

Initiation à la Recherche et aux Projets:

Conception de cartes de communication CAN sur la base d'un microcontrôleur STM32 pour l'émulation de la gestion de charge de la voiture électrique



2024-2025

Table des matières

Introduction	4
A. Contexte du projet	4
B. Objectifs	4
Organisation du projet avec planning prévisionnel de Gantt	5
Etat de l'art	6
A. Solutions actuelles de recharge	6
A.1. Recharge en courant alternatif (AC)	6
A.2. Recharge en courant continu (DC)	6
A.3. Les différents standards de recharge rapide	7
B. Limitations observées	7
Cahier des charges	8
A. Exigences fonctionnelles	8
B. Exigences non-fonctionnelles	8
Conception matérielle	8
A. Choix des composants (STM32L476RG, relais, capteurs...)	8
A.1. Microcontrôleurs	8
A.2. Transceivers CAN	9
A.3. Optocoupleurs	10
A.4. Carte de relais	11
A.5. Composants d'interfaces avec l'utilisateur	12
B. Schémas électriques	12
Simulation du protocole avec deux STM32L476RG	14
A. Diagrammes UML	14
A.1. Diagramme de contexte	14
A.2. Diagrammes de classes	16
A.2.1. Diagrammes de classes pour l'EVSE	16
A.2.2. Diagrammes de classes pour l'EV	17
A.3. Diagrammes de séquences	19
A.3.1. Diagrammes de séquences pour l'EVSE	19
A.3.2. Diagrammes de séquences pour l'EV	20
A.4. Diagrammes de Machine à État Finis (MEF)	21
A.4.1. MEF du véhicule	21
A.4.2. MEF de la borne de recharge	22
B. Trames CAN	24
B.1. Fonctionnement du CAN	25
B.2. Composition des trames CAN	26
C. Résultats de la simulation	27
Développement de l'IHM	28
A. Objectifs de l'IHM	28
B. Langage de programmation choisi	28
C. Fonctionnalités proposées	28

D. Aperçus de l'IHM	29
E. Communication avec la borne	30
E.1. Diagrammes de classes de l'IHM	31
E.2. Diagrammes de séquences IHM-EVSE	33
Design des circuits électroniques	34
A. Arbre des puissances	34
B. Modélisation 3D sous KiCad	35
B.1. Côté de l'EVSE	35
B.2. Côté du véhicule :	37
Tests et validations	38
A. Méthodologie de test	38
B. Résultats	38
C. Éventuelles améliorations à envisager	39
C.1. Simulation du protocole	39
C.2. IHM	39
C.3. Modélisation sous KiCad	39
Bilan du projet	39
Conclusion	39
Annexes	40
Annexe 1 [5]	40
Bibliographie	41

Introduction

A. Contexte du projet

Le développement rapide des véhicules électriques soulève de nombreux défis, notamment celui de la mise en place d'infrastructures de recharge efficaces, sûres et compatibles avec les différents standards en vigueur. Parmi ceux-ci, le protocole CHAdeMO s'est imposé dès les débuts comme une référence en matière de recharge rapide, en particulier grâce à sa fiabilité et à sa capacité à gérer la charge bidirectionnelle.

Dans le cadre de ce projet, nous avons choisi d'explorer ce protocole en nous concentrant sur la communication CAN qu'il impose entre le véhicule et la borne. L'objectif était de simuler une station de recharge conforme à cette norme, en mettant l'accent sur la gestion logicielle et la communication entre dispositifs embarqués.

Ce travail s'inscrit dans la continuité d'un projet déjà initié au LAAS-CNRS, où l'étude portait principalement sur les aspects liés à la puissance et à l'énergie. Notre contribution a consisté à concevoir une carte de contrôle-commande basée sur des microcontrôleurs STM32L4, afin de permettre l'implémentation du protocole CHAdeMO et de gérer le processus de charge, y compris dans sa dimension réversible.

À travers cette démarche, nous avons participé activement à l'avancement du projet global, en apportant une brique essentielle au développement d'une solution complète de recharge intelligente pour véhicules électriques.

B. Objectifs

Ce projet a pour principal objectif de concevoir une carte électronique de contrôle-commande, basée sur un microcontrôleur STM32, destinée à l'implémentation du protocole CHAdeMO. Ce protocole est utilisé pour la gestion de la charge rapide et réversible des véhicules électriques via une communication CAN. À travers ce projet, plusieurs objectifs techniques et pédagogiques sont visés.

Sur le plan technique, il s'agit de :

1. **Concevoir et réaliser deux cartes électroniques** intégrant un microcontrôleur STM32, capables de communiquer via un bus CAN :
 - La première carte jouera le rôle d'une station de recharge (*EVSE*), connectée à un ordinateur.
 - La seconde simulera le BMS (*Battery Management System*) d'un véhicule électrique.
2. **Étudier en profondeur le protocole CHAdeMO**, notamment la structure des trames CAN, les séquences de dialogue entre la borne et le véhicule, ainsi que les

contraintes associées à la charge rapide en courant continu.

3. **Développer le logiciel embarqué** sur STM32 permettant la gestion du protocole, c'est-à-dire la communication et l'échange de données entre les deux cartes.
4. **Créer une interface homme-machine (IHM)** pour permettre le pilotage de la station de charge depuis un ordinateur. L'IHM devra faciliter l'interaction avec les cartes, la visualisation des états du système, et le lancement de scénarios de charge.
5. **Valider expérimentalement l'émulation du protocole CHAdeMO**, en testant les échanges CAN entre les deux cartes et en s'assurant du respect du déroulement prévu par la norme.
6. **Vérifier le bon fonctionnement des circuits électroniques**, en les testant et en les validant dans un environnement contrôlé.

Sur le plan pédagogique, ce projet vise également à mettre en application des compétences acquises en architecture de systèmes embarqués, conception électronique, communication CAN, et développement logiciel embarqué.

Organisation du projet avec planning prévisionnel de Gantt

Afin de pouvoir commencer notre projet, nous avons tout d'abord établi un planning prévisionnel de Gantt. Grâce à cela, nous avons pu nous organiser en conséquence afin de nous répartir les différentes tâches.

		Name	Duration	Start	Finish	Predecessors	Resource Names	2		
								T	F	S
1		Ecriture du rapport	55 days?	10/27/24, 8:00 AM	1/10/25, 5:00 PM					
2		Creation des diagrammes	8 days?	10/27/24, 8:00 AM	11/6/24, 5:00 PM					
3		Prise de rdv janvier	0 days?	1/26/25, 8:00 AM	1/27/25, 5:00 PM					
4		Rdv Novembre	1 day?	11/13/24, 8:00 AM	11/13/24, 5:00 PM					
5		Lecture des documents	8 days?	11/4/24, 8:00 AM	11/13/24, 5:00 PM					
6		Essais et programmation	48 days?	11/19/24, 8:00 AM	1/23/25, 5:00 PM					
7		Preparation des slides	27 days?	2/11/25, 8:00 AM	3/19/25, 5:00 PM					
8		1er Prototype	1 day?	2/4/25, 8:00 AM	2/4/25, 5:00 PM					
9		2nd Prototype	1 day?	2/28/25, 8:00 AM	2/28/25, 5:00 PM					
10		Phase de realisation	58 days?	11/13/24, 8:00 AM	1/31/25, 5:00 PM					
11		Phase de conception	33 days?	2/2/25, 8:00 AM	3/19/25, 5:00 PM					
12		Phase de correction	4 days?	3/19/25, 8:00 AM	3/24/25, 5:00 PM					
13		Rapport de soutenance	92 days?	11/10/24, 8:00 AM	3/18/25, 5:00 PM					

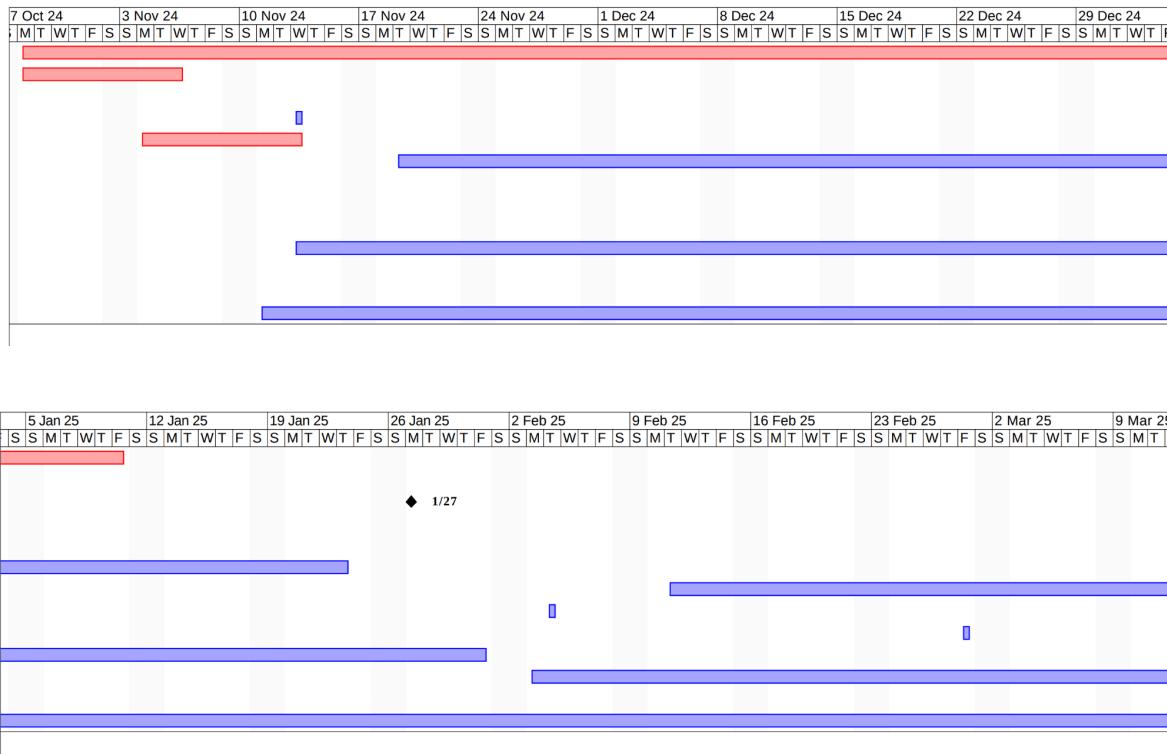


Figure 1. Planning prévisionnel de Gantt

Etat de l'art

A. Solutions actuelles de recharge

Aujourd'hui, plusieurs solutions permettent de recharger un véhicule électrique, chacune ayant ses propres caractéristiques en termes de vitesse, de coût et de compatibilité. Ces solutions reposent principalement sur deux approches : la recharge en courant alternatif (AC) et la recharge en courant continu (DC).

A.1. Recharge en courant alternatif (AC)

La recharge en AC est la plus courante, notamment pour un usage domestique ou dans les parkings publics. Elle repose sur le réseau électrique standard, avec une puissance de charge qui peut aller de quelques kilowatts à une vingtaine de kilowatts selon les installations. Le courant alternatif est converti en courant continu à l'intérieur du véhicule, grâce à un chargeur embarqué.

Ce type de recharge présente l'avantage d'être simple à mettre en œuvre, mais les temps de charge sont généralement assez longs. En pratique, une charge complète peut durer plusieurs heures.

A.2. Recharge en courant continu (DC)

La recharge en DC, quant à elle, est principalement utilisée pour les bornes rapides ou très rapides. Elle permet d'envoyer directement le courant continu dans la batterie, en

contournant le chargeur du véhicule. Cette méthode rend la recharge beaucoup plus rapide et limite les pertes dues à la conversion d'énergie.

Grâce à cette technologie, il est possible de recharger une grande partie de la batterie en moins d'une demi-heure, ce qui est particulièrement adapté aux longs trajets ou à un usage professionnel.

A.3. Les différents standards de recharge rapide

Plusieurs normes coexistent dans le domaine de la recharge rapide, selon les zones géographiques ou les constructeurs. Voici les principales :

- **CHAdeMO** : Originaire du Japon, ce standard a été l'un des premiers à être largement adopté. Il repose sur une communication via le bus CAN, ce qui le rend particulièrement fiable. Il supporte également la recharge bidirectionnelle, permettant par exemple de réinjecter de l'énergie du véhicule vers le réseau.
- **CCS (Combined Charging System)** : Principalement utilisé en Europe et en Amérique du Nord, le CCS combine la recharge en AC et en DC sur un même connecteur. Il est compatible avec un grand nombre de véhicules récents et permet des puissances de charge très élevées.
- **Tesla Supercharger** : Système propriétaire développé par Tesla pour ses propres véhicules, il offre des vitesses de recharge très rapides. Certains modèles commencent à s'ouvrir à d'autres marques, mais l'écosystème reste majoritairement fermé.

B. Limitations observées

Même s'ils sont largement utilisés pour la recharge rapide des voitures électriques, les protocoles Tesla Supercharger et CCS Combo présentent plusieurs limites :

1. Pas de recharge bidirectionnelle (V2G) :

Ni Tesla ni CCS ne permettent actuellement aux voitures de renvoyer de l'électricité vers le réseau (fonction appelée V2G). Cette fonctionnalité est pourtant essentielle pour aider à équilibrer le réseau électrique et utiliser les voitures comme sources d'énergie temporaires.

