

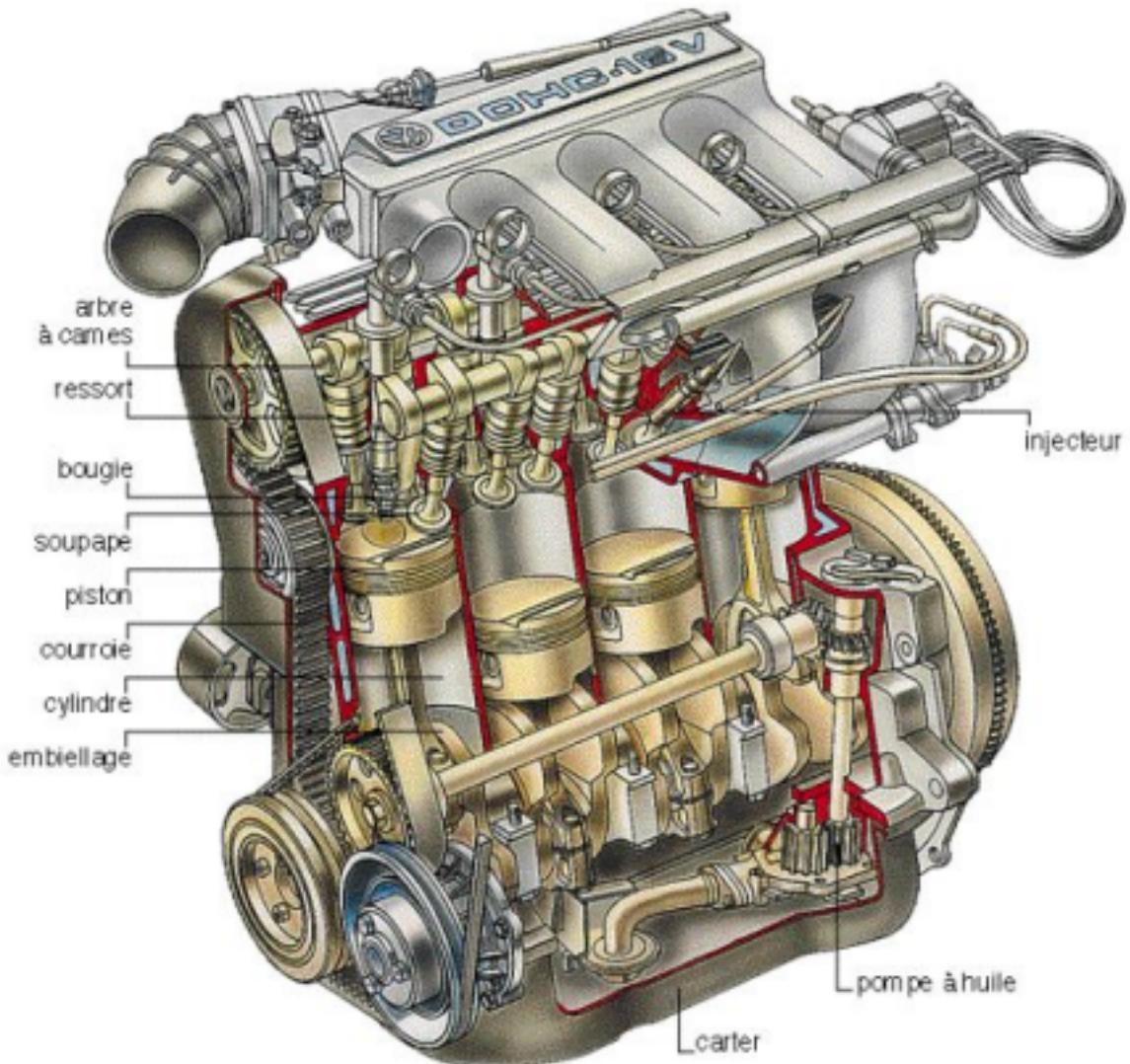
RAPPORT SAE 5

# Interface de mesure pour calculateur automobile

Auteurs : Cyrian LE BRETON & Gaëtan ROY

Référence	Compte-rendu SAE 5
Titre du projet	Étude et réalisation du projet Interface de mesure
Type de projet	Électronique et informatique
Période du projet	18/09/2023 - 1/12/2023
Nature du rapport	Rapport de mi-projet

## RAPPORT SAE 5



### Professeurs référents :

PENNANEACH Gildas et FOURMY Hervé

### Étudiants ayant travaillés sur le projet :

*ROY Gaëtan et LE BRETON Cyrian (2023-2024)*

*BUCHET Nolan et RAISON DU CLEUZIOU Achille (2022-2023)*

*GODEFROY Etienne et JOYEAU Zachary (2021-2022)*

*Julien PERRAUD et Léa BOCQUIER (2019 - 2020)*

*Valentin BRETOOn et Samuel PIPET (2017-2018)*

## RAPPORT SAE 5

**SOMMAIRE :**

Préambule.....	4
<b>INTRODUCTION.....</b>	<b>4</b>
<b>CONTEXTE.....</b>	<b>6</b>
<b>CAHIER DES CHARGES.....</b>	<b>12</b>
Carte d'analyse des temps d'injection.....	12
Carte retard allumage.....	12
<b>ANALYSE FONCTIONNELLE.....</b>	<b>14</b>
Décomposition fonctionnelle.....	14
Conception des blocs.....	14
Analyse du signal.....	15
Gestion de la sortie.....	18
Adaptation de tension.....	19
<b>PARTIE HARDWARE.....</b>	<b>22</b>
Conception de la carte.....	22
Maintenance.....	25
<b>PARTIE SOFTWARE.....</b>	<b>27</b>
BootLoader.....	27
Blink.....	27
DAC.....	28
Fonctionnement général.....	30
Constantes et fonction de base.....	30
Logique de la structure.....	32
Calculs des sorties.....	35
Banc d'essais.....	36
<b>PARTIE MÉCANIQUE.....</b>	<b>39</b>
CAO Carte d'analyse des temps d'injection.....	39
CAO Carte retard allumage.....	40
<b>TEST.....</b>	<b>41</b>
Test Retard Allumage à l'IUT :.....	41
Test Retard Allumage au CIFAM.....	43
<b>GLOSSAIRE.....</b>	<b>45</b>
<b>CONCLUSION.....</b>	<b>46</b>
<b>ANNEXES.....</b>	<b>47</b>

## RAPPORT SAE 5

## Préambule

*Nous avons fait le choix de reprendre dans les grandes largesses les précédents rapport pour ce qui est du contexte, le cahier des charges pour ce qui est attendu de la partie injection, l'analyse fonctionnelle, la partie hardware, ce choix a été fait pour rendre compte des précédents rapports sur le projet dans l'optique où celui-ci arrive à son terme cette année, avec l'idée d'un compte rendu global. Nos apports se situent essentiellement dans la précision du contexte, la partie software, les tests.*

## INTRODUCTION

Ce projet sobrement intitulé Interface de mesure est réalisé dans le cadre du module Situation d'Apprentissage et d'évaluation (SAE) lors des semestres 5 et 6. Différents projets ont été proposés aux étudiants et nous avons fait le choix de celui d'une interface de mesure entre un calculateur automobile et l'humain. C'est une opportunité de mettre en œuvre nos compétences en traitement de signaux, ce qui sort du calculateur, en électronique pour gérer la carte qui va faire l'interface et puis en système embarqué pour programmer son comportement. De plus comme évoqué notre projet est en lien avec le milieu de l'automobile, ce qui nous a semblé être un secteur intéressant pour la formation GEII étant donné son nombre incalculable d'application, de plus c'est un secteur porteur dans lequel la technologie prends plus en plus de place année après année.

Cette année, nous allons donc traiter des signaux qui sortent d'un calculateur d'un véhicule motorisé par quatre cylindres. Pour ce faire, étant donné que l'IUT ne dispose pas de véhicule à disposition pour de tels travaux, nous travaillerons en relation avec le CFA de la chambre de Métiers et de L'artisanat de Loire-Atlantique (CIFAM) qui forme chaque année de nombreux apprentis dans le domaine de la maintenance automobile.

Notre projet a déjà été mené de nombreuses années dans le pôle GEII et est presque arrivé à son terme. Cette année, l'objectif comme de nombreux autres est de conclure et de fournir une documentation complète sur son utilisation pour être utilisée dans le milieu étudiant, ce qui l'est déjà pour partie, et au profit d'étudiants du CIFAM pour observer de manière plus explicites certaines mesures, et dans dans centres de formations du même type.

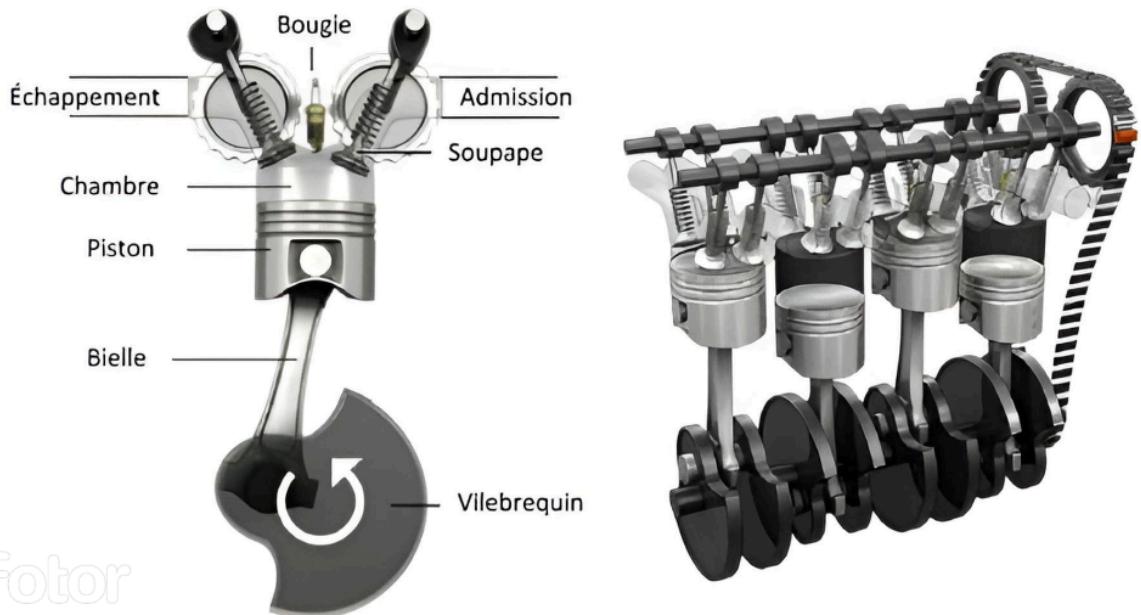
## RAPPORT SAE 5



## RAPPORT SAE 5

### CONTEXTE

Afin de mieux appréhender les exigences inhérentes à notre projet, nous débuterons par élaborer une présentation détaillée du fonctionnement d'un moteur à explosion, en mettant particulièrement l'accent sur les notions cruciales de point mort haut et d'avance à l'allumage. Notre choix s'est porté sur les moteurs les plus répandus dans les véhicules de série, à savoir les moteurs quatre temps à quatre cylindres fonctionnant à l'essence. Pour instaurer une compréhension approfondie du fonctionnement du moteur, nous débuterons par l'utilisation d'un schéma simplifié illustrant ses différentes composantes (voir figures 1 et 2 ci-dessous) :



Dans le contexte du moteur à combustion interne, plusieurs composants essentiels interagissent de manière synchronisée pour assurer le cycle de fonctionnement. Voici une brève description de chaque élément clé :

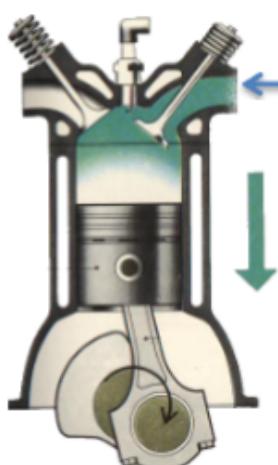
1. La Chambre : C'est l'espace où se déroule le processus d'explosion. C'est à cet endroit que le mélange air-essence est comprimé et enflammé, générant ainsi la puissance nécessaire pour propulser le véhicule.

## RAPPORT SAE 5

2. La Bielle : La bielle constitue le lien mécanique entre le piston et le vilebrequin. Elle permet de convertir le mouvement linéaire du piston en un mouvement rotatif du vilebrequin, assurant ainsi la transmission de l'énergie produite lors de l'explosion.
3. Le Vilebrequin : Le vilebrequin est un composant clé permettant la conversion du mouvement linéaire alternatif du piston en un mouvement rotatif. C'est cette rotation qui sera ensuite transmise aux roues du véhicule.
4. L'Admission : La phase d'admission est cruciale, car elle autorise l'entrée contrôlée d'air et d'essence dans la chambre de combustion. Un mélange précis est nécessaire pour garantir une combustion optimale.
5. La Bougie : La bougie joue un rôle essentiel en générant une étincelle électrique. Cette étincelle enflamme le mélange air-essence, amorçant ainsi le processus d'explosion.
6. L'échappement : Une fois que la combustion a eu lieu, les gaz résultants doivent être évacués. C'est la fonction de la phase d'échappement qui assure le rejet contrôlé des gaz brûlés.
7. Les Soupapes : Les soupapes régulent l'ouverture et la fermeture des passages d'admission et d'échappement. Cette régulation précise garantit le bon timing des phases du cycle moteur.

