UNIVERSITÉ DE SHERBROOKE

Groupe technique

PWM et filtres passe bas

Document pour les groupes techniques Biogénius et CCI

Rédigé par : Jacob Turcotte

En date du : 2 octobre 2024

Université de Sherbrooke

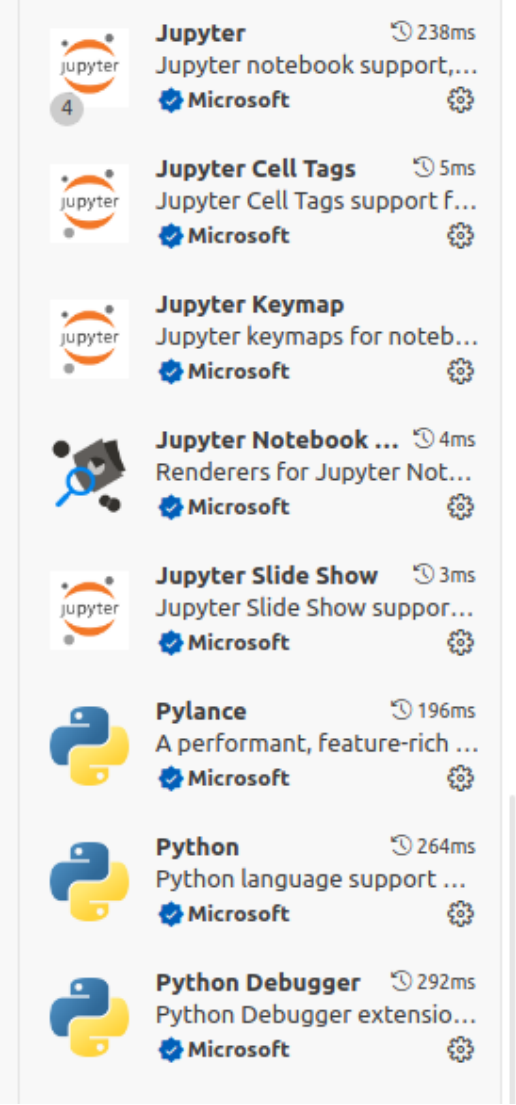
Table des matières

1. Introduction et setup

Ce document vous présente une activité portant sur les signaux PWM (modulation de largeur d'impulsion) et les filtres passe-bas analogiques. Vous aurez le choix entre deux options : la section électrique ou la section programmation. Si vous optez pour la section électrique, vous vous concentrerez sur la conception et l'analyse des circuits, incluant le montage d'un filtre passe-bas. Si vous choisissez la section programmation, vous travaillerez sur la génération et la manipulation des signaux PWM. Dans les deux cas, vous devrez utiliser GitHub pour télécharger le code de départ ainsi que les schémas électriques nécessaires à l'activité.

1.1. Installation nécessaire:

- VScode avec Extension C++ et Platformio (voir ancien atelier pour les images)
- GitHub (https://docs.github.com/en/desktop/installing-and-authenticating-to-github-desktop/installing-github-desktop)
- Optionnel pour le monde qui font la partie électrique:
    - Télécharger Jupyter et Python comme extension VScode



- Ils vous permettront d'ouvrir le notebook Jupyter et de changer les valeurs dans la simulation. Cela étant dit, c'est optionnel et non-nécessaire à l'activité. Tout calcul peut se faire sur calculatrice/ordinateur sans ces extensions.

