Hi-Rel Electronics

**Mesure courant-tension**

Rapport de laboratoire

Master HES-SO

Émilie Gsponer, Yann Maret

2 Mars 2016

version 1.0

Table des matières

[1 Choix du capteur 3](#_Toc451244959)

[1.1 Principes physiques disponibles 3](#_Toc451244960)

[1.2 Principe choisi 3](#_Toc451244961)

[2 Schéma bloc de la chaîne de mesure 3](#_Toc451244962)

[3 Simulation de la mesure de courant 4](#_Toc451244963)

[3.1 Charge variable 5](#_Toc451244964)

[3.2 Décharge de la batterie 6](#_Toc451244965)

[3.3 Courant sinusoïdal 7](#_Toc451244966)

[4 Budget de la chaîne de mesure 8](#_Toc451244967)

[4.1 Liste des composants 8](#_Toc451244968)

[4.2 Budget de masse 8](#_Toc451244969)

[4.3 Budget d’espace 8](#_Toc451244970)

[4.4 Budget de consommation 8](#_Toc451244971)

[5 Dissipation de chaleur 9](#_Toc451244972)

[6 Analyse Monte Carlo 11](#_Toc451244973)

[7 Simulation VHDL du convertisseur A/D 12](#_Toc451244974)

[7.1 Simulation du convertisseur A/D 13](#_Toc451244975)

[7.1.1 Modification de l’entité 13](#_Toc451244976)

[7.1.2 Adaptation de convertisseur pour une lecture par commande 13](#_Toc451244977)

[7.1.3 Timing à respecter 14](#_Toc451244978)

[7.1.4 Simulation de l’ADS1282 15](#_Toc451244979)

[7.2 Simulation du convertisseur piloté par le bus AMBA 16](#_Toc451244980)

[7.2.1 Modification de l’entité 16](#_Toc451244981)

[7.2.2 Modification du contrôleur 16](#_Toc451244982)

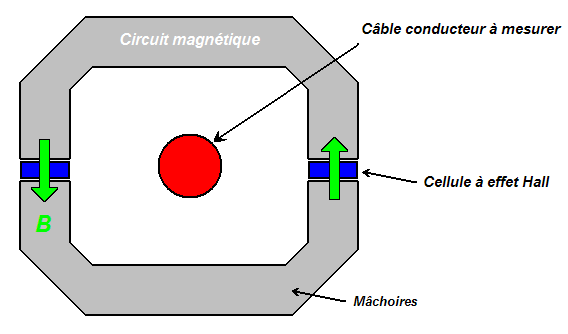
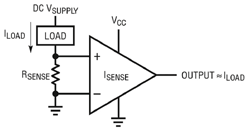
[7.2.3 Amélioration de la machine d’états 17](#_Toc451244983)

[7.2.4 Codage de Hamming 19](#_Toc451244984)

[7.2.5 Simulation du circuit 20](#_Toc451244985)

# Choix du capteur

## Principes physiques disponibles

* Capteur à effet Hall :
  + Lorsqu'un courant traverse un barreau en matériau semi-conducteur (ou conducteur), et qu'un champ magnétique d'induction B est appliqué perpendiculairement au sens de passage du courant, une tension, appelée tension Hall, proportionnelle au champ magnétique et au courant apparaît sur les faces latérales du barreau.[[1]](#footnote-1) [[2]](#footnote-2)
* Mesure Shunt :
  + La mesure de courant par résistance Shunt consiste à insérer une résistance de valeur connue en série avec l’alimentation et à mesurer la tension à ses bornes par mesure différentielle. La loi d’ohm permet de connaître le courant traversant la résistance.[[3]](#footnote-3)

## Principe choisi

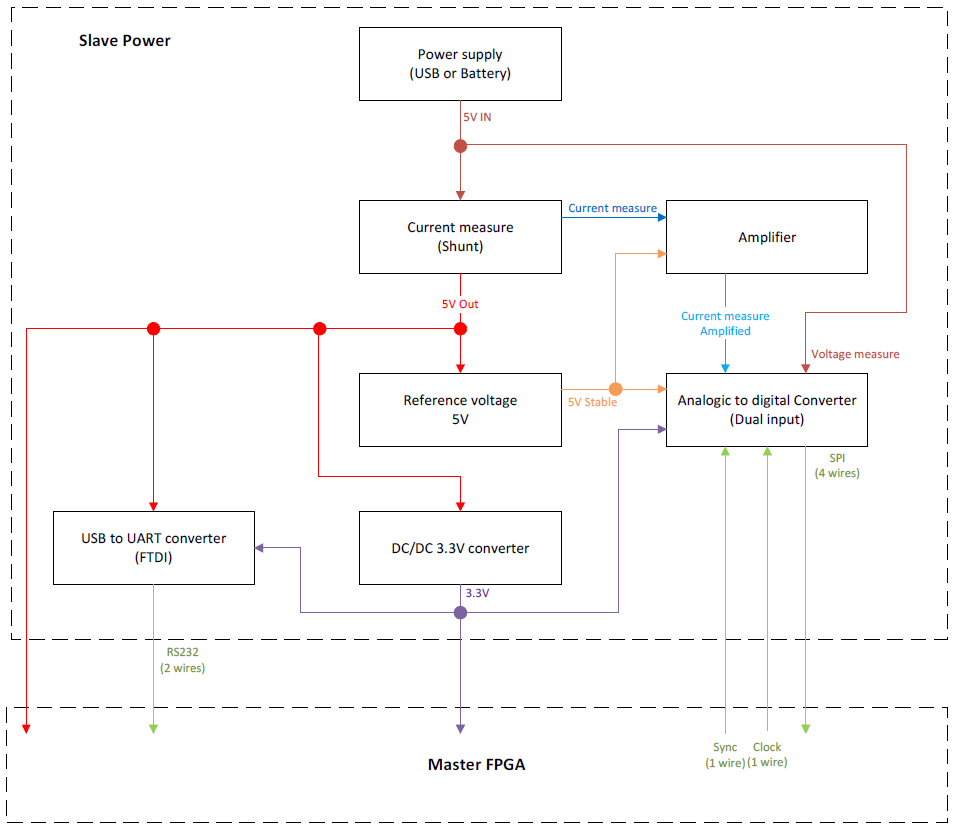
Nous avons décidé de choisir la **mesure shunt**, car nous avons jugé les capteurs à effet hall trop volumineux et non pratiques, car il faut que le fil d’alimentation de la batterie passe à travers le capteur pour la mesure. De plus, avec ce capteur qui mesure à l’aide d’un champ magnétique, il y a plus de chances de mesurer des perturbations qu’avec le principe choisi.

En plus du courant, notre carte devra également mesurer la tension d’alimentation.

# Schéma bloc de la chaîne de mesure

Notre système de mesure devra se placer directement après la source d’alimentation, que ce soit par batterie ou USB. Sans cela, notre circuit ne mesurera pas réellement le courant consommé. C’est pourquoi nous allons devoir refaire la carte *Slave power alimentation* afin d’y insérer notre circuit.

Afin d’avoir assez de lignes pour que le convertisseur A/D puisse communiquer avec la carte FPGA, nous avons décidé de supprimer le switch ainsi que les *user Leds* de la carte d’alimentation existante.