2. Compatibilité internationale limitée :

Le CCS est surtout utilisé en Europe et en Amérique du Nord, mais reste peu répandu en Asie. Pour Tesla, le type de connecteur change selon les régions (NACS en Amérique du Nord, CCS en Europe), ce qui complique les déplacements à l'international.

3. Protocole pas totalement ouvert :

Le système de recharge Tesla est basé sur un protocole privé, donc fermé aux autres marques pendant longtemps, même s'il commence à s'ouvrir un peu. Le CCS,

de son côté, reste contrôlé par certains groupes industriels, ce qui limite parfois l'accès pour d'autres fabricants ou pays.

Cahier des charges

A. Exigences fonctionnelles

Ici, nous allons voir ce que le système doit faire :

1. La carte EVSE doit pouvoir envoyer et recevoir des messages CAN conformes au protocole CHAdeMO.
2. La carte BMS doit être capable de communiquer avec la carte EVSE et de simuler le comportement d'une batterie de véhicule électrique.
3. Les deux cartes doivent être programmées pour gérer les erreurs de communication.
4. Une interface IHM doit permettre l'interaction utilisateur et la surveillance de l'état de charge.

B. Exigences non-fonctionnelles

Ici, nous allons voir comment que le système doit le faire :

1. Les échanges doivent respecter la temporalité imposée par CHAdeMO (temps de réponse, fréquence d'envoi des trames, etc.).
2. Toute entrée ou trame reçue doit être vérifiée (CRC, type de message, etc.) pour éviter les comportements indésirables.
3. Le code embarqué doit être structuré de manière modulaire (séparation de la couche CAN, logique de protocole, etc.) pour faciliter les mises à jour.

Conception matérielle

A. Choix des composants (STM32L476RG, relais, capteurs...)

Afin de mettre en œuvre la simulation du protocole, nous avons utilisé différents composants, ayant chacun un rôle bien précis dans le système.

A.1. Microcontrôleurs

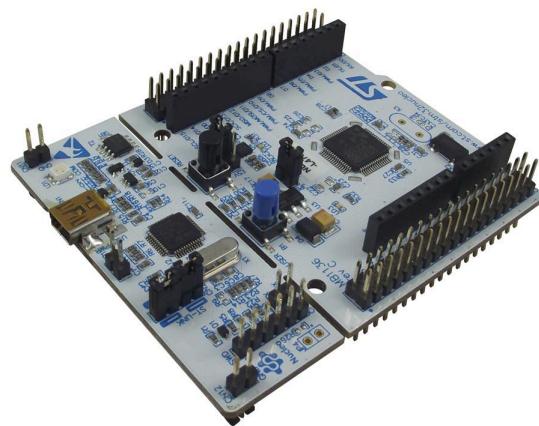


Figure 2 : Carte NUCLEO STM32L476RG

L'élément central du système est une carte NUCLEO STM32L476RG. Cette carte de prototypage de la famille Ultra-Low Power embarque de nombreux périphériques utiles dans ce projet, notamment :

- De nombreux GPIO qui vont servir à interfaçer les différents signaux (commandes, boutons poussoirs, leds)
- un CAN qui sera utile pour la communication d'informations Véhicule ↔ Borne
- Un port série UART pour la communication avec l'IHM sous Matlab
- Un périphérique I2C pour la communication avec l'écran LCD

Ce projet inclut 2 carte NUCLEO STM32 :

- Une qui va simuler la borne de recharge (EVSE)
- Une autre qui va simuler le véhicule à recharger/décharger (EV)

Nous avons utilisé les schémas et documents du site Mbed pour la conception du système [\[1\]](#) :

A.2. Transceivers CAN



Figure 3 : Carte transceivers CAN TJA1051

Même si le STM32 intègre un contrôleur CAN permettant de gérer les trames (identifiants, priorités, format...), il ne peut pas se connecter directement au bus CAN. En effet, il ne génère pas les signaux physiques nécessaires à la communication différentielle utilisée par ce type de réseau. Le transceiver CAN, comme le TJA1051, joue alors un rôle essentiel : il convertit les signaux logiques du microcontrôleur en signaux compatibles avec le bus (CAN_H et CAN_L) et assure la conformité avec les spécifications électriques du standard CAN. Il apporte également une protection contre les perturbations électriques, ce qui est important dans les environnements industriels ou automobiles.

A.3. Optocoupleurs

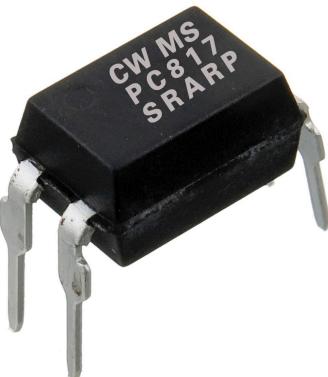


Figure 4 : Optocoupleur PC817

Dans ce projet, nous avons également ajouté des optocoupleurs entre les différentes lignes de commandes (d1, d2 et k). Un optocoupleur est un composant qui permet d'isoler électriquement 2 circuits distincts : celui-ci transmet un signal électrique entre deux circuits isolés galvaniquement en utilisant la lumière : une LED émet de la lumière lorsqu'elle est activée, captée par un phototransistor qui reproduit le signal en sortie (*Figure 5*).

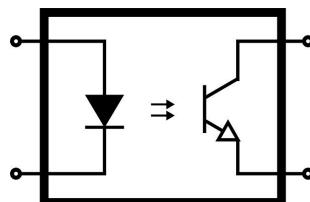


Figure 5 : Schéma électrique d'un optocoupleur

Dans le cadre du système que nous développons, les optocoupleurs permettent d'isoler galvaniquement le véhicule et la borne de recharge. En effet, la masse de la borne étant le neutre du réseau EDF et la masse du véhicule provenant de la batterie, il est nécessaire d'avoir une isolation électrique entre les deux.

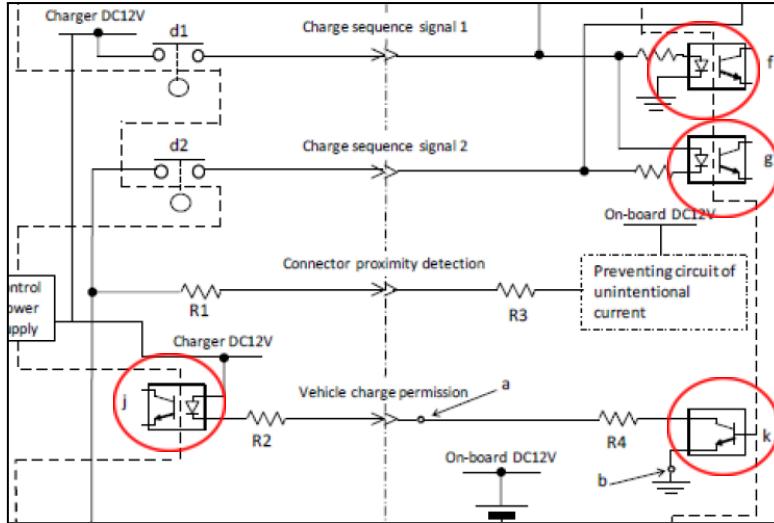


Figure 6 : Entourés en rouge, les différents optocoupleurs implémentés dans le système

A.4. Carte de relais



Figure 7 : Carte de relais 12V

Des relais sont nécessaires pour l'implémentation du protocole CHAdeMo. En effet les lignes analogique d1 et d2 qui permettent une communication entre le véhicule et la borne utilisent un signal +12V. Un microcontrôleur ne pouvant pas délivrer une telle tension, il est nécessaire d'ajouter des relais qui vont pouvoir ouvrir ou fermer une ligne alimentée par un générateur extérieur.

Le système est donc composé de 2 relais qui contrôlent respectivement d1 et d2 (*Figure x*), et un relais qui permet de fermer de 2 relais de charge (*Figure x*).

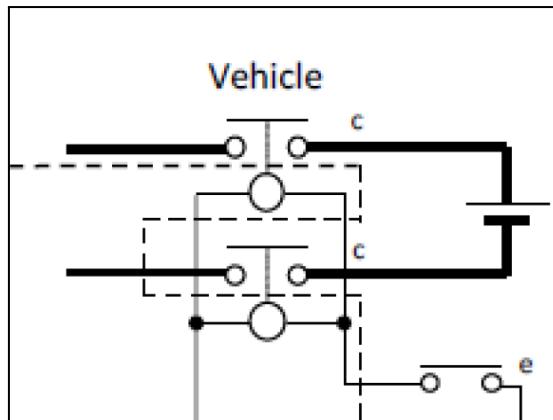


Figure 8 : Le relais “e” contrôle les relais de charge “c” (DC+ et DC-)

N'ayant pas accès à une alimentation +12V pour simuler les lignes d1 et d2, nous avons utilisé une tension de +5V, toujours avec la présence des relais et optocoupleurs. Cette adaptation ne change en rien le fonctionnement du système.

A.5. Composants d'interfaces avec l'utilisateur

Afin d'avoir une communication plus simple avec l'utilisateur, nous avons utilisés :

- En entrées : des boutons poussoirs (déclenchement d'interruption, en pull down)
- En sorties : un écran LCD I2C, des leds (1 verte “Charge” et 1 rouge “ERREUR”)

B. Schémas électriques

Pour réaliser le système, nous nous sommes basés sur le schéma électrique Sequence Circuit du document “*IEEE Standard for Technical Specifications of a DC Quick and Bidirectional Charger for Use with Electric Vehicles*” (Figure x).

La version que nous avons réalisé est légèrement simplifiée :

- La ligne *Connector Proximity Detection* n'est pas implémentée car nous n'avons pas de prise à connecter
- La partie puissance (relais c) n'est pas implémentée car nous ne simulons pas la partie haute tension (tension de charge DC)

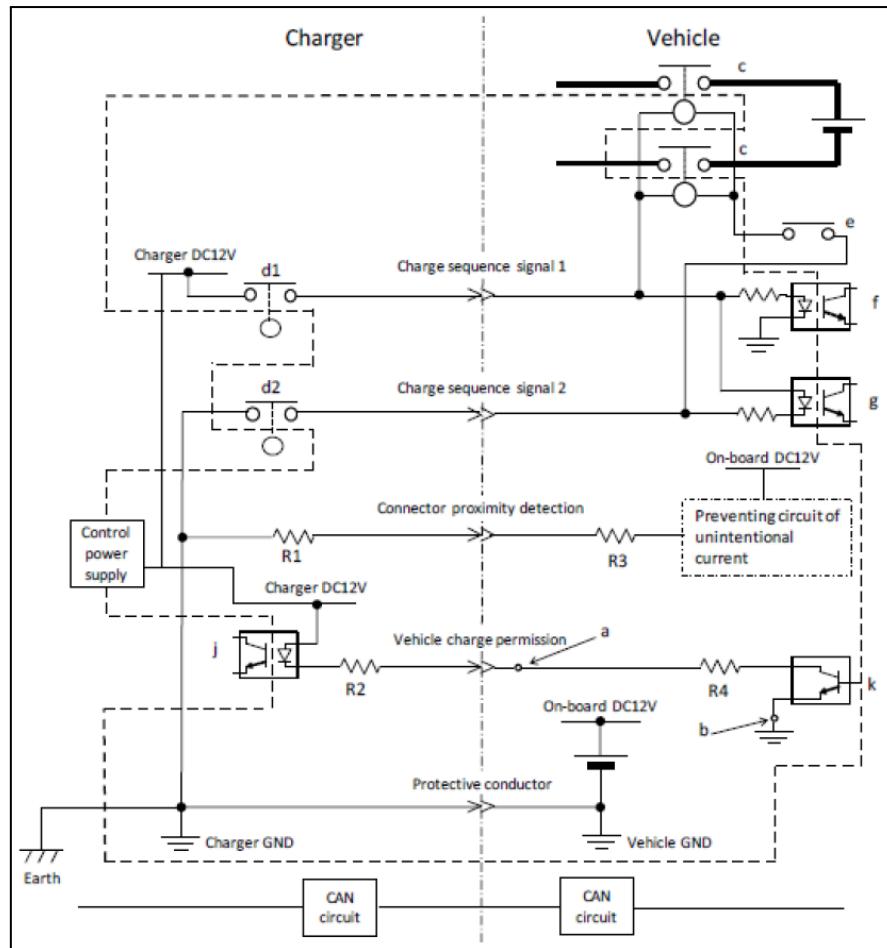


Figure 9 : Sequence Circuit [2]

Voici donc un schémas simplifié du montage réalisé, avec les différents protocoles indiqués, les composants et les lignes de commandes :

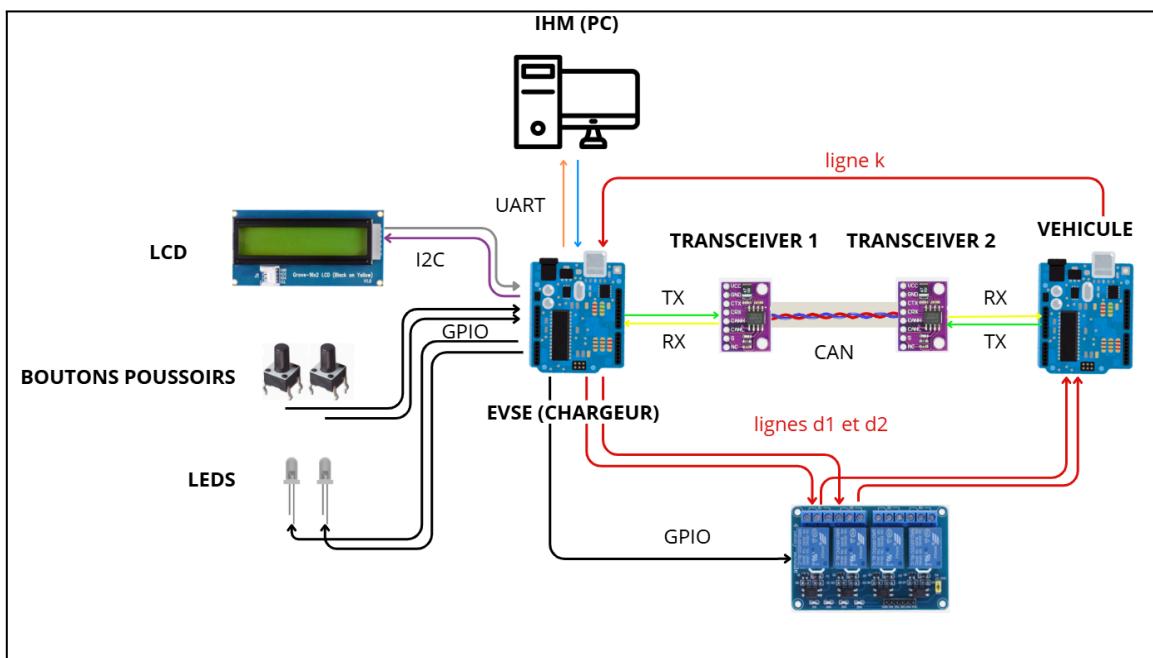


Figure 10 : Schéma électrique simplifié du système

Dans ce schéma, les optocoupleurs ne sont pas intégrés pour plus de compréhension.

