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 Introduction 4](#_Toc452736696)

[1.1 Choix du capteur 4](#_Toc452736697)

[1.1.1 Principes physiques disponibles 4](#_Toc452736698)

[1.1.2 Principe choisi 4](#_Toc452736699)

[1.2 Schéma bloc de la chaîne de mesure 4](#_Toc452736700)

[1.3 Analyse des performances attendues 4](#_Toc452736701)

[2 Simulation de la mesure de courant 4](#_Toc452736702)

[2.1 Charge variable 5](#_Toc452736703)

[2.2 Décharge de la batterie 6](#_Toc452736704)

[2.3 Courant sinusoïdal 7](#_Toc452736705)

[3 Chaîne d’acquisition 8](#_Toc452736706)

[3.1 Schéma électrique 8](#_Toc452736707)

[3.2 Budget de la chaîne de mesure 8](#_Toc452736708)

[3.2.1 Liste des composants 8](#_Toc452736709)

[3.2.2 Budget de masse 9](#_Toc452736710)

[3.2.3 Budget d’espace 9](#_Toc452736711)

[3.2.4 Budget de consommation 9](#_Toc452736712)

[4 Conception détaillée 9](#_Toc452736713)

[4.1 Dissipation de chaleur 9](#_Toc452736714)

[4.1.1 Influence de PDiss sur TJ 11](#_Toc452736715)

[4.1.2 Influence de TSC sur TJ 12](#_Toc452736716)

[5 Failure Mode, effects and criticality analysis (FMECA) 13](#_Toc452736717)

[6 Part Stress Analysis (PSA) 13](#_Toc452736718)

[7 Worst Case Analysis (WCA) 13](#_Toc452736719)

[7.1 Détermination des paramètres à considérer 13](#_Toc452736720)

[7.2 Détermination des performances à analyser 13](#_Toc452736721)

[7.3 Analyse Monte Carlo 13](#_Toc452736722)

[7.4 Evaluation des résultats 14](#_Toc452736723)

[8 Analyse de fiabilité (FRA) 15](#_Toc452736724)

[9 Systèmes numériques 15](#_Toc452736725)

[10 Périphérique de contrôle 15](#_Toc452736726)

[10.1 Convertisseur A/D 15](#_Toc452736727)

[10.1.1 Modification de l’entité 16](#_Toc452736728)

[10.1.2 Adaptation de convertisseur pour une lecture par commande 16](#_Toc452736729)

[10.1.3 Timing à respecter 17](#_Toc452736730)

[10.1.4 Simulation de l’ADS1282 18](#_Toc452736731)

[10.2 Périphérique AMBA 19](#_Toc452736732)

[10.2.1 Modification de l’entité 19](#_Toc452736733)

[10.2.2 Modification du contrôleur 19](#_Toc452736734)

[10.3 Machine d’état du contrôleur ADC 20](#_Toc452736735)

[10.4 Codage de la machine d’états 20](#_Toc452736736)

[10.4.1 Signaux de contrôle 20](#_Toc452736737)

[10.4.2 Fiabilisation 21](#_Toc452736738)

[11 Effets des radiations 21](#_Toc452736739)

[11.1 Registre à décalage pour réception d’un signal série 21](#_Toc452736740)

[11.2 Machine d’état du contrôleur ADC avec contraintes SEU 24](#_Toc452736741)

[11.3 Décodeur de Hamming 25](#_Toc452736742)

[*11.3.1* *Réponse aux questions des slides* 27](#_Toc452736743)

[11.3.2 Mise en commun du codage de Hamming avec correction de séquence 27](#_Toc452736744)

[11.3.3 Transmission de l’erreur au registre de status 28](#_Toc452736745)

[11.4 Simulation de l’ensemble du circuit 28](#_Toc452736746)

[11.5 Estimation du taux de SEU par jour 28](#_Toc452736747)

[12 Conception de circuit numériques 29](#_Toc452736748)

[12.1 Alimentation FPGA 29](#_Toc452736749)

[12.1.1 Circuit d’alimentation 29](#_Toc452736750)

[12.1.2 Séquencement 30](#_Toc452736751)

[12.2 Entrées / Sorties du circuit 31](#_Toc452736752)

[13 Vérification de circuits numériques 31](#_Toc452736753)

[13.1 Matrice de conformité 31](#_Toc452736754)

