TX)
	Uno R4	UART		Serial	✓		
				Serial1		0 (RX0)	1 (TX0)
	Giga R1	UART	?	Serial	✓		
		UART		Serial1		0 (RX)	1 (TX)
		UART		Serial2		19 (RX1)	18 (TX1)
		UART		Serial3		17 (RX2)	16 (TX2)
		UART		Serial4		15 (RX3)	14 (TX3)
ESP32 (devKitC/M)	Wroom-32 D/U	UART	0	Serial	✓	1 *1	3 *1
		UART	1	Serial1		~~9~~ *2	~~10~~ *2
		UART	2	Serial2		16 *1*4	17 *1*4
	C3-Wroom	UART	0	Serial	✓	20 *3 ?	21 *3 ?
		UART	1	Serial1		*1	*1
	S3-Wroom	UART	0	Serial	✓	44*3	43*3
		UART	1	Serial1		18 *1	17*1
		UART	2	Serial2		*1	*1
teensy	4.1	UART	0	Serial	✓		
		UART	6	Serial1		0 (RX1)	1 (TX1)
		UART	3	Serial2		7 (RX2)	8 (TX2)
		UART	2 (MM→4)	Serial3		15 (RX3)	14 (TX3)
		UART	4 (M→2)	Serial4		16 (RX4)	17 (TX4)
		UART	8	Serial5		21 (RX5)	20 (TX5)
		UART	1	Serial6		25 (RX6)	24 (TX6)
		UART	7	Serial7		28 (RX7)	29 (TX7)
		UART	5	Serial8		34 (RX8)	35 (TX8)

*1 On peut configurer une autre broche. *2 Le brochage DOIT être réassigné car déjà utilisé. *3 Le brochage ne peut être réassigné. *4 Le brochage doit être configuré car il ne l'est pas par default.

(version longue)

Les bus I2C des microcontrolleurs et leurs brochages

microcontrolleur		spécification constructeur		Programmation		brochage	
marque	nom	protocole	bus	nom	default*4	SDA[3]	SCL*3
Arduino	Uno R3	I2C		Wire	✓	A4	A5
	Zero	I2C	?	Wire	✓	20	21
	Leonardo	I2C	?	Wire	✓	D2	D3
	Nano	I2C	?	Wire	✓	A4	A5
	Uno R4	I2C	?	Wire	✓	A4	A5
		I2C	?	Wire1		D27	D26
	Giga R1	I2C	?	Wire	✓	D20	D21
		I2C	?	Wire1		D102 (SDA1)	D101 (SCL1)
		I2C	?	Wire2		D9 (SDA2)	D8 (SCL2)
ESP32 (devKitC/M*5)	Wroom-32 D/U	I2C	0	Wire	✓	21*1	22*1
		I2C	1	Wire1		*1 *2	*1 *2
	C3-Wroom	I2C	0	Wire	✓	8*1	9*1
	C6-Wroom	I2C	0	Wire	✓	23*1	22*1
	h2-MINI	I2C	0	Wire	✓	12*1	22*1
		I2C	1	Wire1		*1 *2	*1 *2
	S2/S3-Wroom	I2C	0	Wire	✓	8*1	9*1
		I2C	1	Wire1		*1 *2	*1 *2
teensy	4.1	I2C	0	Wire	✓	18	19
		I2C	1	Wire1		17	16
		I2C	1	Wire1		44*2	45*2
		I2C	2	Wire2		25	24

*1 Il s'agit des broches par default. N'importe quelle autre broche peut être programmée.
*2 Le brochage n'est pas défini par default et doit être configuré manuellement au setup.
*3 C'est aussi le nom de la macro C dans le code source.
*4 Le bus I2C qui est utilisé par défaut par de nombreuses bibliothèque.
*5 Le brochage I2C correspond à celui des cartes de developpement "devKitC" et "devKitM" officielles de espressif.