Simulation du protocole avec deux STM32L476RG

A. Diagrammes UML

A.1. Diagramme de contexte

Le diagramme de contexte montre les interactions entre les différents éléments du système :

Acteurs :

- Utilisateur (technicien ou développeur).
- Station de charge simulée (EVSE).
- Système de gestion de batterie (BMS).
- Interface IHM connectée à un PC.

Description des Éléments Physiques et des Acteurs

- **Cartes électroniques (EVSE et BMS)** : Microcontrôleurs STM32 configurés pour la communication CAN.
- **IHM** : Logiciel de supervision permettant la visualisation des données et l'envoi de commandes.
- **Utilisateur** : Personne qui interagit avec l'IHM pour tester et surveiller le système.

Description des Use Cases

- **Use Case 1 : Démarrer la communication**
 - L'utilisateur initialise la communication entre l'EVSE et le BMS.
 - Les cartes échangent les messages conformes au protocole CAN.

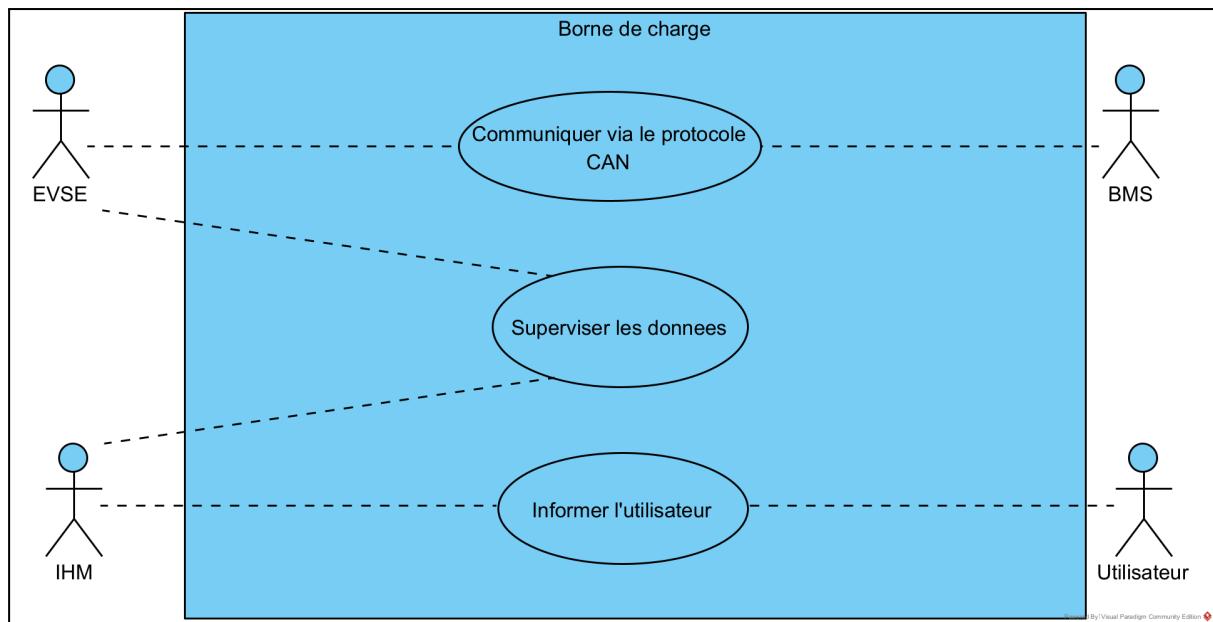


Figure 11 : Diagramme des cas d'utilisation

- **Use Case 2 : Surveillance de l'état de charge**
 - L'IHM affiche en temps réel l'état de la communication et les paramètres de charge.
- **Use Case 3 : Gestion des erreurs de communication**
 - En cas d'erreur, le système envoie une alerte à l'IHM.

A.2. Diagrammes de classes

A.2.1. Diagrammes de classes pour l'EVSE

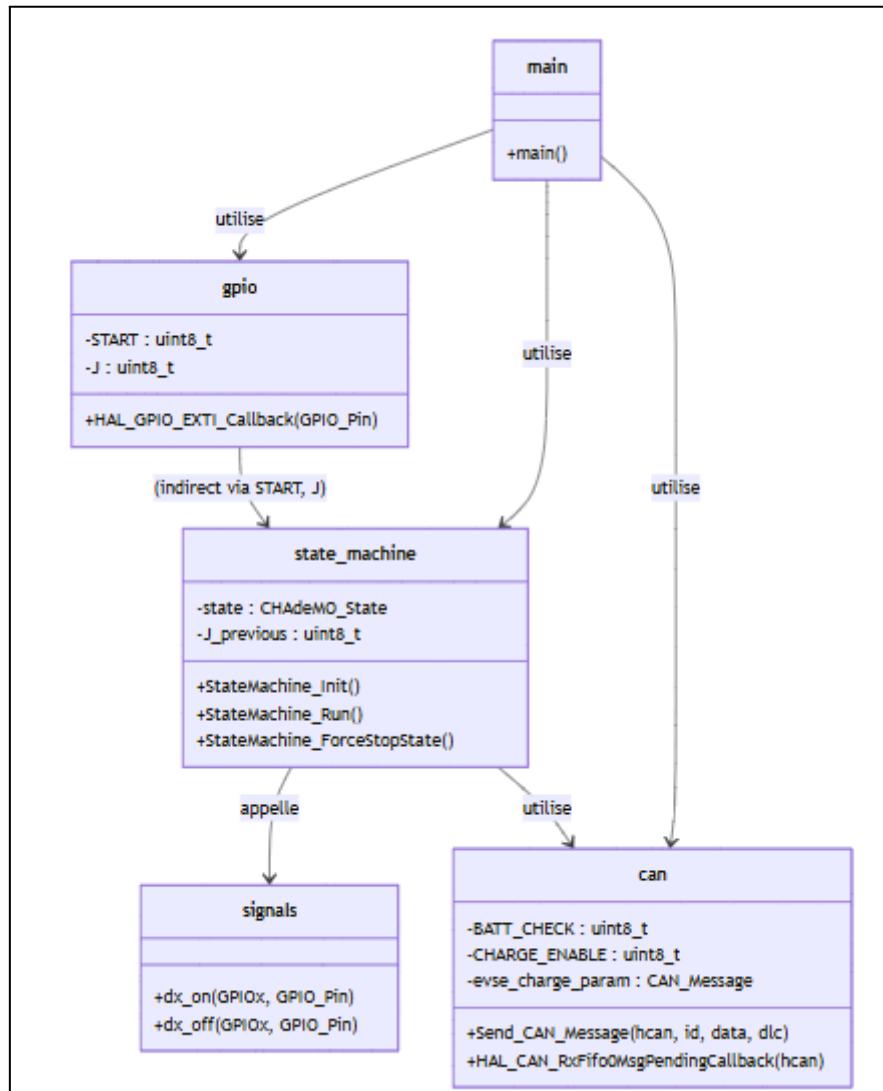


Figure 12 : Diagramme de classe “EVSE”

- **main** initialise l'ensemble du système embarqué, configure les périphériques (CAN, UART, GPIO) et démarre la machine à états via `StateMachine_Init()` puis la fait tourner en boucle grâce à `StateMachine_Run()`.
- Le module **gpio** gère les entrées physiques (boutons ou signaux) via des interruptions. Il met à jour les variables globales **START** et **J** qui sont ensuite utilisées indirectement par la machine à états.
- Le bloc **state_machine** contient la logique principale du protocole CHAdeMO côté station EVSE. Il stocke l'état courant dans la variable **state** (type `CHAdeMO_State`) et réagit aux événements déclenchés par les entrées ou le CAN. Il appelle

directement les fonctions du module **signals** et utilise les fonctions de CAN.

- Le module signals fournit des fonctions de contrôle physique (**dx_on()** et **dx_off()**), permettant d'activer ou désactiver des lignes de signal (par exemple les relais D1 et D2).
- Le module can assure la communication via le bus CAN. Il fournit:
 1. **Send_CAN_Message()** pour envoyer une trame CAN avec identifiant, données et longueur.
 2. **HAL_CAN_RxFifo0MsgPendingCallback()** pour traiter les trames reçues.
 3. Des variables partagées comme **BATT_CHECK** et **CHARGE_ENABLE** qui influencent directement les décisions de la machine à états.
 4. Une trame prédéfinie **evse_charge_param** utilisée lors de l'envoi des consignes de charge.

A.2.2. Diagrammes de classes pour l'EV

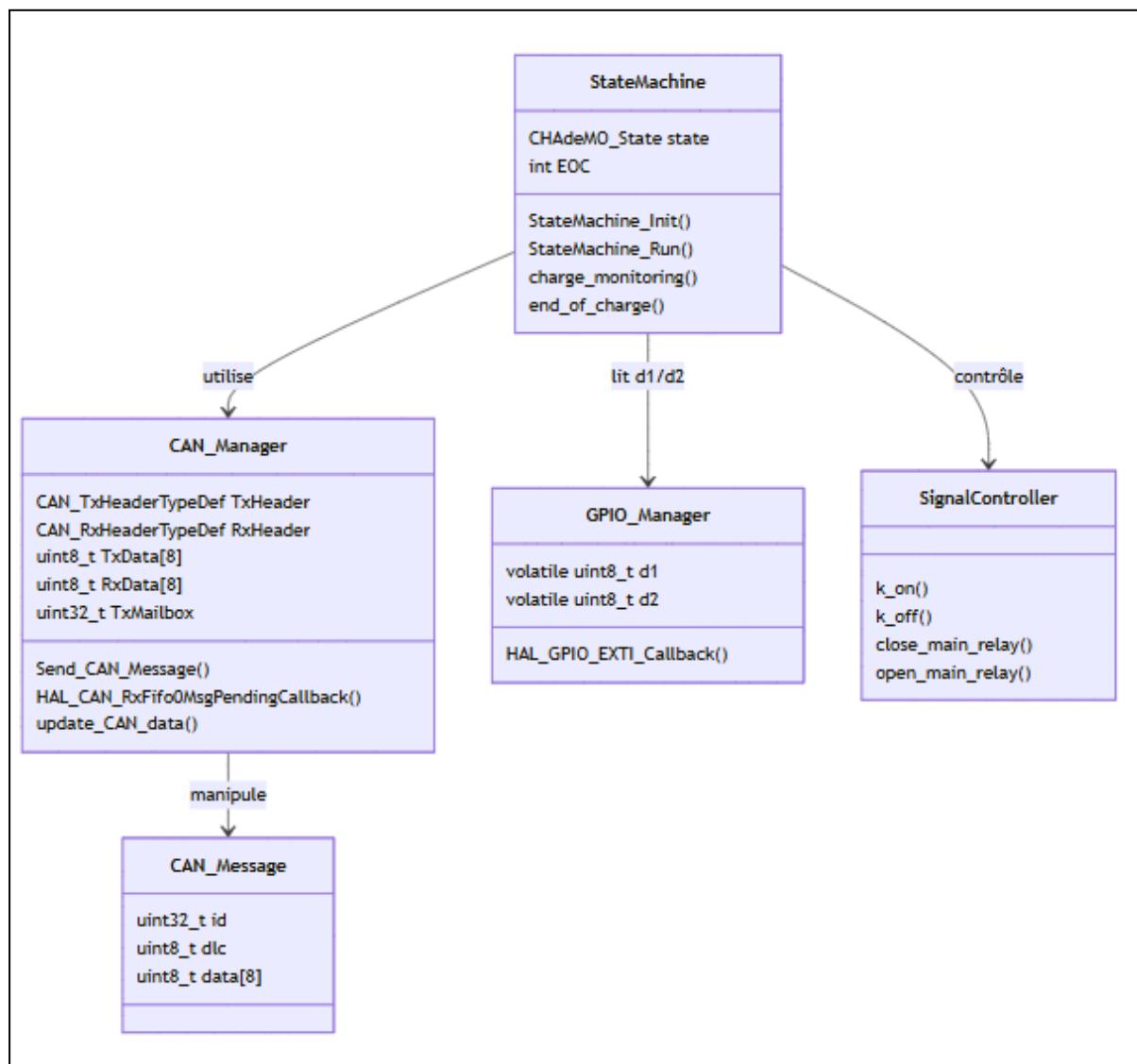


Figure 13 : Diagramme de classes "EV"