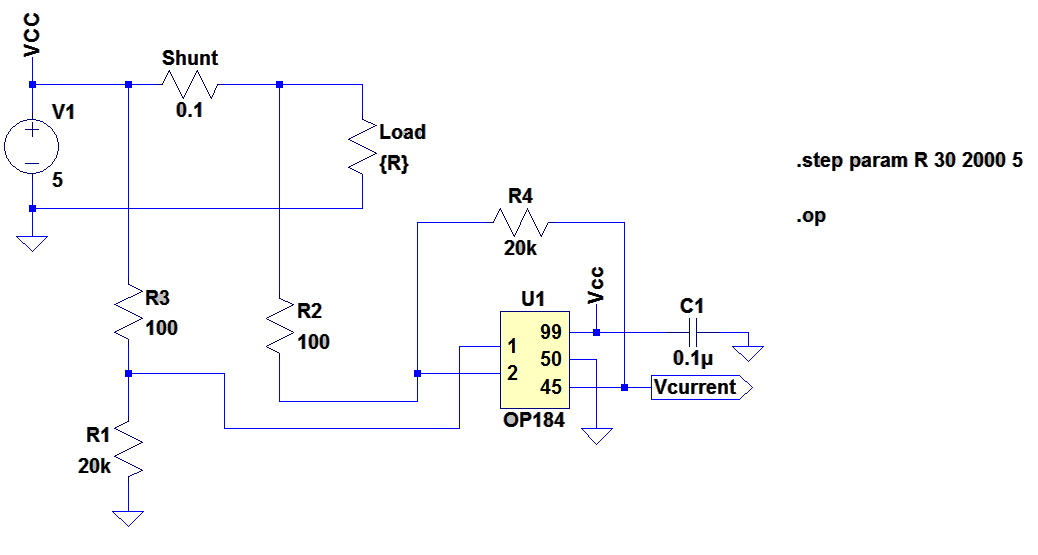


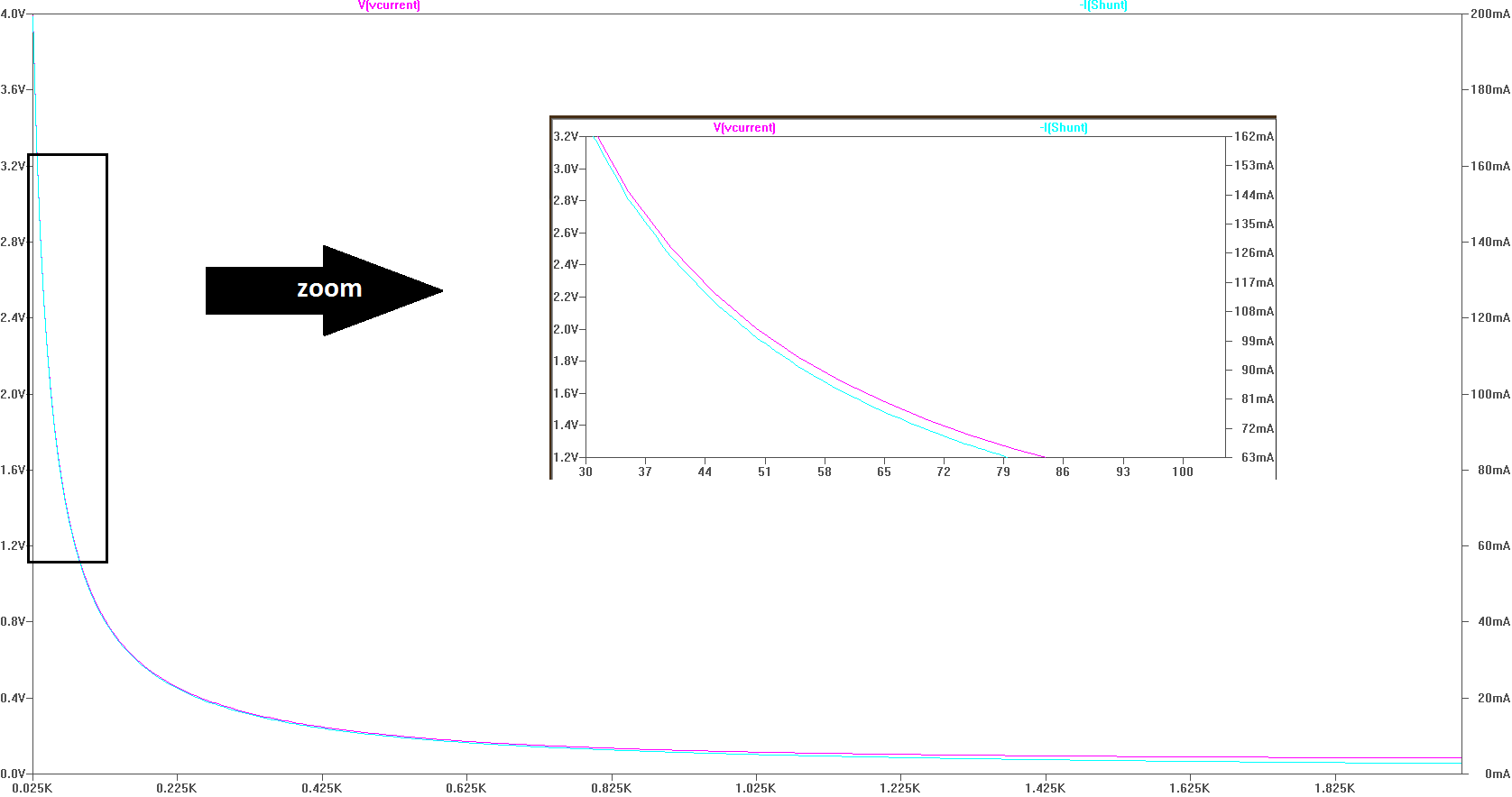
# Simulation de la mesure de courant

Avec le logiciel LTSpice, nous avons pu simuler notre circuit de mesure de courant dans plusieurs cas. Selon les spécifications, le courant consommé par l’ensemble du circuit est aux alentours de 130 mA. Nous avons volontairement gardé une marge, car on ne connait pas encore la consommation de l’amplificateur et du convertisseur A/D. La formule du gain est la suivante :

Nous avons dimensionné notre amplificateur avec un gain de 20, ce qui nous donne 4V à 200 mA. Comme la tension d’alimentation est de 5V et que l’ampli est rail-to-rail, on peut théoriquement mesurer jusqu’à 250 mA.

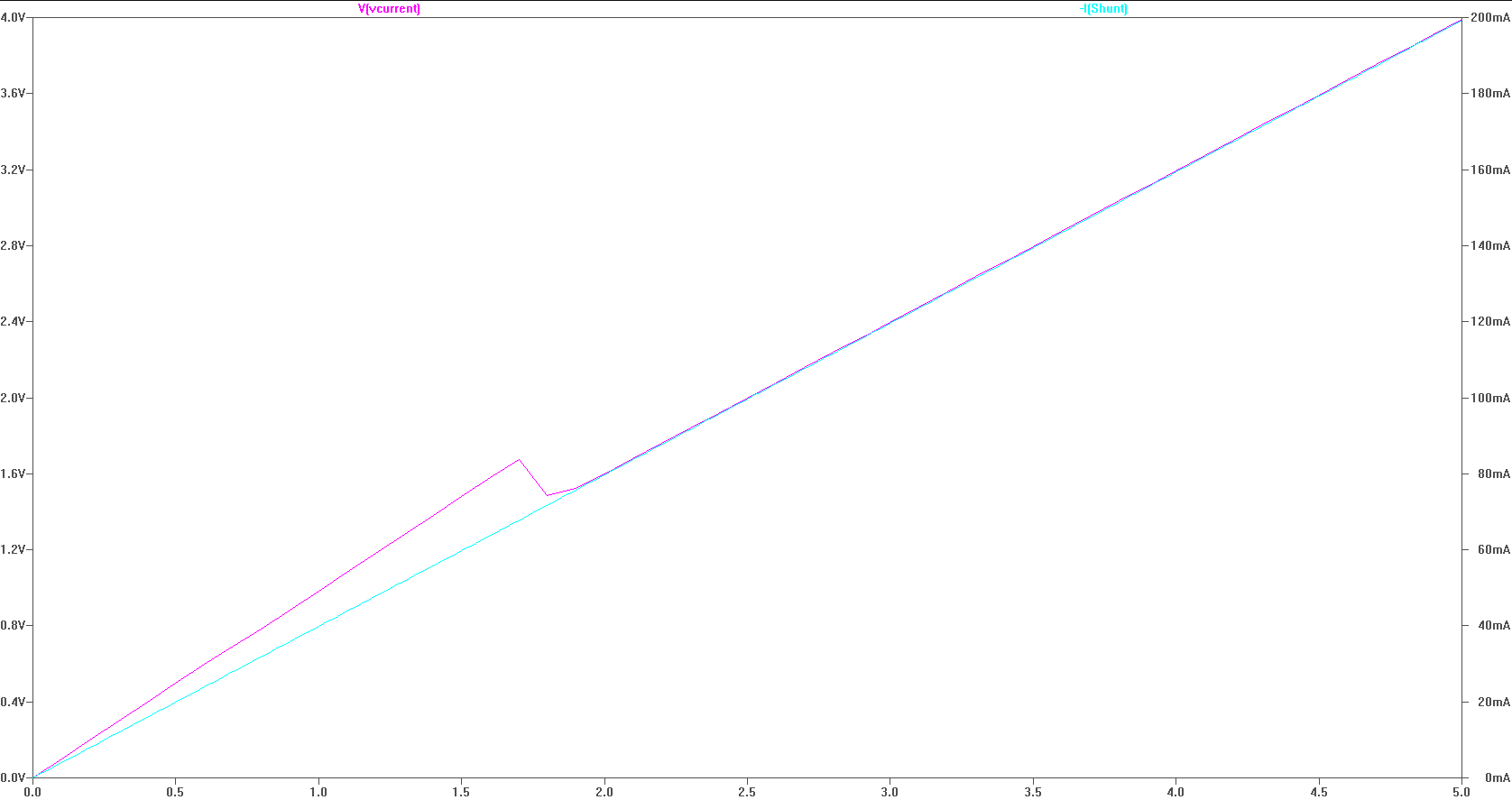
## Charge variable

Dans cette simulation, la charge du circuit varie de façon à avoir un courant consommé variable. Le but est d’observer si notre circuit suit correctement la courbe de courant.