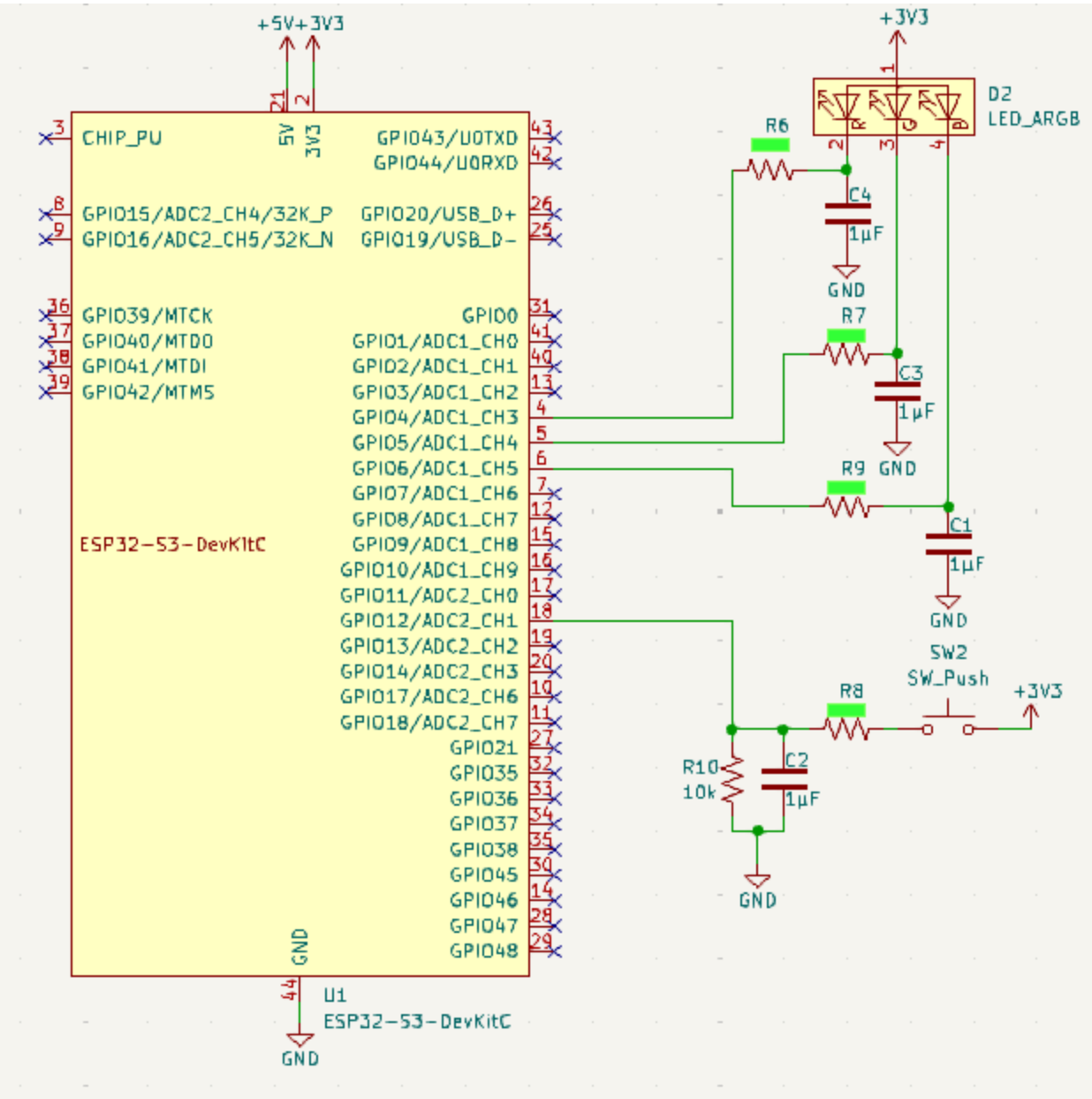
1.2. Comment forker le repo GitHub?

- Vous devez avoir un compte GitHub à votre nom. Si ce n'est pas déjà fait, prenez un petit 5 minutes pour le faire.
- Accédez au repo GitHub officiel de l'activité à l'adresse suivante:
  a. https://github.com/CCI-udes/PWM_Activity
- Cliquez sur le bouton "Fork" en haut à droite de la page du dépôt.
- Sélectionnez votre propre compte GitHub comme destination du fork.

- Clonez le repo que vous venez de fork sur votre machine locale (vous devez avoir GitHub d'installé sur vos ordinateurs). Il a deux options:
    a. (Façon plus facile et recommandé) Ouvrez l'application "GitHub Desktop" et sélectionnez l'option de cloner un repo. Sélectionnez l'option URL, copiez et collez le URL vers VOTRE repo. Appuyez sur clone et c'est fait.
        i. Example de lien:
            - https://github.com/<votre-utilisateur>/ PWM_Activity.git
    b. Ouvrez le terminal de votre ordinateur et naviguez vers le dossier où vous voulez stocker le repo. Une fois fait, tapez cette commande avec vote nom d'utilisateur GitHub:
        i. git clone https://github.com/<votre-utilisateur>/ PWM_Activity.git

2. Activité électronique

2.1. Schéma électrique



2.2. Lecture

Commencez par lire le PDF intitulé “Simulation_PWM_avec_filtre.pdf”, si vous avez installé les extensions optionnelles, ouvrez le document “Simulation_PWM_avec_filtre.ipynb” celui-ci est interactif et vous pouvez changer les valeurs afin de changer les simulations.
Je suis là si vous avez des questions ou si vous voulez d’autres explications.

2.3. Assemblage du circuit

Maintenant que vous avez une certaine compréhension de ce que nous allons faire, assemblé le circuit montré plus haut.