(version longue)

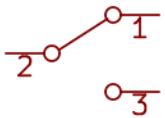
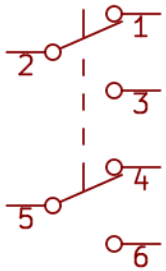
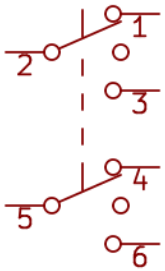
Le brochage des interfaces SPI des microcontrolleurs

microcontrolleur		spécification constructeur		Programmation			brochage			
marque	nom	protocole	bus	instance	bus	default*4	SS[3]	MOSI*3	MISO*3	SCK*3
Arduino	Uno R3/R4 Zero Leonardo Nano	SPI		SPI		✓	10	11	12	13
	Giga R1	SPI	?	SPI		✓	10	90	89	91
		SPI	?	SPI1			10 (SS1)	11 (MOSI1)	12 (MISO1)	13 (SCK1)
ESP32 (devKitC/M*5)	Wroom-32 D/U	SPI	2	SPI	HSPI	✓	5*1	23*1	19*1	18*1
		SPI	3	*7	VSPI		*2			
	C2/C3-Wroom	SPI	2	SPI	FSPI	✓	7*1	6*1	5*1	4*1
	C6-Wroom	SPI	2	SPI	FSPI	✓	18*1	19*1	20*1	21*1
	h2-MINI	SPI	2	SPI	FSPI	✓	0*1	25*1	11*1	10*1
	S2-Wroom	SPI	2	SPI	FSPI	✓	34*1	35*1	37*1	36*1
		SPI	3	*7	HSPI		*2			
	S3-Wroom	SPI	2	SPI	FSPI	✓	10*1	11*1	12*1	13*1
		SPI	3	*7	HSPI		*2			
teensy	4.1	SPI	4	SPI		✓	10	11	12	13
		SPI	3	SPI1			0*6	26*6	1*6	27*6
		SPI	1	SPI2			44*6	43*6	42*6	45*6

*1 Il s'agit des broches par default. N'importe quelle autre broche GPIO peut être programmée à la place.
*2 Non définies par default et doivent être configurées manuellement au setup. Toutes les broches GPIO peuvent être utilisées.
*3 C'est aussi le nom de la macro/variable C dans le code source.
*4 Le bus SPI qui est utilisé par défault par de nombreuses bibliothèques.
*5 Le brochage SPI correspond à celui des cartes de developpement "devKitC" et "devKitM" officielles de espressif.
*6 Aucune macro/variable C n'est définie pour ces broches dans le framework Arduinoteensy.
*7 L'instance SPI1 associée à l'interface SPI n'existe pas par default. Il faut la créer manuellement pour pouvoir l'utiliser :

(version longue)

Différents types d'interrupteur - 1/2

Bouton poussoir		Commutateurs à bascule ou à levier			
interrupteur monostable SPST OFF-MOM SPST OFF-(ON)	interupteur bistable SPST ON-OFF	SPDT ON-ON	SPDT ON-OFF-ON	permutateur DPDT ON-ON	DPDT ON-OFF-ON
					
	 				

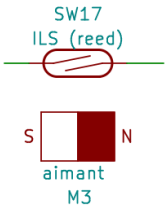
S = Single P = Pole D =Dual/Double T = Throw

ON = connecté à une broche OFF = non connecté MOM (ON) = Momentané

(version longue)

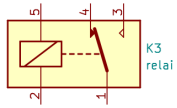
Différents types d'interrupteur - 2/2

Interrupteur magnétique : l'interrupteur se ferme prêt d'une source de champs magnétique.



Interrupteur à lame souple

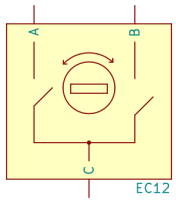
Interrupteur électrique : l'interrupteur se ferme quand un courant passe dans une bobine.



Relai

Consulter les pages 280, 281 et 282 pour plus de détail.

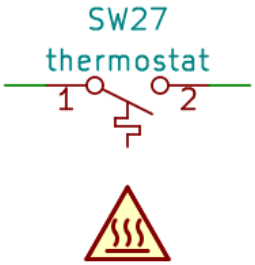
Encodeurs rotatifs : capteur constitué de deux interrupteurs permettant de mesurer la rotation d'un bouton.



Enocodeur EC12

Consulter la page 128 pour plus de détail.