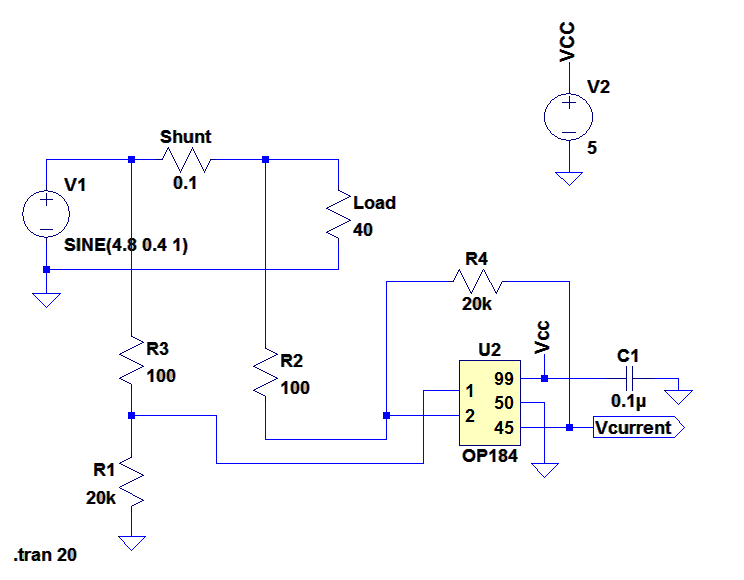
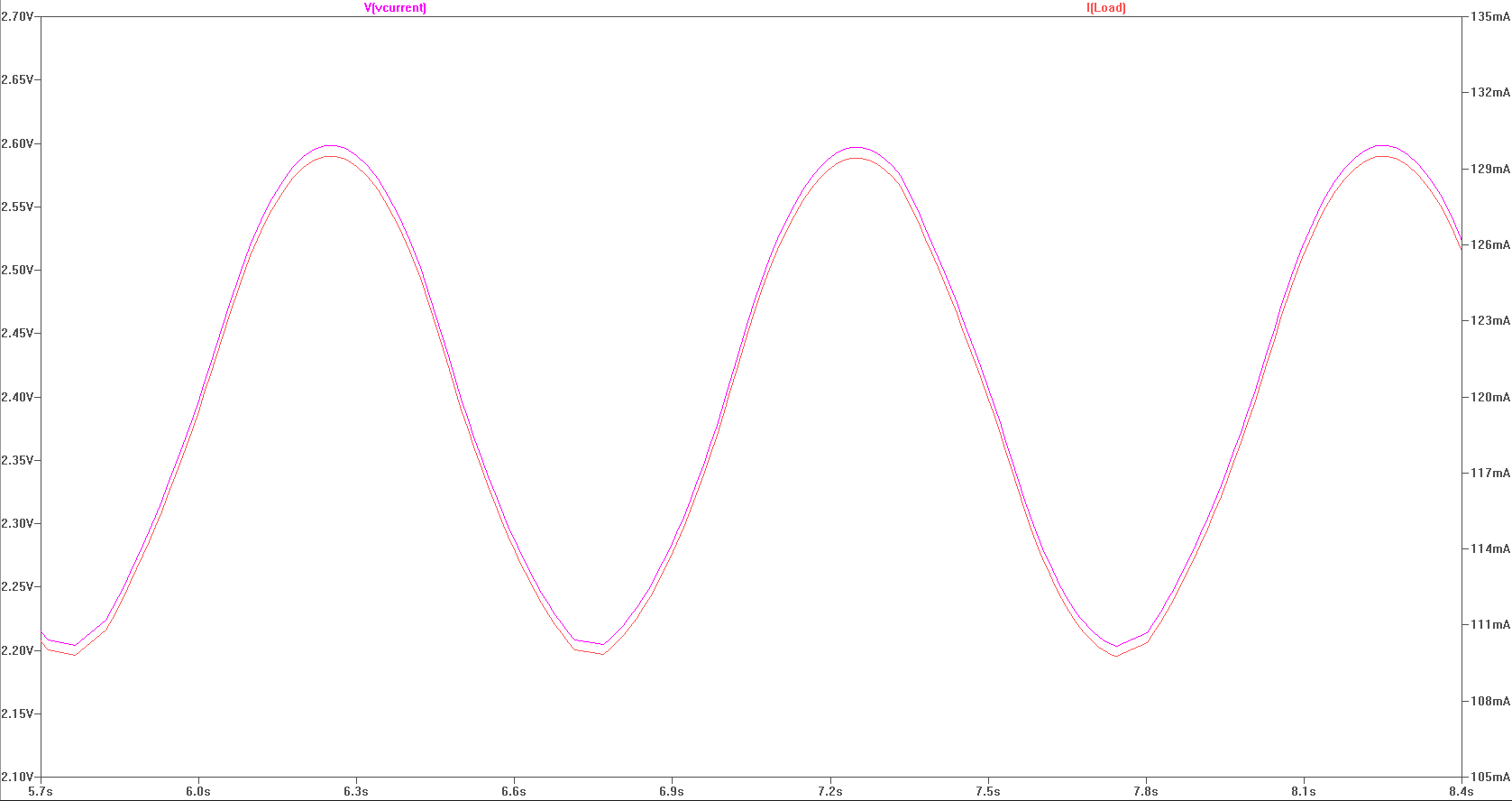
On constate que notre ampli op transforme correctement le courant du shunt en tension jusqu’aux alentours de 5 mA.

## Décharge de la batterie

Dans cette simulation, on observe le comportement du système lorsque la tension d’alimentation diminue.

On constate qu’il se passe un phénomène étrange en dessous de 2V, mais que le circuit mesure correctement au-dessus de ce seuil de tension d’alimentation.

## Courant sinusoïdal

Dans la mesure de courant qui nous a été fournie, on a pu observer que le circuit réel n’avait pas une consommation de courant stable, mais qu’elle oscillait entre 110 et 130 mA. Nous avons tenté de reproduire cette situation.

Cette simulation nous a permis de constater qu’il nous fallait absolument une référence stable de tension 5V pour alimenter notre amplificateur, sinon la mesure n’est pas correcte. Une fois que l’on a ajouté une tension stable sur l’ampli op, on voit dans le résultat que la mesure suit correctement l’oscillation du courant consommé.

C:\Users\Emilie\Documents\Git Hub\HiRel\Rapport\images\image_large.png[[4]](#footnote-4) Ces différentes simulations ont permis de valider le principe de mesure de courant choisi ainsi que le choix de l’amplificateur opérationnel.

# Budget de la chaîne de mesure

## Liste des composants

Avant de pouvoir commencer le budget, nous avons dû établir la liste des composants du circuit. Elle a été incluse dans le fichier Excel du budget sous le nom « Component ».

Nous avons essayé au maximum de choisir des composants respectant la plage de température en vigueur pour le spatial, soit -55°C à 125°C. Malheureusement, une grande partie des composants ne sont pas garantis pour ce plage de température, ils vont plutôt de -40°C à 85°C. Cela n’est pas problématique pour le CanSat qui doit être dimensionné pour passer très peu de temps dans l’espace. Mais si le circuit devait y rester longtemps, cela induirait un vieillissement prématuré des composants.