2.3.1. Valeur manquante

Les valeurs des résistances sont manquantes pour le bouton et les DEL RGB.

DEL RGB
Calculez les valeurs de résistances avec ces conditions:

- Le condensateur a une valeur de 1uF
- La fréquence du PWM est de 50kHz
- La tension source est de 3.3V
- Je veux que la tension de coupure soit d'environs 160Hz
- Je veux que l'ondulation maximal de la tension une fois le signal stabilisé soit moins de 25mV

Toutes les formules sont dans le document que vous venez de lire.

Bouton
Calculez la valeur de résistances avec ces conditions:

- Le condensateur a une valeur de 1uF
- Je veux que la fréquence de coupure soit environs de 50Hz

Une fois les valeurs trouver, venez me voir afin que je puisse valider avec vous et vous donner les résistances nécessaires.

2.4. Comment implémenté un PWM dans un code?

Puisque nous utilisons le Framework « Arduino », gros du travail est déjà fait pour nous.

analogWrite(pin, value)

Description :

- La fonction **analogWrite** permet de générer un signal PWM (modulation de largeur d'impulsion) sur une broche spécifique du microcontrôleur. Contrairement à ce que son nom peut laisser croire, l'ESP32 ne possède pas de sorties analogiques pures, donc cette fonction crée une "illusion" d'une sortie analogique en modulant la largeur d'impulsion d'un signal numérique à haute fréquence.

Paramètres :

- **pin** (int) : La broche sur laquelle vous souhaitez générer le signal PWM. Sur l'ESP32, presque toutes les broches peuvent générer un signal PWM, mais il est recommandé de vérifier les spécifications du microcontrôleur pour éviter les conflits avec d'autres fonctionnalités.

- **value** (int) : La valeur de largeur de l'impulsion en pourcentage. Cela détermine la proportion du temps pendant laquelle le signal est à l'état haut. La plage de valeurs dépend de la résolution PWM définie avec **analogWriteResolution** (par défaut, 0 à 255).

Sortie :

- Cette fonction ne retourne aucune valeur, mais elle configure la broche pour produire un signal PWM en fonction des paramètres donnés.

Utilité :

- **analogWrite** est utilisée pour contrôler des périphériques comme des LED (dimming), des moteurs (contrôle de la vitesse) ou d'autres composants nécessitant une sortie modulée. Sur un ESP32, c'est utile pour les applications qui nécessitent une régulation de puissance ou de la modulation de signaux.

analogWriteFrequency(pin, frequency)

Description :

- La fonction **analogWriteFrequency** permet de définir la fréquence du signal PWM sur une broche spécifique. La fréquence est le nombre de cycles PWM complets générés par seconde, mesuré en Hertz (Hz).

Paramètres :

- **pin** (int) : La broche sur laquelle vous souhaitez régler la fréquence PWM.
- **frequency** (int) : La fréquence du signal PWM, en Hertz. Sur l'ESP32, la fréquence PWM peut être configurée sur une large plage, généralement de quelques Hertz à plusieurs dizaines de kilohertz.

Sortie :

- Cette fonction ne retourne pas de valeur, mais elle définit la fréquence PWM de la broche spécifiée.

Utilité :

- Cette fonction est utile lorsqu'il est nécessaire de contrôler des périphériques qui répondent à des fréquences PWM spécifiques, comme des moteurs ou des circuits de commande d'alimentation. Par exemple, pour les moteurs, une fréquence plus basse peut causer des bourdonnements audibles, tandis qu'une fréquence plus haute résoudra ce problème.

analogWriteResolution(bits)

Description :

- La fonction **analogWriteResolution** permet de définir la résolution de la valeur PWM, c'est-à-dire le nombre de bits utilisés pour représenter la largeur d'impulsion. Par défaut, Arduino utilise une résolution de 8 bits (valeurs entre 0 et 255), mais sur

l'ESP32, il est possible de modifier cette résolution jusqu'à 16 bits (valeurs de 0 à 65535).

Paramètres :

- **bits** (int) : La résolution du PWM en bits. Les valeurs typiques sont 8, 10, 12, 15, ou 16 bits.

Sortie :

- Cette fonction ne retourne pas de valeur, mais elle modifie la plage de la valeur que vous pouvez passer à **analogWrite**.

Utilité :

- Une résolution plus élevée permet un contrôle plus fin de la largeur d'impulsion, ce qui est utile pour des applications de contrôle précis comme la régulation de puissance, la commande de moteurs, ou la gestion de la luminosité de LEDs avec plus de nuances.