- Le bloc **CAN_Message** représente une structure de trame CAN contenant l'identifiant, la longueur (**dlc**) et les données (8 octets maximum). Cette structure est utilisée pour toutes les trames envoyées par le véhicule (batterie, statut, durée de charge).
- La classe **CAN_Manager** regroupe les fonctions de gestion du bus CAN. Elle permet d'envoyer des trames avec **Send_CAN_Message**, de traiter les interruptions de réception avec **HAL_CAN_RxFifo0MsgPendingCallback** et de modifier dynamiquement des octets de trames grâce à **update_CAN_data**.
- **StateMachine** est le cœur logique du véhicule. Elle maintient une variable d'état **state** (**IDLE**, **CHECK_EVSE**, **CHARGING**, **TERMINATING**) et pilote la charge avec les fonctions **StateMachine_Run**, **charge_monitoring** et **end_of_charge**.
- **SignalController** contient les fonctions de commande des signaux physiques (sorties GPIO) : activation du signal K (**k_on**, **k_off**) et contrôle du relais principal (**close_main_relay**, **open_main_relay**).
- La classe **GPIO_Manager** gère les entrées numériques D1 et D2 utilisées comme déclencheurs pour initier la charge ou passer à la phase suivante. Elle capte les événements via **HAL_GPIO_EXTI_Callback**.
- **StateMachine** utilise les services de **CAN_Manager**, **SignalController** et **GPIO_Manager** pour évoluer d'un état à l'autre et envoyer des trames CAN ou contrôler les broches.

A.3. Diagrammes de séquences

A.3.1. Diagrammes de séquences pour l'EVSE

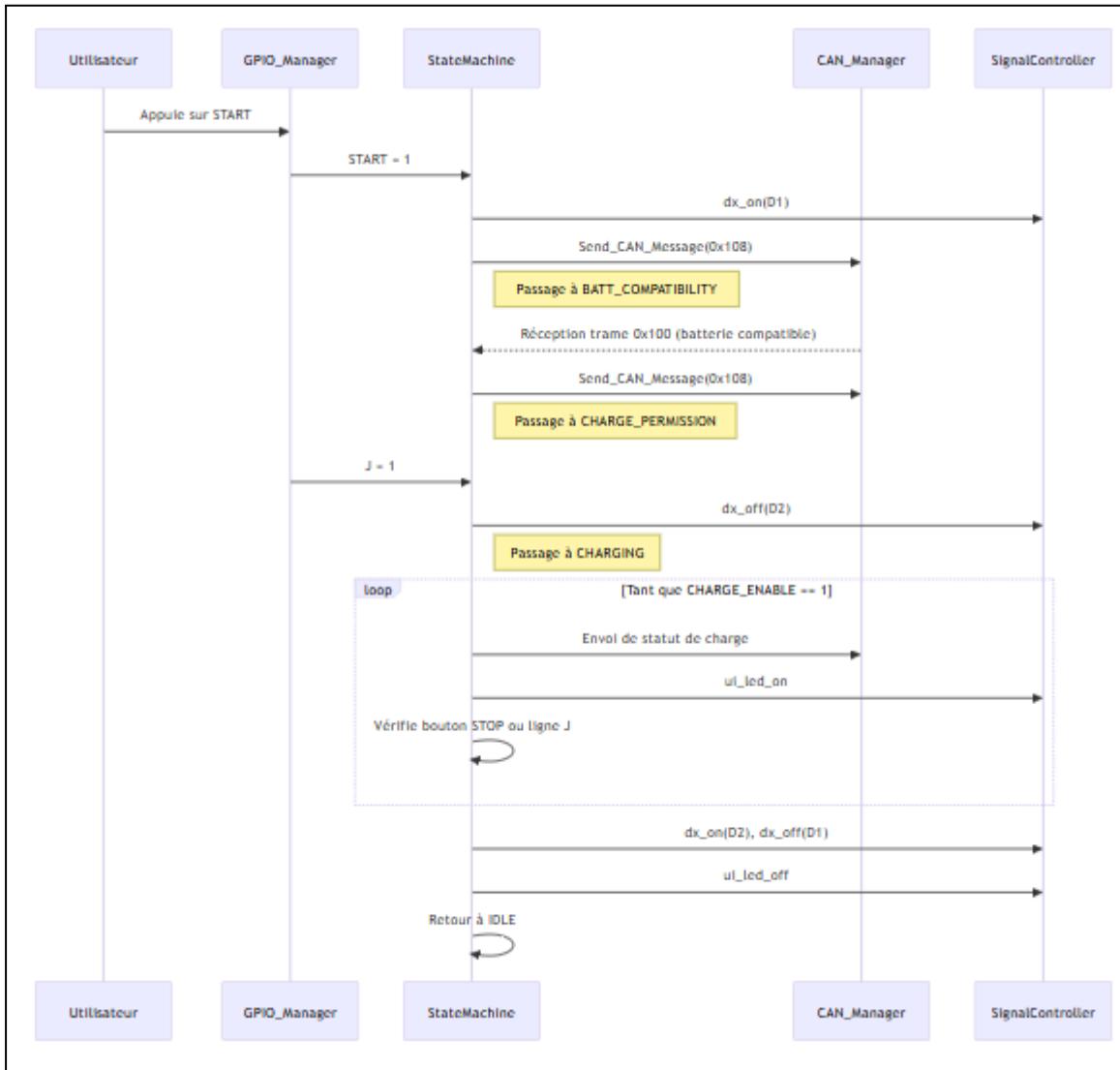


Figure 14 : Diagramme de séquences “EVSE”

L'utilisateur appuie sur le bouton START, déclenchant une interruption captée par **GPIO_Manager**, qui active le flag **START**.

L'état **IDLE** de la **StateMachine** détecte ce flag et passe à **BATT_COMPATIBILITY**. Elle ferme le relais D1 avec **dx_on** et envoie la trame 0x108 contenant les paramètres de charge via **Send_CAN_Message**.

Si une trame 0x100 est reçue de l'EV et indique une compatibilité batterie, la machine passe à l'état **CHARGE_PERMISSION** et envoie à nouveau les paramètres de charge.

Lorsque la ligne J passe à 1 (signal de charge autorisée depuis l'EV), l'état devient **CHARGING**, et le relais D2 est activé via **dx_off**.

Dans l'état **CHARGING**, une boucle maintient la charge tant que **CHARGE_ENABLE** est à 1 : des trames sont envoyées régulièrement, la LED de charge est allumée, et des vérifications sont faites pour détecter un appui sur le bouton STOP ou une déconnexion de la ligne J.

Une fois la charge terminée (ou arrêtée), les relais sont ouverts (D2 et D1), la LED est éteinte, et la **StateMachine** retourne à l'état **IDLE**.

A.3.2. Diagrammes de séquences pour l'EV

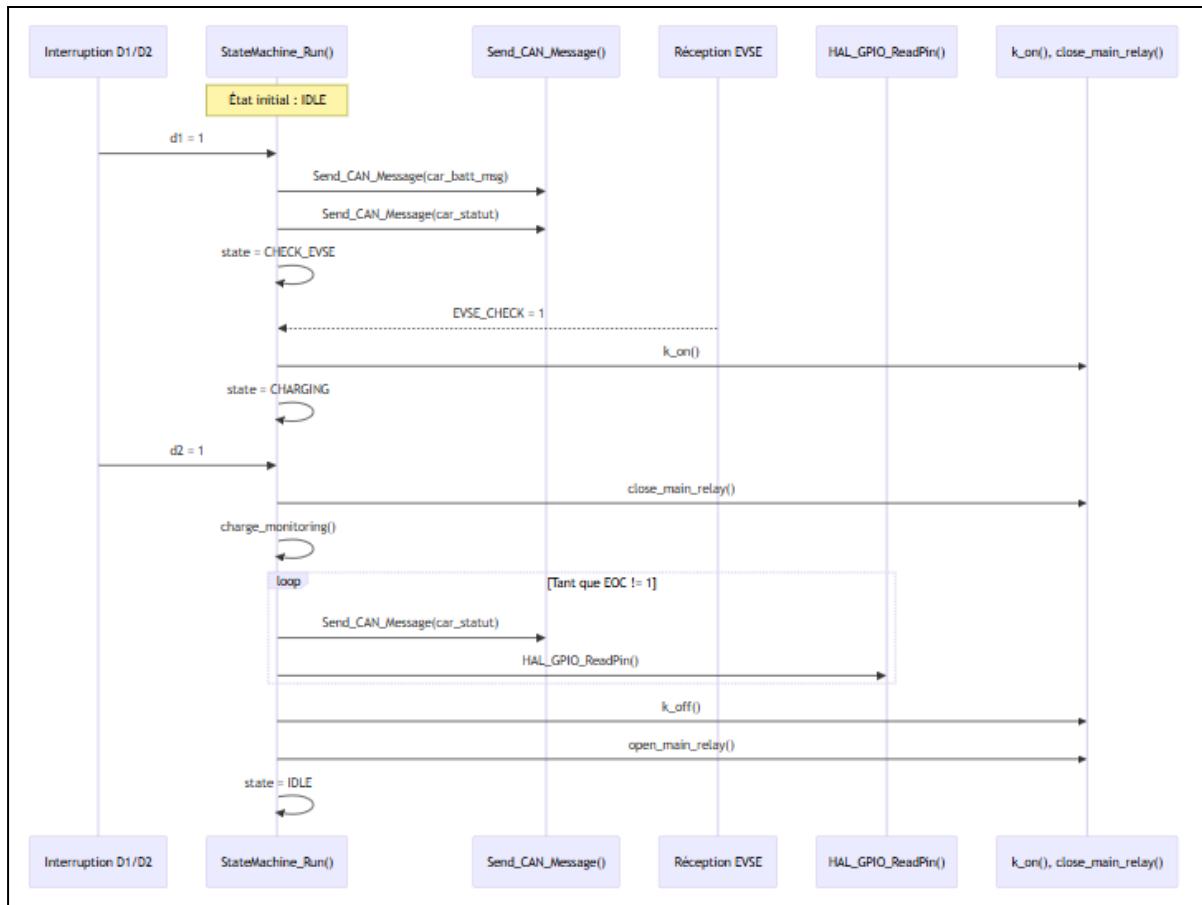


Figure 15 : Diagramme de séquences “EV”

L'exécution débute dans l'état **IDLE**. La machine attend un événement externe déclenché par l'activation de la ligne D1 (activation de la ligne D1 (interruption GPIO)).

Lors de cette activation, la machine passe en mode **CHECK_EVSE** et envoie deux trames au format CAN :

- **car_batt_msg** contenant les paramètres de la batterie ;
- **car_statut** contenant les consignes de charge.

Lorsque l'EVSE confirme la compatibilité de la borne (via la réception de la trame 0x108), le flag **EVSE_CHECK** est activé, et la machine passe à l'état **CHARGING**.

Elle active le signal K via **k_on()** pour autoriser la charge, puis attend un second déclenchement externe via la ligne D2 pour activer le relais de puissance (**close_main_relay**).

La fonction **charge_monitoring** est alors exécutée en boucle :

- Elle demande 125 A à la borne et active le bit de charge via **update_CAN_data**.
- Elle envoie régulièrement la trame **car_statut** pour maintenir la communication avec la borne.
- Elle vérifie la fin de charge en lisant l'entrée GPIO (broche D1).

Une fois la charge terminée, la machine désactive les éléments actifs (**k_off**, **open_main_relay**), puis retourne à l'état **IDLE**.

A.4. Diagrammes de Machine à État Finis (MEF)

Le système est composé de 2 machines à états finis (MEF), qui sont les éléments principaux qui vont gérer la chronologie des échanges entre la borne de recharge et le véhicule. Ces MEF sont implémentés dans la méthode *StateMachine_Run()* de la classe *state_machine.c* de chaque entité (borne et véhicule).

A.4.1. MEF du véhicule

La machine à états du véhicule est responsable de la gestion du processus de recharge en interaction avec la borne via le bus CAN. Elle se compose de quatre états principaux qui s'enchaînent selon des événements logiques ou physiques (reçus via GPIO ou trames CAN) :

La machine à états du véhicule est responsable de la gestion du processus de recharge en interaction avec la borne via le bus CAN. Elle se compose de **quatre états principaux** qui s'enchaînent selon des événements logiques ou physiques (reçus via GPIO ou trames CAN) :

1. IDLE (attente)

Le véhicule est initialement dans un état d'attente. Lorsqu'un signal externe *d1* est activé (par exemple, une action utilisateur ou un branchement du câble), le véhicule passe à l'état suivant **CHECK_EVSE**. À cette transition, il envoie deux trames CAN à la borne :

- Une trame contenant les informations de la batterie (0x100)
 - Une trame indiquant l'état du véhicule (0x102)
- Ces trames initient le processus de demande de charge.