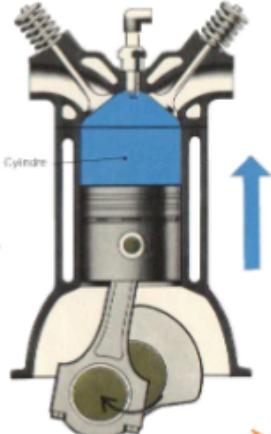
Cette synergie entre tous ces composants permet au moteur à combustion interne de convertir l'énergie chimique du carburant en mouvement mécanique, propulsant ainsi le véhicule.

Parlons donc maintenant comment cette combustion interne se passe exactement dans le moteur à quatre temps. Comme son nom l'indique, son fonctionnement est défini par quatre phases principales. Comme précédemment, nous allons le présenter via des schémas :

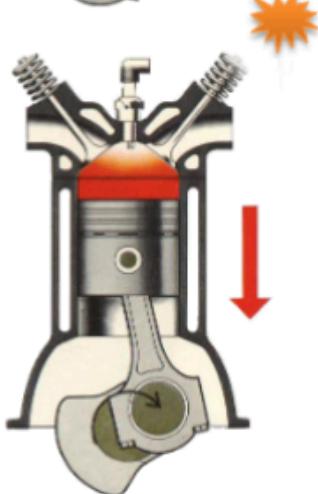


Le mélange air + essence est absorbé lorsque le piston descend. La soupape d'admission est ouverte pour laisser le mélange pénétrer dans le cylindre. C'est l'admission.

## RAPPORT SAE 5



La soupape d'admission se ferme, le piston remonte. L'air et l'essence présents dans le cylindre sont compressés. C'est la phase de **compression**.



Suite à la compression, et à l'étincelle créée par la bougie d'allumage, le mélange air + essence **explose**.  
L'explosion créée fait redescendre le piston. C'est la phase de **détente**. C'est ce temps moteur qui permet la production de l'énergie.



Lors de ce quatrième et dernier temps, le piston remonte. La soupape d'échappement est ouverte et laisse s'échapper le mélange brûlé poussé par le piston. C'est l'**échappement**.

C'est la fin du cycle, un nouveau peut alors recommencer.

## RAPPORT SAE 5

Maintenant, que toutes les parties ont été clairement définies, et que nous connaissons le fonctionnement des différentes phases du moteur, nous allons nous intéresser à l'admission, afin de comprendre comment se réalise le mélange air + essence absorbée pendant cette phase ce qu'on appelle l'injection, le concept de point mort haut et d'étincelle d'allumage. C'est là que l'intérêt du projet entre en jeu.

Voici un schéma présentant l'injection :

L'essence est injectée directement dans le cylindre lors de l'admission, par l'injecteur.

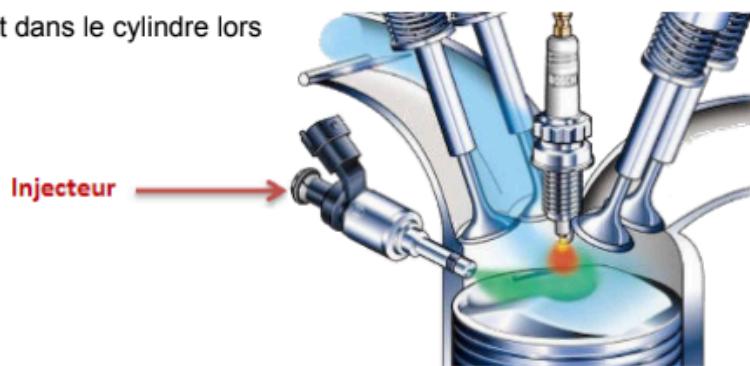


Figure 3 : Injection directe

L'injecteur est un dispositif assurant l'arrivée directe du carburant dans les cylindres d'un moteur. Celui-ci est piloté électroniquement par le calculateur, afin d'injecter la bonne quantité d'essence pour le bon fonctionnement du moteur. La quantité injectée est régulée par le temps d'ouverture de l'injecteur. Si ce temps est plus long, on envoie plus d'essence dans le moteur, et vice versa.

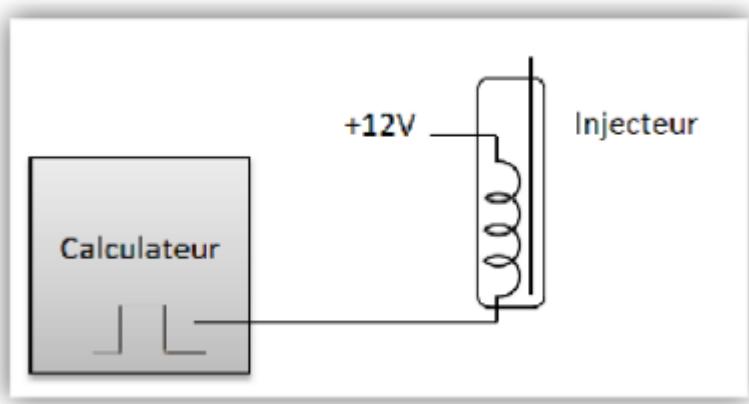
Pour fonctionner correctement, le moteur doit recevoir un mélange air/essence, appelé

$$\frac{Q_{ess}}{Q_{air}} = \frac{1}{15}.$$

dosage, égale à . Ce dosage est réglé par le calculateur, à l'aide d'une sonde à oxygène mesurant la quantité d'air. Pour que ce dosage soit constamment correct, la durée de l'injection varie constamment. Ce sont ces variations que notre projet va permettre d'observer.

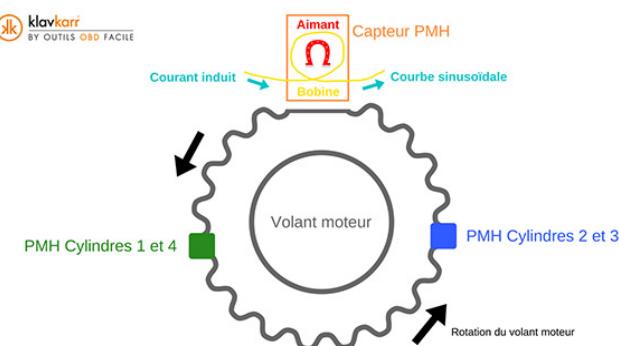
La commande de l'injecteur est, comme vu précédemment, réalisée par le calculateur. Celui-ci va tout simplement envoyer un signal électrique durant quelques millisecondes afin d'ouvrir l'injecteur. Voici le principe :

## RAPPORT SAE 5



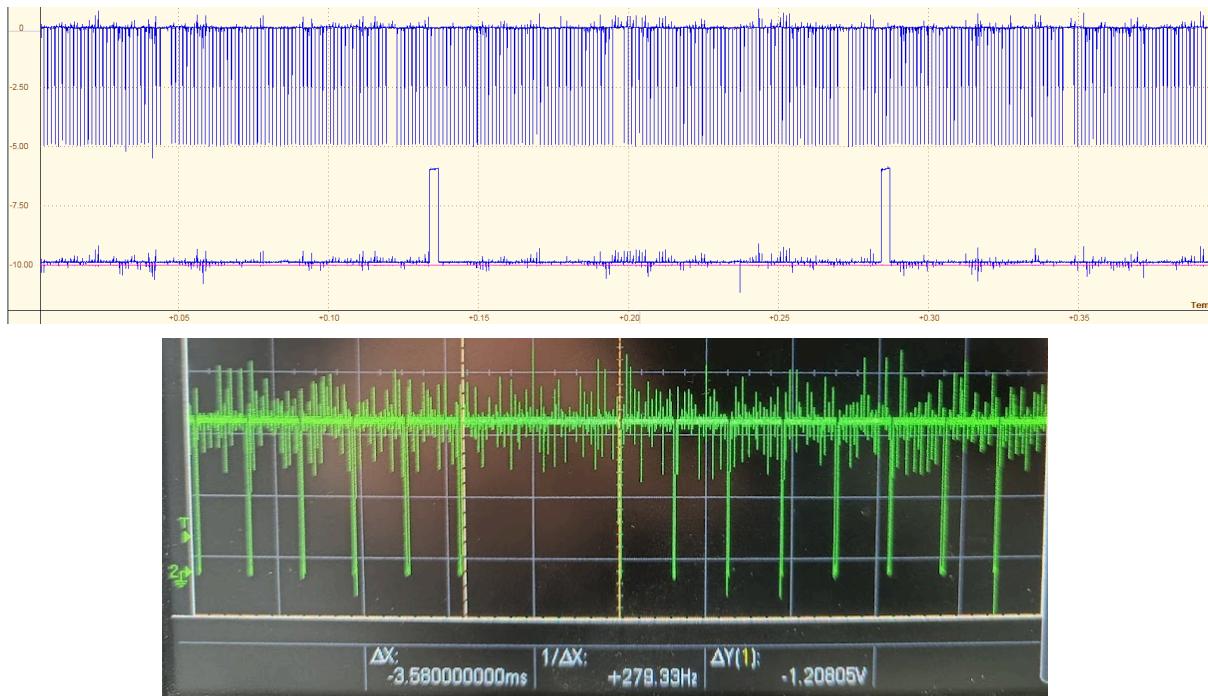
Un autre aspect du projet est de calculer en permanence le régime moteur (nombre de rotation effectué par le vilebrequin précédemment abordé, sur une certaine unité de temps) de l'automobile étudié, pour cela il faut un point de référence pour voir si un tour a été effectué, c'est donc le rôle du capteur PMH (avec pour autre nom capteur vilebrequin, que nous n'utiliserons pas). Il a pour rôle d'informer le calculateur sur la position des pistons pour que ce dernier puisse calculer le régime moteur (c'est à dire le nombre de tour par minute, que l'utilisateur de la voiture voit généralement sur son tableau de bord) et adapter en fonction l'injection de carburant.

Pour connaître la position des pistons, le capteur PMH s'appuie sur la rotation du volant moteur. Un repère est alors placé sur la roue crantée afin d'informer continuellement le calculateur sur la position des pistons : le capteur envoie l'information à chaque fois qu'il détecte le repère et compte le nombre de dents entre chaque intervalle ce qui permet au calculateur de suivre le rythme du moteur.



Ce qu'il faut comprendre c'est que cette roue crantée, il y a une fausse dent qui permet d'avoir un point de repère, comme on peut le voir sur le signal que sort le capteur PMH :

## RAPPORT SAE 5



Mais ça n'est rien de moins qu'un point de référence, en réalité le PMH peut ne pas correspondre à ce moment-là.

[obj]

Enfin le signal d'étincelle d'allumage est un élément crucial du système d'allumage d'un moteur à combustion interne. Il est généré par la bougie d'allumage à des moments spécifiques du cycle moteur pour initier la combustion du mélange air-essence dans la chambre de combustion. Ce signal est synchronisé avec la position du piston, assurant ainsi une combustion efficace. Lorsque le piston atteint le point mort haut (PMH) et que le mélange air-essence est correctement comprimé, la bougie d'allumage produit une étincelle électrique. Cette étincelle enflamme le mélange, générant une explosion contrôlée qui propulse le piston vers le bas, convertissant l'énergie chimique en mouvement mécanique. La précision de ce signal d'allumage est cruciale pour maximiser l'efficacité du moteur, optimiser la consommation de carburant et réduire les émissions. Voici ce signal :



## RAPPORT SAE 5

### **CAHIER DES CHARGES**

#### **Carte d'analyse des temps d'injection**

Dans le but de satisfaire au maximum le client, à savoir M. Fourmy, enseignant au CIFAM, nous avons transmis un cahier des charges pour spécifier toutes leurs attentes concernant ce projet. Voici ce que nous allons réaliser. Notre objectif est de réutiliser les projets des années précédentes en partant de la carte réalisée qui nous permet de récupérer les signaux issus du moteur afin des contraintes que nous allons énoncer plus tard, ce qui nous permettra de conserver la mesure de l'injection et d'y rajouter la mesure du régime moteur et du retard à l'allumage.

Les contraintes pour ce projet sont les suivantes : l'utilisation doit être simple, nous réaliserons une notice de mise en place. Le boîtier doit être facile à brancher sur le véhicule, facile d'entretien, et démontable facilement pour changer la carte électronique en cas de problèmes. Les spécifications de la carte sont que pour un temps d'injection de 1 ms, nous aurons une tension de 1V en sortie, et 1V correspond à 1 000 tours par minute.