2.5. Comment driver les DEL avec PWM?

En vous aidant de vos connaissances passées et de l'autre atelier, complété ces étapes:

2.5.1. Déclarer les pins de sortie pour les DEL

Inspirez-vous du dernier atelier et du schéma électrique pour la déclaration.

2.5.2. Déclarer les pins de d'entrée pour le bouton

Inspirez-vous du dernier atelier et du schéma électrique pour la déclaration.

2.5.3. Déclarer la fréquence du PWM et la définition du PWM

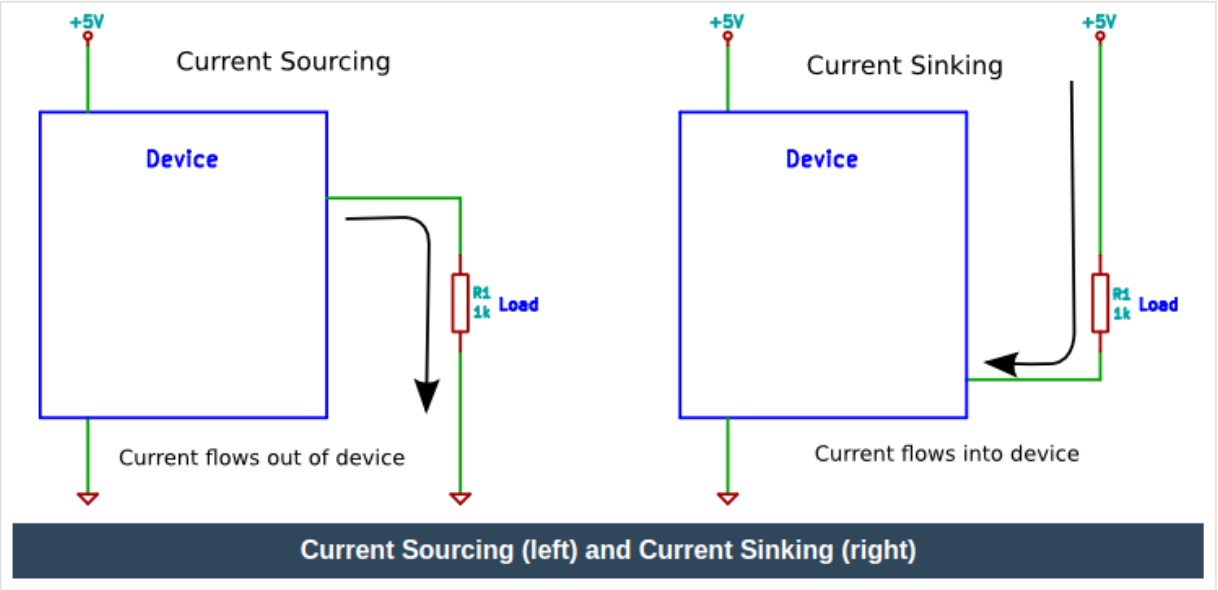
Je vous recommande une fréquence de 50kHz et une définition de 8bit pour l'atelier. Par défaut, les couleurs RGB sont décrite avec 3 int 8 bit. Ainsi, un PWM avec une définiton 8bit pour chaque DEL (rouge, vert et bleu) nous permet de faire toute la gamme RGB.

2.5.4. Tester dans le "main" analogwrite

Testez différentes valeurs, essayer d'allumer les 3 DEL afin de faire différentes couleurs.

2.5.5. Problème!

Si vous essayez d'allumer la DEL rouge à pleine intensité (soit: analogwrite(DEL_ROUGE, 255), la DEL va être éteinte. Si vous faites l'inverse (analogwrite(DEL_ROUGE, 0), la DEL va être à pleine puissance.
Analyser le schéma électrique et essayer de trouver la raison en arrière. (Indice: https://startingelectronics.org/articles/current-sourcing-sinking/#:~:text=An%20example%20of%20current%20sinking,current%20flows%20through%20the%20load.)



2.5.6. Solution

Écrivez une fonction qui résous le problème afin que 255 est ON à 100% et 0 soit OFF à 0%.

2.5.7. Problème PARTIE 2

Prenons la DEL rouge, lorsque la valeur du PWM est à 100, la DEL ne s'allume pas ou très peu, pourquoi?

Rappelez-vous qu'une DEL à une tension de seuil, si la tension fournie par notre circuit notre filtre RC est plus petit que la tension de seuil, la DEL n'émettra pas de lumière.