2. CHECK_EVSE (vérification de la borne)

Dans cet état, le véhicule attend une réponse de la borne, signalée par la variable `EVSE_CHECK` mise à 1. Cela indique que la borne est prête à charger. Le véhicule active alors le **signal K (k_on)** pour autoriser la charge et attend une courte période pour stabiliser les circuits, avant de passer à l'état **CHARGING**.

3. CHARGING (charge en cours)

Le véhicule attend un second déclenchement ($d2$) qui représente une confirmation finale (comme un signal du BMS ou une validation utilisateur). Une fois $d2$ détecté :

- Le relais principal est activé (`close_main_relay()`)
- La fonction `charge_monitoring()` est appelée pour superviser la charge
L'état passe ensuite à **TERMINATING**.

4. TERMINATING (fin de charge)

La charge est terminée. Le véhicule :

- Désactive le signal K (`k_off`)
- Coupe le relais principal (`open_main_relay()`)
- Éteint toute indication lumineuse active (ex. LED).

Il retourne alors à l'état **IDLE**, prêt à initier un nouveau cycle de charge.

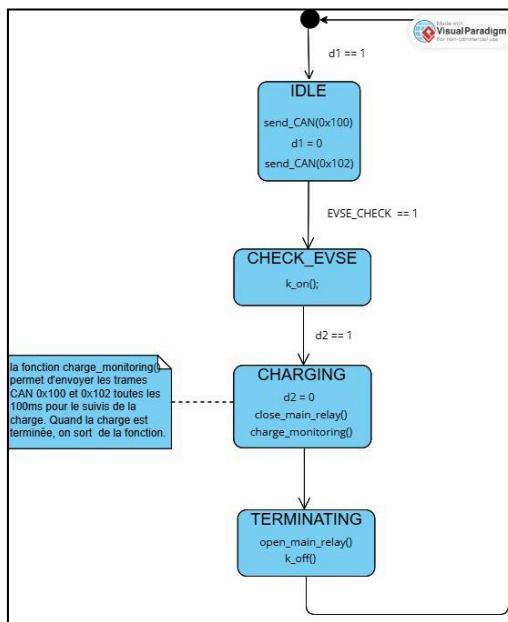


Figure 16 : Diagramme MEF du véhicule

A.4.2. MEF de la borne de recharge

La machine à états de la borne de recharge contrôle l'interaction avec le véhicule et supervise tout le cycle de charge. Elle fonctionne en six états principaux, déclenchés par des signaux utilisateurs, des retours du véhicule ou des conditions internes :

1. IDLE (attente de démarrage)

La borne affiche un message d'invite sur l'écran LCD ("App START").

Lorsque l'utilisateur appuie sur un bouton de démarrage (*START*), elle passe à l'état suivant :

- Ferme le relais D1 (*dx_on(D1)*) pour permettre l'établissement du dialogue initial avec le véhicule.
- Attend une courte stabilisation avant de passer à *BATT_COMPATIBILITY*.

2. BATT_COMPATIBILITY (vérification de compatibilité)

La borne vérifie si la batterie du véhicule est compatible (*BATT_CHECK == 1*) :

- Si oui, elle passe à *CHARGE_PERMISSION*, affiche "batt check ok", et envoie une trame CAN contenant les paramètres de charge (*evse_charge_param*).
- Si non, elle passe directement à l'état *TERMINATING* et affiche "batt incompatible".

3. CHARGE_PERMISSION (autorisation de charge)

La borne attend que la ligne J (signal du véhicule) soit activée (*J == 1*), indiquant que le véhicule autorise la charge.

Elle passe alors à l'état CHARGING :

- Ferme le relais D2 (*dx_off(D2)*) pour activer physiquement la charge.
- Affiche "Ferm. port charge" à l'écran LCD.

4. CHARGING (charge en cours)

C'est la boucle principale de supervision de la charge.

Tant que *CHARGE_ENABLE == 1*, la borne :

- Affiche un pourcentage de charge progressif simulé sur l'écran.
- Allume la LED de charge (*ui_led_on(LED_CHARGE)*).
- Vérifie deux conditions d'arrêt :
 - Si le véhicule se déconnecte (*J* passe de 1 à 0), la borne passe à TERMINATING.
 - Si un arrêt d'urgence est déclenché (*STOP1 == 1* ou bouton STOP appuyé), elle passe à STOP.

5. TERMINATING (fin de charge normale)

Lorsque la charge est terminée, la borne :

- Éteint la LED de charge
- Ouvre les relais D2 et D1 pour couper la puissance
- Affiche "Fin de charge"
Après un court délai, elle retourne à l'état IDLE.

6. STOP (arrêt d'urgence ou erreur)

En cas d'erreur ou d'arrêt forcé :

- La borne affiche "STOP" à l'écran.
- Coupe immédiatement les relais et la LED de charge.
- Fait clignoter la LED d'erreur pendant 5 secondes.
- Reprend ensuite un fonctionnement normal en retournant à IDLE.

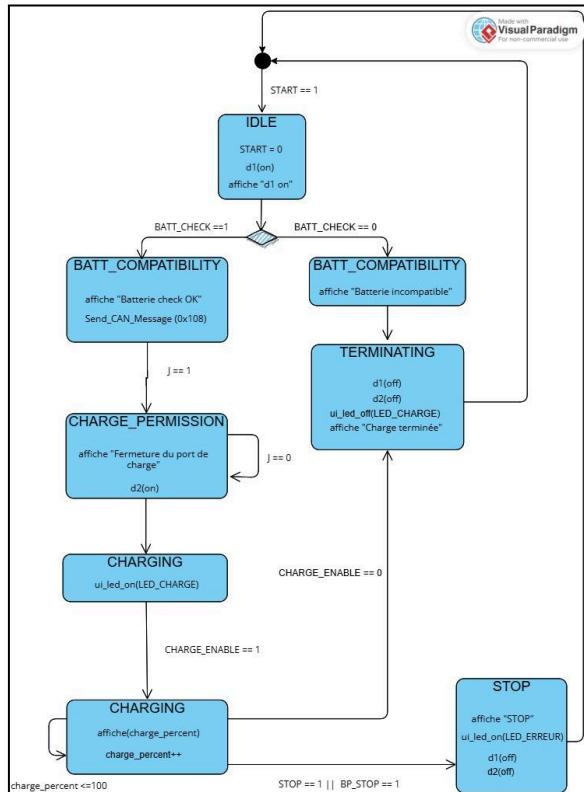


Figure 17 : Diagramme MEF de la borne

B. Trames CAN

Le protocole CHAdeMO étant basé sur le protocole CAN, nous allons voir ici, comment cela fonctionne et quelles sont les données échangées entre le véhicule et la borne.

B.1. Fonctionnement du CAN

Le protocole Controller Area Network (CAN ou CAN Bus) est une méthode de communication par bus série bidirectionnel à deux fils (paire torsadée) qui permet aux sous-systèmes électroniques d'être reliés entre eux et d'interagir dans un réseau.

Le protocole CAN Bus peut être résumé de la manière suivante :

- La couche physique utilise une transmission différentielle sur un fil à paire torsadée
- Un arbitrage bit à bit non destructif est utilisé pour contrôler l'accès au bus
- Les messages sont petits (au plus huit octets de données) et sont protégés par une somme de contrôle
- Il n'y a pas d'adresse explicite dans les messages ; à la place, chaque message porte une valeur numérique qui contrôle sa priorité sur le bus et peut également servir d'identification du contenu du message
- Un système élaboré de gestion des erreurs qui entraîne la retransmission des messages lorsqu'ils ne sont pas correctement reçus
- Il existe des moyens efficaces pour isoler les défauts et supprimer les nœuds défectueux du bus

Le bus CAN dispose d'une capacité multi-maître, ce qui signifie que n'importe quel nœud du bus peut initier une communication avec n'importe quel autre nœud d'un réseau.

Tous les nœuds du réseau CAN doivent fonctionner au même débit binaire nominal, sinon des erreurs se produiront du côté de la réception.

Les spécifications CAN sont des normes internationales. Deux versions sont actuellement utilisées : CAN 2.0A , la version bas débit, parfois appelée CAN de base ou CAN standard, est définie par la norme ISO 11519 ; CAN 2.0B , la version haut débit, également appelée CAN complet ou CAN à trame étendue, est définie par la norme ISO 11898.

Deux nœuds ou plus sont nécessaires sur le réseau CAN Bus pour communiquer. Un message, ou trame, se compose principalement de l'identifiant (ID), qui représente la priorité du message, et d'un maximum de huit octets de données. Un CRC, un ACK (Accusé de Réception) et d'autres surcharges font également partie du message. La trame du message CAN Bus est illustrée à *Figure x*.

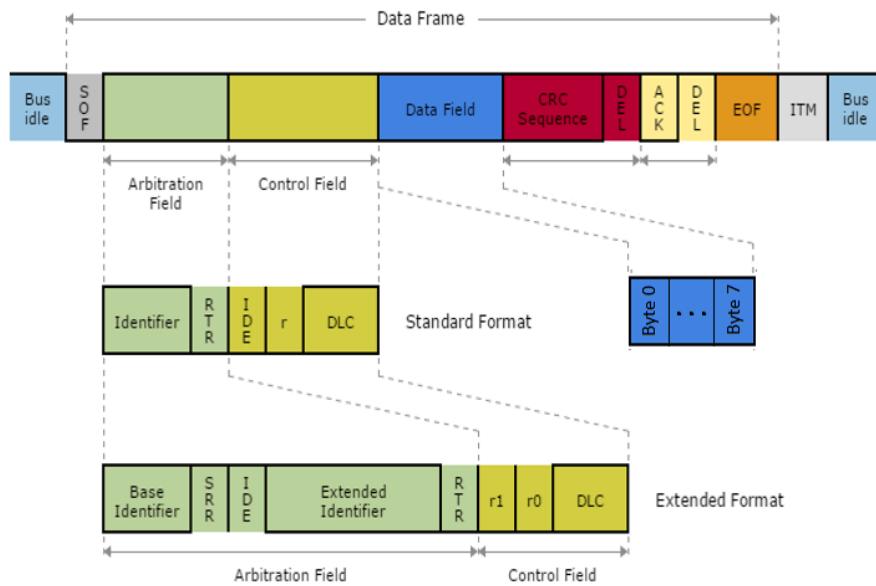


Figure 18 : Trames CAN

Les messages sont étiquetés par un identifiant (ID) attribué à un ou plusieurs nœuds du réseau. Tous les nœuds reçoivent le message et effectuent un filtrage. Autrement dit, chaque nœud exécute un test d'acceptation sur l'identifiant afin de déterminer si le message, et donc son contenu, est pertinent pour ce nœud. Seuls les nœuds concernés par le message le traiteront. Tous les autres l'ignoreront.

L'identifiant a également deux autres fonctions. Il contient des données spécifiant la priorité du message et permet au matériel d'arbitrer le bus. Autrement dit, il permet de déterminer quel nœud peut transmettre si plusieurs nœuds tentent de le faire simultanément. Les identifiants de message doivent être uniques au sein d'un même réseau CAN Bus, sinon deux nœuds poursuivraient la transmission au-delà de la fin du champ d'arbitrage (ID), provoquant une erreur. Plus l'ID numérique est faible, plus la priorité du message est élevée.

Jusqu'à 8 octets de données peuvent être transmis via un seul message de bus CAN. Le champ DLC, ou code de longueur de données, indique le nombre d'octets de données présents dans le champ de données [3].

B.2. Composition des trames CAN

La communication CAN (sans les fonctionnalités V2G) se font principalement via 5 trames différentes. Ces 5 trames ont un ID (identifiant) différent qui permet de distinguer l'information à transmettre. Parmis ces 5 trames, 3 sont réservés à la communication **EV → EVSE** et 2 à la communication **EVSE → EV**.

EV → EVSE :

- *ID 0x100* : Information de la batterie à envoyer au véhicule (courant de charge min, tension de batterie min, tension de batterie max, ...)
- *ID 0x101* : Information sur les temps de recharge (temps max de charge, temps de charge estimé, capacité totale de la batterie)

- *ID 0x102* : Information sur la recharge (version de CHAdeMO, tension cible de batterie, courant de charge demandé, flags d'erreurs)

EVSE → EV :

- *ID 0x108* : Information des capacités de charge de la borne à envoyer au véhicule (tension max disponible pour la charge, courant max disponible pour la charge, tension de seuil)
- *ID 0x109* : Information sur la recharge (version de CHAdeMO, tension de charge délivré à l'instanté, courant de charge délivré à l'instanté, état du chargeur, flags d'erreurs)

Vous pourrez trouver en **Annexe 1** un tableau qui regroupe l'explication de chaque octet des différentes trames vues précédemment.

C. Résultats de la simulation

Après avoir programmé les 2 STM32, on constate que la séquence de charge est bien suivie. L'écran LCD indique les différentes étapes ainsi que la simulation de la charge. On peut également entendre les 3 relais (D1, D2 et main_relay) cliquer, ce qui indique qu'ils se ferment bien.