[14 Tests fonctionnels 31](#_Toc452736755)

[14.1 Environnement de test 31](#_Toc452736756)

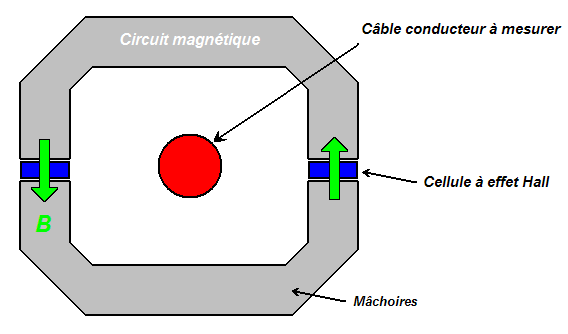
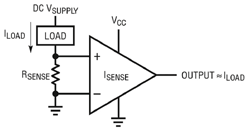
[14.2 Plan de test 31](#_Toc452736757)

[14.3 Procédure de test 32](#_Toc452736758)

# Introduction

## 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

**Dossier de référence :** */SchemaBloc*

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 tout 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.

## Analyse des performances attendues

**Dossier de référence :**

# Simulation de la mesure de courant

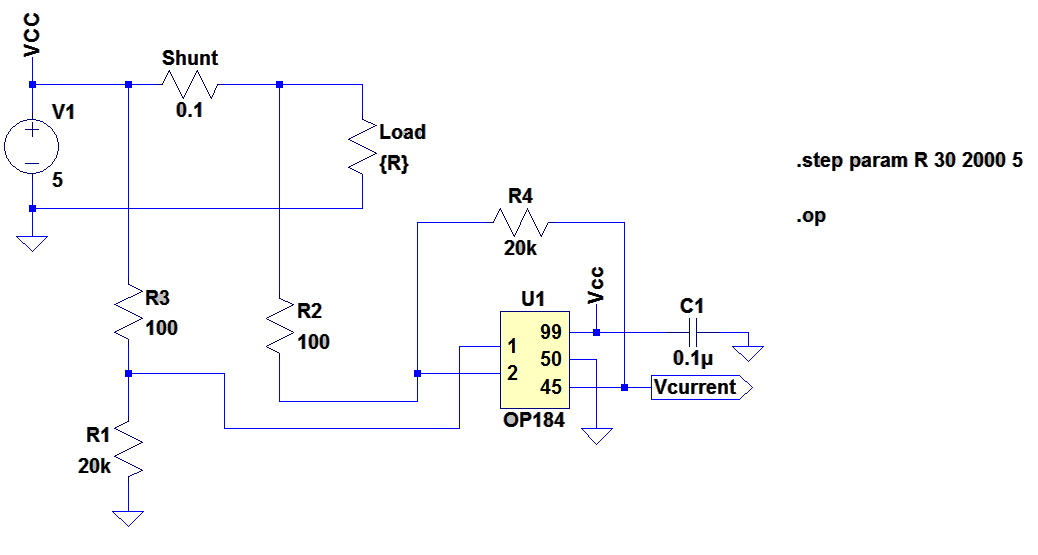
Avec le logiciel LTSpice, nous avons pu simuler notre circuit de mesure de courant dans plusieurs cas. Selon les spécifications données, le courant consommé par l’ensemble du circuit est aux alentours de 130 mA. Nous avons volontairement gardé une marge, car on ne connait pas encore la consommation de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