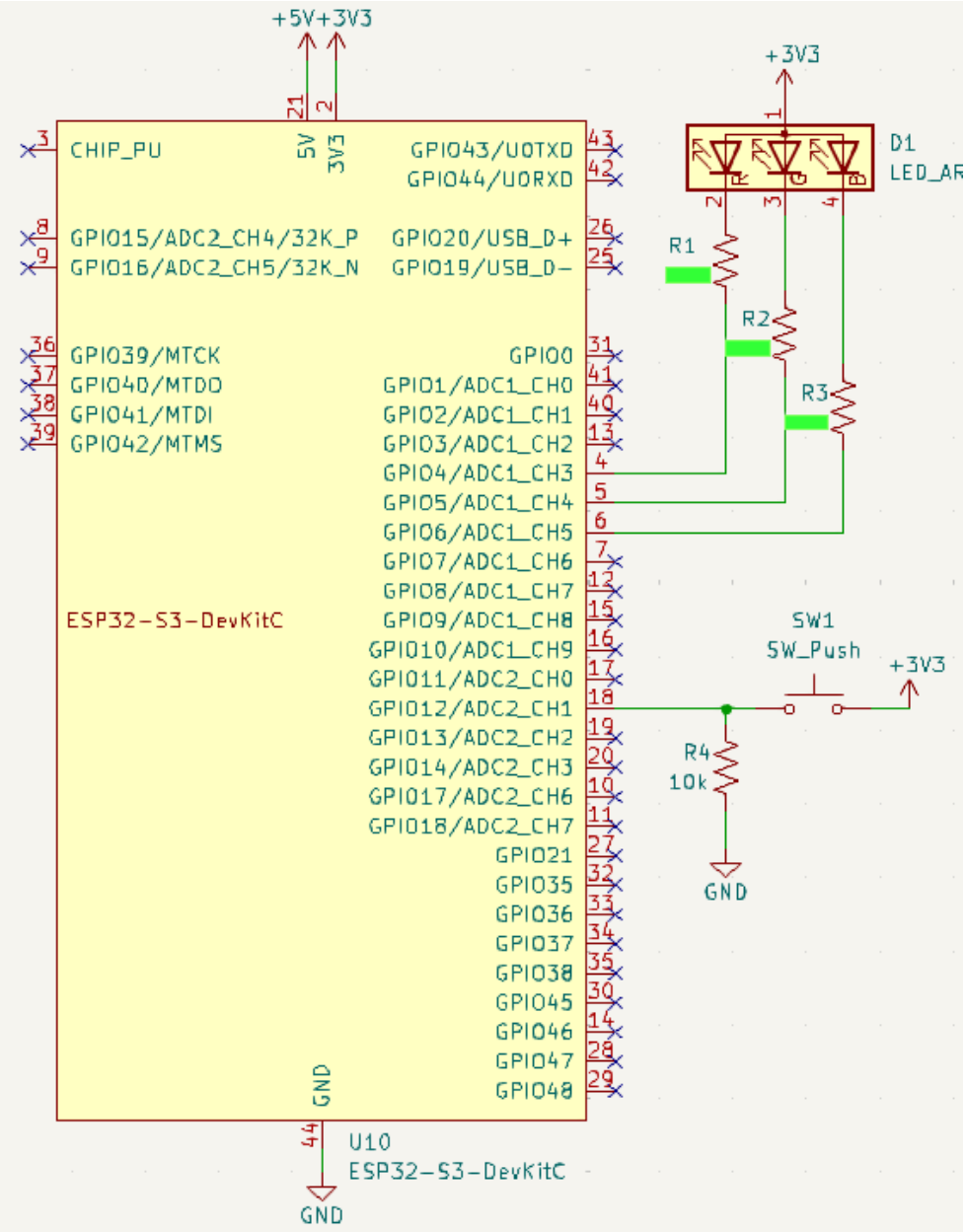
Avec des tests, trouvez la tension de seuil (ou la valeur du PWM) de chacune des DEL afin de s'assurer du bon fonctionnement de notre circuit.

2.5.8. Votre propre fonction

Écrivez une nouvelle fonction qui est appelé lorsque vous appuyez sur le bouton! Vous pouvez aller voir l'exemple fournis pour vous guider/inspirer (pas besoin de faire une aussi grosse fonction).

