

CY IUT – GEII Neuville

Dossier de conception

Projet « Drumpad »

Instrument de percussion numérique automatisé

Document rédigé par l'équipe projet
Mise en forme par « VRONSKYI Volodymyr »
Version : 1.0 – 05/02/2025

Avant-propos

Ce document constitue le dossier de conception du projet « Drumpad », un système électronique conçu pour jouer des sons de manière mélodique dans 3 modes de fonctionnement distants, à la demande de l'IUT de Cergy Pontoise, afin que ce dernier puisse être présenté en tant qu'exemple aux journées de portes ouvertes.

L'objectif de ce dossier est de fournir une vision complète et détaillée des choix techniques, des étapes de conception, des validations et des résultats obtenus tout au long du développement du système. Il comporte également une partie introductive pour rappeler le contexte du projet, cependant ce n'est qu'un rappel de ce qui est décrit dans le cahier des charges de ce projet, où les modalités sont présentées de façon plus détaillée. Ce document sert à la fois de référence pour les parties prenantes du projet et de base pour les futures évolutions ou maintenances du système.

Table des matières

Avant-propos.....	1
Table des matières	1
1. Introduction	3
1.1. Contexte.....	3
1.2. Documents du projet.....	3
1.2.1. Cahier des charges (CDC).....	3
1.2.2. Dossier de conception.....	3
1.2.3. Dossiers de fabrication	3
1.2.4. Autres documents	3
1.3. Spécifications techniques.....	4
2. Conception du système	5
2.1. Choix de logiciels.....	5
2.1.1. Conception des schémas électroniques	5
2.1.2. Conception des circuits imprimés	5
2.1.3. Modélisation 3D.....	5
2.1.4. Programmation.....	6
2.1.5. Documentation.....	6
2.1.6. Diagrammes	6
2.2. Choix de microcontrôleur et langage de programmation.....	6
2.2.1. Choix de microcontrôleur	6
2.2.2. Choix de langage de programmation	7
2.3. Architecture du système.....	9
2.3.1. Approche modulaire.....	9
2.3.2. Structure du système.....	10

2.3.3.	Diagramme de blocks	10
2.4.	Conception physique.....	10
2.5.	Conception électronique	11
2.5.1.	Alimentation.....	11
2.5.2.	Amplificateur	13
2.6.	Conception informatique.....	15
2.6.1.	Détecter l'appuie sur un bouton.....	15
2.6.2.	piézoélectriques.....	15
	Conditionnement du signal du capteur.....	15
	Détection logicielle	16
2.6.3.	Jouer un son	17
2.6.4.	Jouer une séquence de sons préenregistrée.....	21
2.6.5.	Enregistrer une séquence de sons	21
2.6.6.	Envoyer les données en externe.....	21

1. Introduction

1.1. Contexte

Le projet « Drumpad » consiste en la conception d'un instrument de musique de percussion numérique automatisé, développé dans le cadre de la SAE (Situation d'Apprentissage et d'Evaluation) du BUT GEII (Génie Electrique et Informatique Industrielle) à l'IUT de Cergy-Pontoise.

Ce système doit être capable de fonctionner en trois modes : manuel, semi-automatique et automatique, tout en respectant des contraintes techniques, économiques et environnementales.

Ce dossier de conception documente les étapes du projet, de la conception à la réalisation, en passant par les tests et les validations.

1.2. Documents du projet

Cette section référence les documents et ressources associées au projet. Ces documents sont essentiels pour la compréhension, la réalisation, et la maintenance du système. Tous les documents sont en libre accès, et peuvent être consultés sur le répertoire GitHub du projet (voir références à la fin du document).

1.2.1. Cahier des charges (CDC)

Le CDC constitue la base du projet, avec le contexte, les objectifs, les exigences fonctionnelles et techniques, ainsi que les contraintes à respecter. Il définit en détail les attentes en matière de fonctionnement, performance, sécurité, et respect de l'environnement.

1.2.2. Dossier de conception

Le dossier de conception présente tout le parcours de la réalisation du projet. Il détaille les choix pour le système, décrit les schémas électroniques et les calculs associés, la programmation de la partie numérique du système, ainsi que les étapes et les résultats de tests.

1.2.3. Dossiers de fabrication

Les dossiers de fabrication regroupent les plans de fabrication des cartes électroniques, les instructions d'assemblage du boîtier et des composants, ainsi que les procédures de tests. Chaque sous-système du projet comporte son propre dossier de fabrication.

1.2.4. Autres documents

En complément, d'autres documents sont disponibles dans le répertoire du projet, tels que les rapports de tests, les fiches techniques de composants utilisés dans le projet, la documentation logicielle, le manuel d'utilisation d'interface logicielle, et les images concernant la réalisation du projet.

1.3. Spécifications techniques

Cette section comporte un rappel sur les fonctions du système présentées dans le cahier des charges.

Fonctions principales :

- **FP0** : Modifier l'environnement sonore.
- **FP1** : Jouer des sons de manière manuelle.
- **FP2** : Jouer des sons de manière semi-automatique (séquences préenregistrées).
- **FP3** : Jouer des sons de manière automatique (commandes externes).

Fonctions techniques :

- **FT1** : Enregistrer l'appui des touches.
- **FT2** : Jouer le son attribué à la touche.
- **FT3** : Jouer une partition de sons préenregistrée.
- **FT4** : Stocker les sons dans la mémoire interne.
- **FT5** : Modifier le stockage interne depuis l'extérieur.
- **FT6** : Avoir une interface visuelle.
- **FT7** : Permettre le réglage de paramètres.
- **FT8** : Permettre l'alimentation depuis le secteur et depuis la batterie.
- **FT9** : Être synchronisé sur une horloge.

Fonctions secondaires :

- **FS1** : S'intégrer dans l'ensemble d'instruments.
- **FS2** : Assurer le retour visuel d'appuie sur une touche.

Fonctions de contrainte :

- **FC1** : Jouer toutes les notes d'une octave complète avec demi-tons.
- **FC2** : Assurer la justesse de sonorité et afficher la(les) fréquence(s) jouée(s).
- **FC3** : Assurer la rapidité du système et effectuer un calcul de latence.
- **FC4** : Respecter les normes de sécurité électrique et assurer la compatibilité électromagnétique.
- **FC5** : Être économe en énergie et favoriser la réutilisation de matériel.
- **FC6** : Offrir une bonne ergonomie et facilité d'utilisation.