Interrupteur thermique : l'interrupteur s'ouvre prêt d'une source de température.



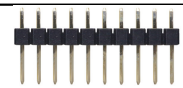
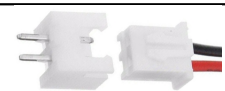
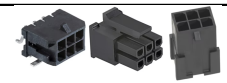
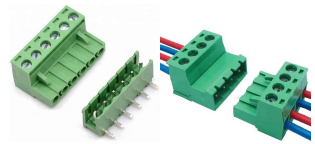
Fusible thermique

Thermostat bilames

Thermostat capillaire à bulbe

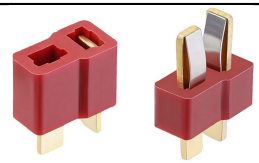
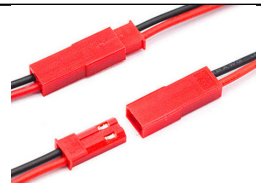
(version longue)

Les connecteurs les plus utilisés

désignation	pour platine d'expérim.	câble ↔ carte	câble ↔ câble	plugable	bornier	awg	A max	V max	empattement (mm)	image
Pin header 2.54	✓	✓		✓		30-22	3	250	2.54	
JST XH	✓	✓		✓		30-22	3	250	2.54	
JST SH		✓		✓		32-28	1	50	1.00	
Molex Micro-fit 3.0		✓	✓	✓		30-18	7.0		3.00	
JST SM	✓		✓	✓		28-22	3	250	2.54	
KF2EDGK-2.54 Pluggable Terminal Block	✓	✓	✓	✓	✓	28-20	4	125	2.54	
KF2EDGK-5.08 Pluggable Terminal Block	✓	✓	✓	✓	✓	24-12	10	300	5.08	
KF128-2.54 PCB Terminal Block	✓	✓			✓	26-18	6	150	2.54	
KF128-5.08 PCB Terminal Block	✓	✓			✓	22-12	10	300	5.08	
Jack DC Power		✓		✓			5-10	50	/	

(version longue)

Connecteurs pour batteries

désignation	A max	V max	AWG	pitch (mm)	résistance de contact (mΩ	prix unit. (euro)	image
XT30	15	500	18	5	0.7	0.3	
XT60	30	500	12	7.2	0.5	0.8	
EC3	25	500	14	8.4	0.6	0.9	
T-Dean	25	500	12			0.97	
JST RCY	3	250	28-22	2.5	20	0.38	

(version longue)