3. Activité de programmation

3.1. Schéma électrique



Comme vous voyez, les résistances R1, R2 et R3 n'ont pas de valeurs. Voici un extrait de la datasheet de la DEL RGB:

LED Chip Typical Electircal & Optical Characteristics: (Ta=25℃)

ITEMS	Color	Symbol	Condition	Min.	Typ.	Max.	Unit
Forward Voltage	Red	V_F	I_F=20mA	1.8	2.0	2.2	V
	Green			3.0	3.2	3.4	
	Blue			3.0	3.2	3.4	

Avec les informations fournies, trouvez les valeurs de R1, R2 et R3 si nous voulons un courant de 1mA passant dans les DEL. Une fois les valeurs trouver, venez me voir afin que je puisse valider avec vous et vous donner les résistances nécessaires.

Utilisez cet article pour vous faciliter la tâche.

Une fois terminer, complétez le montage du circuit. Si jamais vous pensez avoir un bug, ou bien vous voulez simplement une validation, s'il vous plaît me le dire afin que je puisse vous aidez.

3.2. C'est quoi un PWM?

La **Modulation de Largeur d'Impulsion** (PWM, pour *Pulse Width Modulation*) est une technique de modulation qui consiste à moduler la largeur des impulsions d'un signal numérique afin de contrôler la puissance délivrée à une charge. Elle est largement utilisée dans divers domaines, notamment dans le contrôle des moteurs, l'éclairage LED, et les alimentations à découpage.

3.2.1. Principes de fonctionnement

Le PWM fonctionne en alternant rapidement entre un état haut (ON) et un état bas (OFF). La proportion de temps pendant lequel le signal est à l'état haut par rapport à la période totale du signal détermine la duty cycle (rapport cyclique) du signal PWM.

Paramètres clés

- **Fréquence** : La fréquence du signal PWM détermine combien de fois par seconde le signal peut basculer entre l'état haut et l'état bas. Par exemple, une fréquence de 1 kHz signifie que le signal effectue 1000 cycles par seconde.
- **Duty Cycle** : Le rapport cyclique est défini comme le rapport du temps où le signal est à l'état haut par rapport à la période totale du signal. Il est généralement exprimé en pourcentage. Par exemple, un duty cycle de 50% signifie que le signal est à l'état haut pendant la moitié de la période et à l'état bas pendant l'autre moitié.

- **Formule** : Duty Cycle = (Temps en haut / Temps total) × 100 où “Temps en haut” est le temps pendant lequel le signal est à l’état haut et “Temps total” est la période totale du signal.

Applications du PWM

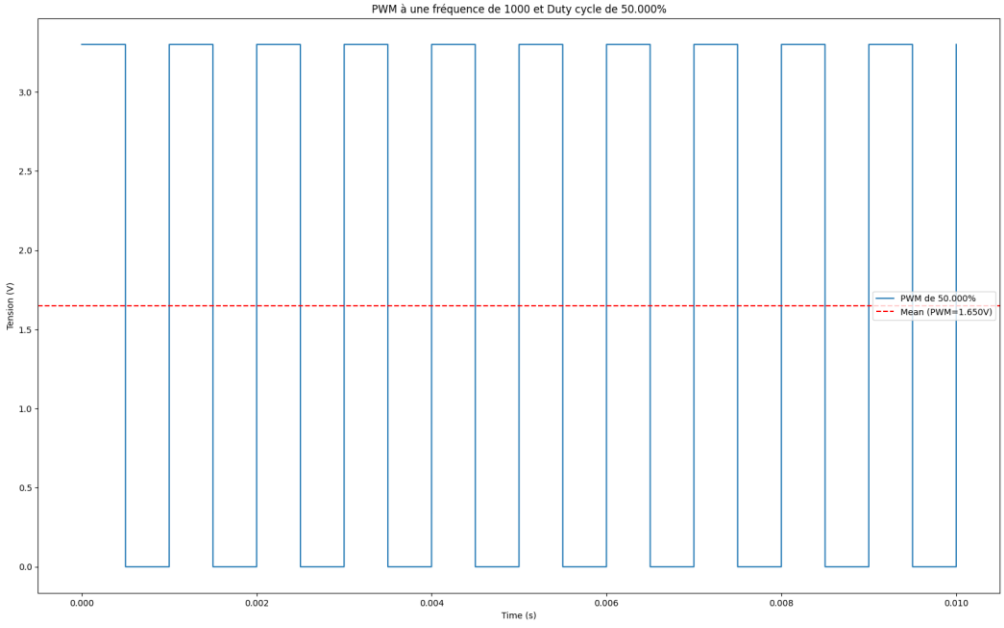
- **Contrôle de la vitesse des moteurs** : En ajustant le duty cycle d’un signal PWM, on peut contrôler la vitesse d’un moteur à courant continu. Un duty cycle plus élevé augmente la puissance moyenne fournie au moteur, ce qui augmente sa vitesse.
- **Dimming des DEL** : Le PWM permet de contrôler la luminosité des LED. En variant le duty cycle, on peut simuler différentes intensités lumineuses.
- **Alimentations à découpage** : Dans les circuits d’alimentation, le PWM est utilisé pour réguler la tension de sortie en contrôlant la durée des impulsions.