2. Conception du système

A rajouter : des alternatives de logiciels pour chaque choix

2.1. Choix de logiciels

Cette section présente les logiciels sélectionnés pour la conception, la simulation, la programmation, et la documentation du projet. Les choix ont été guidés par les exigences du projet, la disponibilité des outils, leur compatibilité avec les composants utilisés, et leur adéquation avec les compétences de l'équipe. Chaque choix est argumenté et des alternatives similaires sont donnés à titre d'exemple pour chaque logiciel.

2.1.1. Conception des schémas électroniques

Pour la conception des schémas électroniques et les simulations, le logiciel Proteus a été retenu. C'est un outil complet qui intègre à la fois un environnement de conception de schémas et un simulateur de circuits. Il permet de valider le fonctionnement des circuits avant leur réalisation physique, réduisant ainsi les risques d'erreurs et de dégradation de matériel, notamment des composants, qui peuvent facilement être endommagés. Proteus présente une bibliothèque large de composants de tout type, ce qui permet d'y retrouver quasiment tout composant qui pourrait être utilisé pour le projet. De plus, ce logiciel est celui utilisé depuis des années dans l'IUT pour apprendre les étudiants à réaliser les schémas et circuits électroniques, dû à la facilité de prise en main, ainsi que l'intégralité de fonctions que ce logiciel offre.

Dans le cadre de ce projet, Proteus sera donc utilisé pour :

- Schématisation de circuits : Création de circuits électroniques de différents sous-systèmes du projet.
- Simulation de circuits : Vérification de fonctionnements attendus de circuits.
- Schémas de présentation : Extraction de schémas en tant qu'image pour la présentation dans les documents.

2.1.2. Conception des circuits imprimés

Pour la conception des circuits imprimés (PCB), le logiciel KiCad a été retenu. C'est un logiciel open-source gratuit et puissant, qui offre toutes les fonctionnalités nécessaires pour concevoir des PCB professionnels. Il inclut un éditeur de schémas, un outil de routage et un visualiseur 3D pour vérifier l'intégration mécanique des composants. Son format de fichiers ouvert et sa compatibilité avec les outils de fabrication en font un choix adapté aux contraintes budgétaires du projet. De plus, KiCad est soutenu par une communauté active, ce qui facilite la résolution des problèmes et l'apprentissage.

Dans le cadre de ce projet, KiCad sera donc utilisé pour :

- Routage des pistes : Traçage des connexions électriques entre les composants.
- Visualisation 3D : Vérification 3D du circuit afin de prévoir les dimensions de circuits et d'éviter les incompatibilités de composants.
- Génération de fichiers de fabrication : Création de fichiers Gerber pour l'impression de circuits électroniques, et création de guides d'assemblage.

2.1.3. Modélisation 3D

La modélisation 3D du boîtier et des composants mécaniques est réalisée avec Autodesk Fusion 360, un logiciel reconnu pour sa polyvalence, sa facilité d'utilisation, et ses fonctionnalités avancées en conception mécanique. Fusion 360 permet de créer des modèles 3D précis, de simuler les contraintes

mécaniques, et de générer des fichiers pour l'impression 3D. De plus, il offre une version gratuite pour les étudiants, ce qui le rend accessible dans le cadre de ce projet. L'importation dans l'espace de travail de modèles 3D de circuits imprimés, comme ceux générés par Proteus ou KiCad, a aussi été un argument de choix pour ce logiciel.

2.1.4. Programmation

Le développement du logiciel embarqué est réalisé avec Visual Studio Code (VS Code), directement sur une Raspberry Pi 4. VS Code est un éditeur de code léger, puissant et extensible, qui supporte une large gamme de langages de programmation (C, C++, Python, etc.). Son intégration avec des outils de versionnement comme Git et sa compatibilité avec les extensions en font un choix idéal pour la programmation. L'utilisation de la Raspberry Pi 4 comme environnement de développement permet de tester le code en conditions réelles, tout en bénéficiant d'une plateforme performante.

2.1.5. Documentation

La rédaction des documents techniques, y compris le cahier des charges, le dossier de conception et les rapports de tests, est réalisée avec la suite Office, notamment Word et Excel. Ces outils sont largement utilisés dans le milieu professionnel et académique pour leur polyvalence et leur compatibilité avec la plupart des systèmes d'exploitation. Word permet une mise en forme claire et professionnelle des documents, tandis qu'Excel est utilisé pour les tableaux de données, les calculs et les graphiques. Leur intégration avec d'autres outils, comme OneDrive, facilite également le partage et la collaboration, ainsi que la rédaction partagée en temps réel.

2.1.6. Diagrammes

Pour la création de diagrammes fonctionnels, d'organigrammes et autres schémas, le site Draw.io a été choisi. Draw.io est un outil en ligne gratuit, simple d'utilisation et ne nécessitant aucune installation. Il permet de créer des diagrammes professionnels rapidement et de les exporter dans différents formats (PDF, PNG, etc.). Sa compatibilité avec les systèmes cloud (comme Google Drive) en fait un outil pratique pour le travail collaboratif.

2.2. Choix de microcontrôleur et langage de programmation

2.2.1. Choix de microcontrôleur

Le système nécessite un microcontrôleur capable de gérer plusieurs entrées/sorties (I/O), de traiter des signaux en temps réel, et de communiquer avec divers périphériques tels que des capteurs, des haut-parleurs et un écran d'interface utilisateur. Après une analyse approfondie des options disponibles, nous avons retenu la Raspberry Pi 4 comme solution optimale pour ce projet.

Contrairement aux microcontrôleurs classiques comme les STM32 ou ESP32, la Raspberry Pi 4 dispose d'un système d'exploitation complet (Linux) qui permet d'exécuter plusieurs processus simultanément. Cette capacité est essentielle pour notre projet, qui nécessite à la fois la gestion des entrées/sorties, le traitement des signaux audio et l'affichage d'une interface utilisateur. De plus, la Raspberry Pi 4 supporte une large gamme de langages de programmation (Python, C, C++, JavaScript, etc.), ce qui facilite le développement et la maintenance du logiciel embarqué.

