

Projet : Moniteur de charge

SAE S4 Alternants

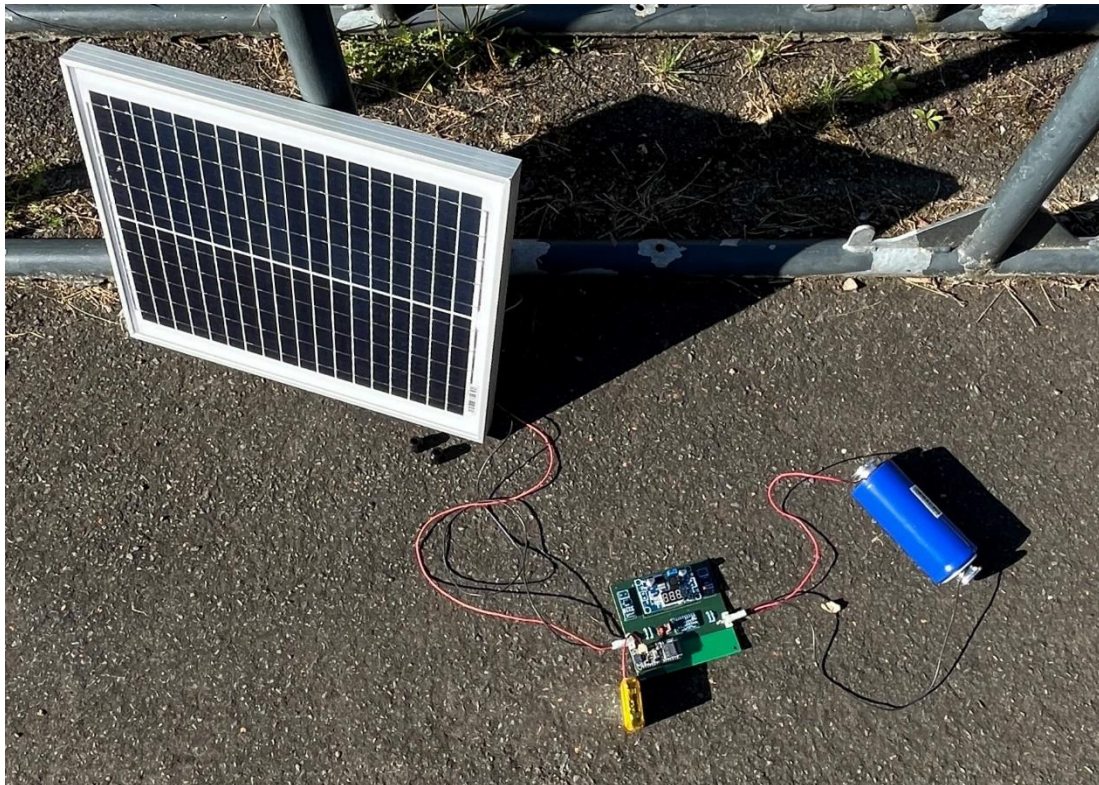


Table des matières

Introduction :	3
Sélection & étude des différents modules :	4
Les différents modules :	4
ACS712 :	4
ADS1115 :	5
Ponts diviseurs :	6
Schéma Structurel :	8
DC/DC :	8
Mesure du courant d'entrée :	8
Mesure du courant de sortie :	9
Module CAN :	9
ESP32 :	10
Génération du 5V :	10
Connecteurs d'entrée & sortie :	11
Routage :	12
Placement des composants :	12
Routage des pistes :	12
Plan de masse :	13
Corrections :	13
Données utiles :	14
Programme :	15
Tests :	18
Schéma fonctionnel :	18
Mesures :	19
Conclusion :	19
Annexe :	20

Introduction :

Les batteries lithium sont aujourd'hui la technologie de stockage d'énergie électrique la plus utilisée. Or, celles-ci sont connues pour être peu tolérantes aux températures négatives, cela réduit la capacité utilisable qu'elles contiennent.

L'une des alternatives les plus intéressantes pour contrer ce problème et qui se voit être de plus en plus étudié, sont les supercondensateurs (autrement appelés supercapa). En effet ici l'absence de procédé chimique permet une insensibilité aux variations de températures, ceux-ci résistent également bien mieux aux cycles de charge & décharge.

Afin d'étudier cela, il est important de pouvoir monitorer la charge en récupérant les valeurs de courant & tension en entrée ainsi qu'en sortie. Tout cela dans le but de surveiller la charge de supercapa, ici elles seront rechargées via un panneau solaire.

Pour se faire, la charge se fera avec un convertisseur de tension DC/DC (buck) qui transformera la tension d'entrée d'environ 21V en une tension de 2.7V pour une seule capa par exemple. Des mesures de tensions & de courant seront faites en entrée & en sortie afin de récupérer des valeurs mesurées.

Afin de faciliter la récupération des données, qui se feront via un module CAN ainsi qu'un ESP32. Nous avons décidé de les envoyer directement sur un serveur via le wifi du μ C. La solution d'hébergement choisi aura été Firebase, de Google.

Sélection & étude des différents modules :

Les différents modules :

Pour pouvoir superviser la charge de nos super condensateurs, nous devons réaliser 3 fonctions différentes : Transformer la tension, mesurer les tensions, et mesurer les courants.

Pour ce faire, nous utiliserons 3 modules.

- Un convertisseur Buck DC/DC
- Un convertisseur tension courant : L'ACS712
- Un convertisseur analogique numérique 16bits pour plus de précision : L'ADS1115

Enfin, le tout sera géré par un esp32 S3 (NO-PSRAM).

En effet, la tension de sortie du panneau solaire est de 21V et au maximum de 25V. Or, la tension de charge d'une super-capac est de 2,7V. Nous devons donc abaisser cette tension pour pouvoir charger nos super-capas.