#### **Carte retard allumage**

Entrées boîtiers :

## RAPPORT SAE 5

- Voie 1 entrée boitier : Signal PMH
- Voie 3 entrée boitier : Signal d'allumage

Sorties boîtier :

- Voie 1 sortie boitier : Déterminer le régime moteur
- Voie 2 sortie boitier : Déterminer la variable d'avance à l'allumage

## **ANALYSE FONCTIONNELLE**

### **Décomposition fonctionnelle**

Tout d'abord, nous commençons par analyser où la carte sera mise en place et

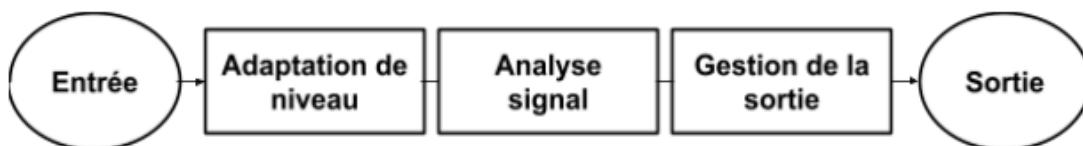
## RAPPORT SAE 5

utilisée, cette carte sera destinée à être placée dans un garage, donc la facilité de branchement et d'utilisation devra être présente. Pour commencer, on utilisera un connecteur USB type B pour avoir un moyen d'alimenter la carte de façon solide et permettre de connecter la carte un peu partout. Les connecteurs USB fournissent du 5V, mais afin de consommer le moins et d'avancer dans les moyens technologiques d'aujourd'hui, nous décidons de réduire la tension de toute la carte à 3.3 V à l'aide d'un régulateur. La carte doit être capable de capter des signaux entre 0 - 12V de la voiture, donc nous mettons en place une adaptation de la tension des signaux afin de protéger notre microcontrôleur. L'analyse des signaux se fera avec un SAMD21 (Microcontrôleur utilisé dans notre établissement), enfin la carte aura 4 sorties, pour simplifier le montage, nous utiliserons un DAC connecteur en i2C afin de gérer les sorties juste avec une histoire d'adresse.

Enfin, chaque signal de sortie passe par un montage à AOP amplificateur non-inverseur. La carte sera utilisée en atelier, nos connecteurs d'entrées et sorties seront des fiches bananes.

## Conception des blocs

Pour détailler l'analyse fonctionnelle faite précédemment, on réalise une décomposition fonctionnelle, voici un schéma simplifié du système que l'on va réaliser :



## Analyse du signal

Nous allons commencer par le cœur du montage, le microcontrôleur, le choix de celui qui va déterminer le reste du montage.

## RAPPORT SAE 5

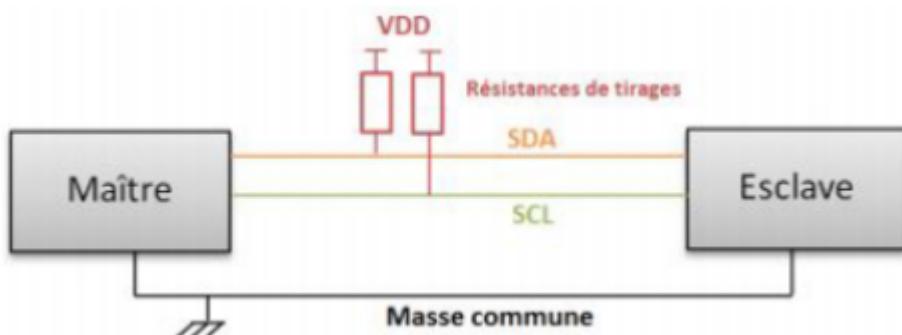
Pour ce projet, nous allons utiliser un ATSAMD21, un microcontrôleur que notre IUT utilise régulièrement en projet. C'est aussi un microcontrôleur très basique et assez performant pour pouvoir réaliser notre carte.

Celui-ci possède :

- 32 bits
- Plusieurs entrées/sorties avec interruption
- Adaptation 3.3 V

La programmation de la carte se fera en Arduino afin de faciliter la reprise du programme en cas de problème.

Nous allons utiliser la fonction I2C du microcontrôleur. Une liaison I2C permet de communiquer entre deux composants électroniques très divers grâce à 3 fils : signal de donnée (SDA), signal de synchronisation d'horloge (SCL) et la masse (GND). Cette communication est de forme half-duplex (1 sens à la fois) et fonctionne par adressage, car chaque périphérique a une adresse unique. L'I2C envoie des données sous la forme d'un octet.



Dans notre cas, le maître sera le microcontrôleur et l'esclave un Convertisseur Analogique Numérique.

Un Convertisseur Analogique Numérique va permettre de convertir la durée d'impulsion en une valeur de tension. Comme les projets précédents, nous allons utiliser un DAC 7578, car il possède 8 sorties (seulement 4 seront utilisées), il peut être alimenté entre 2.7 V et 5 V et est sur 12 bits.

## RAPPORT SAE 5

PW PACKAGE TSSOP-16 (TOP VIEW)				PIN DESCRIPTIONS	
PACKAGE		16-Pin	24-PIN	NAME	DESCRIPTION
LDAC	1	22	LDAC	Load DACs	
ADDR0	2	11	ADDR0	3-state address input	
AV <sub>DD</sub>	3	2	AV <sub>DD</sub>	Power-supply input, 2.7V to 5.5V	
V <sub>OUT</sub> A	4	3	V <sub>OUT</sub> A	Analog output voltage from DAC A	
V <sub>OUT</sub> C	5	4	V <sub>OUT</sub> C	Analog output voltage from DAC C	
V <sub>OUT</sub> E	6	5	V <sub>OUT</sub> E	Analog output voltage from DAC E	
V <sub>OUT</sub> G	7	6	V <sub>OUT</sub> G	Analog output voltage from DAC G	
V <sub>REFIN</sub>	8	7	V <sub>REFIN</sub>	Positive reference input	
DACx578		8	V <sub>REFIN</sub>	Asynchronous clear input	
		9	12	CLR	
		10	13	V <sub>OUT</sub> H	Analog output voltage from DAC H
		11	14	V <sub>OUT</sub> F	Analog output voltage from DAC F
		12	15	V <sub>OUT</sub> I	Analog output voltage from DAC I
		13	16	V <sub>OUT</sub> B	Analog output voltage from DAC B
		14	17	GND	Ground reference point for all circuitry on the device
		15	19	SDA	Serial data input. Data are clocked into or out of the input register. This pin is a bidirectional, open-drain data line that should be connected to the supply voltage with an external pull-up resistor.
		16	20	SCL	Serial clock input. Data can be transferred at rates up to 3.4MHz. Schmitt-trigger logic input.

- **Avdd** est relié au 3.3 V
- **Vrefin** sera relié à une tension de référence (explication ci-dessous)
- **SCL** relié au microcontrôleur avec une résistance de 4.7 KΩ
- **SDA** relié au microcontrôleur avec une résistance de 4.7 KΩ
- **ADDR0** permet de configurer l'adresse du DAC :

SLAVE ADDRESS	ADDR0
1001 000	0
1001 010	1
1001 100	Float

Dans notre cas, il sera relié à la masse donc une adresse de :

$$1001\ 0000_{(2)} \rightarrow 0x90_{(16)} \rightarrow 144_{(10)}$$

Le calcul de Vrefin se fait grâce à une formule donnée dans la datasheet du constructeur :

$$V_{OUT} = \frac{D_{IN}}{2^n} \times V_{REFIN}$$

## RAPPORT SAE 5

Le DAC est sur 12 bits donc :  $2^{12} - 1 = 4095$  Valeurs

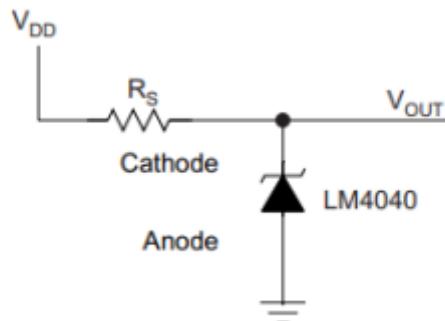
D'après notre premier cahier des charges, nous voulons 0.5 ms pour 1V donc si nous appliquons une tension de référence égale à 1 valeur = 1 mV soit une tension de référence de 4.095 V :

Durée d'impulsion	Valeur DAC	Tension
0 ms	0	0 V
1 ms	1000	1 V
4.095 ms	4095	4.095 V

Afin de conserver une tension d'alimentation de 3.3 V sur le DAC, nous avons décidé de diviser par deux la tension de référence, donc au lieu de 4.095 V, nous aurons 2.048 V. Voici le nouveau tableau explicatif du DAC :

Durée d'impulsion	Valeur DAC	Tension
0 ms	0	0 V
1 ms	500	0.5 V
8.190 ms	4095	4.095 V

Pour avoir une tension de référence propre et non dépendante de l'alimentation comme dans un diviseur de tension, nous utilisons une diode LM4040, avec un  $V_{th}$  de 2.048 V.



## RAPPORT SAE 5

Le choix de  $Rs$  se fait grâce à la formule donnée par le constructeur :

$$Rs = Vdd - \frac{V_{out}}{I_{r_{max}}}$$

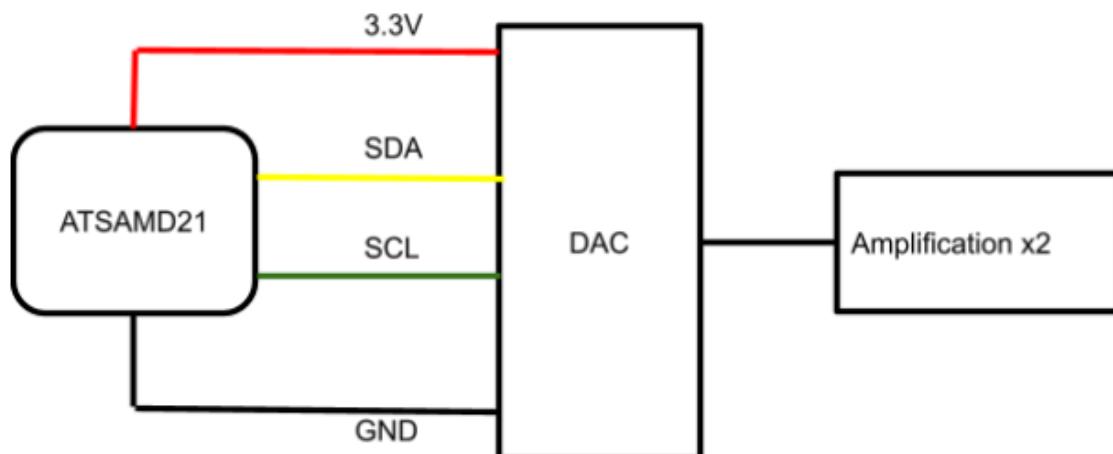
En sachant que :

- $Vdd = 5 \text{ V}$
- $V_{out} = 2.048 \text{ V}$
- $I_r = 9 \text{ mA}$

Donc d'après la formule  $Rs = 5 - \frac{2.048}{0.009} = 328 \Omega$  (résistance de  $330\Omega$ )

### Gestion de la sortie

Comme montré ci-dessus, à la sortie du DAC, nous n'aurons pas une conversion demandée soit de 1V pour 0.5 ms, pour remédier à ce problème, nous mettrons un circuit d'amplification x2 afin de respecter le cahier des charges.

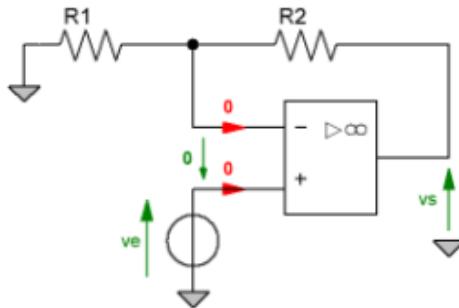


Pour la partie amplification, nous partirons simplement sur un montage avec Amplificateur Opérationnelle, amplificateur non-inverseur.

## RAPPORT SAE 5

Voici ci-dessous un schéma du montage avec l'expression de sortie :

L'amplificateur non inverseur



Expression de  $v_s$  en fonction de  $v_e$  :

$$v_s = + \left( 1 + \frac{R_2}{R_1} \right) \cdot v_e$$

Dans notre cas, nous voulons juste  $V_s = 2 * V_e$ , donc comme montré ci-dessus, il faut juste mettre  $R_1 = R_2$  afin d'avoir un coefficient 2 sur l'amplificateur :

$$V_s = 1 + \frac{R_2}{R_1} * V_e$$

$$\text{Si } R_1 = R_2$$

$$V_s = 1 + \frac{R_1}{R_1} * V_e$$

$$V_s = 1 + 1 * V_e$$

$$V_s = 2 * V_e$$

L'amplificateur opérationnel sert aussi de suiveur de tension, il va permettre au montage de conserver un signal de sortie identique au signal d'entrée, mais sans aucune perturbation possible en fonction de ce qui est relié en sortie de la carte. L'AOP a en effet une impédance d'entrée infinie pour avoir des courants d'entrée nuls. Le signal de sortie ne sera donc pas perturbé par les prélèvements.

Pour la partie amplification, nous prendrons des MCP6L91T, la tension de l'AOP sera de 0 - 5 V.

### Adaptation de tension

Pour chaque entrée, un limiteur de tension est installé afin de protéger les composants.

Nous devons avoir en entrée du microcontrôleur une tension maximum de 3.3 V comme montrée ci-dessous dans la datasheet :

## RAPPORT SAE 5

### Power Supplies

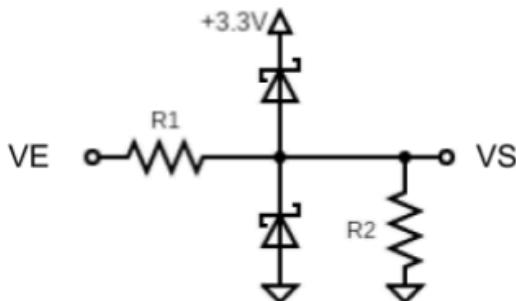
The Atmel® SAM D21 has several different power supply pins:

- VDDIO: Powers I/O lines, OSC8M and XOSC. Voltage is 1.62V to 3.63V.
- VDDIN: Powers I/O lines and the internal regulator. Voltage is 1.62V to 3.63V.
- VDDANA: Powers I/O lines and the ADC, AC, DAC, PTC, OSCULP32K, OSC32K, XOSC32K. Voltage is 1.62V to 3.63V.
- VDDCORE: Internal regulated voltage output. Powers the core, memories, peripherals, DFLL48M and FDPLL96M. Voltage is 1.2V.