Remarque : Le FTDI avec ses deux LEDs et les autres petits composants allant autour pourront être enlever avant d’envoyer le CanSat dans l’espace. Ces composants sont uniquement utiles pour la programmation de la FPGA.

Avec tous nos composants, nous arrivons à un prix total de **265 CHF**. La plus grande partie du prix étant dédiée au convertisseur analogique-digital.

## Budget de masse

Nous n’avions pas fixé de contrainte concernant le poids de la carte. Nos composants ont un poids de **12 g**, ce qui est raisonnable. Avec l’estimation du poids du PCB et de la soudure, on passe à un poids total de **28 g**. Une autre estimation a été faite en calculant le poids en multipliant l’espace occupé par le composant avec l’épaisseur du PCB et sa masse volumique, on obtient un poids de **22 g.**

Il est difficile en l’état actuel du projet de savoir laquelle des deux estimations est la bonne.

## Budget d’espace

La taille du PCB nous est imposée, nous avons le droit à une carte double faces de 35mm sur 83mm. Cela donne deux surface (top et bottom) de 5'810 mm2. Bien sûr, une grande partie de cet espace est utilisé pour les pistes reliant les composants.

En additionnant l’espace pris par tous les composants, on atteint une surface de **3'146 mm2.** Cette valeur est approximative, car un ratio a été ajouté à chaque composant.

C:\Users\Emilie\Documents\Git Hub\HiRel\Rapport\images\image_large.pngOn peut raisonnablement s’attendre à ce que tous nos composants puissent être placés sur la carte et routés.

## Budget de consommation

La consommation du circuit avec la FPGA en fonctionnement nous a été donnée et est de 130mA. L’estimation de consommation de la carte d’alimentation a été estimée dans le pire des cas à **65mA** avec une puissance de **242mW.**

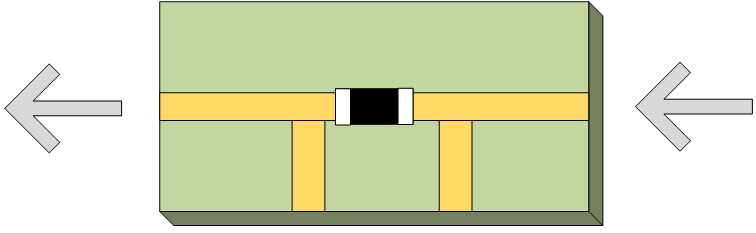
Nous avons également estimé la consommation de la carte d’alimentation actuelle à 50mA, notre circuit de mesure et de conversion analogique digital consomme donc 15mA. Avec ces informations on peut donc estimer la consommation de la carte FPGA à 130mA-50mA = 70mA et une puissance de 3.3V\*70mA = 231mW.

Si l’on met tous ces éléments ensembles, nous avons donc une consommation totale de l’alimentation et de la FGA de 65mA+70mA = **135mA** et une puissance de 242mW+231mW = **473mW**.

C:\Users\Emilie\Documents\Git Hub\HiRel\Rapport\images\image_large.pngLa résistance shunt que nous avons choisie pour la mesure du courant de toutes les cartes peut supporter une puissance maximale de 500 mW. Avec les calculs ci-dessus, on peut valider que la résistance est bien choisie.

# Dissipation de chaleur

Nous avons décidé faire notre analyse de chaleur sur la résistance shunt de notre circuit de mesure de courant. On analysera ce qui se passe d’un bord du PCB à l’autre si la résistance est placée au milieu de celui-ci.



Selon le datasheet, la résistance shunt est constituée à 96% d’aluminium, nous avons donc utilisé les propriétés de ce matériau pour les calculs. Nous avons également deux pistes de cuivre dont les dimensions ont été estimées et le poids calculé avec la masse volumique du matériau, soit ρ= 8.9mg/mm3. Pour la soudure, on a une masse volumique de ρ=5.77mg/mm3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Item** | **Material** | **Dimensions** | **Mass** | **Thermal conductivity [W/(mm⋅K)]** | **Thermal capacity [J/(K⋅g)]** |
| Résistance shunt | 96% Al | 2mm x 1.25mm x 0.65mm | 5.5mg | kAl = 0.237 | Cal = 0.897 |
| Piste cuivre | Cuivre | 2 x 40mm x 0.25mm x 35µm | 2 x 3.2mg | Kcu = 0.401 | Ccu = 0.38 |
| Contact piste-résistance | n/a | (taille pads résistance)  2 x 0.55mm x 1.25mm | n/a | n/a | n/a |

Avec ce tableau, nous pouvons calculer les résistances thermiques et capacitives.

Le calcul est le suivant :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Épaisseur** | **Surface** | **Résistance thermique** | **[K/W]** |
| Résistance shunt | 0.65mm | 2.5mm2 | Rshunt | 2 |
| Piste cuivre | 0.035mm | 20mm2 | Rcu | 0,005 |
| Résistance de contact piste-résistance (cuivre) | Rc ≈ 0.001 K⋅m2/W | 0.0014 m2 | Rcontact | 0,715 |

Nous pouvons également calculer les

Le calcul est le suivant : sse \* Thermal capacity

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Masse** | **Condensateur thermique** | **[J/K]** |
| Résistance shunt | 0.0055 g | Cshunt | 0.005 |
| Piste cuivre | 0.0064 g | Ccu | 0.003 |

Nous obtenons le schéma équivalent suivant :

# Analyse Monte Carlo

L’analyse est dans le fichier « AnalyseMonteCarlo\_Shunt.xls ». L’analyse a été faite sur la mesure shunt en analysant la formule du gain suivante :

Toute nos résistances ont une variation de ±1%. Nous avons donc entré les formules suivantes dans l’analyse :

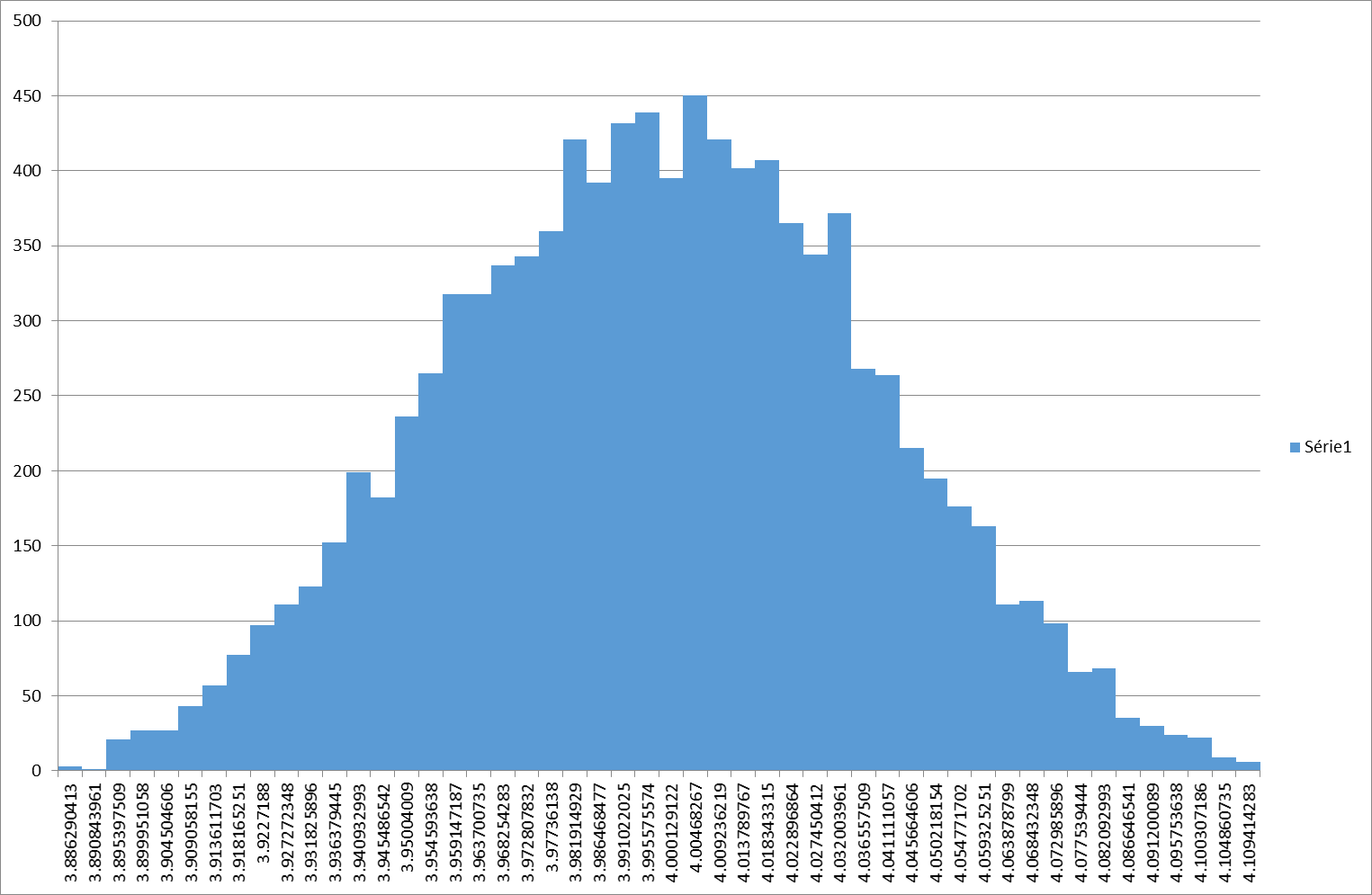
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rsense | R3 | R1 | Iload | Vcurrent |
| =0.1\*(1+0.02\*(ALEA()-0.5)) | =100\*(1+0.02\*(ALEA()-0.5)) | =20000\*(1+0.02\*(ALEA()-0.5)) | 0.2 | = Rsense \* Iload \* R1/ R3 |