Pour cela, nous utilisons un convertisseur/abaisseur Buck. Il est l'élément central de notre système.

De fait, l'objectif étant de superviser la charge de nos super-capas, nous devons mesurer les tensions et courant en entrée en sortie du Buck.

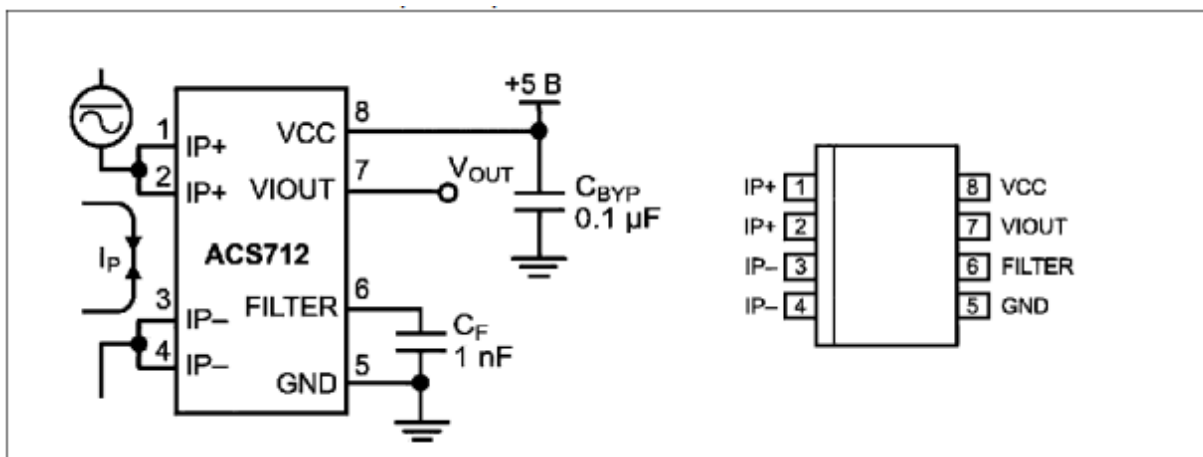
Nous allons étudier le fonctionnement de ces différents modules, et les intégrer dans le système.

ACS712 :

L'ACS712 est un convertisseur courant-tension. Il est branché en série pour mesurer le courant passant grâce à ses capteurs à effet hall, et retourne une tension image du courant mesuré.

Comme précisé précédemment, nous utiliserons deux modules ACS712 ; un en entrée et un en sortie du Buck.

Documentation et caractéristiques importantes :



Il est précisé dans la documentation du capteur que 185mV de tension différentielle correspond à 1A de courant passant.

De plus, pour éviter d'obtenir des tensions négatives en sortie, il y a un offset défini par sa tension d'alimentation ($VCC/2$). Ici, nous alimenterons nos capteurs en 5V, ce qui signifie que 2,5V sera équivalent à un courant nul (0A).

Il est important de vérifier la tension maximum que notre capteur pourra fournir en sortie, car l'ESP32 n'est capable de recevoir que 3,3V en entrée.

- Puissance max panneau = 30W $\Leftrightarrow$ Courant Max = 1.5A
- $VACS_{max} = 2.54(\text{offset}) + 1.5 * 0.185V \Leftrightarrow VACS_{max} = 2.8175V$
- Nous n'avons donc pas besoin d'utiliser un autre pont diviseur pour limiter la tension à 3.3V

ADS1115 :

Ce module ADS1115 est un convertisseur Analogique Numérique 16bits. L'utilisation de ce module nous permettra d'être plus précis que le CAN de l'esp32 intégré qui lui est en 12bits, et de plus présente une non-linéarité et un offset de quelques mV.

(Voir : <https://lucidar.me/fr/esp32/linearity-of-the-esp32-adc/>)

Il communique en I2C et possède plusieurs modes de fonctionnement : Simple conversion, mode différentiel, et un mode de comparaison renvoyant un flag en fonction de la configuration de celui-ci.

Nous utiliserons la simple conversion puisque dans notre cas, nous aurons besoin des 4 entrées analogiques (A0...A3). Voici un tableau des différentes connexions :

- PIN A0 : Sortie du Pont diviseur de tension de Vpanneau
- PIN A1 : Sortie du Pont diviseur de tension en sortie du BUCK (3.7V)
- PIN A2 : Sortie du module ACS712 (1)
- PIN A3 : Sortie du module ACS712 (2)



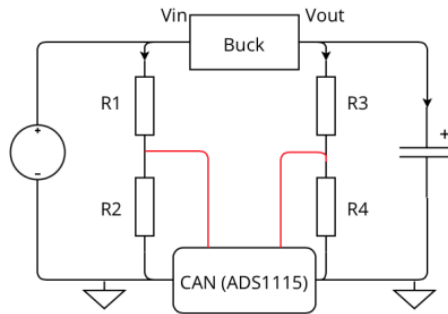
Toutes ces entrées analogiques seront converties en signaux numériques et envoyés à travers deux fils (SDR et SDA) à l'ESP32. Ce module nous permet non seulement d'être plus précis sur nos conversions mais il permet aussi de convertir jusqu'à quatre signaux en utilisant seulement deux fils.

Ponts diviseurs :

Cependant, comme pour l'ACS712, les tensions d'entrées ne doivent pas dépasser les 3,3V. Les pins A2 et A3 étant reliés aux deux modules ACS712, il n'y aura pas de risque de dépasser cette tension.

En revanche, les pins A0 et A1, eux sont connectés aux tensions d'entrée et de sortie qui sont toutes les deux supérieures à 3,3V.

Nous sommes donc obligés d'utiliser deux ponts diviseur de tension pour ramener ces tensions à 3,3V.