On peut ensuite observer les différentes trames échangées, avec un Picoscope que l'on configure en déclenchement seul à l'état descendant et en mode décodage CAN, 500kbits/s à l'état LOW :

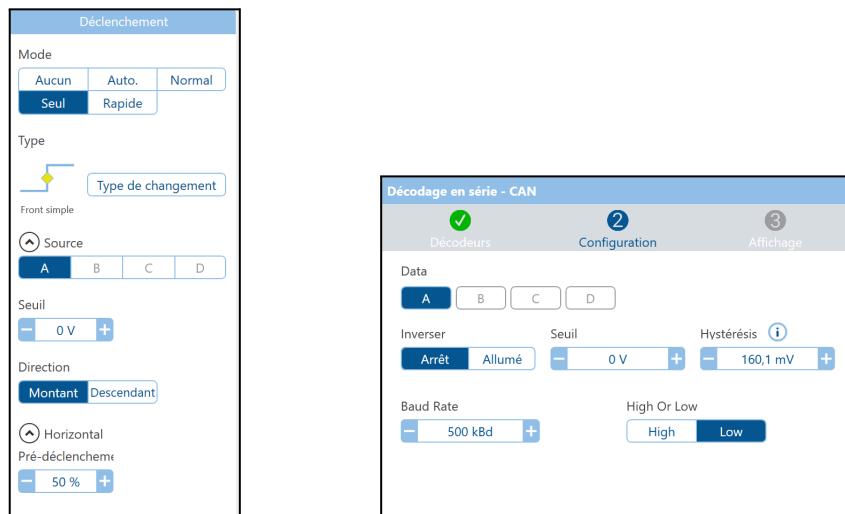


Figure 19 : Configuration du Picoscope

On peut ensuite observer différentes trames CAN avec les différents octets de données configurés dans le code (can.c) de chaque entité du système :

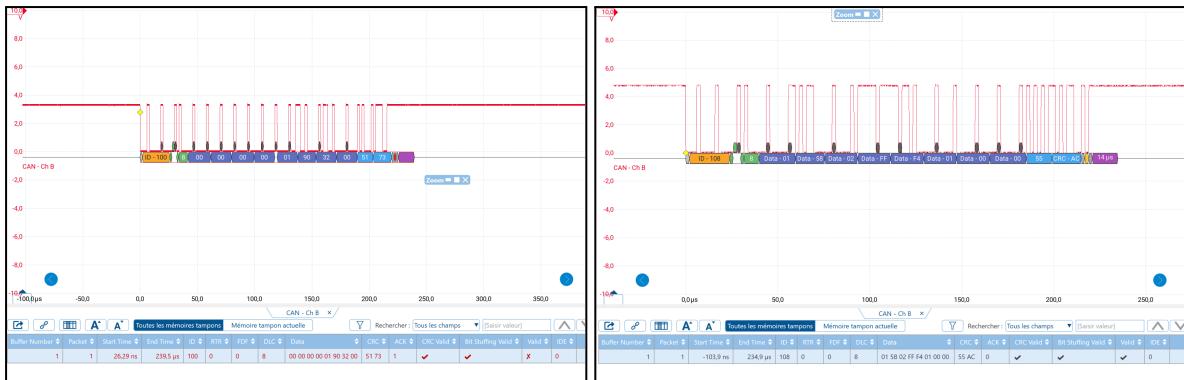


Figure 20 : Trames 0x100 et 0x108

Développement de l'IHM

A. Objectifs de l'IHM

L'interface homme-machine (IHM) a pour rôle de faciliter l'interaction entre l'utilisateur et le système de recharge simulé. Elle permet de visualiser en temps réel l'état du protocole CHAdeMO, de suivre les échanges de trames CAN, de déclencher des séquences de charge, d'afficher les erreurs, et de simuler un scénario de charge typique. Elle joue également un rôle d'un outil pédagogique et technique efficace pour interagir avec le système de charge CHAdeMO, en offrant un aperçu clair du déroulement du protocole.

Elle permet de superviser l'état du protocole, de simuler un scénario complet, et de valider les trames échangées.

Elle constitue également une base extensible pour un futur système plus complet de supervision des communications CAN, avec intégration possible de diagnostics, d'alertes, et d'un suivi multi-session.

B. Langage de programmation choisi

L'IHM a été développée sous **MATLAB**, plus précisément avec **App Designer**, qui offre un environnement de conception graphique adapté à la création rapide d'interfaces interactives. MATLAB a été choisi pour sa capacité à :

- Communiquer en série (UART) avec les cartes STM32 via l'USB,
- Afficher dynamiquement des courbes ou des indicateurs (SOC, tension, courant),
- Prototyper rapidement une IHM visuelle, extensible et portable.

C. Fonctionnalités proposées

L'interface homme-machine développée sous MATLAB vise à faciliter l'interaction entre l'utilisateur et le système simulé de recharge CHAdeMO. Elle est structurée en trois modules complémentaires :

- **CAN_Interface_App_avec_logo** : interface principale jouant le rôle de tableau de bord, permettant d'importer et visualiser des signaux CAN, de lancer des modules de simulation et d'accéder à la supervision.
- **StationCHAdemo_App** : IHM dédiée à la station de recharge avec affichage des paramètres de charge, LED d'état, commandes UART (START / STOP) et indicateurs dynamiques.
- **Simulateur_chademo** : simulateur complet de scénario de recharge, autonome, reproduisant l'évolution du SOC, courant et PWM avec animation graphique et export CSV.

Cette IHM permet de simuler visuellement une session de recharge, de tester le comportement des trames CAN, et d'échanger des commandes avec la carte STM32 via UART.

D. Aperçus de l'IHM

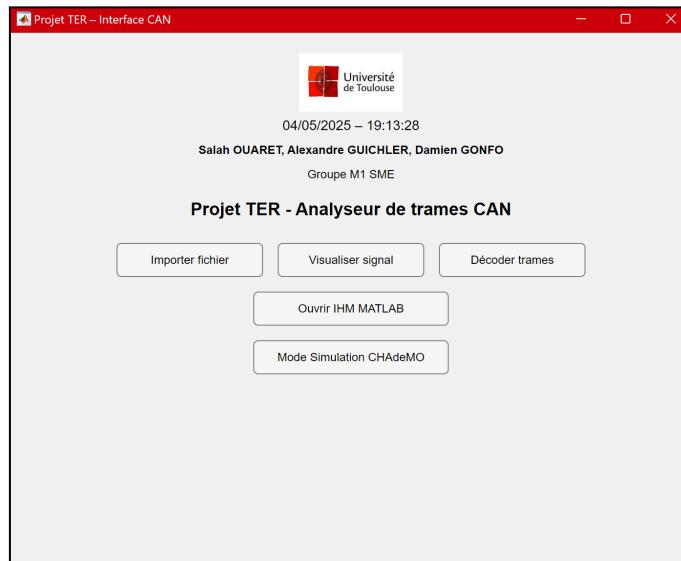


Figure 21 : Vue de l'IHM principale **CAN_Interface_App_avec_logo**. Contient les boutons pour importer un fichier, visualiser un signal CAN, décoder les trames, ouvrir l'IHM station ou lancer le mode simulation.

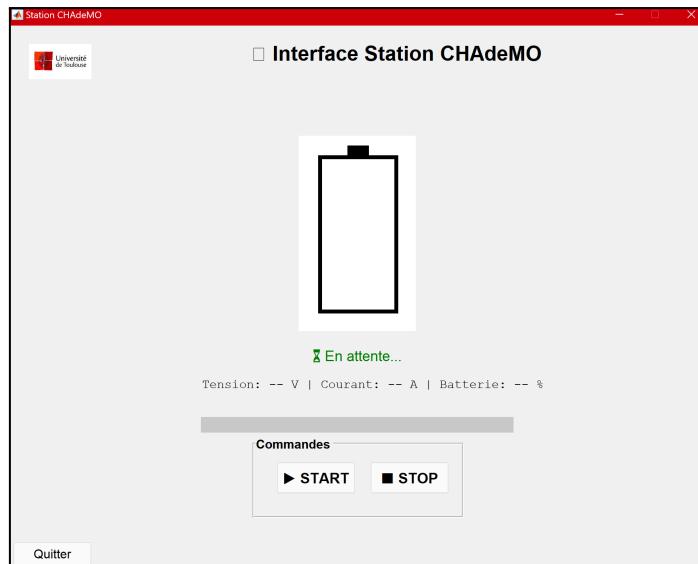


Figure 22 : Vue de **StationCHAdemo_App**. Affiche une image de batterie, une LED d'état, une barre de progression SOC, un message d'état dynamique, et deux boutons (START / STOP) pour la commande via UART.

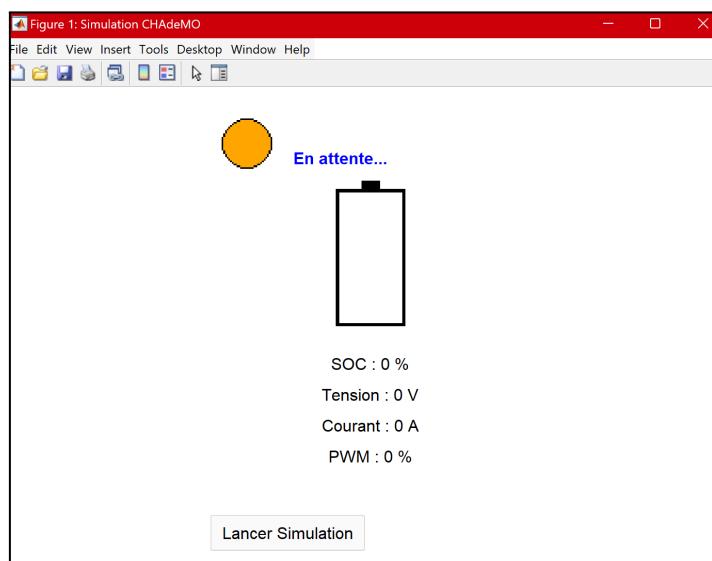


Figure 23 : Vue de **simulateur_chademo**. Affiche une LED, une image de batterie évolutive, des indicateurs SOC/tension/courant/PWM, et génère automatiquement un fichier CSV.

E. Communication avec la borne

Les échanges entre l'IHM et la carte STM32 simulant la borne se font via UART (port COM3) :

1. **La fonction envoyerStartUART()** envoie le caractère "S" pour initier la charge.
2. **La fonction envoyerStopUART()** envoie le caractère "X" pour l'arrêter.
3. **Un timer** lit toutes les 2 secondes les valeurs transmises par la carte (SOC, tension, courant, PWM) via la fonction **lireDonneesUART()** (à intégrer).

Ces échanges permettent à l'IHM de se synchroniser avec l'état réel ou simulé de la borne.

E.1. Diagrammes de classes de l'IHM

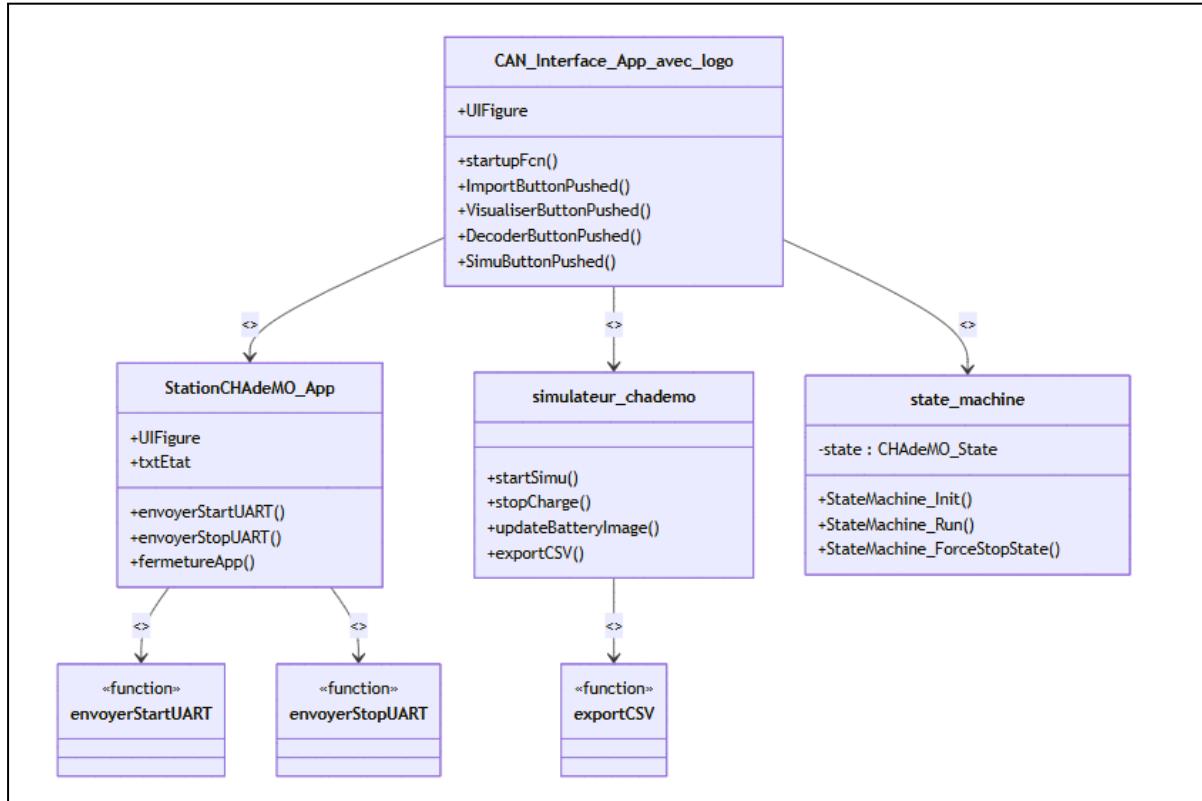


Figure 24 : Diagramme de classes "IHM"

Le diagramme de classes ci-dessus présente l'**architecture logicielle de l'IHM** du projet, structurée autour de **4 modules principaux** :

1. **CAN_Interface_App_avec_logo**.
2. **StationCHAdemo_App**.
3. **simulateur_chademo**
4. **state_machine**.