La valeur de sortie de l’analyse est Vcurrent, soit la tension représentant le courant mesuré amplifié avec un gain de 20. Nous avons mis comme paramètre d’entrée Iload à 200mA. Avec ce courant, l’amplificateur devrait produire une tension de 4V.

En faisant 10'000 répétitions, l’analyse fourni les résultats suivants :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Rsense | R3 | R1 | Iload | Vcurrent |
| **10000** | 0.100287259 | 99.53533642 | 19966.74546 | 0.2 | 4.023516148 |
| Mean | 0.100001467 | 100.001792 | 20000.26507 | 0.2 | 4.000175488 |
| Standard error | 5.82704E-06 | 0.005816951 | 1.154807463 | 3.17524E-16 | 0.000402379 |
| Median | 0.100006133 | 100.0046554 | 19999.47035 | 0.2 | 4.000400396 |
| Standard deviation | 0.000582704 | 0.581695084 | 115.4807463 | 3.17524E-14 | 0.040237852 |
| Variance | 3.39544E-07 | 0.338369171 | 13335.80275 | 1.00821E-27 | 0.001619085 |
| Skewness | -0.004851911 | -0.002379161 | -0.005482447 | -1 | 0.001078677 |
| Kurtosis | 1.775340068 | 1.782954626 | 1.799654297 | 1 | 2.609424224 |

On peut constater que sur les 10'000 répétitions, on obtient une tension de sortie très proche des 4V espérés. La figure ci-dessous présente ces mêmes résultats de manière graphique.

C:\Users\Emilie\Documents\Git Hub\HiRel\Rapport\images\image_large.png

Le résultat de l’analyse Monte Carlo valide le choix des résistances pour le gain de l’amplificateur. Leur imprécision cause une variation maximale de seulement ±115 mV sur la tension de sortie. Cela représente une imprécision de ±5mA sur le courant mesuré. Si l’on ramène cela par rapport aux 200mA appliqués à la simulation, on peut donc déduire que notre mesure aura une précision d’environ ±3% ce qui nous convient.

# Simulation VHDL du convertisseur A/D

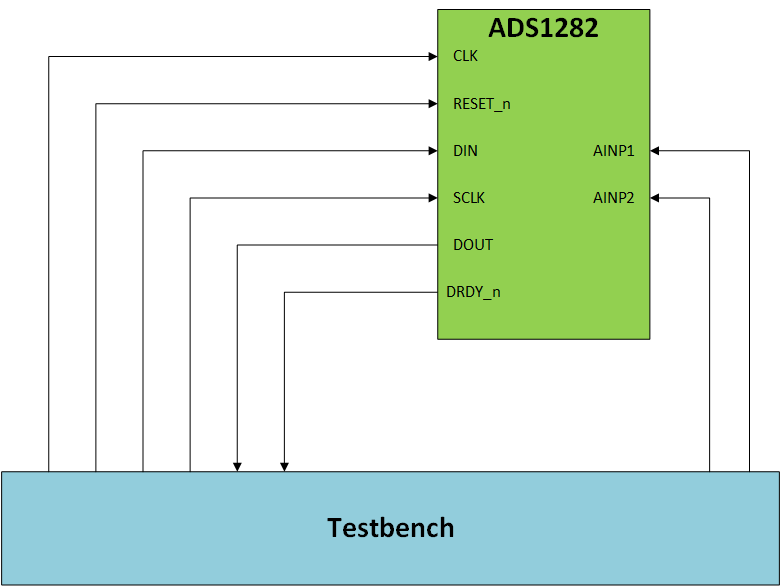
Nous avons reçu un canevas de base du cansat contenant une représentation de notre circuit de conversion A/D.

Nous avons commencé par essayer de simuler le circuit de base. Pour cela, nous avons utilisé le logiciel Questasim dont la licence nous a été fournie dans un autre cours.

L’exécutable compilant le projet, *modelsim.bat*, a dû être modifié pour utiliser questasim et non modelsim.

## Simulation du convertisseur A/D

### Modification de l’entité

Une fois le circuit base compilé et simulé, nous avons modifié l’entité représentant le convertisseur ADS1282. Elle a été modifiée pour correspondre à l’image ci-dessous.

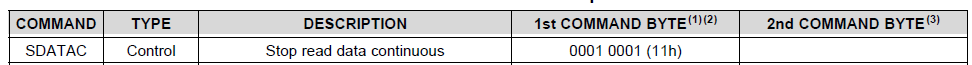
Nous avons enlevé certains signaux tels que SYNC ou PWD\_n pour refléter au mieux la réalité de notre circuit. Dans notre schématique, nous ne disposions que de six pins pour communiquer entre le convertisseur et la FPGA. Nous n’avons donc pas pu relier absolument tous les signaux de contrôle du convertisseur. Nous avons donc conservé uniquement les indispensables.

### Adaptation de convertisseur pour une lecture par commande

Nous avons ensuite modifié le fichier *ads1282\_sim.vhd* contenant une modélisation du comportement du convertisseur A/D. Le comportement implémenté ne correspond pas à celui que nous voulons. Nous voulons utiliser les deux canaux du convertisseur pour obtenir en alternance une mesure du courant et une autre de la tension de notre système.

Par défaut, le convertisseur est en mode *« Read Data Continuous »* sur le canal 1. Nous voulons pouvoir modifier la configuration pour lire les deux canaux. Pour cela, il faut utiliser les commandes suivantes :

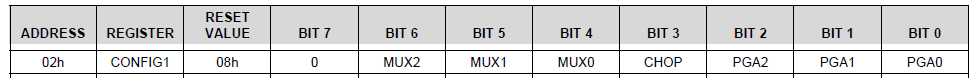
1. Stopper la lecture en continu[[5]](#footnote-6)

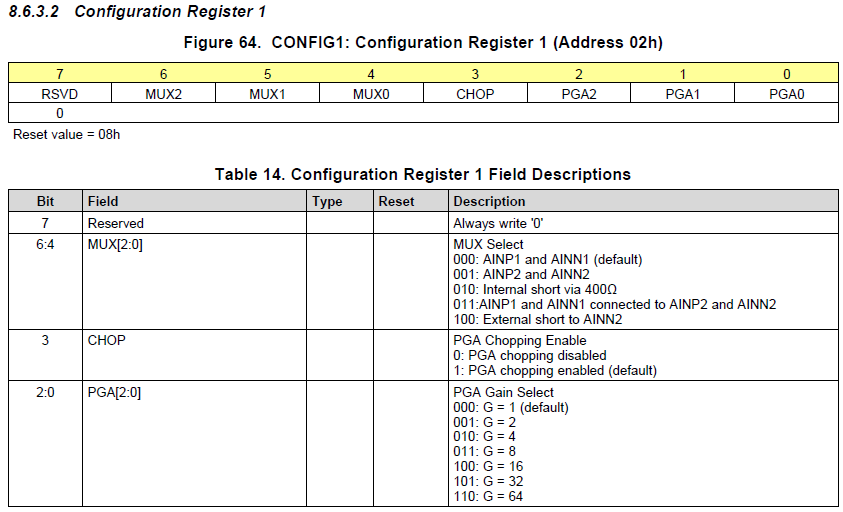


1. Configurer le canal à utiliser :



Pour cela, il faut utiliser la command WREG pour écrire le registre CONFIG1.