Avantages du PWM

- **Efficacité énergétique** : Le PWM est une méthode très efficace pour contrôler la puissance car il minimise les pertes d’énergie, contrairement aux méthodes analogiques comme la résistance variable.
- **Précision** : Permet un contrôle précis de la puissance et de la vitesse, rendant les systèmes plus réactifs. • Facilité de mise en œuvre : Les circuits PWM sont simples à concevoir et à mettre en œuvre dans des applications électroniques.

Paramètres de l’exemple

- **Fréquence** : 1000 Hz (fréquence de l’onde carrée)
- **Rapport cyclique PWM** : 0,5 (0,0 à 1,0)
- **Source** : 3,3 V (amplitude maximale du signal)



3.3. Comment implémenté un PWM dans un code?

Puisque nous utilisons le Framework « Arduino », gros du travail est déjà fait pour nous.

analogWrite(pin, value)

Description :

- La fonction **analogWrite** permet de générer un signal PWM (modulation de largeur d'impulsion) sur une broche spécifique du microcontrôleur. Contrairement à ce que son nom peut laisser croire, l'ESP32 ne possède pas de sorties analogiques pures, donc cette fonction crée une "illusion" d'une sortie analogique en modulant la largeur d'impulsion d'un signal numérique à haute fréquence.

Paramètres :

- **pin** (int) : La broche sur laquelle vous souhaitez générer le signal PWM. Sur l'ESP32, presque toutes les broches peuvent générer un signal PWM, mais il est recommandé de vérifier les spécifications du microcontrôleur pour éviter les conflits avec d'autres fonctionnalités.
- **value** (int) : La valeur de largeur de l'impulsion en pourcentage. Cela détermine la proportion du temps pendant laquelle le signal est à l'état haut. La plage de valeurs dépend de la résolution PWM définie avec **analogWriteResolution** (par défaut, 0 à 255).

Sortie :

- Cette fonction ne retourne aucune valeur, mais elle configure la broche pour produire un signal PWM en fonction des paramètres donnés.

Utilité :

- **analogWrite** est utilisée pour contrôler des périphériques comme des LED (dimming), des moteurs (contrôle de la vitesse) ou d'autres composants nécessitant une sortie modulée. Sur un ESP32, c'est utile pour les applications qui nécessitent une régulation de puissance ou de la modulation de signaux.

analogWriteFrequency(pin, frequency)

Description :

- La fonction **analogWriteFrequency** permet de définir la fréquence du signal PWM sur une broche spécifique. La fréquence est le nombre de cycles PWM complets générés par seconde, mesuré en Hertz (Hz).

Paramètres :

- **pin** (int) : La broche sur laquelle vous souhaitez régler la fréquence PWM.
- **frequency** (int) : La fréquence du signal PWM, en Hertz. Sur l'ESP32, la fréquence PWM peut être configurée sur une large plage, généralement de quelques Hertz à plusieurs dizaines de kilohertz.

Sortie :

- Cette fonction ne retourne pas de valeur, mais elle définit la fréquence PWM de la broche spécifiée.

Utilité :

- Cette fonction est utile lorsqu'il est nécessaire de contrôler des périphériques qui répondent à des fréquences PWM spécifiques, comme des moteurs ou des circuits de commande d'alimentation. Par exemple, pour les moteurs, une fréquence plus basse peut causer des bourdonnements audibles, tandis qu'une fréquence plus haute résoudra ce problème.

analogWriteResolution(bits)

Description :

- La fonction **analogWriteResolution** permet de définir la résolution de la valeur PWM, c'est-à-dire le nombre de bits utilisés pour représenter la largeur d'impulsion. Par défaut, Arduino utilise une résolution de 8 bits (valeurs entre 0 et 255), mais sur l'ESP32, il est possible de modifier cette résolution jusqu'à 16 bits (valeurs de 0 à 65535).

Paramètres :

- **bits** (int) : La résolution du PWM en bits. Les valeurs typiques sont 8, 10, 12, 15, ou 16 bits.

Sortie :

- Cette fonction ne retourne pas de valeur, mais elle modifie la plage de la valeur que vous pouvez passer à **analogWrite**.