La classe **CAN_Interface_App_avec_logo** joue le rôle de **tableau de bord principal**, depuis lequel l'utilisateur peut :

- Importer un fichier de trames CAN.
- Visualiser et décoder les signaux.
- Lancer une simulation de recharge.
- Ouvrir l'interface de supervision station.

Le module **StationCHAdemo_App** est une IHM dédiée à la **station de charge**, qui utilise :

- La fonction **envoyerStartUART()** pour envoyer la commande "S" à la carte STM32.
- La fonction **envoyerStopUART()** pour envoyer la commande "X".
- Un **timer** interne pour recevoir et afficher en temps réel les données UART (SOC, tension, courant).

Le module **simulateur_chademo** permet de simuler une session de charge complète sans carte physique, en affichant :

- Des valeurs de SOC, courant, PWM,
- Une image dynamique de batterie,
- Et en exportant les résultats via la fonction **exportCSV()**.

La classe **state_machine** représente la machine à états CHAdeMO implémentée sur la carte STM32, avec 3 fonctions clés :

- **StateMachine_Init()** : réinitialisation.
- **StateMachine_Run()** : exécution de la logique d'état.
- **StateMachine_ForceStopState()** : arrêt d'urgence.

Les fonctions autonomes comme **envoyerStartUART()** ou **exportCSV()** sont représentées en tant que «**function**» externes, car elles ne sont pas rattachées à une classe App Designer mais sont appelées par celles-ci.

Chaque relation du diagramme indique une **dépendance fonctionnelle réelle** observée dans le code : appel direct de fonction, ouverture d'interface, ou déclenchement de simulation.

E.2. Diagrammes de séquences IHM-EVSE

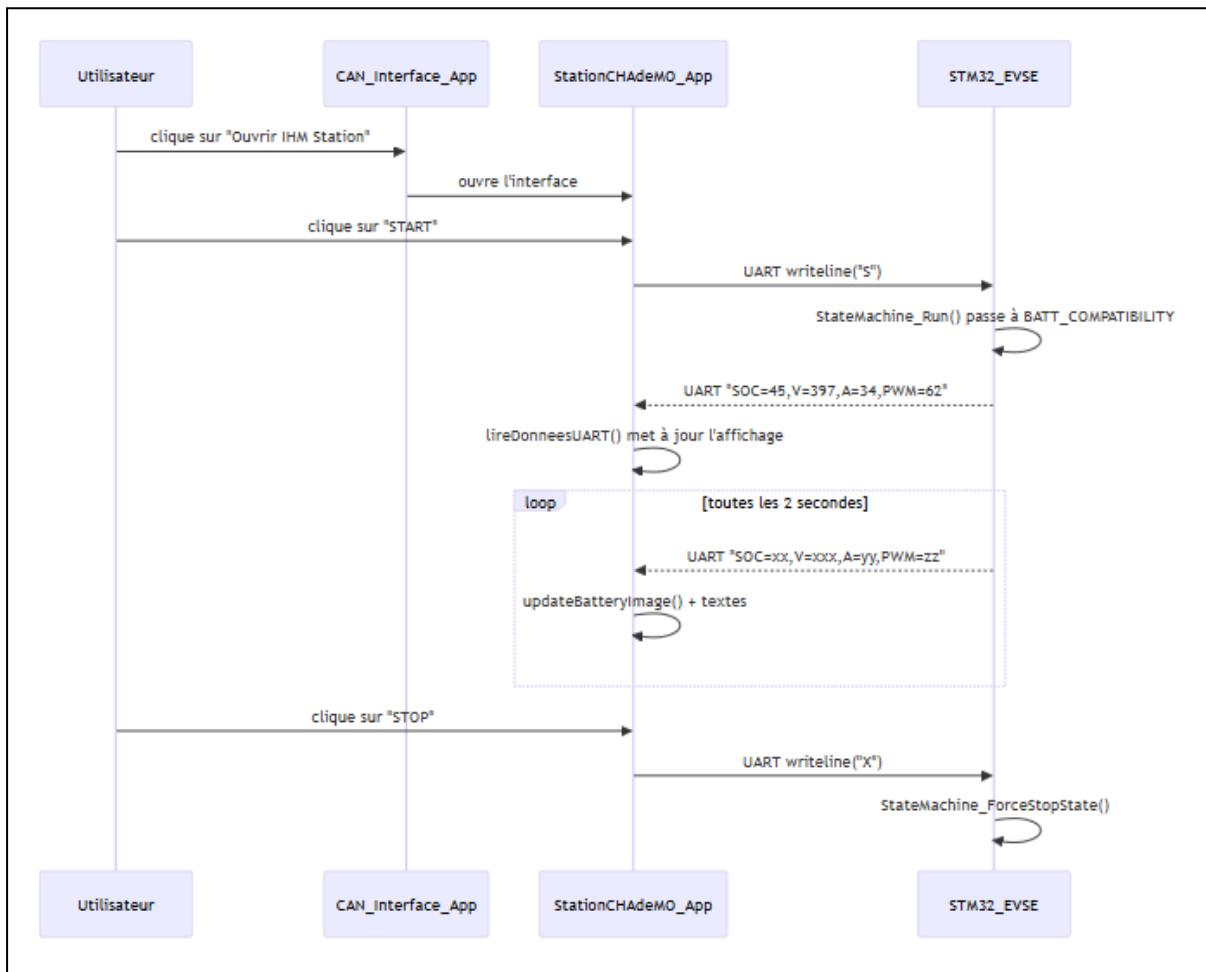


Figure 25 : Diagramme de séquences "IHM"

Le diagramme de séquence présente le **scénario d'interaction** entre les différentes entités du projet : **Utilisateur**, **CAN_Interface_App**, **StationCHAdemo_App** et **STM32_EVSE** (la carte STM32 simulant la borne).

1. Lancement de l'IHM principale :

- L'utilisateur interagit avec l'interface **CAN_Interface_App**.
- En cliquant sur le bouton "**Ouvrir IHM Station**", la fonction MATLAB correspondante ouvre une nouvelle interface secondaire : **StationCHAdemo_App**.

2. Déclenchement de la charge :

- Dans l'interface **StationCHAdemo_App**, un clic sur le bouton "**START**" appelle la fonction **envoyerStartUART()**.
- Cette fonction envoie le caractère "**S**" par le port série UART vers la carte **STM32_EVSE**.

3. Réaction du STM32 :

- Le microcontrôleur reçoit le signal "**S**" et exécute la fonction **StateMachine_Run()**
- La machine à états passe de **IDLE** à **BATT_COMPATIBILITY**, puis à **CHARGE_PERMISSION** si les conditions sont réunies.

4. Transmission des données de charge :

- Pendant l'état **CHARGING**, la carte **STM32_EVSE** envoie régulièrement des trames UART contenant les données simulées : SOC, tension, courant, PWM.
- Par exemple, une trame peut contenir : "**SOC=45, V=397, A=34, PWM=62**".

5. Réception et affichage côté MATLAB :

- Dans l'application **StationCHAdeMO_App**, la fonction **lireDonneesUART()** lit ces trames toutes les 2 secondes (grâce à un timer).
- Elle met à jour dynamiquement l'affichage : texte, image de batterie, LED d'état, barre de progression.

6. Interruption manuelle de la charge :

- Si l'utilisateur clique sur "**STOP**", la fonction **envoyerStopUART()** envoie "X" à la carte.
- Cela force l'exécution de la fonction **StateMachine_ForceStopState()**, qui fait passer l'état interne à **STOP** et met fin à la session de charge.

Ce diagramme illustre fidèlement le **flux d'exécution temps réel** entre les IHM MATLAB et le système embarqué STM32, mettant en évidence les rôles respectifs des fonctions de commande, de lecture UART et de supervision visuelle.

Design des circuits électroniques

A. Arbre des puissances

La conception d'un arbre de puissance, nous permet de visualiser les dissipations et déperditions énergétiques partant de notre borne EVSE à la portion de charge du véhicule.

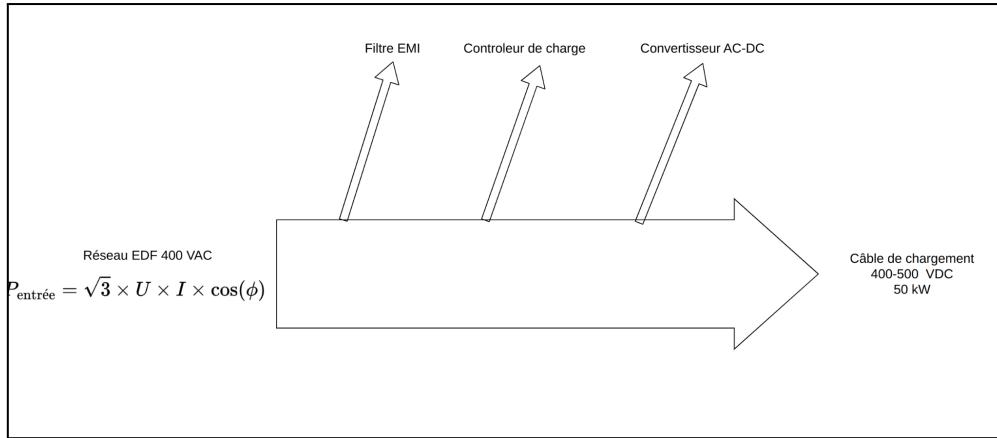


Figure 26 : Arbre des puissances

B. Modélisation 3D sous KiCad

La modélisation sous KiCad, se compose des deux parties de notre système : la partie EVSE et la partie EV. Nous allons donc créer deux projets KiCad et concevoir deux schémas électriques, ainsi que deux PCB. Nous nous sommes inspirés de l'article [\[4\]](#) pour modéliser le système.

B.1. Côté de l'EVSE

Du côté de l'EVSE, nous avons notre microcontrôleur STM32 à notre gauche, responsable des communications CAN. Il est relié aux deux relais pour la charge du véhicule, ainsi qu'à l'optocoupleur. Chacune des sorties amène à notre droite, aux plugs de la borne de recharge CHAdeMO. De plus, la particularité de CHAdeMO résidant dans la charge haute tension, en bas de notre schéma, nous pouvons observer la partie du circuit électrique amenant cette charge au véhicule.

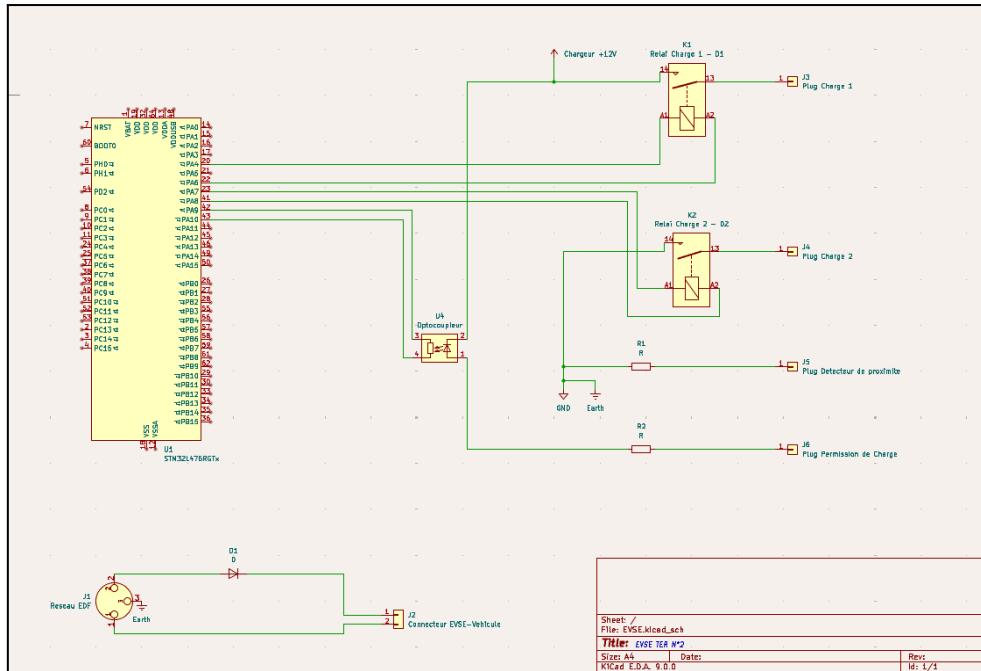


Figure 27 : Schéma électrique de l'EVSE

Grâce au schéma précédent, nous avons maintenant une première itération de notre PCB pour l'EVSE. Elle se compose des Composants Montés en Surface (CMS) au centre, en haut à gauche, nous avons les plugs de communication, au centre les relais et à droite les plugs de charge. De plus, le plug principal de charge est symbolisé par l'emplacement en bas à gauche, l'empreinte adéquate n'étant pas encore disponible sur KiCad.