Nb : Nous calculerons les résistances en cherchant à obtenir 3V et non 3.3V pour être sûr de ne pas les dépasser.

- Pour Vin (Vpanneau) :
 - Tension max d'entrée = Vmax du panneau (on prendra 30V pour être sûr de ne jamais dépasser les 3.3V)
 - $V_s = V_e \times R_2 / (R_2 + R_1) \iff 3V = 30V \times R_2 / (R_2 + R_1) \iff R_1 = 9 \times R_2$
 - Il faut donc choisir deux résistances avec un rapport de 9 pour pouvoir obtenir 3.3V.
- Pour Vout (sortie du Buck) :
 - Tension max d'entrée = 2.7V (pour la charge d'une super-capac)

Dans le cas où nous avons plusieurs super-capac en série et donc une tension supérieure à 3.3V admissible par l'ESP32, il faudra mettre le pont diviseur suivant :

$$V_s = V_e \times R_3 / (R_3 + R_4) \iff 3V = 3.7V \times R_3 / (R_3 + R_4) \iff R_1 = 0.2 \times R_2$$

Sinon, il faudra court-circuiter les résistances, la tension de sortie étant à 2.7V ($2.7 < 3.3$)

Pour éviter toute consommation supplémentaire, nous utiliserons des résistances hautes.

Pour la tension d'entrée, nous utiliserons une résistance de 910k, et une de 100k.

Pour la tension de sortie, nous utiliserons une résistance de 200k, et une de 1M.

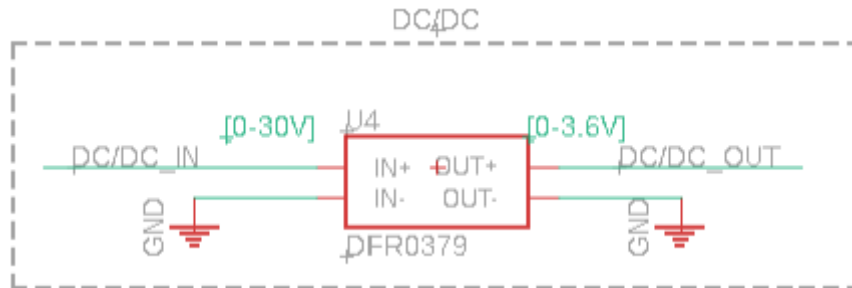
Calculs à partir de ces résistances :

- Pour Vin (Vpanneau) :
 - $V_s = V_e \times R_2 / (R_2 + R_1) \iff V_s = 30V \times 1000 / (1000 + 9100) \iff V_s = 2,97V$
 - Il faudra donc multiplier la tension mesurée par $30/2,97 = 10,1$.
- Pour Vout (sortie du Buck) :
 - Dans le cas évoqué ci-dessus où il y a plusieurs super capac, il faudra réaliser le même calcul qu'en entrée et mettre les résistances adaptées pour abaisser la tension de sortie

Après avoir sélectionné les différents composants utiles à nos modules, nous réaliserons la programmation des ACS712 ainsi que celle de l'ADS1115.

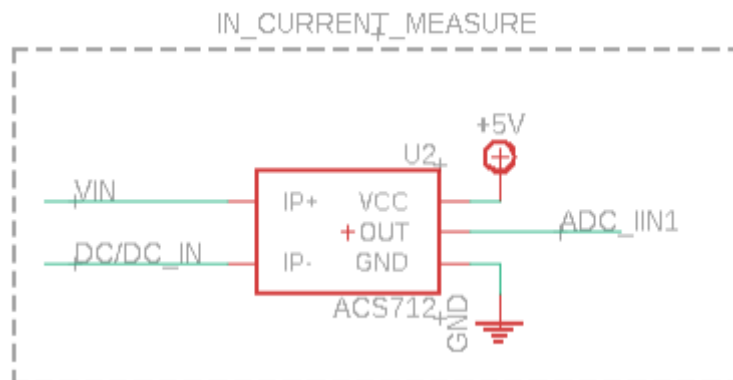
Schéma Structurel :

DC/DC :



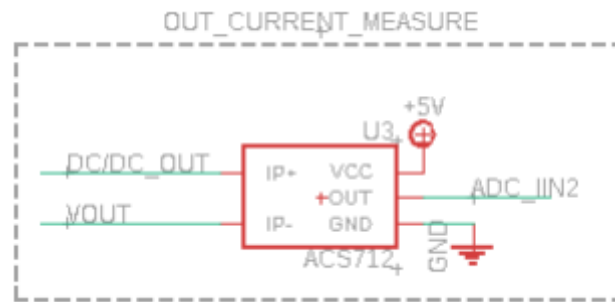
Le composant principal de notre carte électronique est ici le DC/DC (buck), il transformera la tension continue d'entrée provenant du panneau solaire [0-30V]. En une tension de sortie continue de 2.7V (sous réserve que l'entrée soit elle aussi à 2.7V au minimum, ce module buck ne faisant pas boost).