The same voltage must be applied to both VDDIN, VDDIO and VDDANA. This common voltage is referred to as  $V_{DD}$  in the datasheet.

Les signaux récupérés du calculateur sont des signaux carrés de 0 à 12V, donc pour protéger notre carte, nous allons appliquer un pont diviseur de tension avec des diodes Schottky de protection :

Voici le schéma structurel :



Pour connaître les valeurs des résistances, on part de la formule du pont diviseur de tension :

$$V_S = \frac{V_E * R_2}{R_1 + R_2}$$

Afin de ne pas perturber le calculateur, nous mettons en place une résistance R1 très grande afin d'avoir une grande impédance d'entrée.  
(100 KΩ)

Pour trouver R2, nous résolvons une équation avec 1 inconnu :

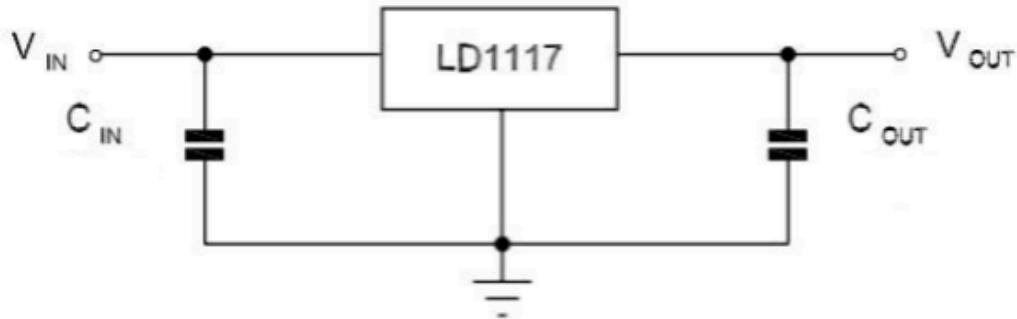
$$\Rightarrow 3.3 = \frac{12 * R_2}{100K + R_2} \Rightarrow R_2 = \frac{3.3 * 100K}{12 - 3.3} = 37\ 931\Omega$$

La résistance la plus proche est 39 KΩ

En finissant la partie conception des blocs, nous ajoutons un régulateur de tension de 5V vers 3.3 V, pour ne pas avoir d'alimentation externe.

## RAPPORT SAE 5

Pour celui-ci, nous utilisons un régulateur que nous utilisons en projet, le LD1117 :



Chaque entrée de composant possède des condensateurs de découplage, comme le montre l'image ci-dessus, pour permettre de compenser les pics de courant.

## RAPPORT SAE 5

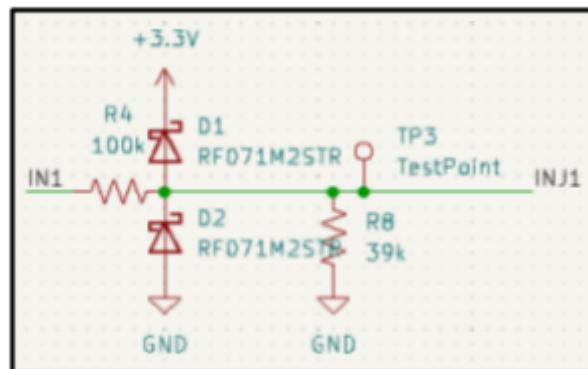
### PARTIE HARDWARE

Pour la partie Hardware, nous avons utilisé le logiciel KiCad qui est gratuit. KiCad est un logiciel multi-plateforme. Il utilise la bibliothèque graphique libre. Il permet de réaliser toutes les étapes nécessaires à la conception d'un circuit imprimé : réalisation du schéma électrique, association des empreintes de composants, routage, export au format Gerber.

#### Conception de la carte

Voici le schéma électrique global de chaque partie représenté sous Kicad :

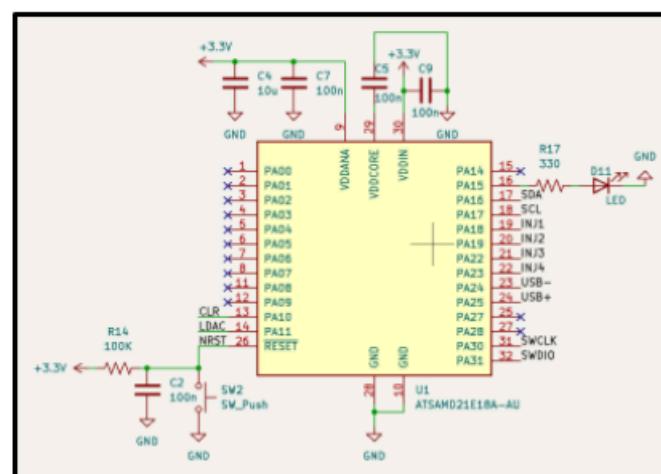
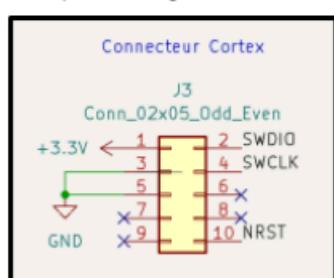
**Adaptation de tension :**



[Changement des résistances](#) pour les entrées, on remplace R8 qui était une résistance 39k par une résistance 180k pour obtenir un niveau de 0 à 5 V en lecture.

**ATSAMD21 :**

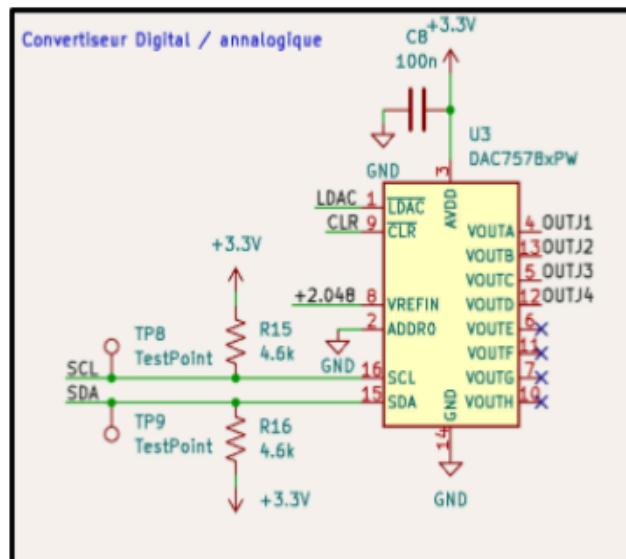
Un bouton RESET et une Led de débogage ont été ajoutés afin de tester les différents programmes. Un connecteur cortex a aussi été rajouté afin de boot le microcontrôleur et d'utiliser potentiellement le mode débogage pas à pas du logiciel Atmel Studio.



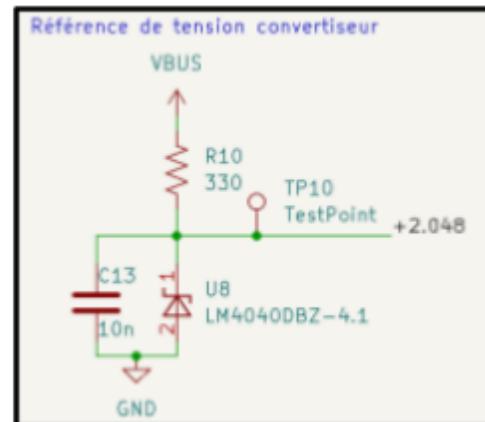
## RAPPORT SAE 5

### Le DAC :

Le DAC et CLR sont connectés au microcontrôleur afin de paramétriser les modes de fonctionnement du DAC.

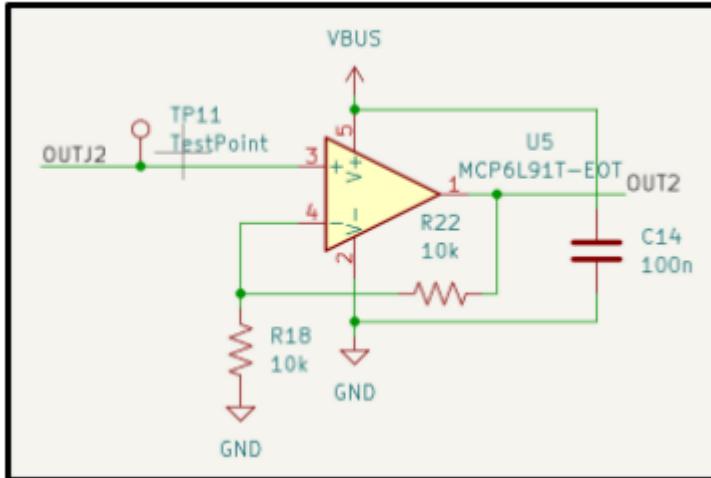


Et, la référence de tension du DAC.

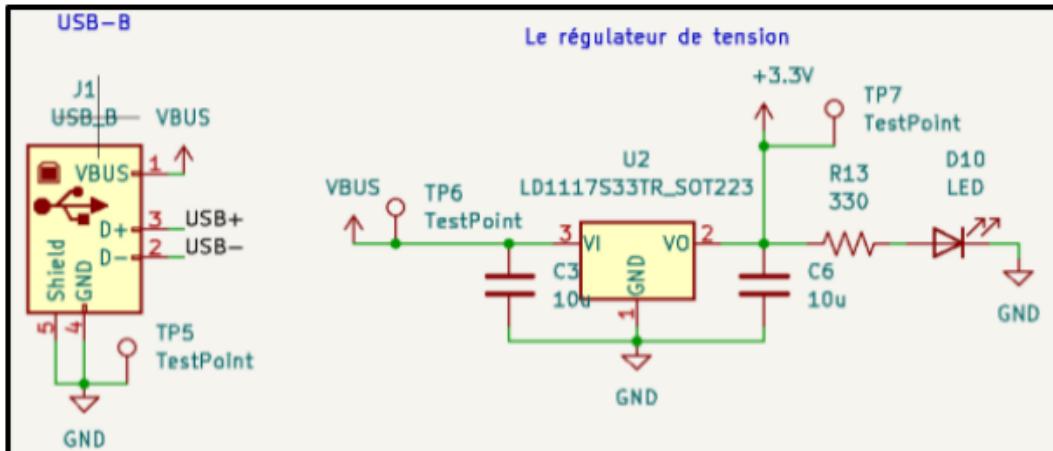


## RAPPORT SAE 5

### La sortie avec amplification :

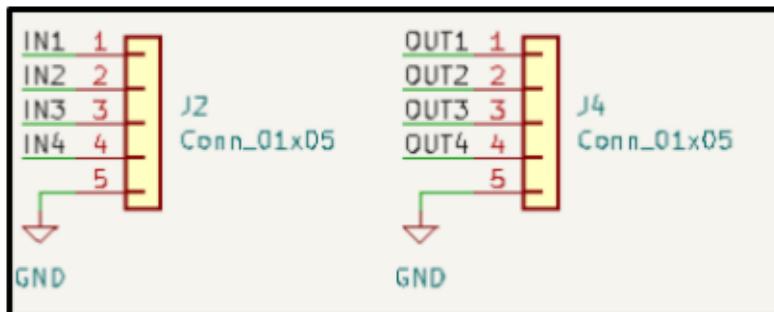


### La partie alimentation :



Une Led a été ajoutée qui s'allume lorsque la carte est alimentée.

### L'entrée et sortie :



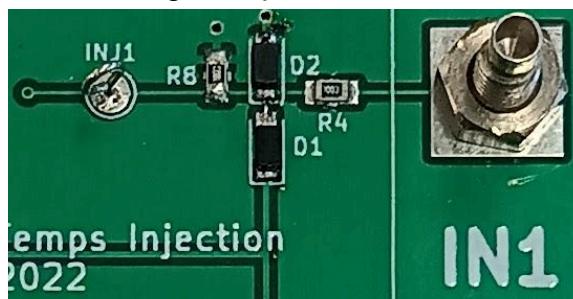
Voici nos connecteurs de sortie, 4 entrées avec leurs 4 sorties associées et une masse de référence.

## RAPPORT SAE 5

### Maintenance

#### ➤ Vérification des Diodes d'Entrée

L'examen méticuleux des diodes utilisées sur les entrées de la carte a été une étape essentielle de notre processus de maintenance. Cette vérification méticuleuse avait pour objectif de garantir le bon état de ces composants clés. Des images détaillées des diodes vérifiées ont été consignées pour référence future.