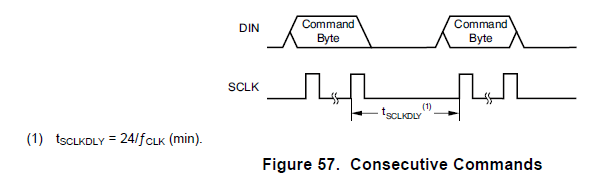


Donc pour configurer la conversion sur le canal 1, il faut envoyer la suite de bytes suivante : 0h420008. Et pour le canal 2, la séquence 0h420018.

1. Lire par commande :



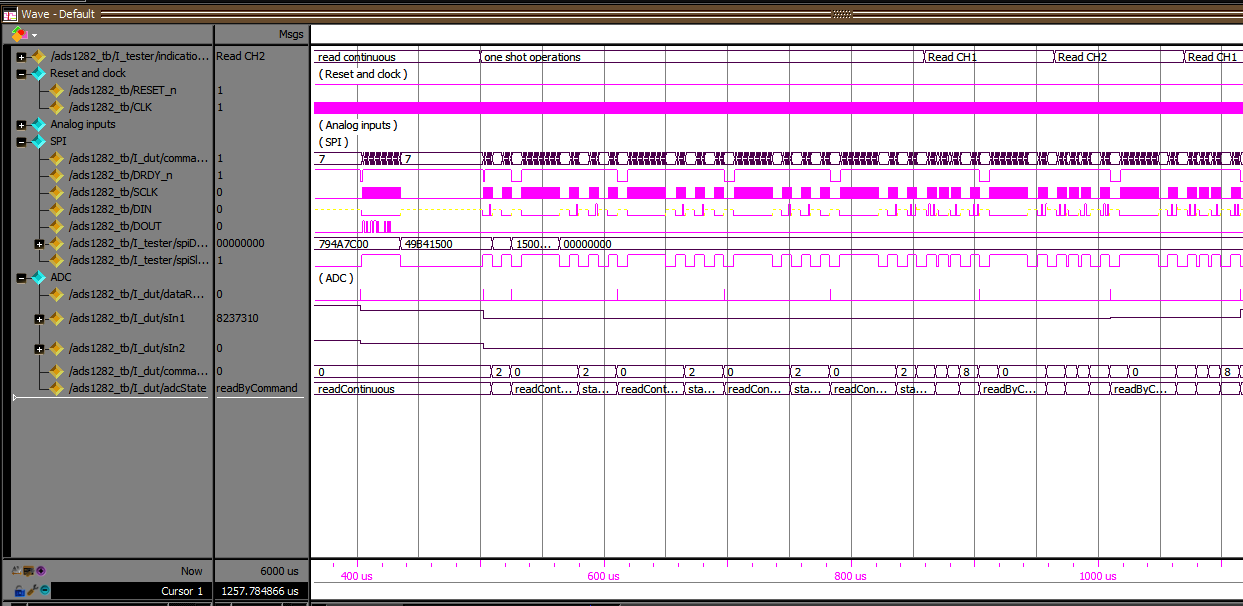
### Timing à respecter

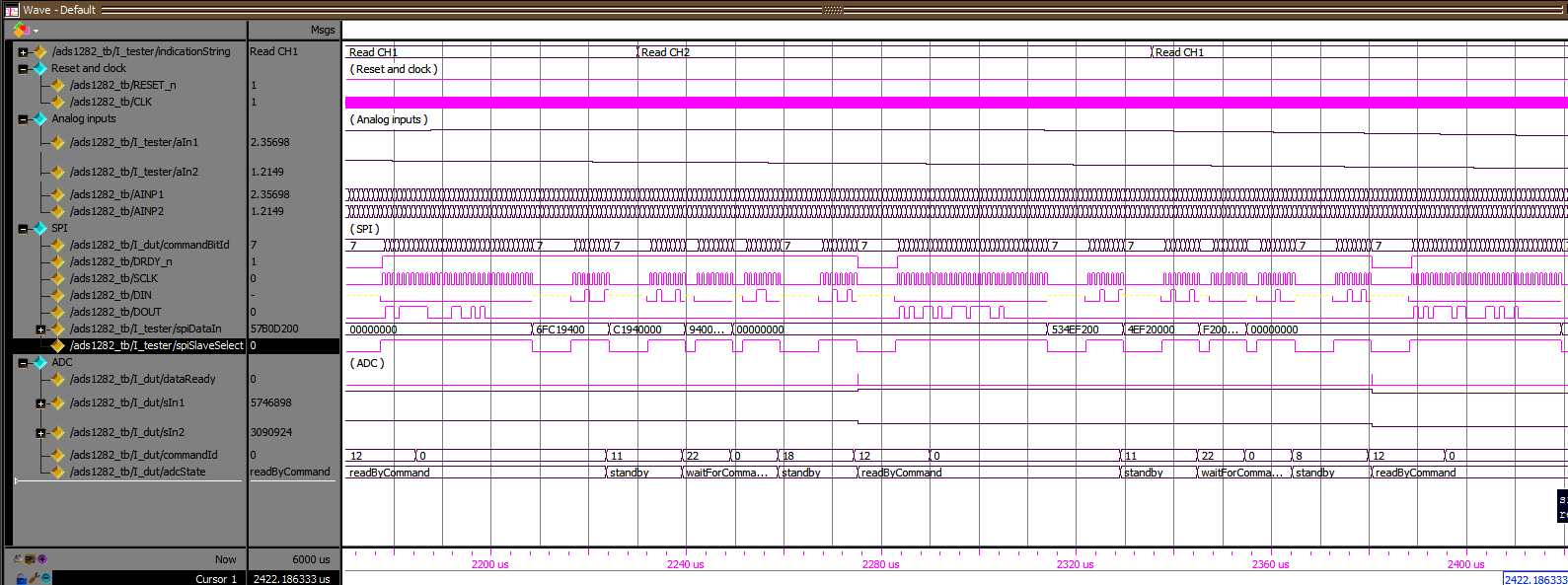
Pour envoyer plusieurs commandes à la suite, il faut respecter 24 coups de clock entre deux commandes :

### Simulation de l’ADS1282

Nous avons donc modifié la représentation de notre convertisseur pour qu’il reconnaisse les trois commandes décrites précédemment. Nous avons fait en sorte de pouvoir envoyer une commande contenant plusieurs bytes pour la configuration du canal. Le canal à lire est modifié lors de la configuration du registre CONFIG1.