Mesure du courant d'entrée :



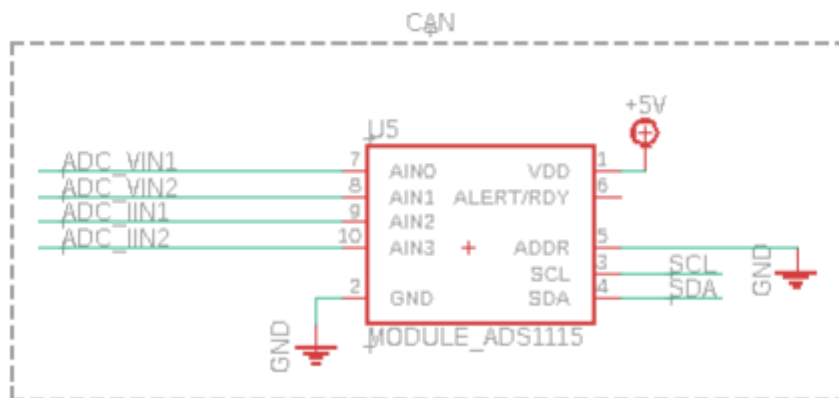
Afin de pouvoir monitorer notre charge, nous avons besoin de connaître la tension ainsi que le courant d'entrée (ici celui sortant du panneau solaire), comme cela a été dit dans l'écrit sur le choix de nos composants, nous avons sélectionné l'ACS712, un petit module intégrant un capteur à effet hall mesurant le courant en entrée du module, et sortant un signal de tension qui lui est proportionnel au courant mesuré. Cela permettra au module CAN de convertir cette tension en valeur numérique et au μC de pouvoir lire la valeur du courant mesuré.

Mesure du courant de sortie :



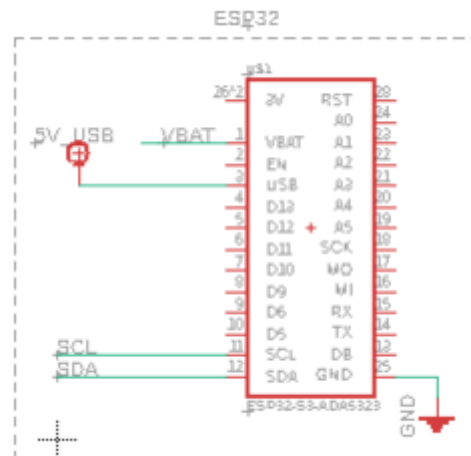
Dans le même objectif, nous avons besoin de mesurer le courant en sortie du buck (celui qui entrera dans les super-capac pour les charger). Pour cela nous plaçons donc un deuxième ACS712 en sortie du buck afin de mesurer le courant de charge et nous le redirigeons vers une deuxième broche du CAN ADS1115.

Module CAN :



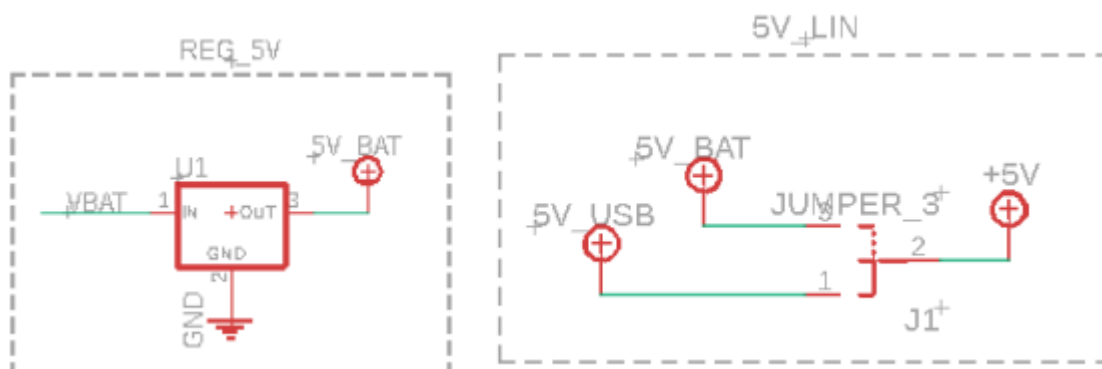
Dans notre application, le CAN (ADC1115) recevra les signaux analogiques de tension proportionnelles au courant d'entrée & sortie de notre système (mesurées par les deux ACS712) et les convertira en valeur numériques qui seront ensuite envoyées vers le μ C.

ESP32 :



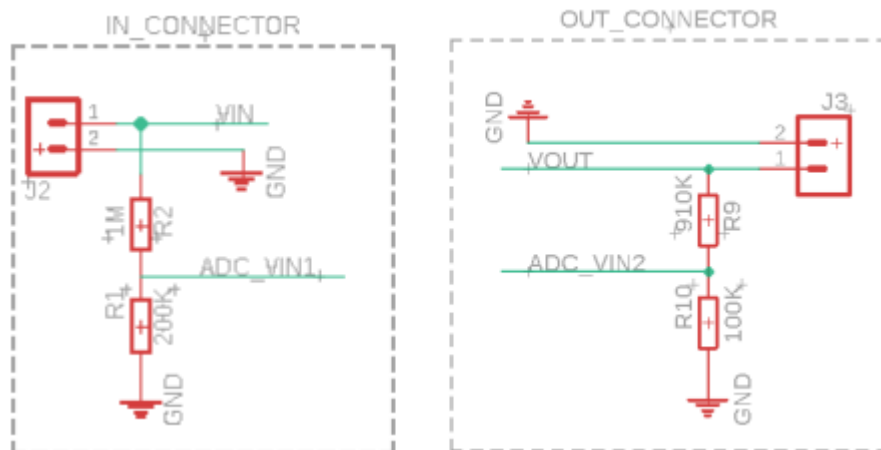
Le micro contrôleur sera ici chargé de centraliser toutes les mesures et de les traiter (traçage de courbes par exemple, ou tableau de mesures). Il liera les valeurs numériques de tension envoyées par l'ADC1115 et les interprétera à l'aide d'un coefficient multiplicateur pour qu'elles deviennent des valeurs de courant lisibles par tous.