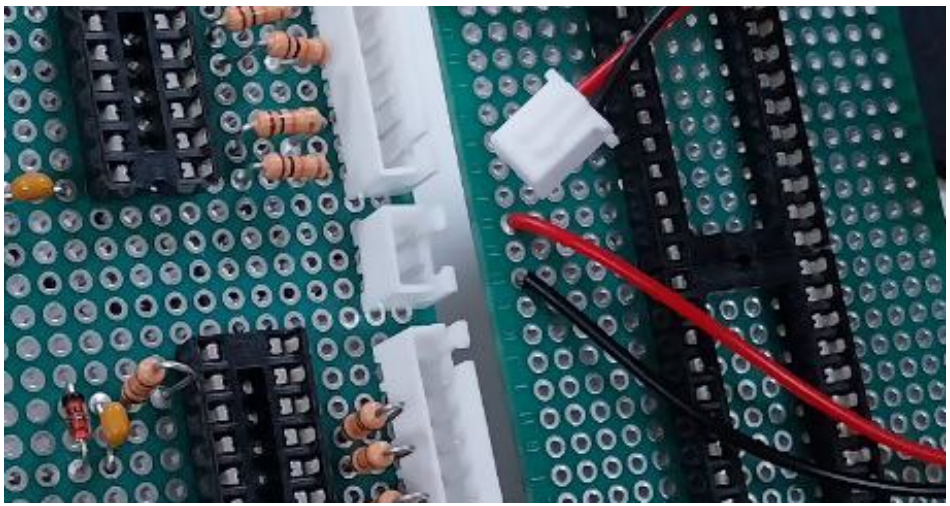
Comment utiliser les connecteurs

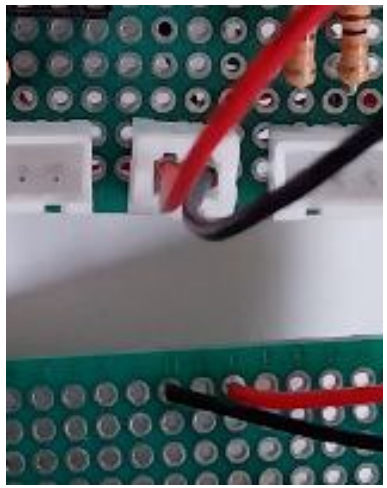
Le plus simple est d'utiliser des câbles avec des connecteurs déjà sertis sur un seul bout.

Ensuite, vous soudez :

- le connecteur de carte sur la carte mère;
- les fils sertis sur les cartes filles.

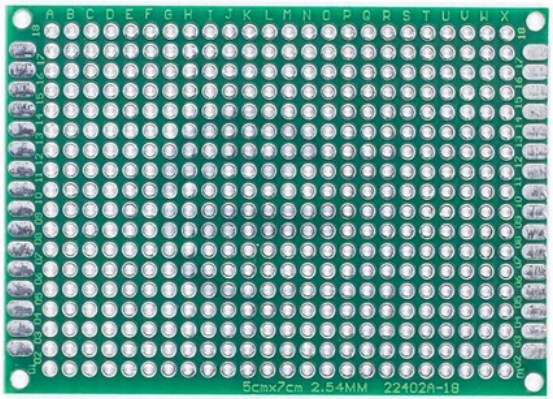
Il vous suffit alors de connecter le connecteur de la carte fille sur le connecteur du pcb de la carte mère.

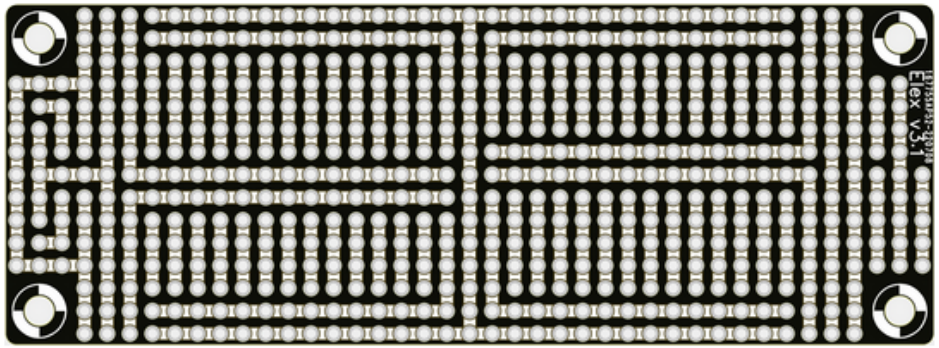


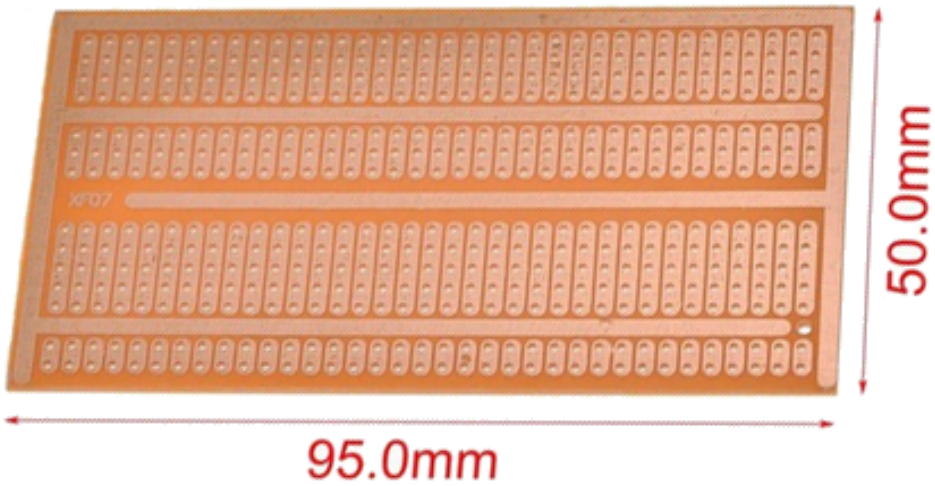


Les plaques d'expérimentation à souder

Voici quelques format de plaques d'expérimentation à souder :