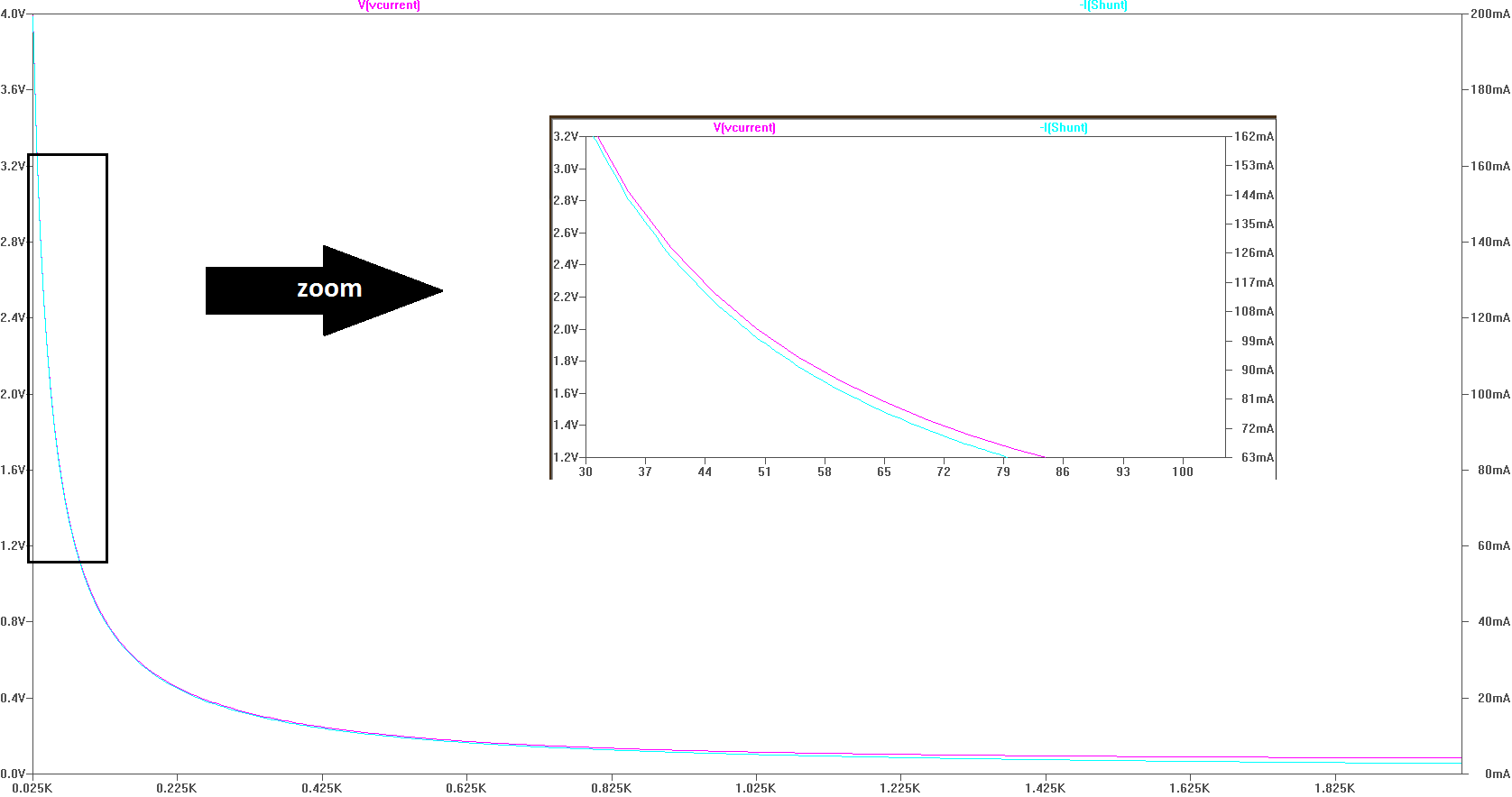
Nous avons dimensionné notre amplificateur avec un gain de 20 et choisi une résistance shunt de 0.1 ohm. Cela nous donne une tension de 4V avec un courant de charge de 200 mA. Comme la tension d’alimentation est de 5V et que l’ampli est rail-to-rail, on peut théoriquement mesurer jusqu’à 250 mA.

[[4]](#footnote-4) Le composant de notre Amplificateur OP184 a été importé. Il se peut qu’il faille le réimporter pour pouvoir reproduire nos simulations. C’est pourquoi nous avons fourni ci-dessous des captures d’écran de nos simulations.