Génération du 5V :



Afin d'alimenter les différents modules de notre carte électronique (2x ACS712, ADC1115, ESP32) nous devons faire venir du 5V sur celle-ci, sans pour autant récupérer cette tension sur notre tension de charge afin de ne pas empiéter sur celle-ci. Pour ce faire nous allons connecter une batterie accu (3.7V) à un petit module « buck boost », ce module permettra d'élever notre tension de 3.7V en une tension de 5V. L'ESP32 quant à lui pourra ensuite redistribuer ce 5V jusqu'au différents modules. Néanmoins, il est nécessaire de veiller à ce que la tension de 5V sortant du buck boost ne vienne pas rentrer en conflit avec celle redistribuée par l'ESP32, pour se faire nous avons mis en place un jumper avec de chaque côté nos deux tensions 5V, ici 5V_BAT & 5V_USB, qui seront ensuite commutés non simultanément en une tension appelée « 5V », cela évitera toute différence de potentielle pouvant endommager l'un de nos systèmes alimentés.

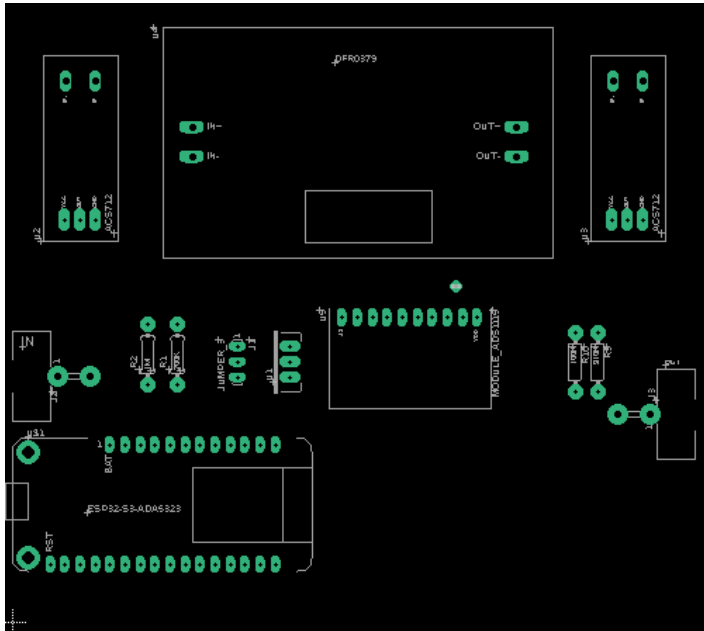
Connecteurs d'entrée & sortie :



Pour faciliter l'expérience utilisateur, nous avons décidé d'intégrer deux connecteurs afin d'éviter de connecter directement la source de charge (dans notre application le panneau solaire) & l'appareil chargé (ici les supercapa) sur le BUCK principal. Ici nous aurons juste à connecter la source de charge dans le connecteur « IN » et l'appareil chargé dans le connecteur « OUT ».

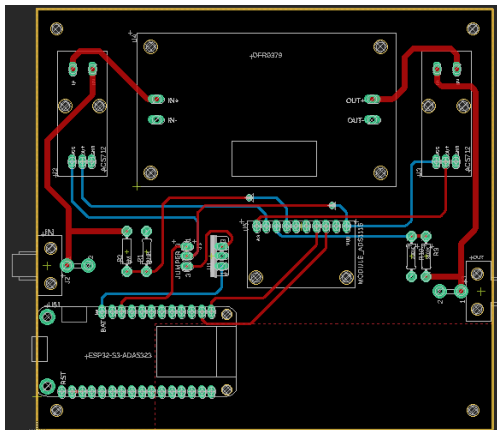
Routeage :

Placement des composants :



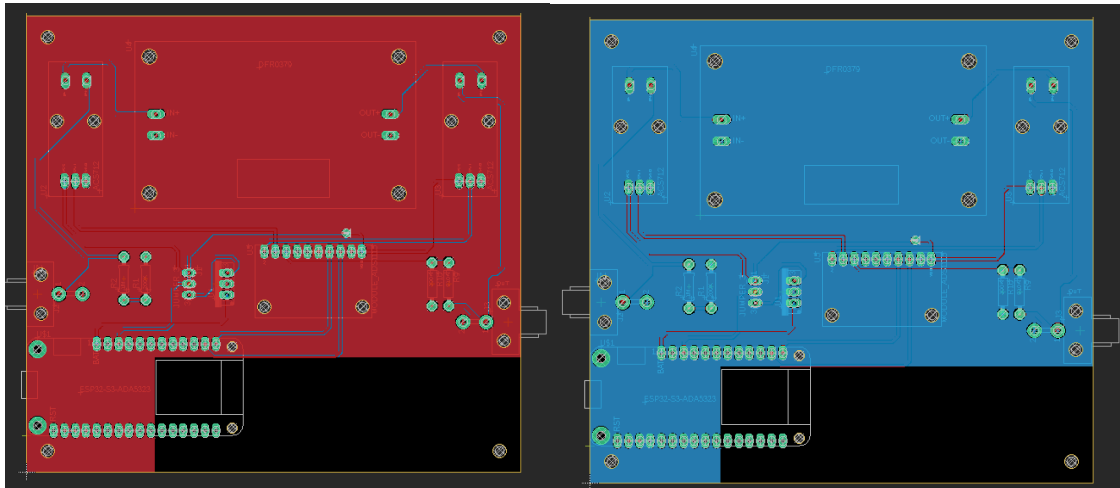
Les composants ont été placés de façon que la carte soit lisible aux premiers abords. Concrètement le module de mesure de courant d'entrée a été placé à gauche, celui de sortie à droite, les connecteurs d'entrée & sortie ainsi que le connecteur USB-C de l'ESP32 ont été placé de la même manière et en bord de carte afin de faciliter l'accessibilité de ceux-ci.

Routeage des pistes :



Pour le routage des pistes, celles-ci sont toutes en taille 0.5, exceptée les pistes de puissances qui ont été mises en 1.5 puisqu'elles feront passer le courant de charge. Le nombre de via a été réduit au maximum, il n'y en a seulement un sur la carte. Les GND eux ne sont pas reliés par des pistes car ils le seront par le plan de masse visible ci-dessous.