En comparaison avec les FPGA, bien que ceux-ci offrent des performances supérieures pour le traitement parallèle, ils nécessitent une programmation bas niveau (VHDL/Verilog) et ne disposent pas d'un système d'exploitation intégré. Cela rendrait la gestion des tâches complexes, comme l'interface utilisateur ou la communication réseau, beaucoup plus contraignante.

La Raspberry Pi 4 est un micro-ordinateur monocarte (SBC) développé par la Fondation Raspberry Pi. Elle offre une combinaison unique de puissance de calcul, de connectivité et de flexibilité, ce qui en fait un choix idéal pour notre application. Voici ses principales caractéristiques :

Processeur : Un processeur Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) cadencé à 1,5 GHz, offrant des performances haut niveau pour les systèmes embarqués.

Entrées/Sorties (GPIO) : 40 broches GPIO configurables, incluant :

- **UART** : 6 interfaces (dont certaines partagées avec d'autres fonctions).
- **SPI** : 6 interfaces matérielles.
- **I²C** : 6 interfaces matérielles.
- **PWM** : Jusqu'à 2 canaux matériels, avec possibilité d'en ajouter via logiciel.
- **GPIO** : 28 broches utilisables pour des entrées/sorties générales.

Connectivité :

- 2 ports USB 3.0 et 2 ports USB 2.0 pour connecter des périphériques externes.
- 2 ports micro-HDMI supportant une résolution jusqu'à 4K pour l'affichage.
- Ethernet Gigabit et Wi-Fi 5 GHz pour la communication réseau.
- Un slot microSD pour le stockage du système d'exploitation et des données.

Système d'exploitation : Compatible avec plusieurs distributions Linux, ce qui permet une gestion multitâche et une programmation flexible.

Les raisons principales de choix de ce microcontrôleur sont donc :

- **Puissance de calcul** : Le processeur quad-core de la Raspberry Pi 4 est suffisamment puissant pour gérer le traitement des signaux audio, l'analyse des fréquences et l'exécution de l'interface utilisateur en temps réel.
- **Connectivité étendue** : Les nombreuses interfaces GPIO, USB et réseau permettent de connecter facilement les capteurs, les haut-parleurs, l'écran et d'autres périphériques nécessaires au fonctionnement du système.
- **Multitâche** : La capacité à exécuter plusieurs processus en parallèle est cruciale pour gérer simultanément les entrées des touches, la génération des sons et l'affichage de l'interface.
- **Facilité de développement** : L'environnement Linux et la compatibilité avec plusieurs langages de programmation simplifient le développement, les tests et la maintenance du logiciel.
- **Coût et accessibilité** : La Raspberry Pi 4 est une solution économique et en disposition à l'IUT, ce qui est en adéquation avec les contraintes budgétaires du projet.

2.2.2. Choix de langage de programmation

La programmation du système sera réalisée en C, un choix motivé par plusieurs facteurs techniques et pratiques. Le langage C est particulièrement adapté aux exigences du projet, notamment en termes de performance, de contrôle matériel et de portabilité.

Le langage C est réputé pour sa performance et son efficacité, ce qui en fait un choix idéal pour les systèmes embarqués où les ressources sont limitées. Contrairement à des langages de plus haut niveau

comme Python, le C permet un contrôle précis de la mémoire et des ressources matérielles, ce qui est essentiel pour garantir un traitement en temps réel des signaux audio, une latence minimale lors de la génération des sons, ce qui conduit aussi à une consommation d'énergie plus basse grâce au temps de traitement réduit.

Le projet nécessite une interaction directe avec les broches GPIO de la Raspberry Pi 4 pour lire les entrées des touches et contrôler les sorties audio. Le langage C offre un accès de bas niveau au matériel, permettant une gestion fine des entrées/sorties et des périphériques. Cela est crucial pour garantir une réponse rapide et fiable du système aux interactions de l'utilisateur.

Le langage C bénéficie d'une large communauté de développeurs et d'une abondance de ressources documentaires, bibliothèques et outils de développement. Cela facilite la résolution des problèmes, l'apprentissage et l'intégration de fonctionnalités supplémentaires. Par exemple, des bibliothèques comme « WiringPi » ou « pigpio » permettent de simplifier la gestion des GPIO sur la Raspberry Pi.

Parmi de nombreux langages, ceux qui étaient gardés pour le choix final :

- **Python** : Bien que Python soit plus facile à apprendre et à utiliser, il est moins performant que le C en termes de vitesse d'exécution et de contrôle matériel. Pour un projet nécessitant une latence minimale et un traitement en temps réel, le C est plus adapté.
- **C++** : Le C++ offre des fonctionnalités supplémentaires comme la programmation orientée objet, mais il est également plus complexe à mettre en œuvre. Dans le cas de ce projet, le C est suffisamment puissant et plus simple à gérer.

Assembleur : Bien que l'assembleur offre un contrôle maximal du matériel, il est beaucoup plus difficile à écrire et à maintenir. Le C offre un bon compromis entre performance et facilité de développement.

De plus, des environnements de développement intégrés (IDE) comme Visual Studio Code (VS Code) offrent un support complet pour le C, facilitant l'écriture, le débogage et la gestion du code.

2.3. Architecture du système

Contrôle de fabrication : schémas élec 3D, implantation élec, soudage, etc.

L'architecture du système Drumpad a été conçue pour répondre aux exigences de performance, de modularité et de maintenance. Nous avons opté pour une approche verticale, qui permet une meilleure organisation des sous-systèmes, une réparabilité simplifiée et une maintenance plus efficace grâce à une accessibilité simplifiée. Cette section explique les raisons de ce choix et décrit en détail l'architecture retenue.

2.3.1. Approche modulaire

Afin de rendre le système facilement réparable une approche de système modulaire a été choisie. C'est une approche largement utilisée dans les systèmes électroniques. Elle permet d'augmenter la réparabilité, et rend la maintenance du système plus rapide et simple.

Il existe deux approches modulaires, l'approche horizontale, et l'approche verticale :

Approche horizontale :

Dans cette approche, les fonctionnalités sont réparties sur des modules indépendants qui se connectent à une carte principale. Par exemple, les shields pour la Raspberry Pi 4 (comme un shield GPIO, un shield audio, etc.) sont des modules horizontaux qui ajoutent des fonctionnalités spécifiques à la carte principale. Cette approche est flexible et permet une extension facile du système, mais elle peut entraîner une complexité accrue dans la gestion des interconnexions et des interfaces.