## Charge variable

**Dossier de référence :** */Simulation/chargeVariable*

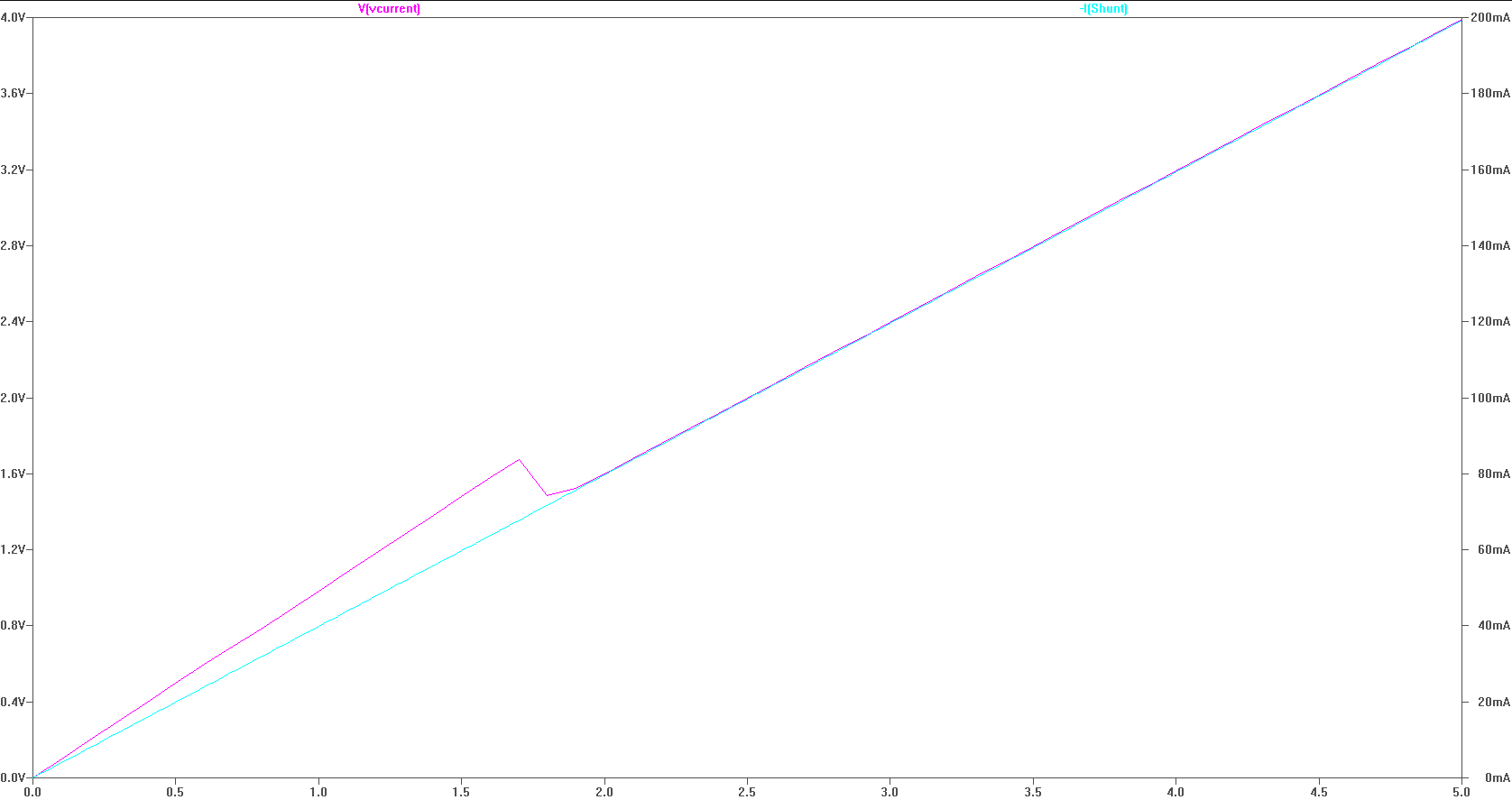
Dans cette simulation, la charge du circuit varie de façon à avoir un courant consommé variable. Le but est d’observer si la tension en sortie de l’amplificateur suit correctement la courbe de courant.

On constate que notre ampli op transforme correctement le courant du shunt en tension jusqu’aux alentours de 5 mA. En dessous de cette valeur, la tension fournie reste constante à 100 mV.