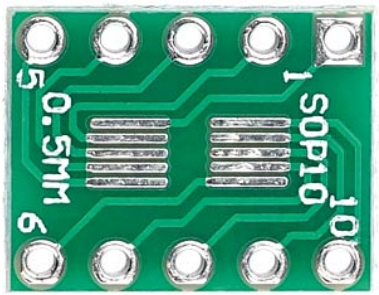


Le matériel compatible avec les platines d'expérimentation

Liste de matériels compatibles avec les platines d'expérimentation :

- le fil AWG 22 (0.64 mm de diamètre), mono-brin;
- les composants dont l'empattement est un multiple de 2.54 mm;
- les connecteurs duponts (pin headers) 2.54 mm : cf. page 657;
- les connecteurs JST XH : cf. page 657;
- les borniers KF128 (2.54 et 5.08) : cf. page 657;
- les borniers plugables KF2EDGK (2.54 et 5.08) : cf. page 657.

Les cartes adaptateurs SMD → DIP:



Piles ne nécessitant pas de protection (sans danger)

	désignation	matiere	assemblage nb de cellule	tension max	tension typique	tension déchargé	capacité (mAh)	rechargeable	prix (euro)
pile 1.5 V	LR8D425 AAAA	Alcaline ZNMnO2	1 × 1.5	1.5	1.3	0.9	250-650		
	LR03 AAA	Alcaline	1 × 1.5	1.5	1.1-1.3	0.9	350-1200		
	LR6 – AA	Alcaline	1 × 1.5	1.5	1.1-1.3	0.9	1200-2870		
	LR14 – C	Alcaline	1 × 1.5	1.5	1.1-1.3	0.9	5000-8200		
	LR20 – D	Alcaline	1 × 1.5	1.5	1.1-1.3	0.9	9000-16000		
	HR03 AAA	NiMH	1 × 1.2	1.4	1.2	1.0	800-1000	✓	
	HR6 – AA	NiMH	1 × 1.2	1.4	1.2	1.0	1900-2300	✓	
	HR14 – C	NiMH	1 × 1.2	1.4	1.2	1.0	2500-3000	✓	
	HR20 – D	NiMH	1 × 1.2	1.4	1.2	1.0	2500-3000	✓	
pile 4.5V	3LR12	Alcaline	3 × 1.5	4.5	3.4-3.9	2.7	1900-2500		
	3R12	carbone zinc	3 × 1.5			2.7	1000-2000		
pile 9V	6LR61	Alcaline	6 × 1.5	9.5	7-8	5.4	260-600		
	HR22	NIMH	6 × 1.2	9.5	8.4	7.0	175	✓	
	6F22	Carbon Zinc		9.5	6-8	5.4	270-370		

Ces chiffres sont tirés d'un mix des spécifications des piles VARTA et ENERGIZER.

Piles au lithium - 1 seule cellule : 1S

La liste de piles et batteries qui suivent sont au lithium. Ces piles sont dangereuses . Lisez la page 525 avant de choisir ces piles.

Les piles les plus sures sont les piles avec protections intégrés L91 et L533 et peuvent être utilisées telle quel (sans ajout de circuit de protection supplémentaire). Ensuite viennent les piles LiFePo4 car leurs températures d'emballement thermique sont les plus hautes (cf. figure page 525) mais il faut ajouter un circuit de protection présenté à la page 532.