Tableau 1: Exemple d'un système modulaire horizontal

Approche verticale :

Dans cette approche, les sous-systèmes sont organisés en couches fonctionnelles, où chaque couche a une responsabilité spécifique (alimentation, traitement, entrées/sorties, etc.). Par exemple, une carte mère traditionnelle utilise une approche verticale, avec des composants comme la carte graphique, la carte son et la RAM intégrés directement sur la carte. Cette approche offre une meilleure intégration des fonctions, une réduction des interconnexions et une maintenance simplifiée.



Tableau 2: Exemple d'un système modulaire vertical

Pour le projet, l'approche verticale a été choisie pour les raisons suivantes :

- **L'intégration optimisée** : L'approche verticale permet une meilleure intégration des sous-systèmes, réduisant les risques d'interférences et améliorant la fiabilité globale du système.
- **Réparabilité et maintenance** : En séparant les fonctions sur des modules distincts, il est plus facile de diagnostiquer et de remplacer un sous-système défaillant sans affecter l'ensemble du

système. Par exemple, si le régulateur de tension tombe en panne, il suffit de remplacer le module correspondant sans toucher aux autres parties du système.

- **Optimisation de l'espace** : Un système modulaire permet d'utiliser l'espace vertical, ce qui permet d'optimiser l'utilisation d'espace de manière générale.
- **Evolutivité** : L'approche modulaire permet d'intégrer des améliorations au système très facilement. Un module de sous-système peut être remplacé sans devoir changer le reste du système.

2.3.2. Structure du système

Comme détaillé dans la partie [Approche modulaire](#), un système modulaire est composé d'une carte « mère », et de modules qui se connectent dessus. Dans le cadre de ce projet, la « carte mère » sera la carte de distribution d'énergie et d'information, appelée dans la suite « carte de distribution ». Le système comporte donc les sous-systèmes suivants :

- **Carte de distribution** : Cette carte centralise l'alimentation électrique et les signaux de communication entre les différents modules. Elle assure une distribution efficace de l'énergie et des données, étant la base pour les modules.
- **Modules séparés** :

Chaque sous-système est implémenté sur un module distinct, qui se connecte à la carte de distribution. Les modules incluent :

- **Gestion de la charge de la batterie** : Pour gérer la recharge de la batterie et assurer une alimentation stable lors du fonctionnement depuis la batterie.
- **Alimentation de la Raspberry Pi 4** : Pour alimenter la Raspberry Pi 4 de manière sécurisée.
- **Amplificateur audio** : Pour amplifier le signal sonore avant l'arrivée vers les haut-parleurs.
- **Convertisseur numérique – analogique** : Pour convertir le signal audio numérique en sortie de la Raspberry Pi 4 en signal analogique.
- **Interface de la matrice de boutons** : Pour réguler la tension envoyée par les capteurs vers le microcontrôleur et gérer les adresses des boutons.

2.3.3. Diagramme de blocks

Représenter le système sous forme de diagramme

- Entrées
- Traitement (Rpi4)
- Sorties (hautparleurs, écran, retour visuel)
- Alimentation (secteur et batterie)
- Communication (MIDI)

2.4. Conception physique

Conception du boîtier



Figure 1 : Boîtier - modèle 3D

2.5. Conception électronique

Tout en rapport avec l'électronique

2.5.1. Alimentation

Cette partie traite sur le dimensionnement et la réalisation des sous-systèmes d'alimentation. Tout d'abord on fait une estimation de consommation par le système :

Sous-système	RPi 4 + écran	CNA	Amplificateurs	Haut-parleurs	Recharge batterie	Total
Tension	5V	12V	12V	X	12V	X
Courant	3A (max)	0.1A	0.4A (max)	0.67A	0.5A	4.67A
Puissance	15W	1.2W	3W	8W	6W	33.2W

Tableau 3 : Calcul de puissance consommée par le système

On estime donc une consommation de 33,2W avec un courant maximum de 4,67A. On fait le choix de fixer l'alimentation à 12V, 5A, ce qui donne une puissance maximale de 60W.

Alimentation principale

L'alimentation du système est basée sur une conception classique, utilisant un transformateur, un pont de diodes, et un régulateur de tension. Cette section décrit en détail la conception de cette alimentation, dimensionnée à partir du calcul de puissance précédent.

Le schéma suivant permet de transformer les 230V alternatifs en 12V 5A continus à l'aide d'un régulateur de tension LM338.

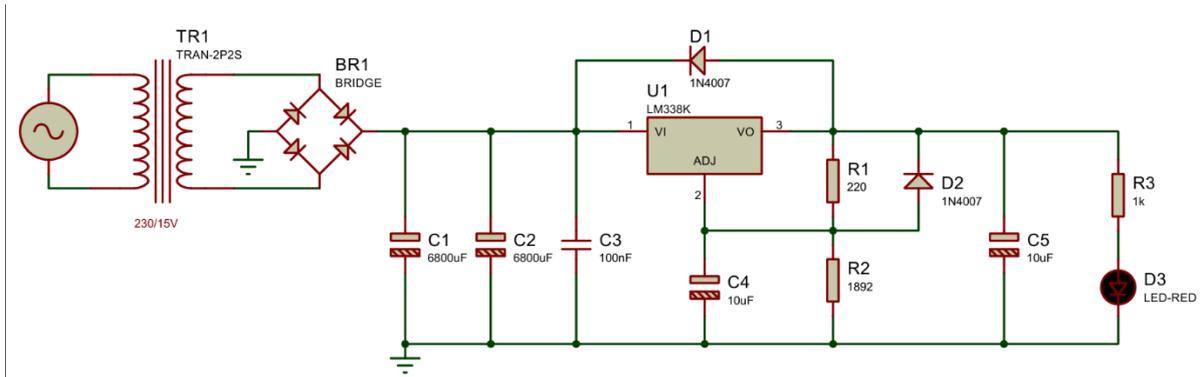


Figure 2: Schéma - Alimentation 12V 5A

Le transformateur permet d'abaisser la tension du secteur (230V AC) à une tension adaptée à l'alimentation du système. Pour obtenir une tension de sortie de 12V après la régulation, on choisit un transformateur avec une tension secondaire de 15V. La puissance requise est :

$$P = V * I = 15V * 5A = 75 VA$$

On choisira donc un transformateur avec un secondaire de 15V et une puissance d'au moins 75VA, afin d'éviter la surchauffe.