## Décharge de la batterie

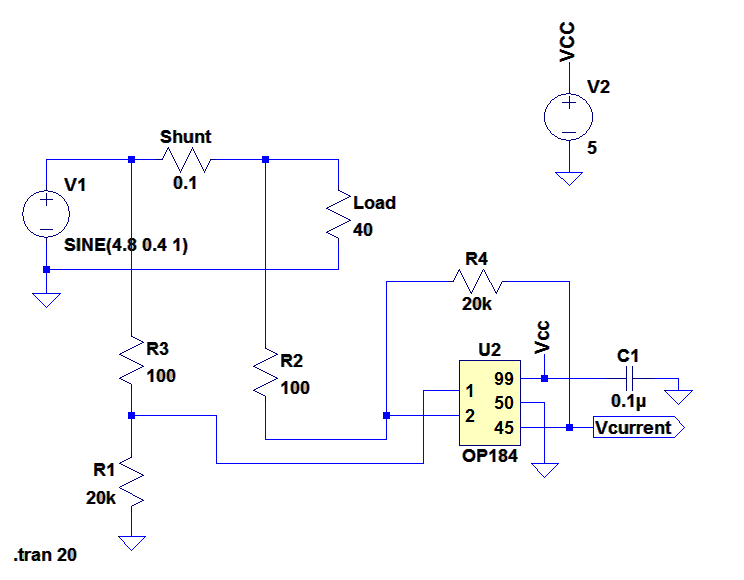
**Dossier de référence :** */Simulation/dechargeBatterie*

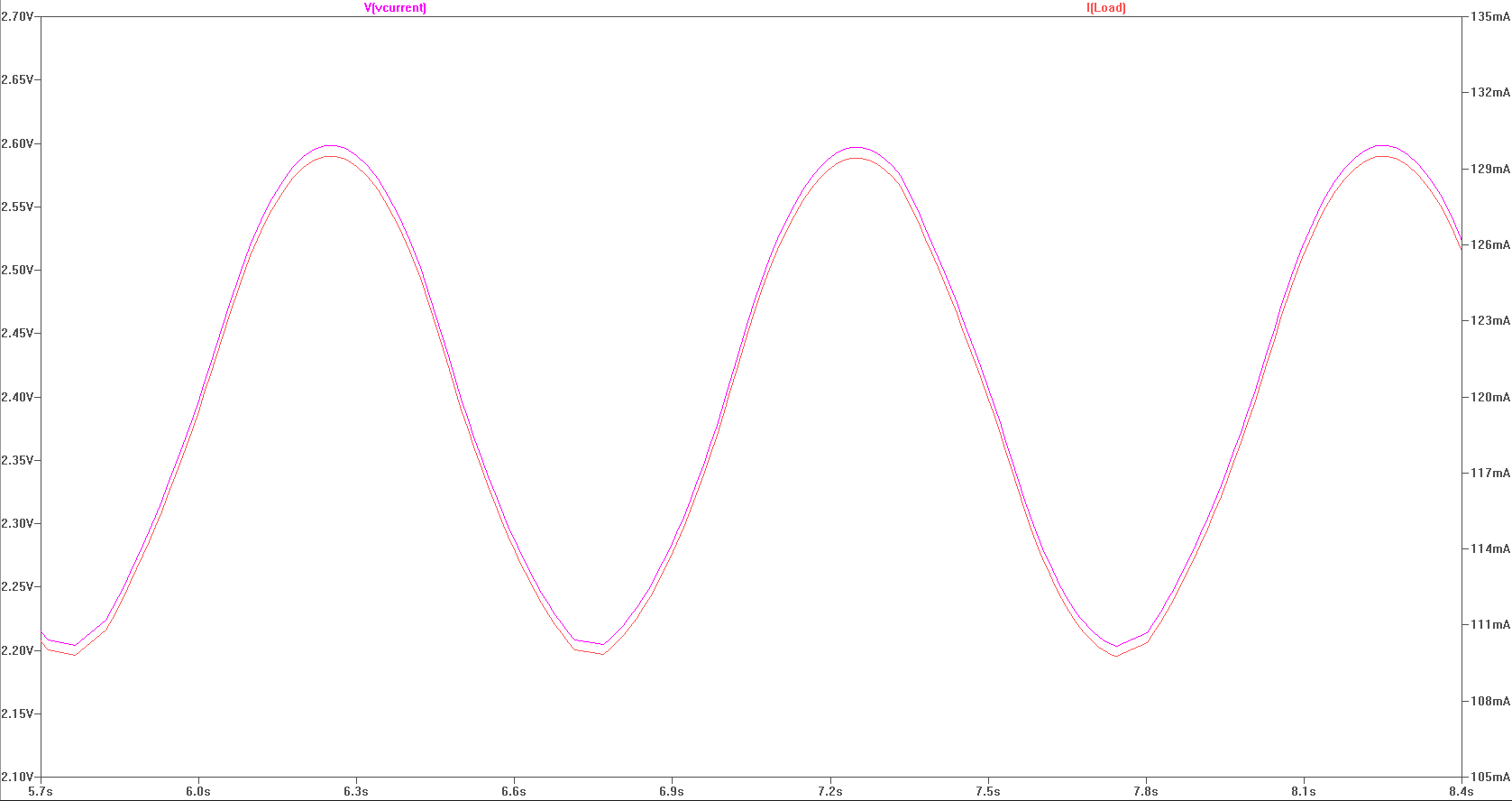
Dans cette simulation, on observe le comportement du système lorsque la tension d’alimentation diminue.