Si vous mettez plusieurs cellules en séries, il faut utiliser des BMS spécifiques pour piloter et vérifier chaque cellule indépendamant les une des autres lors de la charge et la décharge !

désignation		matiere	assembl. nb de cellule	tension max	tension typique	tension déchargé	capacité (mAh)	recharg-eable	protection integré	prix (euro)
pile 1.5 V	L91 AA	LiFeS2	1 × 1.5	1.8	1.4-1.5	1.2	3500		✓	
pile 3.2V	LiFePo4 18650	LeFePo4	1 × 3.2	3.65	3.2	2.6	1400-1500	✓		
pile 3.2V	LiFePo4 26650	LiFePo4	1 × 3.2	3.65	3.2	2.6	1900-2300	✓		
pile 3.7V	Li-ion 18650	Li-ion	1 × 3.7	4.2	3.4-3.7	3.0	2350-2600	✓		
pile 3.7V	LiPo	LiPo	1 × 3.7	4.2		3.0	1000-2000*1	✓	*2	
pile 9V	L522	LiMnO2	1 × 9	9.0	7-8.2	5.4	750–1200		✓	

*1 Les tailles des cellules Lithium Polymer ne sont pas normalisées. Leurs capacité dépendent des constructeurs.
*2 Selon les constructeurs, ces cellules peuvent contenir des protections intégrés, mais jamais de circuits de charges et décharges.

(version longue)

Piles au lithium - assembl. de plusieurs cellules : 2S,3S,$\cdots$

A FAIRE : Finir la table.

Les séries E12, E24, E48 et E96 de résistances

E12 (10% tolérance) : 10, 12, 15, 18, 22, 27, 33, 39, 47, 56, 68, 82

E24 (5% tolérance): 10, 11, 12, 13, 15, 16, 18, 20, 22, 24, 27, 30, 33, 36, 39, 43, 47, 51, 56, 62, 68, 75, 82, 91

E48 (2% tolérance): 100, 105, 110, 115, 121, 127, 133, 140, 147, 154, 162, 169, 178, 187, 196, 205, 215, 226, 237, 249, 261, 274, 287, 301, 316, 332, 348, 365, 383, 402, 422, 442, 464, 487, 511, 536, 562, 590, 619, 649, 681, 715, 750, 787, 825, 866, 909, 953

E96 (1% tolérance) : 100, 102, 105, 107, 110, 113, 115, 118, 121, 124, 127, 130, 133, 137, 140, 143, 147, 150, 154, 158, 162, 165, 169, 174, 178, 182, 187, 191, 196, 200, 205, 210, 215, 221, 226, 232, 237, 243, 249, 255, 261, 267, 274, 280, 287, 294, 301, 309, 316, 324, 332, 340, 348, 357, 365, 374, 383, 392, 402, 412, 422, 432, 442, 453, 464, 475, 487, 499, 511, 523, 536, 549, 562, 576, 590, 604, 619, 634, 649, 665, 681, 698, 715, 732, 750, 768, 787, 806, 825, 845, 866, 887, 909, 931, 953, 976

Table de drivers et modules pour moteur

désignation	DC 1 sens	DC 2 sens	pas à pas 4 fils	pas à pas 5 fils	brushless	Nombre de demi-ponts*1	Nombre d'interrupteurs	Transistors intégrés*7	Diodes intégrés	Courrant max (A)	Tension d'alimentation (V)	total Ron (Ω) ou chute de tension	frequence max PWM (kHz)	mesure le courant	génére les PWM*2	nb max micro-step	régulateur PI couple/pos./vit.	protection optocoupleur	sortie 5V ou 3.3V	Dificulté*5	Prix (euros)
LM293 LM293D	✓	✓	✓	✓		4×1		b	× ✓	1 (2) 0.6 (1.2)	7-36	2.6V-3.6V		✓						m	
L298N	✓	✓	✓	✓		2×2		b		2 (3)	7-46	1.8V-4.9V	40	✓						d	
Module adafruit L298N	✓	✓	✓	✓		2×2		b	✓	2 (3)	7-46	1.8V-4.9V	40						5 1A	m	
L6205	✓	✓	✓	✓		2×2		m	✓	2.8 (5.6)	8-52	0.62-1.12	100	✓						d	
ULN2003	✓			✓			7	b	✓	0.5	50	0.9V-1.7V								m	
TB6600			✓	✓		1×4		m	✓	4 (4.0)	8-42	0.4-0.6			✓	16				d	
Microstep TB6600			✓	✓		1×4		m	✓	4 (4.0)	9-42	0.4-0.5			✓	16		✓		f	
A4988			✓	✓		1×4		m	✓	(2)	8-35	0.32-0.43		✓	✓	16				d	
module A4988			✓	✓		1×4		m	✓	(2)	8-35	0.32-0.43			✓	16				f	
DRV8825			✓	✓		1×4		m	✓	(2.5)	8.2-45	0.4-0.64	30	✓	✓	32			3.3 10mA	d	
module DRV8825			✓	✓		1×4		m	✓	(2.5)	8.2-45	0.4-0.64	30		✓	32				f	
DRV8313	✓	✓			✓	3×1		m	✓	1.2 (2.5)	8-60	0.48-0.78	250	✓					3.3 10mA	td	
L6234	✓	✓			✓	3×1		m	✓	4 (5)	7-52	0.48-0.78	150	✓						td	
IR2104	✓	✓*3	✓*3	✓*3	✓*3	1×1				∞*4	12-600	*4	76*6							ttd	
TMC4671	✓	✓	✓	✓	✓	4×1				∞*4	*4	*4	100	✓	✓*8		✓*8			ttd	