Plan de masse :

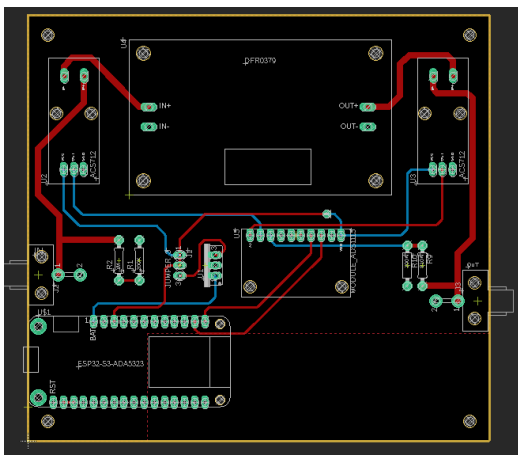


Ici il était nécessaire de veiller à ce que le plan de masse ne passe pas sur l'antenne wifi de l'ESP32 afin de s'assurer du bon fonctionnement de celle-ci, il en était de même pour les pistes & via.

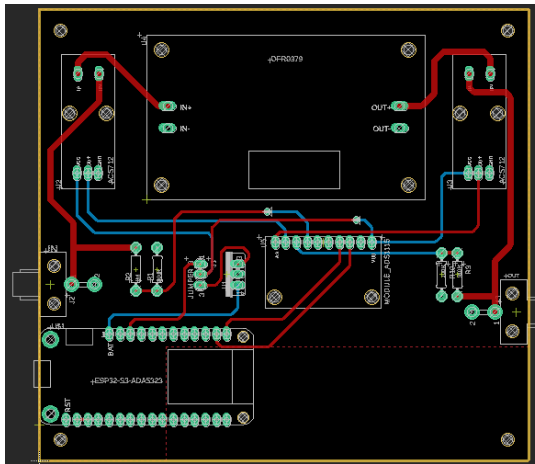
Corrections :

Lors de la réception de la carte, nous nous sommes aperçus qu'il y avait un problème, lorsque l'ESP32 était alimenté par la batterie le système ne fonctionnait plus. Le souci provenait du jumper sélectionnant la source d'alimentation 5V, une erreur dans l'empreinte du composant nous avait fait relier les mauvaises pistes entre-elles, la commutation avec le 5V_USB était donc non fonctionnelle. Nous avons également fait une autre petite erreur en oubliant de relier le second pont diviseur jusqu'au module CAN, la mesure de courant en sortie ne se faisait donc pas.

Nous avons corrigés ces deux soucis et avons fait ré imprimer un PCB, nous avons ensuite dessoudé les modules de l'ancienne carte et les avons soudés sur la nouvelle, ainsi les soucis ont été réglés.



Ancienne version du routage



Version corrigée du routage

Données utiles :

Taille du PCB : 107x116mm

Perçage (4 coins) : 3.1mm

Programme :

Afin de superviser la charge de nos super-capacités, l'objectif est de recevoir les mesures converties par l'ADS1115 puis de les traiter numériquement afin de les envoyer sur un serveur web.

Dans le cadre de notre SAE, nous utiliserons un service Google facile d'utilisation appelé Firebase pour stocker des données sur un site en ligne.

La première étape a donc été de réaliser une fonction permettant de convertir ses valeurs numériques en valeurs de tension « réelles » à l'aide des documentations techniques et des coefficients des ponts diviseurs.

```
void measureAndPrint() {
    u0 = ads.readADC_SingleEnded(0); // collect voltage value on PIN0
    tension0 = ads.computeVolts(u0) * 6.06;
    /* Serial Print For Debug */
    Serial.print("Tensions entrée : ");
    Serial.print(tension0);
    Serial.print("V          ");

    u1 = ads.readADC_SingleEnded(1); // collect voltage value on PIN1
    tension1 = ads.computeVolts(u1);
    /* Serial Print For Debug */
    Serial.print("Tension Sortie : ");
    Serial.print(tension1);
    Serial.println("V");

    u2 = ads.readADC_SingleEnded(2); // collect voltage value on PIN2
    tension2 = ads.computeVolts(u2);
    IE = (tension2 - 2.54) / SENSIBILITY; // convert to a current value
    /* Serial Print For Debug */
    Serial.print("Courant entrée : ");
    Serial.print(IE);
    Serial.print("A          ");

    u3 = ads.readADC_SingleEnded(3); // collect voltage value on PIN3
    tension3 = ads.computeVolts(u3);
    IS = (tension3 - 2.54) / SENSIBILITY; // convert to a current value
    /* Serial Print For Debug */
    Serial.print("Courant sortie : ");
    Serial.print(IS);
    Serial.println("A");
}
```

Nous retrouvons donc quatre variables u1, u2, u3, et u4, qui contiendront les valeurs envoyées par le CAN. Ces valeurs seront converties en tension grâce à la fonction `ads.computeVolts(un)`.

Pour les courants, un simple calcul est réalisé en prenant en compte l'offset de 2,54V (2,54 et non 2,5, suite à des mesures expérimentales), ainsi que le coefficient 0,185 (ici définit plus tôt dans le code dans « SENSIBILITY »).

Puis, pour un premier débogage, nous afficherons ces valeurs dans le terminal série de l'IDE Arduino.