Rajouter explication de résistance

Alimentation de RPi4

La Raspberry Pi 4 nécessite une alimentation stable de 5 V pour fonctionner correctement. Pour répondre à cette exigence, nous utilisons un régulateur de tension LM7805, un composant largement utilisé pour sa simplicité et sa fiabilité. Cette section décrit la conception de cette alimentation, en expliquant le rôle de chaque composant et en justifiant les choix techniques.

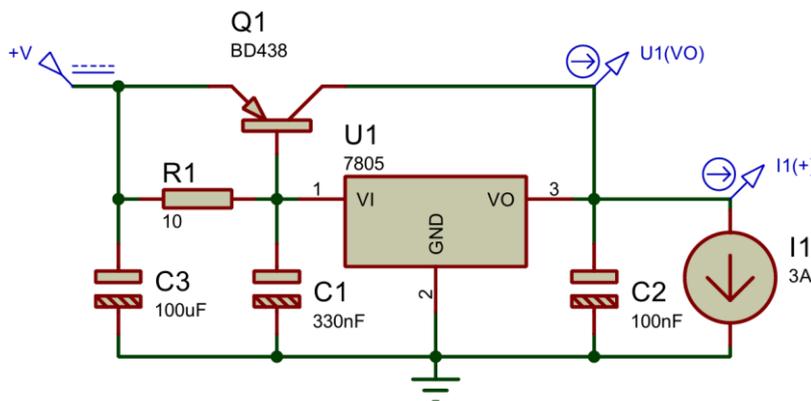


Figure 3: Schéma - Alimentation RPi4 5V3A

Le schéma de l'alimentation pour la Raspberry Pi 4 est composé des éléments suivants :

1. **Source d'entrée** : Une tension d'entrée de **12 V** provenant de la batterie ou de l'alimentation principale.
2. **Régulateur LM7805** : Stabilise la tension à **5 V** pour alimenter la Raspberry Pi 4.
3. **Condensateurs de filtrage** : Utilisés pour réduire les ondulations et assurer une tension stable en entrée et en sortie du régulateur.

Le LM7805 est un régulateur de tension linéaire qui fournit une tension de sortie fixe de **5 V**. Il est capable de délivrer un courant maximal de **1,5 A**, ce qui est suffisant pour alimenter la Raspberry Pi 4 (qui nécessite jusqu'à **3 A** en charge maximale). Cependant, pour garantir une marge de sécurité, nous utiliserons deux LM7805 en parallèle pour partager le courant.

- **Tension d'entrée** : 12 V.
- **Tension de sortie** : 5 V.
- **Courant de sortie** : Jusqu'à 1,5 A par régulateur (3 A en parallèle).
- **Dissipation thermique** : Le LM7805 doit être monté sur un dissipateur thermique pour évacuer la chaleur générée.

Les condensateurs de filtrage sont essentiels pour réduire les ondulations et assurer une tension stable en entrée et en sortie du régulateur.

- **Condensateur d'entrée (C1)** : Un condensateur de **0,33 µF** est placé à l'entrée du LM7805 pour filtrer les hautes fréquences.
- **Condensateur de sortie (C2)** : Un condensateur de **0,1 µF** est placé à la sortie du LM7805 pour stabiliser la tension de sortie.

Le LM7805 dissipe une puissance importante en raison de la différence entre la tension d'entrée et la tension de sortie. Un dissipateur thermique est nécessaire pour éviter la surchauffe.

Puissance dissipée :

$$P_{dissipée} = (V_{entrée} - V_{sortie}) * I = (12 - 5) * 1.5 = 10,5W$$

2.5.2. Amplificateur

Pour le prototype, l'objectif est d'envoyer un signal via une GPIO PWM de la Raspberry. Cependant, les GPIO de celle-ci ne fournissent qu'un courant de 2mA à une tension de 3,3V, soit une puissance de 6,6mW, ce qui est totalement insuffisant pour faire fonctionner le haut-parleur 4W. Il est donc nécessaire d'amplifier le signal en puissance. L'amplificateur sera composé de deux étages : un premier étage d'amplification en tension, puis un étage d'amplification en courant.

Amplificateur en tension

L'amplification en tension est réalisée par un AOP, cette solution sera privilégiée à un montage à transistors car elle présente moins de défauts et est moins difficile à implémenter. Dans ce cas, un montage d'amplification à transistor n'aurait pas d'avantage par rapport à un montage à AOP. Celui qui a été choisi pour prototyper est le MCP6091. Il a été choisi pour des raisons pratiques, puisqu'il était disponible dans la salle lors des tests. Cependant, sa tension d'alimentation est limitée à 6V, ce qui n'est pas adapté à ce système qui possède une alimentation 12V. Ainsi, il faudra changer de modèle pour la version finale du système. Dans la suite du développement de la partie amplificateur, on fera les calculs en admettant qu'on utilise un AOP avec une tension d'alimentation de 12V, afin de ne pas les rendre obsolètes dès que l'on changera l'AOP, ce qui devrait arriver très prochainement.