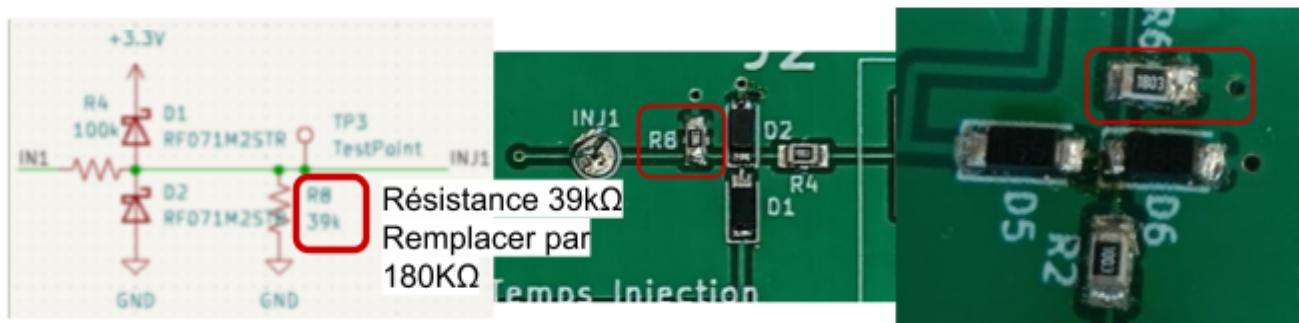


#### ➤ Vérification et Réajustement Précis des Valeurs de Résistance

Dans le cadre de cette opération de maintenance, nous avons minutieusement mesuré et vérifié les valeurs des résistances sur les entrées de la carte de retard à l'allumage. Après avoir constaté des écarts par rapport aux valeurs attendues, des ajustements ont été entrepris pour rectifier ces différences.

#### ➤ Adaptation des Résistances pour une Lecture Conforme

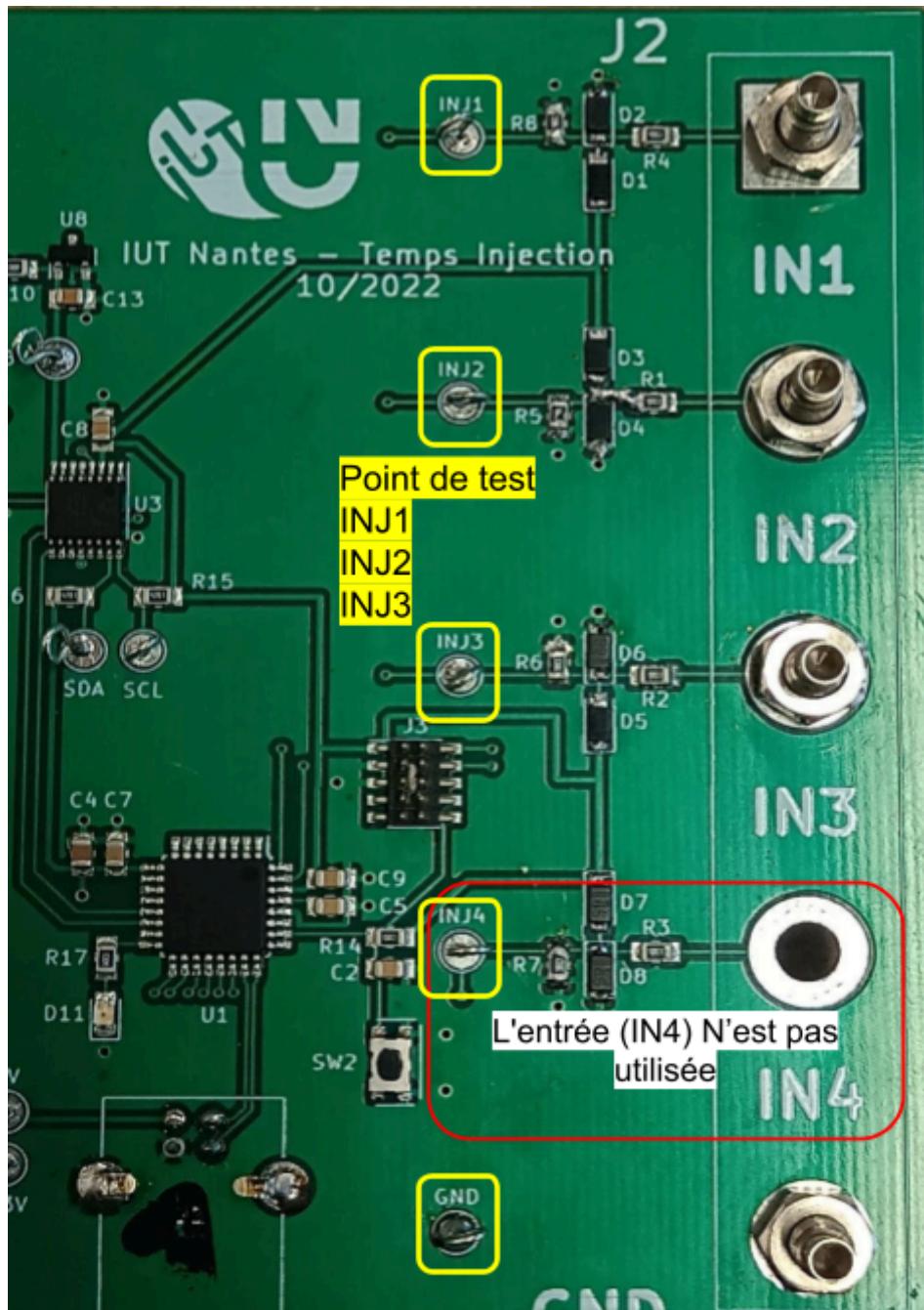
Après des analyses approfondies, nous avons identifié la nécessité de revoir certaines valeurs de résistance afin d'obtenir des niveaux de tension en entrée adéquats pour nos besoins spécifiques. Plus précisément, la résistance R8, initialement notée à  $39\text{ k}\Omega$ , a été remplacée par une résistance de  $180\text{ k}\Omega$ . Cette adaptation a permis d'obtenir une plage de tension en lecture allant de 0 à 5V, conforme à nos spécifications pour le véhicule.



## RAPPORT SAE 5

### Utilisation d'Appareils de Mesure pour une Évaluation Précise

Pour effectuer des mesures précises et garantir l'exactitude de nos ajustements, nous avons utilisé des sondes d'oscilloscope sur des points de test spécifiques. Ces outils nous ont permis de mesurer avec précision les niveaux de tension et de surveiller les signaux à différents stades des modifications apportées à la carte.



## RAPPORT SAE 5

### **PARTIE SOFTWARE**

*Dans cette section, nous explorerons en détail le développement et la prise en main logiciel du projet, mettant en lumière les algorithmes, les obstacles rencontrés et les stratégies utilisées pour réaliser l'attente du cahier des charges*

#### BootLoader

Pour pouvoir programmer en langage Arduino la carte nous devons booter la carte à l'aide d'un fichier binaire de configuration.

Pour charger le fichier boot sur le microcontrôleur, il faut une sonde de programmation JLink (voir ci contre) connecter au connecteur cortex installé précédemment



Puis soit passer par le logiciel Atmel Studio ou le logiciel Jlink Flash, qui permet à tous deux de charger le bootloader dans un microcontrôleur.

Disponible en fichier dans le dossier du projet. Pour la programmer ensuite sur Arduino, il faut installer les pilotes de la carte sur le logiciel :

- Ajouter en préférence la carte :  
[https://www.mattairtech.com/software/arduino/package\\_MattairTech\\_index.json](https://www.mattairtech.com/software/arduino/package_MattairTech_index.json)
- Installer la carte via Outils / Type de carte / Gestionnaire de carte
- Chercher " MattairTech SAMD|L|C core for Arduino "
- Et, l'installer

#### Blink

Afin d'essayer de voir si la programmation de la carte est disponible, nous passons par un programme de base qui vise juste à faire clignoter une Led de debug (sur notre carte cette Led est verte).

Si nous branchons la carte, le système d'exploitation détecte un périphérique, "MattairTech", cela signifie que notre carte est bien détectée.

## RAPPORT SAE 5

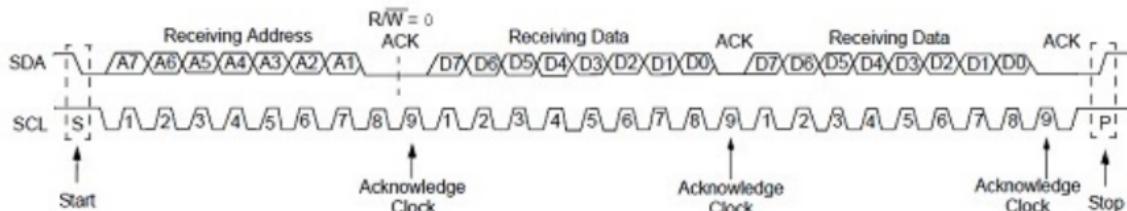
Sur le logiciel Arduino, nous devons sélectionner votre carte et ses paramètres :

- Outils
- Type de carte
- Sélectionner “MattairTech MT-D21E(rev B)”
- Sélectionner en oscillateur → internal oscillator
- Enfin, le port de communication

## DAC

Pour avoir une transmission plus facile et avoir une conversion juste, nous avons décidé de mettre en place un DAC (Digital Analog Converter), celui-ci va permettre de transmettre une tension précise juste avec une information que le microcontrôleur va lui envoyer. Pour rappel, le microcontrôleur est le maître et le DAC l'esclave.

L'I2C permet de communiquer entre ces deux circuits, le fonctionnement de l'I2C expliquer ci-dessous :



La liaison fonctionne sur une base de trame avec un bit de start et de stop et avec une horloge qui permet de synchroniser les circuits entre eux.

Pour être clair tel que nous l'avons configuré est le suivant : Nous ouvrir une communication du DAC avec le reste des composants électroniques du montage :

```
Wire.beginTransmission(Address_dac); // Commence la transmission sur l'adresse du DAC
```

Grâce à quoi nous pouvons configurer le comportement du DAC selon son entrée et sa sortie :

## RAPPORT SAE 5

```
#define OUT1 0x30 // Adresse DAC sortie 1
#define OUT2 0x31 // Adresse DAC sortie 2
#define OUT3 0x32 // Adresse DAC sortie 3
#define OUT4 0x33 // Adresse DAC sortie 4

#define IN1 18    // Pin entrée 1
#define IN2 19    // Pin entrée 2
#define IN3 22    // Pin entrée 3
#define IN4 23    // Pin entrée 4
```

Si vous voulez vous assurer que le DAC fonctionne correctement il faut alors tester un code comme ceci, ou il faut précédemment s'assurer qu'on ait le moyen de regarder la sortie du convertisseur :

```
#include <Wire.h>                                // Bibliotheque I2C
#define Address_Dac 0x90                            // Adresse DAC

void setup()
{
    Wire.begin();                                  // Demarrage protocole I2C
}

void loop()
{
    Wire.beginTransmission(Address_Dac);           // Demarrage I2C sur Dac
    Wire.write(0x55);                             // Envoie valeur 0x55 sur la trame
    delay(200);                                 // Attendre I2C
    Wire.endTransmission();                       // Fin de transmission sur DAC
}
```

Enfin il faut savoir que dans le cadre de notre projet le but est de retranscrire le comportement du moteur en niveau de tension sur la sortie du DAC, dont l'interprétation sera transmise, à cet effet l'écriture d'1 volt sur la sortie du DAC 1 s'écrit tel quel :

```
DacWrite(1000, OUT1);
```

## RAPPORT SAE 5

## Fonctionnement général

### Constantes et fonction de base

Comme entrevu précédemment , le code a été configuré selon les *constantes* afin de faciliter son utilisation et sa compréhension, a été également écrit des fonctions de configurations générique qui sont utilisés dans le programme principal et pour toutes les tests réalisés, voici comment cela a été organisé :

```
#include <Arduino.h>
#include <Wire.h> // Bibliothèque I2C
```

L'inclusion de la bibliothèque *Arduino.h* fournit des définitions de base pour les fonctions d'E/S (Entrée/Sortie), les opérations de temporisation, la gestion des interruptions, etc. Et celle de *Wire.h* nous permet d'accéder de communiquer entre le micro-contrôleur sur lequel nous écrivons et le DAC, comme nous le verrons par la suite, étant donc essentiel dans notre projet.

Ensuite comme évoqué précédemment il y a la définition des entrées et sorties du DAC, permettant alors de transmettre les informations que l'on a traité de l'entrée du DAC, en niveau de tension sur la sortie :

```
#define Address_dac 0x48 // Adresse du DAC
#define OUT1 0x30 // Adresse DAC sortie 1
#define OUT2 0x31 // Adresse DAC sortie 2
#define OUT3 0x32 // Adresse DAC sortie 3
#define OUT4 0x33 // Adresse DAC sortie 4

#define IN1 18    // Pin entrée 1
#define IN2 19    // Pin entrée 2
#define IN3 22    // Pin entrée 3
#define IN4 23    // Pin entrée 4
```

## RAPPORT SAE 5

Il y a donc logiquement l'utilisation de ces variables afin d'initialiser les différentes pins en tant qu'entrées et sorties :

```
void ES_Init() // Fonction initialisation PIN
{
    pinMode(IN1, INPUT);      // Signal 1
    pinMode(IN2, INPUT);      // Signal 2
    pinMode(IN3, INPUT);      // Signal 3
    pinMode(IN4, INPUT);      // Signal 4

    pinMode(15, OUTPUT);      // LED test fonctionnement SAMD21
    digitalWrite(15, LOW);     // Initialisation Led éteinte
}

void DacInit() // Fonction initialisation DAC
{
    pinMode(10, OUTPUT);      // Clear en sortie du DAC
    pinMode(11, OUTPUT);      // Ldac en sortie du DAC
    digitalWrite(10, LOW);     // Clear à 0
    digitalWrite(11, LOW);     // Ldac à 0
    digitalWrite(10, HIGH);    // Remise à 0 du DAC
}
```

On a également mis en place le fait de pouvoir allumer et éteindre une LED sur la carte SAMD21 afin d'avoir un retour physique de notre programme, car nous le verrons par la suite l'écriture numérique en même temps que l'écriture sur le DAC constitue un problème.