Deux autres fonctions seront définies pour initialiser le serveur web et pour récupérer l'heure actuelle en interrogeant un serveur dédié.

```

void initFirebase() {
    config.api_key = API_KEY;
    auth.user.email = USER_EMAIL;
    auth.user.password = USER_PASSWORD;
    Firebase.begin(&config, &auth);
}

```

NB : Les identifiants du serveur Firebase seront définis plus tôt dans le code.

```

void printLocalTime() {
    struct tm timeinfo;
    if (!getLocalTime(&timeinfo)) {
        Serial.println("Failed to obtain time ");
        return;
    }
    Serial.println(&timeinfo);
    Serial.println(mktime(&timeinfo));
    content.set("fields/Time/doubleValue", mktime(&timeinfo));
    strftime(timeStringBuff, sizeof(timeStringBuff), "%A, %B %d %Y %H:%M:%S", &timeinfo);
    String asString(timeStringBuff);
    content.set("fields/Date/stringValue", asString);
}

```

Enfin, dans notre boucle générale du programme, nous utiliser ces différentes fonctions pour envoyer les données récupérées sur le serveur Firebase.

```

void setup() {
    ads.begin();
    ads.setGain(GAIN_ONE);
    Serial.begin(9600);
    delay(100);

    connectToWiFi();
    initFirebase();
    initTime("CET-1CEST,M3.5.0,M10.5.0/3");

    timeClient.begin();
    timeClient.setTimeOffset(3600);

    printLocalTime();
}

```

```

void loop() {
  measureAndPrint();

  if (Firebase.ready() && (millis() - dataMillis > 60000 || dataMillis == 0)) {
    dataMillis = millis();

    String documentPath = "measure/"; // create a field name measure

    content.set("fields/VoltageInput/doubleValue", tension0); // Create a
    doubleValue wich contains tension0
    content.set("fields/CurrentInput/doubleValue", IE); // Create a doubleValue
    wich contains IE
    content.set("fields/VoltageOutput/doubleValue", tension1); // Create a
    doubleValue wich contains tension1
    content.set("fields/CurrentOutput/doubleValue", IS); // Create a
    doubleValue wich contains IS
    printLocalTime();

    if (Firebase.Firestore.createDocument(&fbdo, FIREBASE_PROJECT_ID, "",
    documentPath.c_str(), content.raw()))
      Serial.printf("ok\n%s\n\n", fbdo.payload().c_str());
    else
      Serial.println(fbdo.errorReason());
  }
}

```

Les fonctions « content.set(...) » nous permettent de créer un des champs contenant des valeurs (entier, double, timestamp value, etc...).

Tous ces champs seront insérés dans un chemin nommé « measure » créée juste au-dessus.

La fonction printLocalTime nous permettra aussi d'insérer un champ contenant l'heure de la mesure ainsi qu'une valeur timestamp au format UNIX qui correspond au nombre de seconde écoulées depuis le 1^{er} Janvier 1970.

Voici un exemple de mesures envoyées sur le serveur Fireabase :

> measure > UoeircCetQmwK. Plus de fonctionnalités dans Google Cloud 		
(default)	measure	UoeircCetQmwKCIH7S3x
+ Commencer une collection	+ Ajouter un document	+ Commencer une collection
measure >	UoeircCetQmwKCIH7S3x >	+ Ajouter un champ
		CurrentInput: 0.68581 CurrentOutput: 0.66351 Date: "Friday, June 07 2024 09:28:40" Time: 1717745320 VoltageInput: 3.24665 VoltageOutput: 1.65563 (chiffre)

Tests :

Schéma fonctionnel :

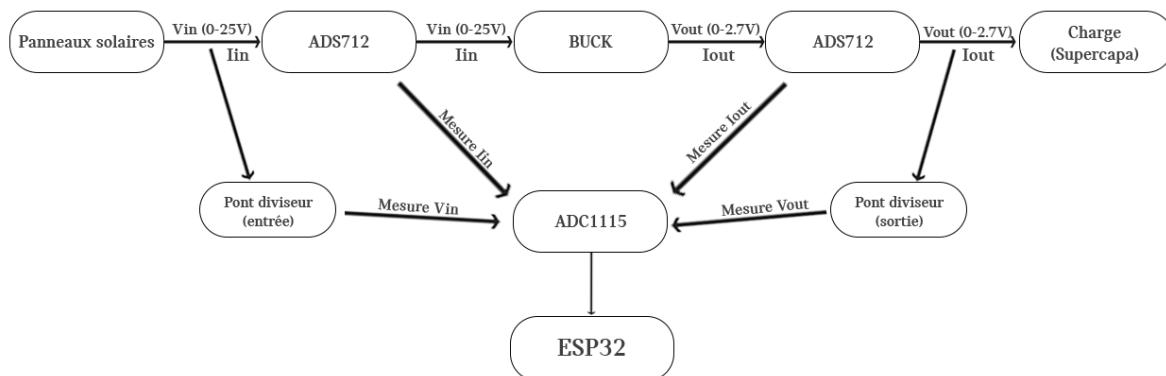


Schéma fonctionnel du système

Une fois les tests des différents modules ainsi que du code effectué sur la breadboard, avec une résistance en sortie du buck en tant que charge pour remplacer la super-capacitor, nous avons décidé de remplacer cette charge par une super-capacitor afin de faire quelques mesures de courant & tension quand celle-ci serait en train de charger.

NB : Il a été observé lors des tests préliminaires avec le buck, le panneau solaire ainsi qu'une super-capacitor que lorsque la tension de la super-capacitor est inférieure à 2.7V, nous observons une chute de tension en entrée du système.

L'alimentation stabilisée mise en entrée afin de simuler la tension que délivrerait le panneau solaire, fait chuter sa tension et n'envoie donc que 3.8V en entrée pour un courant limité à 1A. Cela n'empêchant pas de charger la super-capacitor, et une fois celle-ci chargée à la tension de 2.7V, l'alimentation pourra délivrer les 20V et continuera sa charge.

Mesures :

Une fois la super capa câblée en sortie de notre montage, une mesure de courant a été effectuée en entrée du buck, celui-ci était de 1A, valeur que l'on retrouve dans le terminal Arduino grâce aux mesures de l'ACS712 d'entrée & de la conversion effectuée par l'ADC1115.