Dimensionnement

Le montage utilisé pour ce circuit sera un montage amplificateur non-inverseur. Il est important que celui-ci soit non-inverseur, car s'il l'était, il faudrait soit que l'étage suivant (amplification en courant) possède une tension d'alimentation négative, ce que le système ne possède pas, soit que l'on réinverse le signal par la suite, ce qui ajouterai une complexité non nécessaire. La fonction de transfert d'un montage amplificateur non-inverseur est :

$$V_{out} = V_{in} \left(1 + \frac{R2}{R1} \right)$$

La tension d'entrée est la sortie de la GPIO de la Raspberry, soit 3,3V et on veut une tension d'entrée la plus élevée possible. Pour éviter les problèmes pouvant être générés avec une tension de sortie trop proche de la tension d'alimentation, on prendra $V_{in} = 3,3V$ et $V_{out} = 11,5V$. Ainsi, on obtient :

$$\frac{V_{out}}{V_{in}} - 1 = \frac{R2}{R1}$$

$$\frac{R2}{R1} = \frac{11,5}{3,3} - 1 \approx 2,5$$

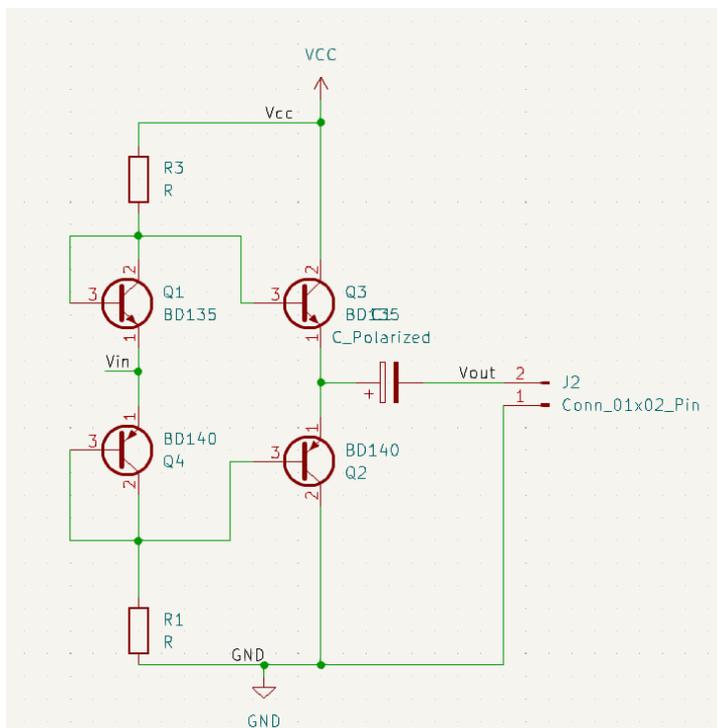
On pourra donc prendre $R2 = 250k\Omega$ et $R1 = 100k\Omega$.

Amplification en courant

L'objectif est d'avoir une puissance de 4W à fournir au haut-parleur, avec une tension de 11,5, on peut calculer le courant nécessaire :

$$I = \frac{4W}{11,5V} = 348mA$$

On arrondira à 300mA, puisqu'il n'est pas nécessaire d'utiliser le haut-parleur au maximum de sa puissance, et le signal sera parfaitement audible avec un tel courant. Etant donné que la plupart des AOP, sauf ceux prévus pour les puissances élevées ne possède qu'un courant de sortie de quelques dizaines de milliampères, il est nécessaire d'ajouter un étage d'amplification en courant. Le montage est un amplificateur push-pull de classe AB dont le schéma se trouve ci-dessous.



Pour cette application, il est préférable d'utiliser des transistors de puissances. Ce qui ont été choisis pour ce projet sont le BD135, transistors NPN, et son homologues PNP, le BD140. Ceux-ci ont été choisis en fonction de leur disponibilité à l'IUT et en fonction de leur spécification.

- Gain suffisant : entre 150 et 250 entre 25°C et 150°C

- Puissance tolérée suffisante : maximum 12W

Dimensionnement

Pour le dimensionnement, on admettra que le courant d'entrée est de 30mA, celui-ci sera réajusté en fonction du courant de sortie de l'AOP choisi dans la version finale. Le courant minimal de polarisation de l'émetteur est de 10mA, il faut alors dimensionner les résistances pour avoir un courant de polarisation de légèrement supérieur à celui-ci, on prendra $I_c = 12\text{mA}$

$$I_c = \frac{V_{cc} - 2V_{be}}{2R} = \frac{12 - 4}{2R}$$
$$R = \frac{8}{2I_c} = \frac{8}{24\text{mA}} = 333\text{k}\Omega$$

En arrondissant à 330k Ω , valeur standardisée la plus proche, on obtient un courant de polarisation de 12,1mA, ce qui ne s'éloigne que de manière non significative de la valeur voulue. On met ensuite en condensateur de 47 μf , une valeur assez élevée pour ramener le signal de sortir autour de 0V, pour ne pas abimer le haut-parleur.

2.6. Conception informatique

Tout en rapport avec la programmation de RPi4.

2.6.1. Détecter l'appuie sur un bouton

Pour détecter l'appuie sur un bouton, on utilisera des capteurs piézoélectriques. Ce type de capteur est particulièrement adapté à ce système puisque l'objectif est de faire un instrument sur lequel on vient frapper les touches, et non simplement appuyer dessus comme on pourrait le faire sur un clavier de piano par exemple. Ensuite, il faut conditionner le signal généré par le capteur piézoélectrique pour l'envoyer sur une GPIO de la Raspberry pour générer une interruption. Pour ce faire, on utilisera un monostable : le NE255.

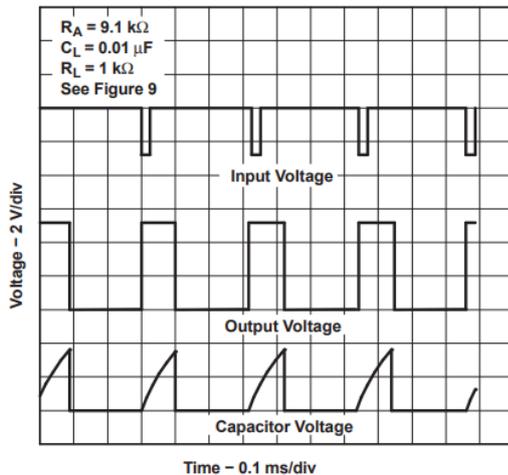
2.6.2. piézoélectriques

Les capteurs choisis seront les Murata 7BB-20-3. Leur dimension de 2cm permet de les intégrer facilement dans le système également, la tension générée lorsqu'ils sont frappés est largement suffisante, puisqu'elle peut monter au delà de 10V.