Nous avons ensuite modifié le fichier ads1282\_tester\_test.vhd contenant le testbench du convertisseur. Nous l’avons adapté pour qu’il envoie les commandes précédemment implémentées et ainsi tester le bon fonctionnement du système. Nous obtenons la simulation suivante :



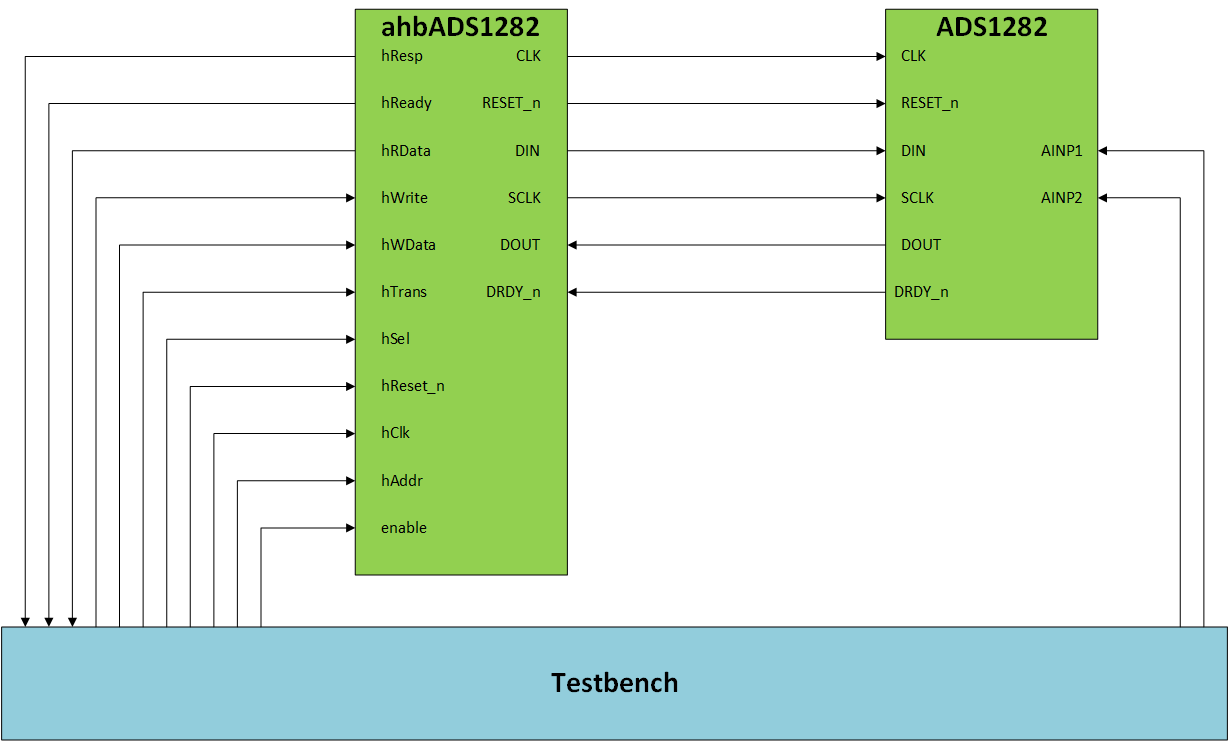
Nous n’avons pas altéré le fonctionnement de base du circuit. On peut toujours faire une lecture en continu ainsi qu’un « *one shot operation »*. Nous avons complété le banc de test pour lire le canal 1 suivi du 2 une vingtaine de fois.

C:\Users\Emilie\Documents\Git Hub\HiRel\Rapport\images\image_large.pngEn regardant de plus près la lecture des canaux, on observe que l’on a bien une valeur qui est envoyé sur la sortie DOUT lors d’une demande de lecture. Nous pouvons donc à ce stade valider la bonne implémentation des commandes de contrôle ainsi que la lecture par commande.

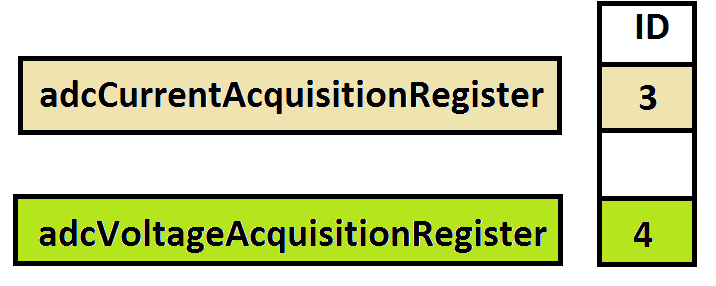
## Simulation du convertisseur piloté par le bus AMBA

### Modification de l’entité

Nous avons également reçu une simulation de convertisseur avec un composant nommée ahbAds1282 qui permet au bus AMBA d’envoyer des instructions pour piloter le convertisseur.

Nous avons modifié son entité pour qu’elle corresponde à l’image ci-dessous :

### Modification du contrôleur

Nous avons ensuite modifié le contenu du contrôleur, *ahbAds1281\_RTL.vhd*. Nous avons ajouté deux registres pour permettre de récupérer la valeur du courant et de la tension.

Ces deux registres sont accessibles uniquement en lecture. Chaque fois que l’entrée enable est active, le circuit fait une lecture d’un canal du convertisseur et stocke sa valeur dans le registre correspondant, voltage pour le canal 1 et current pour le canal 2. Le canal à lire est alterné après chaque lecture.

Comme le code n’était pas pensé pour envoyer plusieurs commandes à la suite, ni des commandes plusieurs bytes. Deux processus ont été implémentés pour combler ce manque.

Celui-ci permet d’envoyer une commande de 3 bytes. Elle est faite sur mesure pour la configuration du registre CONFIG1, mais pourrait être améliorée en ajoutant un signal contenant la taille de la commande voulue.

adcLongCmd: process(adcState,reset, clock)

begin

if reset = '1' then

adcConfigured <= '0';

adcConfigByteNbr <= 0;

elsif rising\_edge(clock) then

adcConfigured <= '0';

if adcState = sendConfigCH and adcSending = '0' and adcCommandWait = '0' then

adcConfigByteNbr <= adcConfigByteNbr+1;

if adcConfigByteNbr >= 2 then

adcConfigByteNbr <= 0;

adcConfigured <= '1';

end if;

end if;

end if;

end process adcLongCmd;

Ce deuxième processus attend que la commande actuelle soit envoyée en levant un signal empêchant la machine d’état d’envoyer d’autres commandes.

adcWaitCmdSend: process(reset,clock)

begin

if reset = '1' then

adcCommandWait <='0';

elsif rising\_edge(clock) then

adcCommandWait <='0';

if adcSendCommand = '1' and adcSending = '0' then

adcCommandWait <='1';

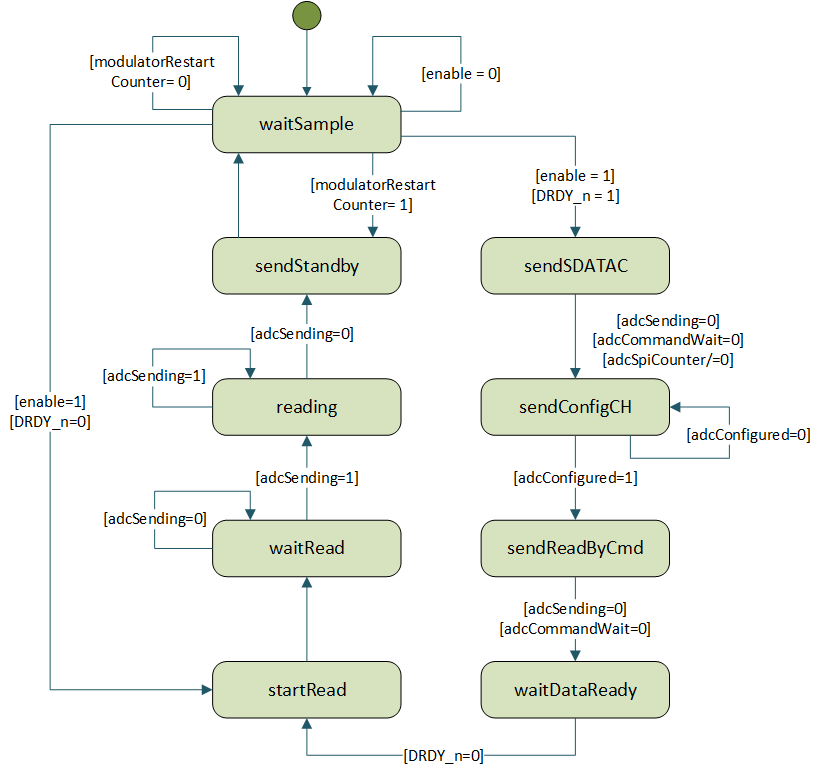
end if;

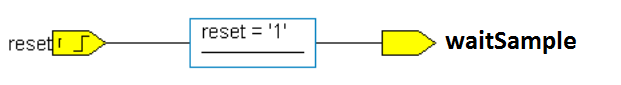
end if;

end process;

### Amélioration de la machine d’états

La machine d’état contenue dans le fichier a été modifiée pour correspondre à celle-ci-dessous :

La machine d’état comporte maintenant neuf états. Dès que l’entrée enable est active, on va configurer le convertisseur A/D et faire une lecture simple sur un de ses canaux.