Par la suite, nous initialisons le DAC en configurant les broches du clear et ldac du DAC, qui servent pour le premier à initialiser les sorties à zéro, établissant un point de départ défini. Et pour le deuxième à synchroniser le chargement des nouvelles données dans le DAC, évitant des variations abruptes.

Enfin il y a la fonction qui est responsable de l'écriture en sortie sur le DAC, elle permet de sortir le niveau de tension voulue sur une sortie donné

## RAPPORT SAE 5

```

void DacWrite (unsigned int data, unsigned int channel)
{
    int MSBD, LSBD;          // Variable bit de start et stop

    MSBD = (data & 0xFF0) >> 4;           // 8 bits de poids fort
    LSBD = ((data & 0x00F) << 4 ) & 0x0FF; // 8 bits de poids faible

    Wire.beginTransmission(Address_dac);   // Commence la transmission sur l'adresse du DAC
    Wire.write(channel);                  // Commence la transmission sur la sortie
    Wire.write(MSBD);                   // On envoie 8 bits de poids forts
    Wire.write(LSBD);                   // On envoie 8 bits de poids faible
    Wire.endTransmission();             // On termine la transmission
}

```

La fonction *DacWrite* commence par diviser la valeur *data* en deux parties, MSBD (Most Significant Byte) et LSBD (Least Significant Byte), qui représentent respectivement les 8 bits de poids fort et les 8 bits de poids faible. Ces deux parties sont obtenues en effectuant des opérations de décalage et de masquage sur la valeur *data*. Ensuite, la communication avec le DAC est initiée en utilisant la bibliothèque *Wire*. La transmission débute par l'envoi du numéro de canal (*channel*) suivi des octets MSBD et LSBD. Une fois ces données transmises, la communication I2C est terminée. Ainsi, la fonction *DacWrite* assure la préparation et la transmission appropriées des données vers le DAC pour contrôler les sorties analogiques en fonction des besoins du système de contrôle moteur.

### Logique de la structure

L'idée est la suivante :

On reçoit, dans une première approche de notre projet deux signaux du moteur de la voiture :

Le retour du capteur PMH et celui du retard à l'allumage. Le concept est d'à la fois calculer en temps réel le régime moteur grâce à ce qui nous est reçu du capteur PMH et de calculer le retard à l'allumage en degré, par rapport à sa position en fonction du point de référence que l'on a pris pour le PMH. Un des problèmes auxquels on doit faire face est que l'on change en permanence ce fameux point de référence sur le PMH, et donc on change le calcul du retard à l'allumage. Il faut alors donner une priorité à ces calculs pour ne pas "sauter" des moments de calcul et tout fausser.

C'est là qu'intervient la notion d'interruption: Les interruptions sur des broches (pins) sont une fonctionnalité essentielle dans les microcontrôleurs, permettant au processeur d'interrompre son flux d'exécution principal pour traiter des événements

## RAPPORT SAE 5

spécifiques en temps réel. Lorsqu'une interruption est déclenchée, le processeur suspend temporairement l'exécution du programme en cours pour exécuter un autre morceau de code appelé "routine d'interruption" associé à l'événement déclencheur. Sur des broches, ces événements peuvent être des changements d'état logique, tels que des fronts montants (de LOW à HIGH) ou descendants (de HIGH à LOW). Dans le contexte des microcontrôleurs Arduino, des fonctions telles que attachInterrupt() sont utilisées pour associer une broche à une routine d'interruption spécifique. Cela est particulièrement utile pour la gestion de signaux externes en provenance de capteurs, de boutons-poussoirs, ou d'autres périphériques, permettant au microcontrôleur de répondre immédiatement à ces signaux sans interrompre le déroulement normal du programme.

Dans notre application, l'utilisation des interruptions sur les broches (pins) se révèle particulièrement avantageuse pour la gestion des signaux PMH (Point Mort Haut) et AA (Allumage). La caractéristique concise de l'état haut de l'AA, indiquant l'allumage, facilite son déclenchement par une interruption sur front montant ou descendant. En revanche, le signal PMH génère des états hauts successifs, nécessitant une vérification à chaque état bas. Ainsi, nous configurons les interruptions de la manière suivante :

- Pour l'AA, nous utilisons attachInterrupt() sur le front montant ou descendant, en fonction de la caractéristique du signal d'allumage.
- Pour le PMH, nous utilisons attachInterrupt() sur le front bas, ce qui nous permet de détecter chaque transition de l'état haut à l'état bas du signal PMH et de prendre les mesures nécessaires en conséquence.

Voici comment ça se présente :

```
attachInterrupt(digitalPinToInterrupt(IN1), premier_interrupt, RISING); // procédure d'interruption entré 1, Pin 18
attachInterrupt(digitalPinToInterrupt(IN2), deuxième_interrupt, FALLING); // procédure d'interruption entré 2, Pin 19
```

En illustrant la configuration des interruptions, les lignes de code ci-dessus montrent comment nous utilisons la fonction attachInterrupt() pour définir des procédures d'interruption sur les broches spécifiques IN1 et IN2. Pour IN1 (Pin 18), l'interruption est déclenchée sur le front montant (RISING), correspondant à la caractéristique du signal AA. Pour IN2 (Pin 19), l'interruption est déclenchée sur le front descendant (FALLING), correspondant à la caractéristique du signal PMH.

Il est essentiel de souligner que l'écriture continue sur la sortie du DAC n'est pas nécessaire. Une fois que l'information a été transmise au DAC pour générer une tension spécifique sur une sortie donnée, il maintiendra cette sortie à cette valeur jusqu'à ce qu'une nouvelle décision soit prise. Cette caractéristique nous permet de minimiser les écritures sur le DAC, ce qui est crucial pour éviter tout conflit potentiel entre les calculs en cours et les opérations d'écriture. En restreignant le moment où

## RAPPORT SAE 5

nous effectuons des écritures sur le DAC, nous limitons le moment critique qui ne peut être réservé qu'à ça.

C'est pourquoi nous avons fait le choix d'écriture le DAC, que lorsque les calculs ne sont pas en train d'être faits. Pour cela on désactive les calculs en détachant les broches d'interruptions

```
// Détache les interruptions pour éviter les conflits
detachInterrupt(digitalPinToInterruption(IN1));
detachInterrupt(digitalPinToInterruption(IN2));
```

Et on fait l'écriture dans une fonction dédié :

```
void dacWriteBuffer()
{
    DacWrite(buffer_regime, OUT1);
    DacWrite(buffer_allumage, OUT2);

    attachInterrupt(digitalPinToInterruption(IN1), premier_interrupt, FALLING); // Procédure d'interruption entrée 1, Pin 18
    attachInterrupt(digitalPinToInterruption(IN2), deuxieme_interrupt, RISING); // Procédure d'interruption entrée 2, Pin 19
}
```

Et donc on réactive les 2 interruptions qui sont responsables des calculs.

## RAPPORT SAE 5

### Calculs des sorties

Voici donc le cœur du calcul de ce que l'on renvoie en sortie du DAC et donc la partie visible de l'utilisateur :

Pour ce qui est du PMH :

```
// Sauvegarde de la durée de la dent précédente pour la comparaison
DP_PMH = DC_PMH; // Enregistre le temps de la dent précédente pour la comparaison

// Enregistrement du temps du front montant actuel
T1_PMH = micros(); // Temps du front montant

// Calcul de la durée de la dent actuelle
DC_PMH = T1_PMH - T2_PMH;

// Mise à jour du temps du front montant précédent
T2_PMH = T1_PMH;

// Si la durée de la dent actuelle est plus de deux fois la durée de la dent précédente
if (DC_PMH > DP_PMH * 2)
{
    // Sauvegarde des temps pour le calcul du régime moteur
    temp_0 = temp_1;
    temp_1 = T1_PMH;

    // Calcul du régime moteur (N) en tr/min
    PMH = temp_1 - temp_0;
    buffer_regime = (60000000 / PMH);
}
```

Et pour ce qui est de l'avance à l'allumage :

## RAPPORT SAE 5

```

// Enregistrement du temps du front descendant
TI = micros();

// Calcul de l'avance en microsecondes
avance = TI - temp_1;

//Serial.println(avance);

// Calcul du rapport entre le temps entre la période du PMH et son écart avec l'allumage
rapport = avance / PMH;

// Ajuste le rapport si supérieur à 1, one le ramène en dessous
rapport = (rapport > 1) ? (rapport - 1) : rapport;

// Conversion du rapport en degrés
deg = rapport * 360;

// Calcul de la valeur de l'allumage en fonction du décalage et du gain

buffer_allumage = (deg - 105) * Gain_Out2;

// Réinitialisation du drapeau d'interruption
IFLAG = 0;

// Ecris dans les DAC et réattache les interruptions pour la prochaine itération
dacWriteBuffer();

```

## Banc d'essais

Nous avons expérimenté diverses méthodes d'amélioration. Récemment, nous avons intégré des potentiomètres pour permettre la variation dynamique du signal, élargissant ainsi nos capacités d'analyse et de test.

```

// Définition des broches pour les signaux PMH et
ALLUMAGE ainsi que pour les potentiomètres A0 et A1
#define PMH 12
#define ALLUMAGE 11
#define POT_A0 A0
#define POT_A1 A1

int potValue1;
int potValue2;

```

## RAPPORT SAE 5

```

int period; // Période initiale du signal en microsecondes
int multi;
int base;
int toothCount = 0; // Compteur de dents
int allumageTrigger = 0; // Compteur pour déclencher ALLUMAGE

void setup() {
    // Configuration des broches en entrée ou en sortie
    pinMode(PMH, OUTPUT);
    pinMode(ALLUMAGE, OUTPUT);
    pinMode(POT_A0, INPUT);
    pinMode(POT_A1, INPUT);
}

void loop() {
    // Lecture des valeurs des potentiomètres pour ajuster la période du signal
    potValue1 = analogRead(POT_A0);
    potValue2 = analogRead(POT_A1);
    // Mapping des valeurs lues du potentiomètre pour ajuster la période entre 1000 et 10000
    microsecondes
    multi = map(potValue2, 0, 1023, 10, 100);
    base = map(potValue1, 0, 1023, 50, 16);
    period = base * multi;
    // Affichage des valeurs lues des potentiomètres et de la période calculée dans le
    moniteur série
    Serial.print(multi);
    Serial.print(" ");
    Serial.print(base);
    Serial.print(" ");
    Serial.println(period);
    // Génération du signal PMH
    generatePMHSignal();

    // Génération du signal ALLUMAGE un peu après chaque période de PMH
    if (toothCount == 20 && allumageTrigger == 2) {
        digitalWrite(ALLUMAGE, HIGH);
        delayMicroseconds(100); // Réduire ce délai pour retarder le signal ALLUMAGE
        digitalWrite(ALLUMAGE, LOW);
    }
}

```

## RAPPORT SAE 5

```

allumageTrigger = 0; // Réinitialisation du compteur pour le prochain déclenchement
}

}

void generatePMHSignal() {
if (toothCount < 58) { // Génère les 58 dents
    digitalWrite(PMH, LOW);
    delayMicroseconds(period / 2); // On maintient le signal HIGH pendant la moitié de la
periode
    digitalWrite(PMH, HIGH);
    delayMicroseconds(period / 2); // Puis le signal LOW pendant l'autre moitié de la période
toothCount++;
} else { // Génère les 2 dents manquantes
    digitalWrite(PMH, LOW);
    delayMicroseconds(period*2); // On maintient le signal HIGH pendant la moitié de la
periode
    digitalWrite(PMH, HIGH);
    delayMicroseconds(period*2);
    toothCount = 0; // Réinitialisation du compteur pour recommencer le cycle
    allumageTrigger++; // Marque le déclenchement pour le signal ALLUMAGE
}
}

```

- On définit des broches pour les signaux PMH et ALLUMAGE ainsi que pour les potentiomètres A0 et A1.
- On utilise les valeurs des potentiomètres pour ajuster la période du signal PMH.
- La boucle loop génère le signal PMH en fonction de la période définie, puis déclenche brièvement le signal ALLUMAGE après chaque période complète du PMH.

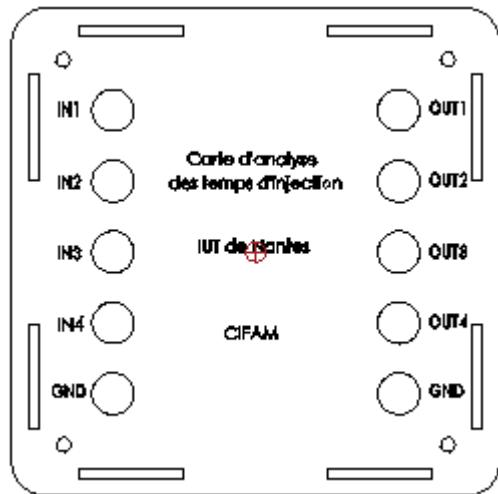
Cela permet générer un signal de type PMH (Point Mort Haut) avec une période ajustable avec les potentiomètres A0 et A1, et déclenche le signal ALLUMAGE en fonction du compteur de dents du signal PMH.