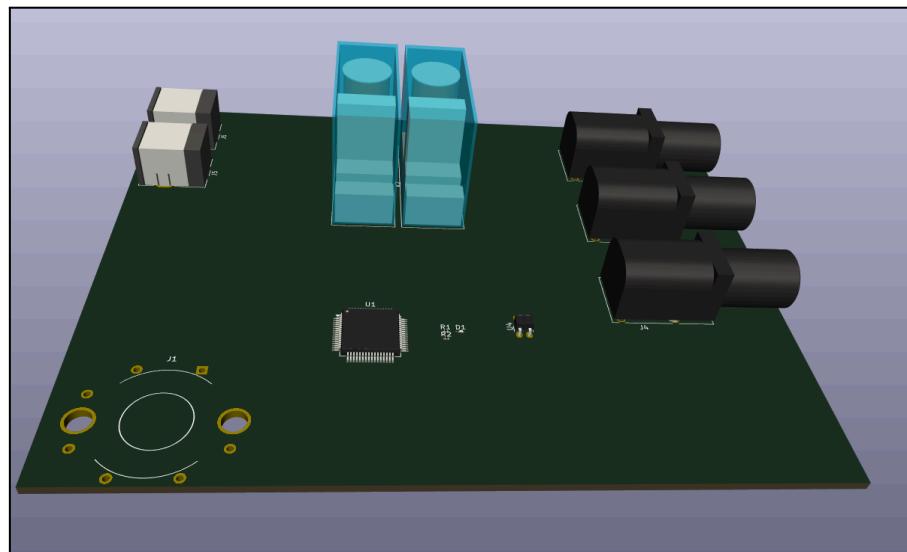


Figure 28 : Vue 3D de la PCB de l'EVSE

B.2. Côté du véhicule :

Pour le circuit électrique de la partie EVSE, les plugs de charge et de communication étaient placés à droite, pour la partie EV, ils sont placés à gauche, dans la continuité du module CHAdeMO. Au centre, sont placés les relais permettant la charge de la voiture, au centre-droit les optocoupleurs et à droite notre STM32, qui agit en tant contrôleur CAN. Cette configuration permet une homogénéité de fonctionnement entre les parties EVSE et EV.

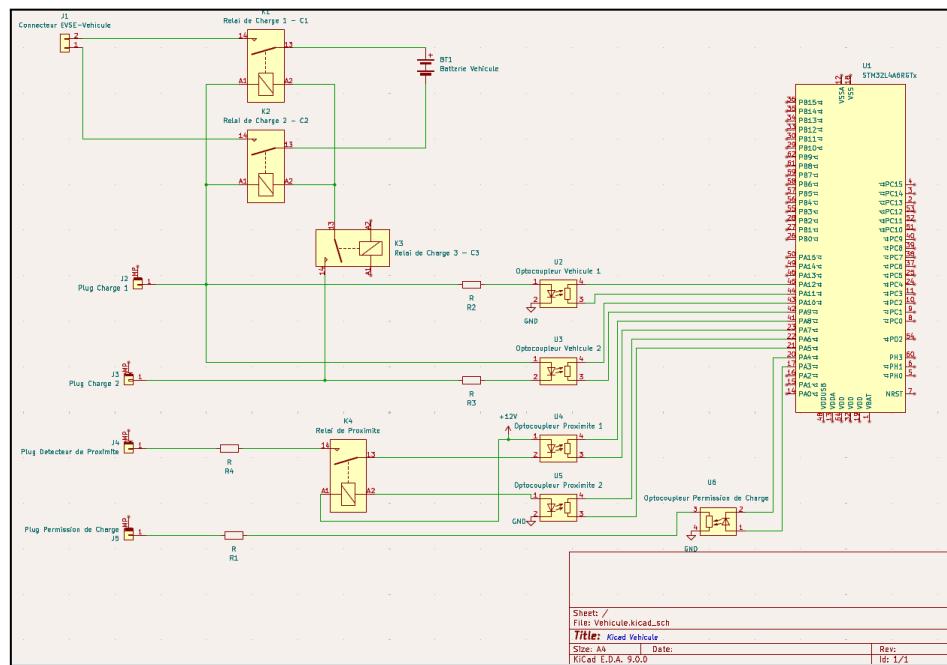


Figure 29 : Schéma électrique de la PCB de charge du véhicule

En gardant les mêmes codes de couleurs et composants pour chacune des fonctions précédemment attribuées, nous avons en haut à droite les plugs de communication, au centre les relais du véhicule, en bas à droite les plugs de charge et au centre les CMS. La batterie du véhicule est symbolisée par l'empreinte de charge de piles AA, le modèle CHAdeMO n'étant pas encore disponible sur KiCad.

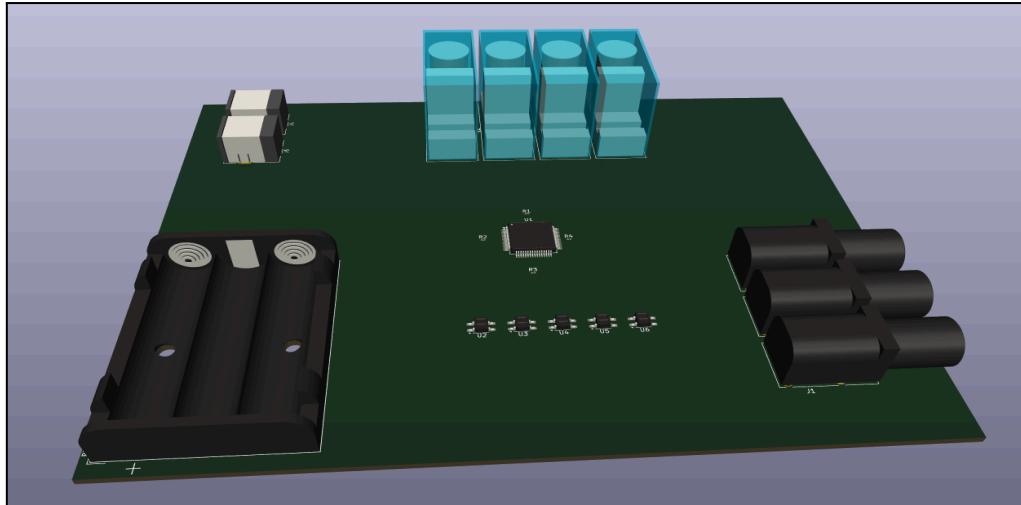


Figure 30 : Vue 3D de la PCB de charge du véhicule

Tests et validations

A. Méthodologie de test

Les tests ont été réalisés selon deux axes :

- **Tests avec STM32** : la carte embarquée envoie des trames CAN réelles décodées côté IHM via UART. On a vérifié la compatibilité du message "S" envoyé depuis l'IHM avec la réception STM32.
- **Tests en mode simulation** : le module `simulateur_chademo` a été testé indépendamment pour valider les enchaînements visuels et l'évolution des variables SOC/courant/PWM sans carte physique.

Chaque bouton de commande a été vérifié individuellement pour s'assurer du comportement attendu.

B. Résultats

- La communication UART s'est montrée fiable et réactive.
- Les données envoyées depuis la carte STM32 ont bien été interprétées et affichées en temps réel dans l'IHM.
- Le simulateur CHAdeMO a produit un fichier CSV contenant les valeurs attendues.
- La simulation s'interrompt bien en cas d'arrêt manuel ou d'erreur simulée.

C. Éventuelles améliorations à envisager

C.1. Simulation du protocole

- Ajouter la partie “Puissance” avec 2 relais de charges et les ligne DC+ et DC-
- Ajouter la mise à jour des données de tension de batterie et de durée estimée de charge sur les trames CAN pendant la phase de recharge
- Ajouter la ligne “Connector Proximity Detection” pour une simulation plus réaliste
- Ajouter une alimentation +12V pour remplacer le +5V des signaux de contrôles (D1, D2, K)

C.2. IHM

- Ajouter une zone de console temps réel pour afficher les trames CAN reçues ligne par ligne.
- Intégrer un journal des événements avec horodatage.
- Améliorer l'aspect graphique avec des animations fluides (progress bar, LED clignotante).
- Offrir une sauvegarde automatique des sessions ou une exportation en PDF.
- Développer une version tactile ou compatible App Designer mobile.

C.3. Modélisation sous KiCad

- Impression 3D des supports EV, EVSE et des câbles et ports de charge.
- Conception des PCB EV et EVSE.
- Concevoir de nouvelles empreintes KiCad pour s'adapter aux PCBs CHAdeMO.

Bilan du projet

La réalisation de l'IHM a permis d'atteindre plusieurs objectifs pédagogiques :

- La maîtrise de la communication série UART.
- La conception d'une interface MATLAB avancée.
- La gestion de données CAN, et interaction en temps réel avec un système embarqué.

L'IHM a contribué à rendre le système intelligible, interactif et exploitable pour les tests du protocole.

Conclusion

Ces travaux d'Initiation à la Recherche et aux Projets nous ont permis d'avancer sur la conception d'une maquette pédagogique pour les futurs étudiants des promotions de Master SME. Ce projet de première année de Master Systèmes et Microsystèmes Embarqués nous a donné l'opportunité de nous plonger dans le monde vibrant de

l'automobile, qui est en perpétuelle extension. Suite aux cours d'informatique industrielle et d'ingénierie projet, ainsi que de la présentation des travaux de l'ingénieur en systèmes embarqués (AUTOSAR), nous avons pu acquérir des connaissances et compétences qui nous seront utiles dès maintenant.

Annexes

Annexe 1 [5]

Table A.47—CAN communication message transmission default value set of vehicle

ID	Byte, bit	Item	Content
100	0	Minimum current	Set “minimum current” defined by vehicle
	1	Not used	Set 0x00
	2	Minimum battery voltage (Low byte)	Lower limit voltage for backup to stop by a charger (Low byte)
	3	Minimum battery voltage (High byte)	Lower limit voltage for backup to stop by a charger (High byte)
	4	Maximum battery voltage (Low byte)	Upper limit voltage for backup to stop by a charger (Low byte)
	5	Maximum battery voltage (High byte)	Upper limit voltage for backup to stop by a charger (High byte)
	6	Charged rate constant value	Set fixed value (0x64: 100 %) related to charged rate
	7	Not used	Set 0x00
101	0	Not used	Set 0x00
	1	Maximum charging time (unit: 10 s)	Set 0x00 (Maximum charging time that vehicle permits charger)
	2	Maximum charging time (unit: 1 min)	Set 0x00 (Maximum charging time that vehicle permits charger)
	3	Estimated charging time (unit: 1 min)	Estimated time until stop of charging (Set “0x00” if vehicle does not require this parameter to be displayed on charger)
	4	Not used	Set 0x00
	5	Total capacity of battery (Low byte)	Set total capacity of battery (Low byte)
	6	Total capacity of battery (High byte)	Set total capacity of battery (High byte)
	7	Not used	Set 0x00
102	0	CHAdeMO protocol number	Set 0x03
	1	Target battery voltage (Low byte)	Target value of charging voltage (Low byte)
	2	Target battery voltage (High byte)	Target value of charging voltage (High byte)
	3	Charging current request	Set 0x00
	4	Fault flag	Set present vehicle state and result of abnormality detection
	5	Status flag	Set a flag such as position of vehicle shift lever, present vehicle state, and charge enabled state
	6	State of charge	Set state of charge of battery (unit: %)
	7	Not used	Set 0x00
NOTE 1—Set “maximum charging time” by the time switch (k) turns on.			
NOTE 2—Set “maximum battery voltage” by the time switch (k) turns on.			
NOTE 3—Set “target battery voltage” after receiving “available output voltage” of a charger.			

Table A.48—CAN communication message transmission default value set of charger

ID	Byte, bit	Item	Content
108	0, 0	Welding detection	Set “1”
	1	Available output voltage (Low byte)	Set low byte of “available output voltage” of charger
	2	Available output voltage (High byte)	Set high byte of “available output voltage” of charger
	3	Available output current	Set “available output current” of charger
	4	Threshold voltage (Low byte)	Set low byte of “threshold voltage” of charger
	5	Threshold voltage (High byte)	Set high byte of “threshold voltage” of charger
	6	Not used	Set 0x00
	7	Not used	Set 0x00
109	0	CHAdeMO protocol number	Set 0x03
	1	Present voltage (Low byte)	Set output voltage measured by charger (Low byte)
	2	Present voltage (High byte)	Set output voltage measured by charger (High byte)
	3	Present charging current	Set output current measured by charger
	4	Not used	Set 0x00
	5	Status/fault flag	Set a state of a charger and a results of fault status
	6	Remaining charging time (Unit: 10 s)	Setting value of charger (initial value)
	7	Remaining charging time (Unit: 1 min)	Setting value of charger (initial value)

Bibliographie

[1] <https://os.mbed.com/platforms/ST-Nucleo-L476RG/>.

[2] IEEE Std 2030.1.1™-2021 (Revision of IEEE Std 2030.1.1-2015) p31

[3]

https://www.typhoon-hil.com/documentation/typhoon-hil-software-manual/References/can_bus_protocol.html

[4]

https://www.researchgate.net/publication/319162700_Rapid_EV_Chargers_Implementation_of_a_Charger

[5] IEEE Std 2030.1.1™-2021 (Revision of IEEE Std 2030.1.1-2015) p91-92