Pour éviter d’avoir des incertitudes avec les états, nous avons fait en sorte d’être dans l’état waitSample lors d’un reset ou lorsque l’on est dans un état non défini.

Pour rendre notre machine d’état un peu plus robuste nous avons codé les états avec un code de Gray où un seul bit change entre les états pour la séquence :

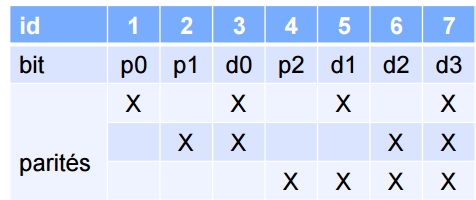
waitSample**->**sendSDATAC**->**sendConfigCH**->**sendReadByCmd**->**waitDataReady**->**startRead**->**reading**->**sendStandby

Wikipedia explique parfaitement bien l’intérêt d’utiliser un code Gray :

*Le fait de modifier plusieurs bits lors d'une simple incrémentation peut mener, selon le circuit logique, à un état transitoire indésirable dû au fait que le chemin logique de chaque bit dispose d'un délai différent. … Ce code permet de contourner cet aléa en forçant la commutation d'un seul bit à la fois, évitant ainsi les états transitoires.[[6]](#footnote-7)*

### Codage de Hamming

Comme notre système peut potentiellement être soumis à des radiations, notre machine d’état peut changer d’état sans que ce ne soit voulu. C’est pourquoi nous avons implémenter une correction d’état par codage de Hamming.

Nous avons neuf états, on a donc besoin de quatre bits pour les coder. Selon le tableau ci-dessous, nous devons donc ajouter 3 bits de parité pour le codage des états.

Chaque état a donc été codé sur 7 bits, le quatre d’état selon le code de Gray et la parité en appliquant un XOR en suivant le tableau ci-dessus. Cela nous donne le code suivant :

constant waitSample : hamming\_stateType := "0000000";

constant sendSDATAC : hamming\_stateType := "0001011";

constant waitDataReady : hamming\_stateType := "0011110";

constant startRead : hamming\_stateType := "0010101";

constant waitRead : hamming\_stateType := "0110011";

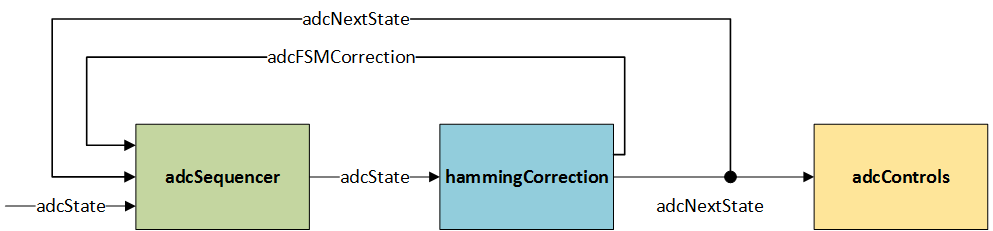
constant reading : hamming\_stateType := "0111000";

constant sendStandby : hamming\_stateType := "0101101";

constant sendConfigCH : hamming\_stateType := "0100110";

constant sendReadByCmd : hamming\_stateType := "1100001";

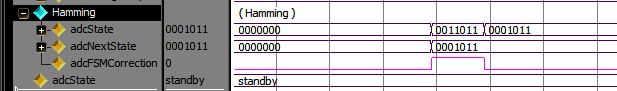
Nous avons ensuite implémenté un processus nommée hammingCorrection qui va analyser l’état courant et indiquer à l’aide du signal *adcFSMCorrection* si une erreur a été détectée sur l’état. Cela permet à la machine de recharger l’état correct. L’état correct est calculé et est donnée sur le signal *adcNextState*. L’image ci-dessous présente le principe de notre codage de Hamming.



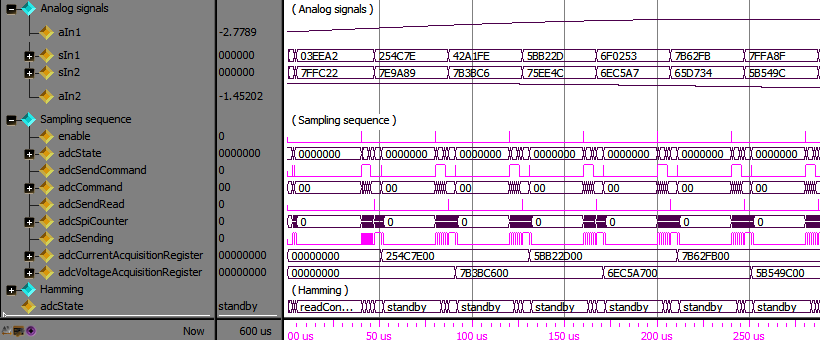
Afin de valider le bon fonctionnement de notre correction d’état, nous avons volontairement introduit une erreur dans notre machine sur l’état sendSDATAC.

--adcState <= sendSDATAC; //Correct state = 0001011

adcState <= "0011011"; //Wrong state

Voici le résultat de la simulation, on voit que l’état est corrigé correctement :

### Simulation du circuit

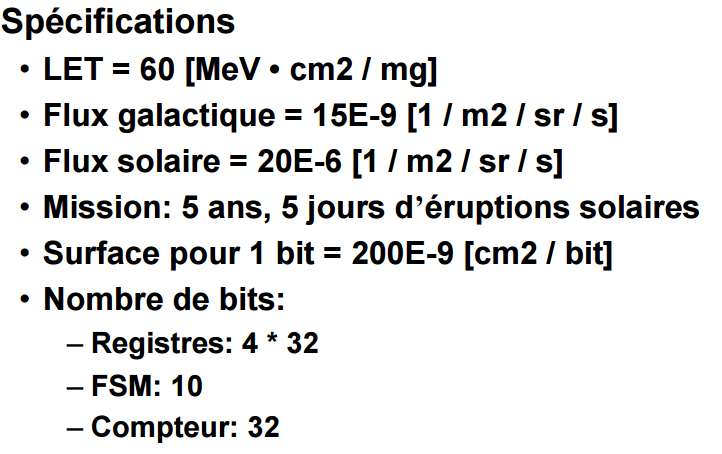
Avec cette simulation, on valide qu’à chaque enable, on arrive à récupérer correctement la valeur échantillonnée par le convertisseur A/D. On voit dans l’image que l’on lit correctement en alternance la valeur de sIn1 et sIn2 correspondant aux deux canaux du convertisseur. Les valeurs sont correctement récupérées dans les deux registres correspondants *adcCurrentAcquisitionRegister* et *adcVoltageAcquisitionRegister*.

C:\Users\Emilie\Documents\Git Hub\HiRel\Rapport\images\image_large.pngÀ ce stade on a donc :

Une machine d’état robuste dont l’état peut être corrigé en cas de perturbation extérieure.

C:\Users\Emilie\Documents\Git Hub\HiRel\Rapport\images\image_large.pngUn périphérique permettant de récupérer alternativement les valeurs du courant et de la tension d’alimentation du circuit. Les résultats sont accessibles par le bus AMBA grâce à des registres.

# Probabilité de SEU

Pour le calcul de probabilité de SEU (*single event upset*), nous avons repris les paramètres donnés dans les slides du cours. Nous ne savions pas comment calculer cela par rapport à notre circuit. La donnée est donc :

Nous avons introduit ces données dans le tableau Excel d’exemple et obtenu un taux de SEU de 3.85E-09 SEU/jour. On a donc une très faible probabilité de SEU.

1. <https://fr.wikipedia.org/wiki/Effet_Hall> [↑](#footnote-ref-1)
2. <http://docplayer.fr/docs-images/24/4149845/images/5-0.png> [↑](#footnote-ref-2)
3. <http://www.electronique-mag.com/IMG/gif/3-2.gif> [↑](#footnote-ref-3)
4. Lien : <http://www.expertmultimedia.ch/ressources/graphisme-symboles-logos/symboles-1/symbole-vu/image_large> [↑](#footnote-ref-4)
5. Source des images : Datasheet du ADS1282 [↑](#footnote-ref-6)
6. <https://fr.wikipedia.org/wiki/Code_de_Gray> [↑](#footnote-ref-7)