## RAPPORT SAE 5

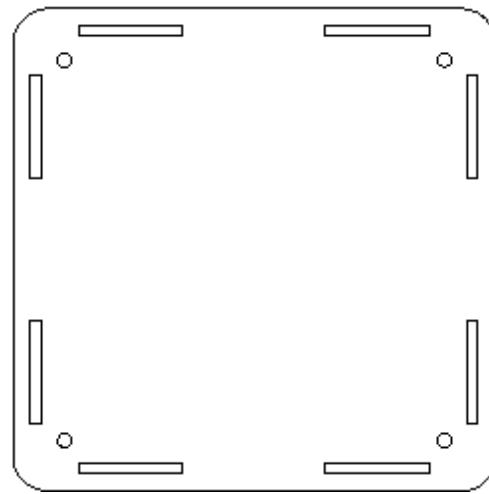
### PARTIE MÉCANIQUE

#### CAO Carte d'analyse des temps d'injection

Le premier boîtier a été fait en 2017. Malheureusement, nous avons dû en refaire une autre, car nous n'avons pas mis la même empreinte pour les entrées et sorties. Nous avons choisi une empreinte avec laquelle les espaces entre les trous sont plus grands.

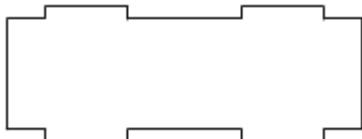


Face avant

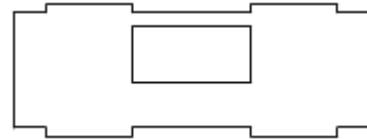


Face arrière

Coté du boitier



Avant du boitier

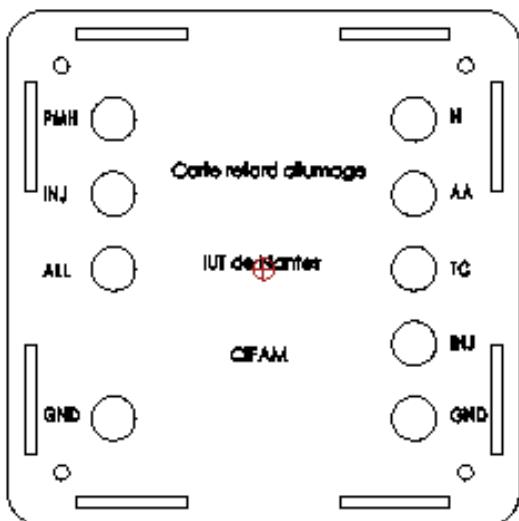


## RAPPORT SAE 5

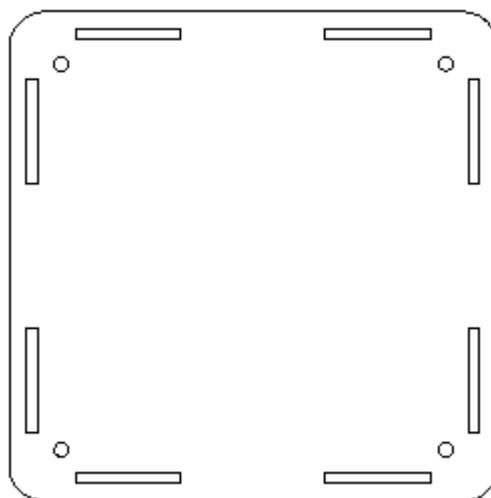
### CAO Carte retard allumage

Le deuxième boîtier a été conçu en prenant comme référence le boîtier utilisé pour la carte d'analyse des temps d'injection. Les principales différences résident dans l'absence d'un trou d'entrée spécifique et dans les inscriptions ajustées pour refléter les différentes entrées et sorties nécessaires pour cette carte.

En plus de ces adaptations, une attention particulière a été portée au choix des matériaux pour ce boîtier. Contrairement au modèle original en plastique, nous avons exploré différentes options, notamment le bois et le plexiglas, pour ce deuxième boîtier.



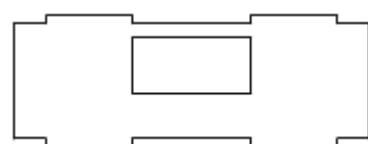
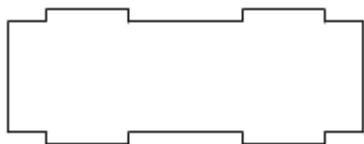
Face avant



Face arrière

Côté du boitier

Avant du boitier



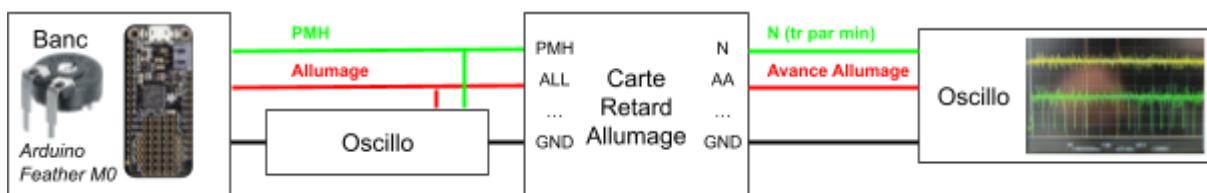
## RAPPORT SAE 5

### **TEST**

#### Test Retard Allumage à l'IUT :

Pour réaliser des tests sur la carte de retard à l'allumage au sein de l'IUT, nous réalisons ce montage :

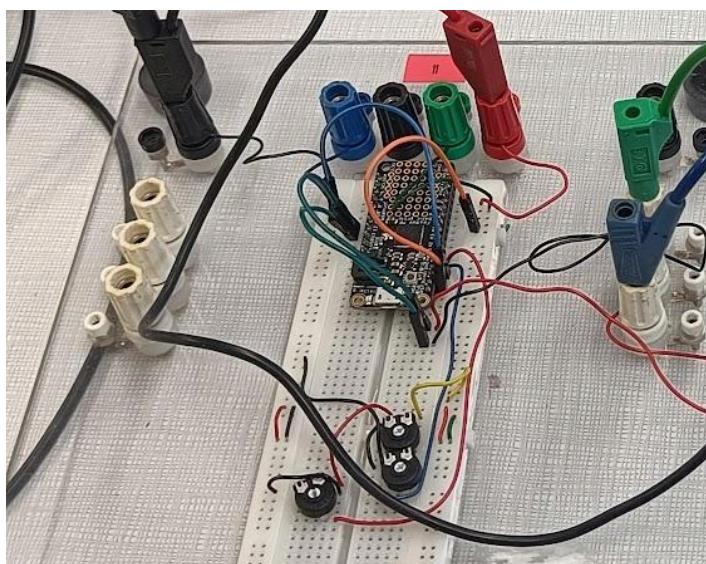
Montage IUT



Au laboratoire de l'IUT, nous avons élargi les capacités de notre banc d'essais en ajoutant une fonctionnalité de variation des signaux. Nous avons incorporé des potentiomètres au système, offrant ainsi la possibilité de régler et varier les signaux simulés du PMH et de l'allumage.

#### ➤ Montage du banc essais et Simulation des signaux

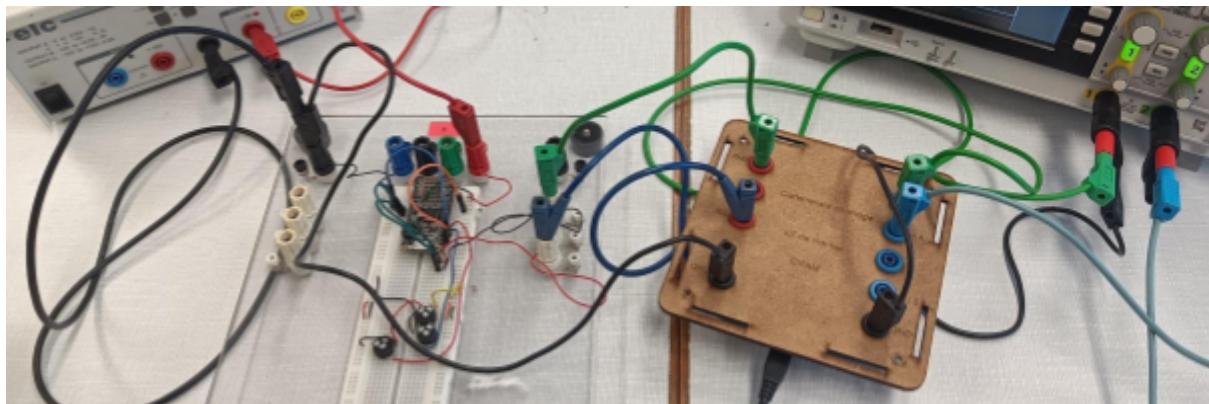
Au laboratoire de l'IUT, nous avons utilisé une plaque d'essai équipée d'un Adafruit Feather M0 (SAMD21) pour simuler les signaux du PMH et de l'allumage. Ces signaux ont été adaptés à notre carte, passant de 0 à 12V à l'aide d'un level shifter pour être compatibles avec le microcontrôleur du banc d'essai fonctionnant sous 3,3V. La programmation des interruptions de timer sur le SAMD21 nous a permis de générer précisément ces signaux simulés.



## RAPPORT SAE 5

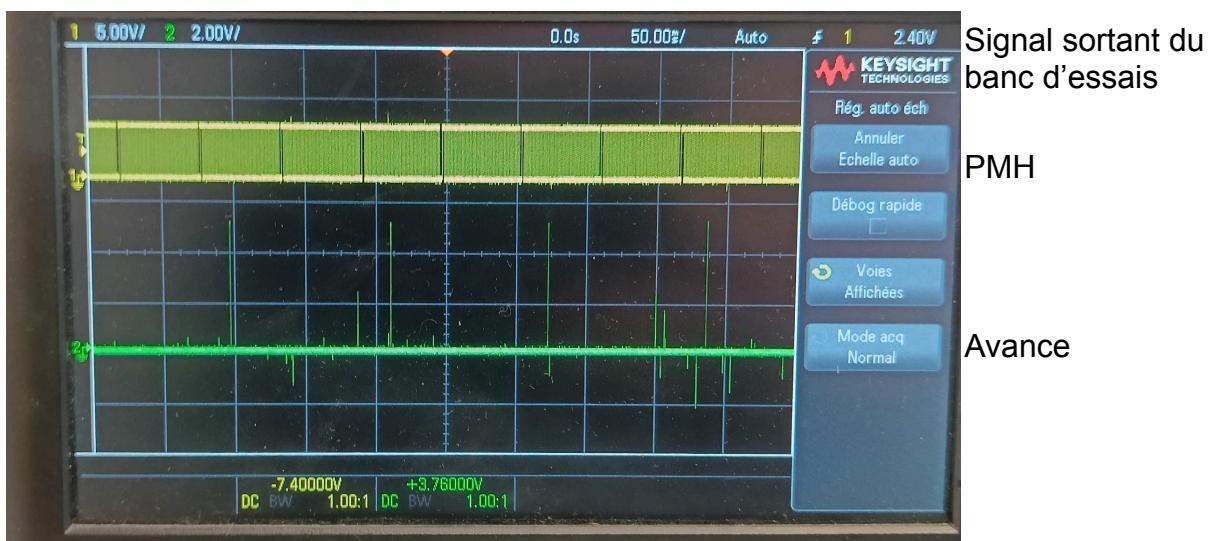
### ➤ Adaptation des Tensions pour la Compatibilité de la carte d'acquisition

Un ajustement crucial a été nécessaire pour garantir la compatibilité des tensions entre notre carte et le microcontrôleur du banc d'essai. Alors que notre carte d'acquisition fonctionnait sur une plage de tensions de 0 à 12V à ce moment, le microcontrôleur du banc de test opérait sous 3,3V. Pour rendre nos signaux de test compatibles avec le système de contrôle du moteur, nous avons utilisé un level shifter pour éllever les signaux de 0 à 12V.



### ➤ Méthode de Génération des Signaux

Pour reproduire au plus près les conditions réelles du fonctionnement d'un moteur, nous avons programmé des interruptions de timer sur le SAMD21. Cette programmation nous a permis de générer simultanément et de manière contrôlée deux signaux distincts : le signal du PMH et celui de l'allumage. Cette approche a été cruciale pour simuler des conditions de fonctionnement similaires à celles rencontrées dans un environnement automobile.