En revanche, en ce qui concerne la mesure du courant de sortie affichée dans le terminal [0.04A], elle n'est pas cohérente avec ce que l'on devrait obtenir. L'ACS712 étant un composant mesurant le courant et le convertissant en une tension de sortie reflétant le courant mesuré (tension proportionnelle à ce courant mesuré), une mesure de tension a donc été faite en sortie de l'ACS712 de sortie, celle-ci affichant 2.49V, soit 0A (l'ACS712 mesurant des courant négatifs & positifs, 2.49V étant son point milieu, soit un courant nul). Cette mesure n'étant pas cohérente, cela peut laisser penser que celle-ci soit erronée.

Après vérification, nous nous sommes assurés que le souci ne venait pas du code, cela venait donc du câblage, l'ACS712 de sortie qui est donc relié à la super capa, était câblé en parallèle alors qu'une mesure de courant se fait évidemment en série., cela justifie la mesure de courant nulle observée. Le câblage a été rectifié et le module de mesure de courant a donc été câblé en série, voici donc les mesures obtenues après rectification. Le terminal Arduino renvoie ici 0.90A, ce qui est une valeur tout à fait cohérente, la mesure a été confirmée en mettant le multimètre en série en sortie du montage (position ampèremètre).

En ce qui concerne les mesures de tensions, elles n'ont-elles pas posées de soucis, la tension de 21V réglée sur l'alimentation en entrée est bien celle mesurée par le premier pont diviseur. En sortie, on retrouve bien notre tension de charge, ici réglée à 2.7V.

Conclusion :

Les principaux objectifs que nous nous étions fixés en début de projets ont été accomplis.

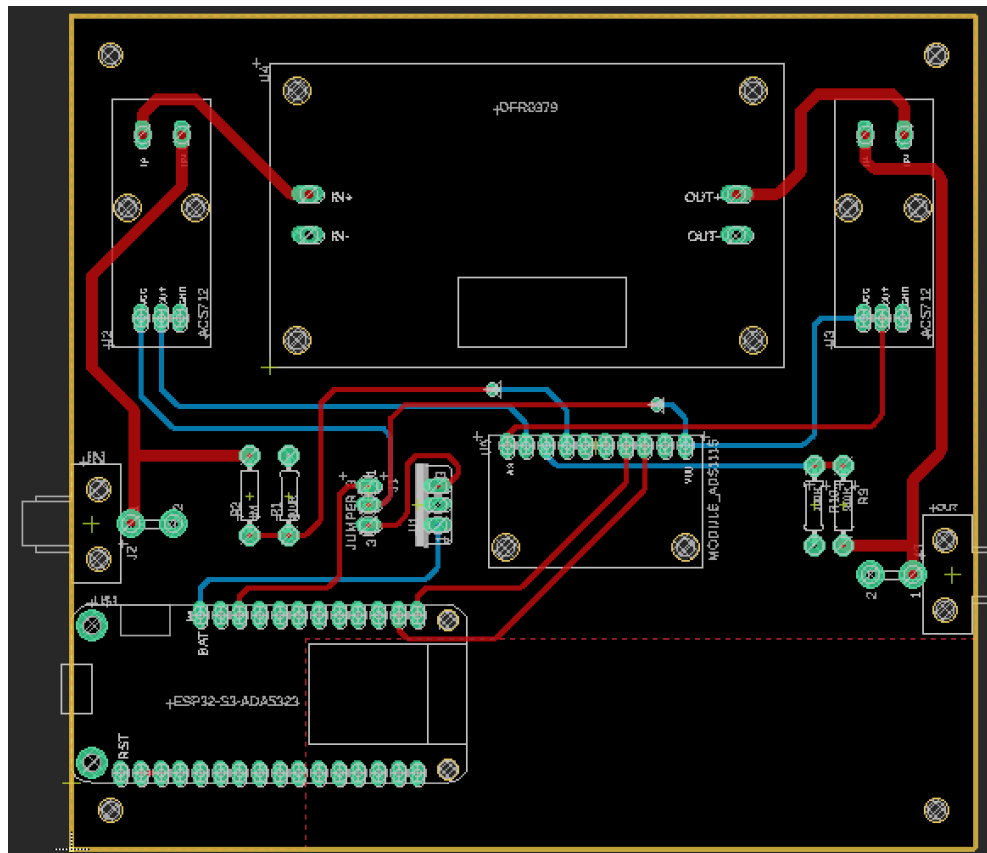
La charge s'effectue sans embuches et nous récupérons bien les mesures de courant & de tension d'entrée & sortie sur un serveur afin de pouvoir les lire à distance.

Néanmoins, nous pouvons évidemment voir des pistes d'améliorations, ouvrant d'éventuels perspectives pour de futurs repreneurs du projet. Premièrement l'intégration de l'affichage de l'heure sous la forme timestamp (nombre de secondes depuis le 1^{er} janvier 1970)

Également, nous pourrions envisager l'envoi des mesures que nous récupérons dans un fichier Excel afin de pouvoir tracer des courbes par exemple. Cela permettrait d'avoir un rendu plus visuel de la charge, pouvant ainsi mieux détecter d'éventuels anomalies.

Pour finir, la conception & fabrication d'un boîtier pourrait en effet permettre d'avoir un produit mieux fini, dégageant ainsi une meilleure impression et un sentiment de qualité en voyant le produit. Mais aussi et surtout afin de pouvoir éventuellement étanchéifier ce boîtier afin de faciliter l'intégration de notre système en extérieur, cela en le laissant tourner en autonomie, ainsi qu'en récupérant les valeurs des mesures à distance.

Annexe :



Routage complet