*1 : $X \times Y$ signifie que le driver permet de piloter indépendamant X groupes de Y demi-ponts. Par exemple, 2×2 siginfie que 2 ponts en H sont pilotables indépendament; *2 : Il génére donc les chronogrames aussi. *3 : Il faut utiliser 2, 4 et 3 IR2104 pour commander respectivement un moteur DC dans les 2 sens, un pas à pas 4/5 fils et un brushless; *4 : Dépends des transistors externes choisis. Consultez The Art of Electronics: The X-Chapters, page 242. *5 : f=facile, m=moyen, d=difficile, td=très difficile, ttd=très très difficile; *6 : fréquence de pwm calculée pour laquelle la durée de comutation ne transmettant

(version longue)

Table de microcontrôleurs

A FAIRE :

Les différentes familles de microcontrolleur de Espressif

Microcontrôleur	pinouts devkit	datasheet	freq. (MHz)	nb de coeurs	SRAM (KB)	ROM (KB)	nb GPIO	nb PWM	nb Touch	nb ADC 12-bit	nb DAC 8-bit	nb SPI	nb UART	nb I2C	nb I2S	USB OTG	Wi-fi 802.11 2.4 GHz	802.15.4	Bluetooth
ESP32	➡	➡	240	2	520	448	34	16	10	2	2	4	3	2	2		b/g/n		4.2 BR/ EDR LE
ESP32-C2 ESP8684	➡	➡	120	1	272	576	14	6	0	1	0	3	2	1	0		b/g/n		LE 5.3
ESP32-C3	➡	➡	160	1	400	384	22	6	0	2	0	3	2	1	1		b/g/n		LE 5 mesh
ESP32-C5	➡	?	240	1	384	320	29	6	0	1	0	2	2	1	0		ax 2.4GHz 5GHz	✓	LE 5
ESP32-C61	➡	?	160	1	320	256	25	6	0	1	0	2	3	1	1		ax		LE 5
ESP32-C6	➡	➡	160	1	512	320	30	6	0	1	0	1+2	1+1	1+1	1		b/g/n ax	✓	LE 5.3 mesh
ESP32-H2	➡	➡	96	1	320	128	19	?	0	1	0	1+2	2	2	1			✓	LE 5.3 mesh
ESP32-S2	➡	➡	240	2	320	128	43	8	14	2	2	4	2	2	1	✓	b/g/n		∅
ESP32-S3	➡	➡	240	2	512	384	45	8	14	2	0	2+2	3	2	2	✓	b/g/n		LE 5 mesh

Pour télécharger les documents techniques de ces microcontrolleurs, rendez-vous ici : Page de téléchargement des documentations techniques des ESP32.

(version longue)

Table de transistors usuels

A FAIRE :

Table d'amplificateurs et comparateurs usuels

A FAIRE :

Liste de materiel électronique pour débuter

A FAIRE :

Composant DIP / SMD

A FAIRE :

Plan

(version longue)

Index

(version longue)

(version longue)

(version longue)

(version longue)