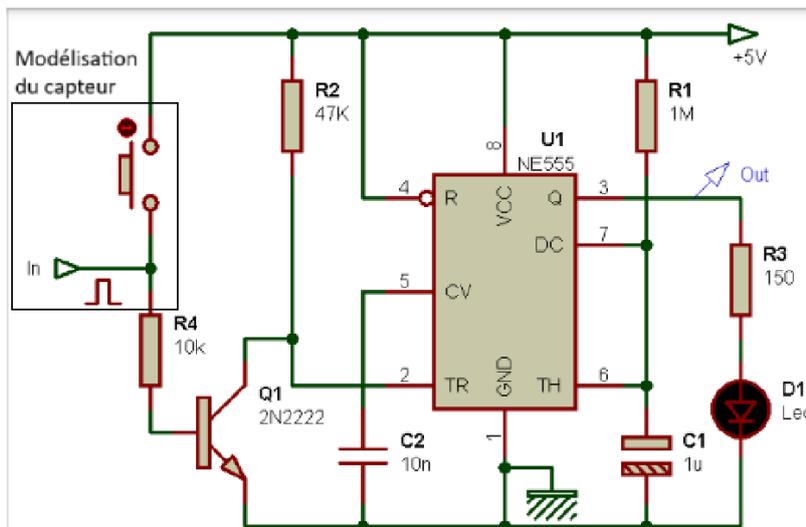
Conditionnement du signal du capteur

Pour que signal puisse être envoyé vers une GPIO de la Raspberry, on utilise un monostable dont la tension d'alimentation peut aller jusqu'à 18V, qui prendra en entrée le signal de sortie du capteur, et qui enverra sur sa sortie un signal stabilisé à 3,3V pendant un temps déterminé. . Celui-ci possède un temps de stabilisation réglable et à une sortie de 3,3V, ce qui est la tension idéale à envoyer sur une

GPIO. La sortie en fonction de l'entrée de ce composant peut se modéliser par le schéma suivant :



On peut remarquer que la sortie se déclenche lorsque l'entrée passe d'un état haut à un état bas. Il faut alors ajouter un transistor monté en inverseur pour avoir l'entrée désiré. On ajoute également une diode zener en parallèle du capteur pour limiter la tension sur la broche de détection du NE255. Le montage est celui-ci, la sortie est directement envoyée sur la broche d'interruption de la Raspberry :



Détection logicielle

Pour détecter une interruption sur une GPIO, on utilisera la librairie wiringPi, elle possède une fonction permettant de configurer une interruption et d'exécuter une fonction de callback quand celle-ci est détectée. Voici un exemple simple de code permettant de faire cela. [CODE](#)

```

1. #include <stdio.h>
2. #include <string.h>
3. #include <errno.h>
4. #include <stdlib.h>
5. #include <wiringPi.h>
6. #include "wavsound.c"
7.
8. #define OUT_PIN 1
9. #define IN_PIN 29
10.
11. static volatile int globalCounter = 0 ;
12.
13. /*
14.  * myInterrupt:
15.  * *****
    
```

```

16. */
17.
18. void myInterrupt (void)
19. {
20.     digitalWrite (OUT_PIN, 1) ;
21.     ++globalCounter ;
22.     digitalWrite (OUT_PIN, 0) ;
23. }
24.
25. /*
26. ****
27. * main
28. ****
29. */
30.
31. int main (void)
32. {
33.     int myCounter    = 0 ;
34.     int lastCounter = 0 ;
35.
36.     if (wiringPiSetup () < 0)
37.     {
38.         fprintf (stderr, "Unable to setup wiringPi: %s\n", strerror (errno)) ;
39.         return 1 ;
40.     }
41.
42.     pinMode (OUT_PIN, PWM_OUTPUT) ;
43.     pinMode (IN_PIN,  INPUT) ;
44.
45.     fprintf(stderr,"mode du pin IN : %d\n",getAlt(IN_PIN));
46.
47.     fprintf(stderr,"mode du pin OUT : %d\n",getAlt(OUT_PIN));
48.
49.     if (wiringPiISR (IN_PIN, INT_EDGE_FALLING, &myInterrupt) < 0)
50.     {
51.         fprintf (stderr, "Unable to setup ISR: %s\n", strerror (errno)) ;
52.         return 1 ;
53.     }
54.
55.     for (;;)
56.     {
57.         printf ("Waiting ... ") ; fflush (stdout) ;
58.         printf ("%d", digitalRead(IN_PIN)) ; fflush (stdout) ;
59.
60.         while (myCounter == globalCounter)
61.             delay (100) ;
62.
63.         printf (" Done. counter: %6d: %6d\n",
64.             globalCounter, myCounter - lastCounter) ;
65.         lastCounter = myCounter ;
66.         myCounter    = globalCounter ;
67.     }
68.
69.     return 0 ;
70. }
    
```

2.6.3. Jouer un son

Pour jouer un son, nous utilisons un fichier .wav sur la RaspberryPi, qui le converti en signal PWM et l'envoi sur une broche GPIO. Un fichier .wav est un format audio numérique qui représente un signal sonore sous forme d'échantillons successifs. Chaque échantillon correspond à une valeur d'amplitude à un instant donné et est généralement encodé sur 16 bits avec un taux d'échantillonnage de 44,1 kHz.

La Raspberry Pi ne disposant pas de convertisseur numérique-analogique (DAC) intégré, elle utilise le PWM pour simuler un signal analogique. Ce signal peut ensuite être filtré et amplifié, ou encore envoyé vers un DAC I2S pour obtenir une restitution sonore de meilleure qualité.

Pour implémenter ce système, nous avons testé deux bibliothèques : WiringPi et pigpio. Cependant, nous avons rencontré des problèmes de compatibilité, car les deux bibliothèques ne fonctionnent pas bien ensemble, et l'une d'elles ne gère pas correctement les interruptions. Finalement, nous avons décidé d'utiliser WiringPi, car elle prend en charge les interruptions, ce qui est essentiel pour la lecture du fichier audio via GPIO.