Utilité :

- Une résolution plus élevée permet un contrôle plus fin de la largeur d'impulsion, ce qui est utile pour des applications de contrôle précis comme la régulation de puissance, la commande de moteurs, ou la gestion de la luminosité de LEDs avec plus de nuances.

3.4. Comment driver les DEL avec PWM?

En vous aidant de vos connaissances passées et de l'autre atelier, complété ces étapes:

3.4.1. Déclarer les pins de sortie pour les DEL

Inspirez-vous du dernier atelier et du schéma électrique pour la déclaration.

3.4.2. Déclarer les pins de d'entrée pour le bouton

Inspirez-vous du dernier atelier et du schéma électrique pour la déclaration.

3.4.3. Déclarer la fréquence du PWM et la définition du PWM

Je vous recommande une fréquence de 50kHz et une définition de 8bit pour l'atelier. Par défaut, les couleurs RGB sont décrite avec 3 int 8 bit. Ainsi, un PWM avec une définiton 8bit pour chaque DEL (rouge, vert et bleu) nous permet de faire tout la gamme RGB.

3.4.4. Tester dans le "main" analogwrite

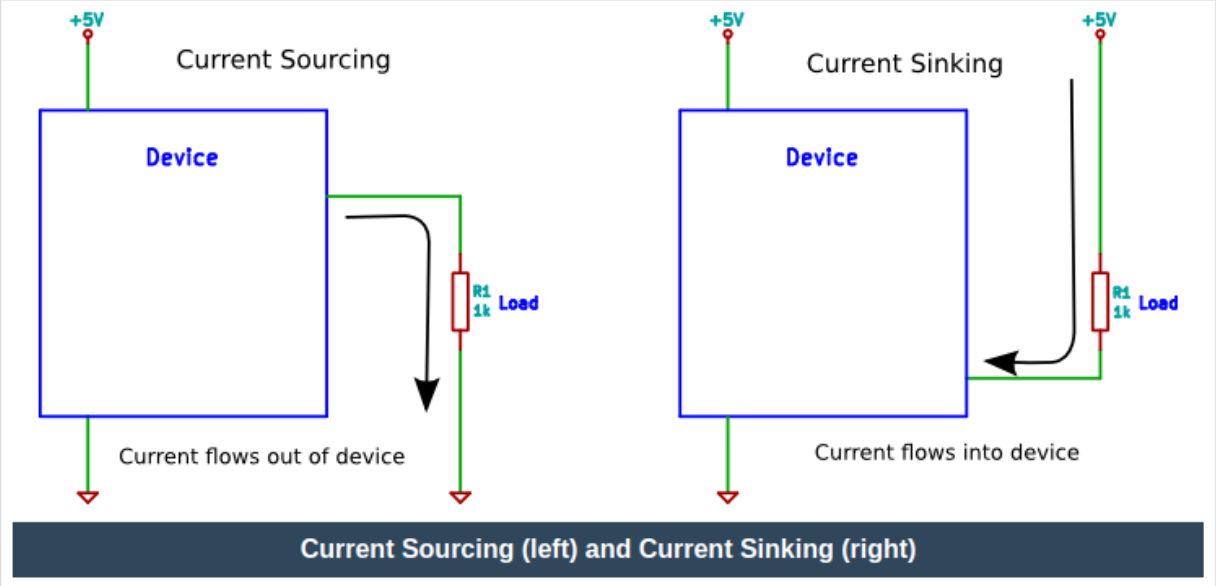
Testez différentes valeurs, essayer d'allumer les 3 DEL afin de faire différentes couleurs.

Indices:

- Utiliser ce lien pour vous aider à choisir les valeurs pour une couleurs désiré: https://www.rapidtables.com/web/color/RGB_Color.html

3.4.5. Problème!

Si vous essayez d'allumer la DEL rouge à pleine intensité (soit: analogwrite(DEL_ROUGE, 255), la DEL va être éteinte. Si vous faites l'inverse (analogwrite(DEL_ROUGE, 0), la DEL va être à pleine puissance.
Analyser le schéma électrique et essayer de trouver la raison en arrière. (Indice: https://startingelectronics.org/articles/current-sourcing-sinking/#:~:text=An%20example%20of%20current%20sinking,current%20flows%20through%20the%20load.)



3.4.6. Solution

Écrivez une fonction qui résous le problème afin que 255 est ON à 100% et 0 soit OFF à 0%.

3.4.7. Votre propre fonction

Écrivez une nouvelle fonction qui est appelé lorsque vous appuyez sur le bouton! Vous pouvez aller voir l'exemple fournis pour vous guider/inspirer (pas besoin de faire une aussi grosse fonction).