Signal sortant du banc d'essais

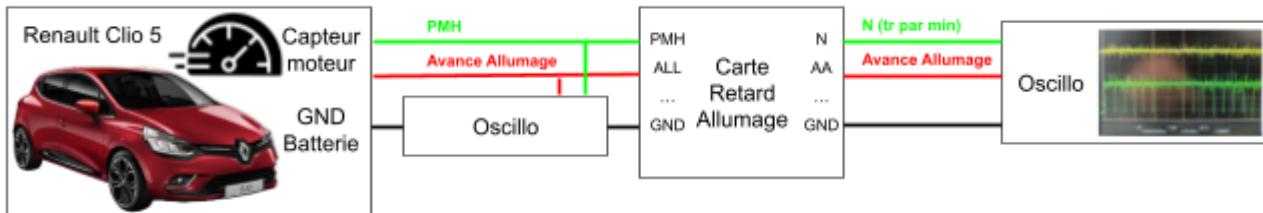
PMH

Avance

## RAPPORT SAE 5

### Test Retard Allumage au CIFAM

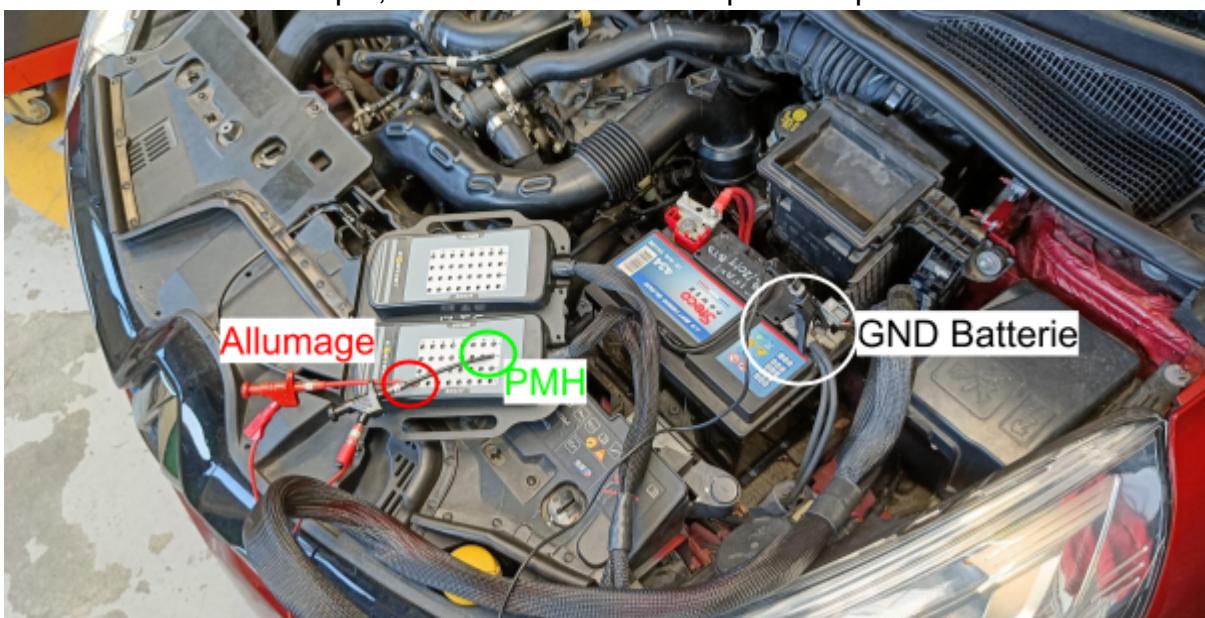
Montage réalisé au CIFAM



#### ➤ Acquisition des Signaux sur Différents Véhicules au CIFAM

À plusieurs reprises, nous nous sommes rendus au CIFAM pour mener des essais sur divers véhicules, dans le but d'observer les signaux du PMH et de l'injection à l'aide de notre carte de retard à l'allumage. Ces tests ont mis en évidence des variations significatives de tensions entre les différents modèles de voitures testés.

Pour obtenir des mesures précises et spécifiques, nous avons opté pour la méthode consistant à brancher nos sondes sur un bornier de mesure de la 5. Ce bornier est connecté au faisceau de la voiture, ce qui nous a permis d'obtenir, à l'aide de la documentation technique, les bornes des capteurs qui nous intéressaient.



#### ➤ Choix d'un Véhicule pour des Mesures

Suite à nos observations et afin de garantir des mesures stables et représentatives pour affiner le traitement des signaux de la carte, nous avons pris la décision de sélectionner spécifiquement une Clio 4 rouge de Renault comme véhicule de test principal au CIFAM. Cette décision a été prise après accord sur le choix de ce véhicule, s'alignant ainsi sur une méthodologie consensuelle pour obtenir des données cohérentes et exploitables.

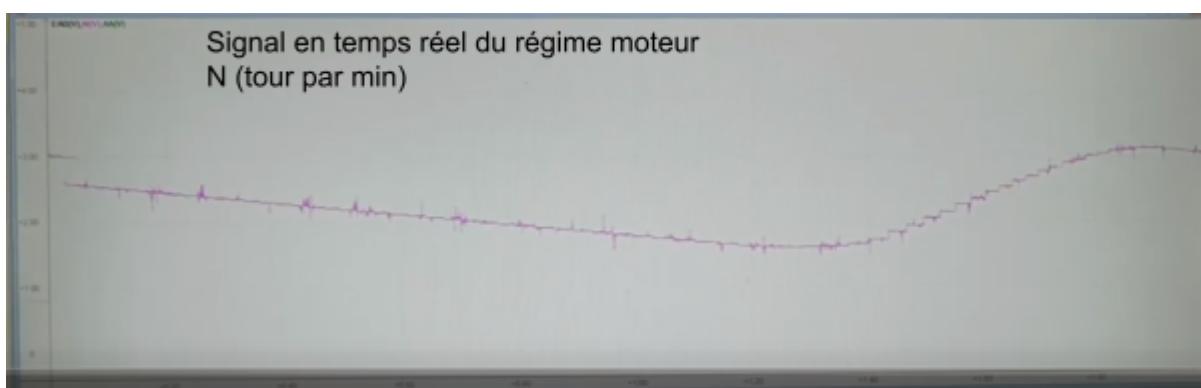
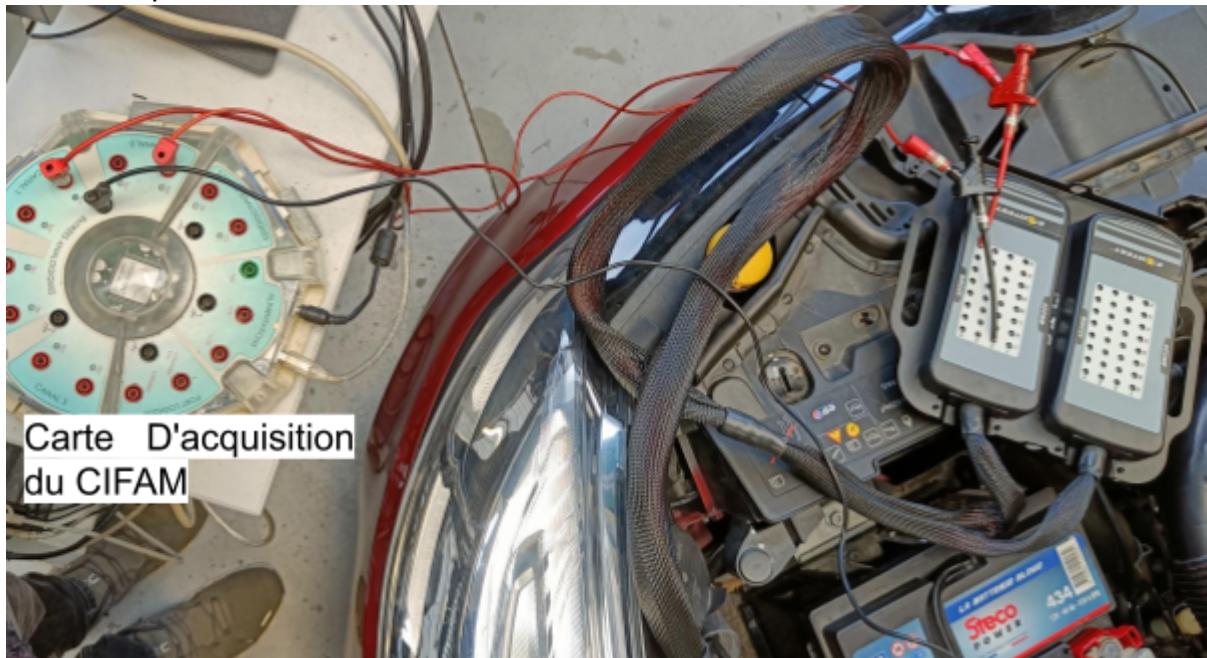
## RAPPORT SAE 5

### Adaptation des Composants suite aux Observations

Les tensions relevées sur la Clio 4 étaient comprises entre 0 et 5V, ce qui différait considérablement de nos attentes de 0 à 12V pour les composants initialement dimensionnés. Pour évaluer l'impact de ces variations sur notre conception, nous avons procédé à des mesures des valeurs des résistances en entrée à l'aide d'un ohmmètre. [Maintenance de la carte](#)

### Interaction Directe avec le Véhicule pour des Observations Précises

Pour mieux comprendre le comportement des signaux dans des conditions réelles, nous avons interagi activement avec la Clio 4 choisie. Nous avons démarré le moteur et manipulé la pédale d'accélérateur pour observer les réponses des signaux générés par le véhicule et donc ainsi observer les signaux en temps réel avec la carte d'acquisition du CIFAM.



## RAPPORT SAE 5

### GLOSSAIRE

Terme Technique	Traduction
PMH	Points morts (état du piston )
AA	Avance à l'allumage
SAMD21	Un microcontrôleur (Composant électronique )
AOP	Amplificateur Opérationnel (Composant électronique )
I2C	Inter-Integrated Circuit : Une série de plusieurs équipements, maîtres ou esclaves, pouvant être connectés à un réseau bus.
Résistance de Tirage	Résistance permettant de fixer l'état d'une pin
DAC	Digital Analog Convertisseur : Convertisseur de valeur numérique vers analogique
Data sheet	Document de renseignement technique d'un composant électronique
CAO	Conception Assistée par Ordinateur
Routage	Le fait de lier les pattes d'un composant à un autre sur le logiciel de CAO
Gerber	Fichier permettant la fabrication d'une carte (renseignement d'emplacement, trou ...)
Connecteur cortex	Connecteur permettant de connecter un JLink (Programmateur et déboguer du code)
CMS	Composant Monté en Surface
THT	éléments électroniques dans le PCB à travers des trous pré-percés
Sérigraphie	Commentaire sur une carte électronique
Bibliothèque	Autre fichier permettant l'ajout d'autre fonction dans un programme
Duty cycle	Rapport cyclique

## RAPPORT SAE 5

## **CONCLUSION**

Au cours du premier semestre de notre projet de 3ème année en Génie Électrique et Informatique Industrielle (GEII), nous avons repris un projet déjà initié en 2017. Notre première étape a consisté à prendre en main le travail existant, à comprendre les enjeux et les attentes qui nous étaient présentés. L'objectif principal était de finaliser la partie logicielle de la deuxième carte, responsable du calcul du régime moteur et de l'avance à l'allumage. Nous avons également prévu de concevoir un boîtier pour sécuriser l'ensemble des cartes et de donner un aspect utilisable propre pour des étudiants et d'enfin faire une documentation complète pour leur utilisation.

À ce stade, des progrès significatifs ont été réalisés, notamment la résolution de certains aspects logiciels qui nécessitent encore des approfondissements. La conception du boîtier est planifiée pour la clôture du projet, une fois que les manipulations fréquentes des cartes ne seront plus nécessaires. Nous avons identifié comme point négatif de notre réalisation l'organisation entre les sessions à l'IUT et au CIFAM. Pour le second semestre, notre intention est de consacrer davantage de temps à la conception logicielle en dehors des séances, tout en maximisant notre présence au CIFAM pendant les périodes de séances.

De plus, nous avons pris plaisir à concevoir des solutions innovantes pour résoudre des problèmes, que ce soit sur le plan logiciel ou matériel. Cette démarche nous a permis d'apprécier le processus intellectuel d'analyse et de résolution des problèmes, en identifiant les défis et en élaborant des approches novatrices pour les surmonter.

## RAPPORT SAE 5

### **ANNEXES**

#### **Annexes 1 : Les composants utilisés**

Type	Quantité	Boitier	Valeur	Prix unité	Farnell
Condensateur	9	0805	100nF	0,06 €	
Condensateur	3	0805	10uF	0,12 €	
Condensateur	1	0805	10nF	0,14 €	
Résistance	8	0805	10KΩ	0,70 €	
Résistance	5	0805	100KΩ	0,05 €	
Résistance	4	0805	39KΩ	0,01 €	
Résistance	3	0805	330Ω	0,01 €	
Résistance	2	0805	4,6KΩ	0,19 €	
Diode	8	SOD123	diode (BAT43W-W)	0,50 €	3373133
Led	2	0805	rouge - vert	0,18 €	
U5, U6, U7, U9	4	SOT-23-5	Amplificateur (MCP6L91T)	0,74 €	1715865
U1 SAMD21	1	TQFP-32_7*7mm_P0.8mm	Microcontrôleur (SAMD21)	4,96 €	2409244
U2	1	SOT-223-3	Régulateur de tension (Id1117s33tr sot223)	0,23 €	2253484
U3	1	TSOP-16	DAC7578	12,53 €	3118638
U8	1	SOT-23	LM4040DELT-2.0	1,00 €	3993631
SW2	1		Boutons poussoir	0,60 €	
Connecteur Banane	2		connecteur entré/sortie	2,93 €	
J1	1	USBBBSRA	USB-B	1,63 €	
J3	1		connecteur cortex	0,10 €	

## RAPPORT SAE 5

## Annexes 2 : Schéma carte