Partie programmation

- Les bibliothèques utiliser :

Le programme commence par inclure plusieurs bibliothèques :

- `stdio.h` et `stdlib.h` : pour les fonctions de gestion des entrées/sorties et de gestion de la mémoire.
- `string.h` et `errno.h` : pour la gestion des erreurs et des chaînes de caractères.
- `stdint.h` : pour utiliser des types de données spécifiques comme `int16_t`.
- `sndfile.h` : bibliothèque pour lire les fichiers audio en format WAV.
- `wiringPi.h` : bibliothèque pour accéder aux broches GPIO de la Raspberry Pi.
- `unistd.h` : pour la fonction `usleep()` qui permet de gérer les délais en microsecondes.
- Code pour lire un fichier WAV :

On utilise **libsndfile** pour ouvrir et lire un fichier WAV.

```
1. int16_t* read_wav(const char* filename, int* sample_rate, int* num_samples) {
2.     SNDFILE* file;
3.     SF_INFO sfinfo;
4.
5.     file = sf_open(filename, SFM_READ, &sfinfo);
6.     if (!file) {
7.         fprintf(stderr, "Erreur: Impossible d'ouvrir le fichier WAV (%s)\n",
sf_strerror(NULL));
8.         return NULL;
9.     }
10.
11.     if (sfinfo.format != (SF_FORMAT_WAV | SF_FORMAT_PCM_16)) {
12.         fprintf(stderr, "Erreur: Le fichier n'est pas au format PCM 16 bits\n");
13.         sf_close(file);
14.         return NULL;
15.     }
16.
17.     *sample_rate = sfinfo.samplerate;
18.     *num_samples = sfinfo.frames;
19.
20.     int16_t* samples = (int16_t*)malloc(sfinfo.frames * sfinfo.channels * sizeof(int16_t));
21.     if (!samples) {
22.         perror("Erreur d'allocation mémoire");
23.         sf_close(file);
24.         return NULL;
25.     }
26.
27.     sf_read_short(file, samples, sfinfo.frames * sfinfo.channels);
```

```
28.     sf_close(file);
29.
30.     return samples;
31. }
32.
```

Cette fonction utilise la bibliothèque libsndfile pour ouvrir et lire un fichier WAV. Elle vérifie que le format du fichier est bien **PCM 16 bits**, ce qui est requis pour une lecture correcte.

Le fichier est ensuite lu et stocké dans un tableau de type `int16_t` (représentant des échantillons audio sur 16 bits).

La fonction retourne ce tableau et met à jour la fréquence d'échantillonnage (`sample_rate`) et le nombre d'échantillons (`num_samples`).

- Génération du signal PWM sur GPIO :

La Raspberry Pi ne dispose pas de DAC intégré. Cependant, elle peut utiliser le **PWM (Pulse Width Modulation)** pour simuler un signal analogique. Ce signal PWM peut ensuite être utilisé pour alimenter un amplificateur et produire un son.

```
1. void play_audio(int16_t* audio_data, int num_samples, int sample_rate) {
2.     int delay_us = 1000000 / sample_rate; // Calcul du délai entre chaque échantillon
3.
4.     for (int i = 0; i < num_samples; i++) {
5.         int pwm_value = (audio_data[i] + 32768) * 1024 / 65536; // Normalisation
6.         pwmWrite(GPIO_OUT, pwm_value); // Écriture PWM sur GPIO
7.         usleep(delay_us); // Attente entre chaque échantillon
8.     }
9.
10.    pwmWrite(GPIO_OUT, 0); // Remise à zéro après lecture
11. }
```

Cette fonction joue le son en envoyant les échantillons audio sur la broche **GPIO_OUT** sous forme de signal PWM.

Normalisation des échantillons : Le fichier WAV contient des valeurs entre **-32768** et **+32767** (16 bits). Comme le signal PWM nécessite des valeurs entre **0** et **1023**, la normalisation est effectuée avec la formule suivante :

```
1. int pwm_value = (audio_data[i] + 32768) * 1024 / 65536;
```

Cela permet de convertir les valeurs audios en valeurs PWM appropriées. Ensuite, `pwmWrite(GPIO_OUT, pwm_value)` envoie la valeur PWM sur la broche GPIO. La fonction `usleep(delay_us)` crée un délai entre chaque échantillon pour respecter la fréquence d'échantillonnage du fichier audio (en microsecondes).

- Initialisation de WiringPi et configuration des GPIO

```
1. if (wiringPiSetup() < 0) {
2.     fprintf(stderr, "Erreur : Impossible d'initialiser WiringPi (%s)\n", strerror(errno));
3.     return 1;
4. }
```

Cette section initialise la bibliothèque **WiringPi**, qui est utilisée pour gérer les GPIO de la Raspberry Pi. Si l'initialisation échoue, le programme affiche un message d'erreur et se termine.

Ensuite, les broches GPIO sont configurées :

- **GPIO_OUT** est définie comme une sortie PWM (pour l'audio).
- **GPIO_IN** est définie comme une entrée (pour l'interruption du bouton).

- Gestion de l'interruption

```
1. void button_pressed(void) { play_flag = 1; }
```

Cette fonction est appelée lorsqu'un bouton est pressé, grâce à l'interruption sur la broche GPIO_IN. Elle définit la variable `play_flag` à 1, ce qui déclenche la lecture du son dans la boucle principale.

- Configuration de l'interruption sur la broche GPIO_IN

```
1. if (wiringPiISR(GPIO_IN, INT_EDGE_FALLING, &button_pressed) < 0) {  
2.     fprintf(stderr, "Erreur : Impossible de configurer l'ISR (%s)\n", strerror(errno));  
3.     return 1;  
4. }
```

Cette ligne configure une interruption sur la broche **GPIO_IN**. Lorsque le bouton est pressé (un front descendant sur GPIO_IN), la fonction `button_pressed` est appelée, ce qui met le flag `play_flag` à 1 pour commencer la lecture du son.

- Boucle principale et gestion de l'état

```
1. while (1) {  
2.     if (play_flag) { // Si le bouton est pressé  
3.         printf("Lecture du son...\n");  
4.         play_audio(audio_data, num_samples, sample_rate);  
5.         play_flag = 0; // Réinitialisation du flag  
6.         printf("Lecture terminée. En attente...\n");  
7.     }  
8.  
9.     delay(100); // Pause pour éviter de monopoliser le CPU  
10. }
```

La boucle principale attend l'interruption du bouton. Lorsque le bouton est pressé, la variable `play_flag` est mise à 1, et la lecture du fichier WAV commence. Après la lecture, le flag est réinitialisé à 0 pour attendre une nouvelle pression du bouton.

Le `delay(100)` permet de ralentir la boucle principale afin de réduire l'utilisation du CPU.

2.6.4. [Jouer une séquence de sons préenregistrée](#)

2.6.5. [Enregistrer une séquence de sons](#)

2.6.6. [Envoyer les données en externe](#)