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

**Dossier de référence :** */Simulation/courantSinus*

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 reproduit 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. On voit ci-dessus résultat que la mesure suit correctement l’oscillation du courant consommé.

C:\Users\Emilie\Documents\Git Hub\HiRel\Rapport\images\image_large.png[[5]](#footnote-5) Ces différentes simulations ont permis de valider le principe de mesure de courant et l’amplificateur opérationnel choisis.

[[6]](#footnote-6)C:\Users\Emilie\Documents\Git Hub\HiRel\Rapport\images\croix-rouge.png Nous n’avons pas trouvé de modèle LTSpice de notre convertisseur A/D. Nous n’avons donc pas pu simuler entièrement le comportement de notre circuit de mesure.

# Chaîne d’acquisition

## Schéma électrique

**Dossier de référence :**

## Budget de la chaîne de mesure

**Dossier de référence :** */BudgetChaineMesure*

### Liste des composants

La liste des composants du circuit a été incluse dans le fichier Excel sous l’onglet « 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. 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 va passer très peu de temps dans l’espace.

Le FTDI avec ses deux LEDs et les autres petits composants allant autour pourront être enlevés 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 total de **12 g**, ce qui est raisonnable. Avec l’estimation du poids du PCB et de la soudure, on passe à un poids estimé à **28 g**. Une autre estimation a été faite en multipliant l’espace occupé par le composant avec l’épaisseur du PCB et sa masse volumique, on obtient un poids de **22 g.**

### 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.

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 totale 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.

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**. Cela fait que la shunt dissipe une puissance de 0.1ohm\*135mA2**=1.8mW**

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.

# Conception détaillée

## Dissipation de chaleur

**Dossier de référence :** */DissipationChaleur*

Nous avons décidé faire notre analyse de chaleur sur le chip FTDI, avec ses 75mW c’est le composant qui consomme le plus. On analysera comment la chaleur va se dissiper au travers du PCB. Une simulation est disponible dans le dossier de référence.

Le datasheet du FTDI nous fournit la résistance thermique de contact JC (jonction-case). Il donne également la résistance JA (jonction-air), mais elle n’est pas utile dans ce contexte, car la dissipation par convection et radiation est négligeable dans l’espace. On a bien plus de dissipation par conduction.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Item** | **Material** | **Dimensions** | **Mass** | **Thermal conductivity [W/(m⋅K)]** | **Thermal capacity [J/(K⋅kg)]** |
| FT232RL | n/a | 7.8mm x 10.2mm x  2mm | 2268mg | component  specific | n/a |
| PCB conducting layers | Cu,  ρ = 8.9mg/mm3 | 35mm x 83mm,  4 layers 35µm | mCu,35 = 90.5mg (35µm layer,  fill factor 0.1) mCu,plane = 301mg (35µm layer,  fill factor 0.85) | kcu = 190 | Ccu = 380 |
| PCB substrate | FR4,  ρ = 2.5mg/mm3, prepreg material, ρ= 1.5mg/mm3 | 35mm x 83mm  1 FR4 of 150 µm+  2 prepreg of 135 µm | mPCB = 1089 mg FR4 + 1177 mg Prepreg  +90.5mg Cu  = 2.4g | kFR4 = 0.85 | CFR4 = 830 |

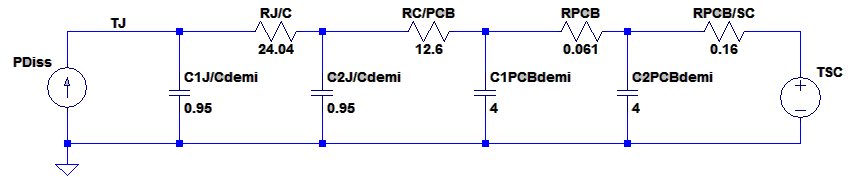
Avec ce tableau, nous pouvons calculer les résistances thermiques et capacitives.

Le calcul est le suivant :

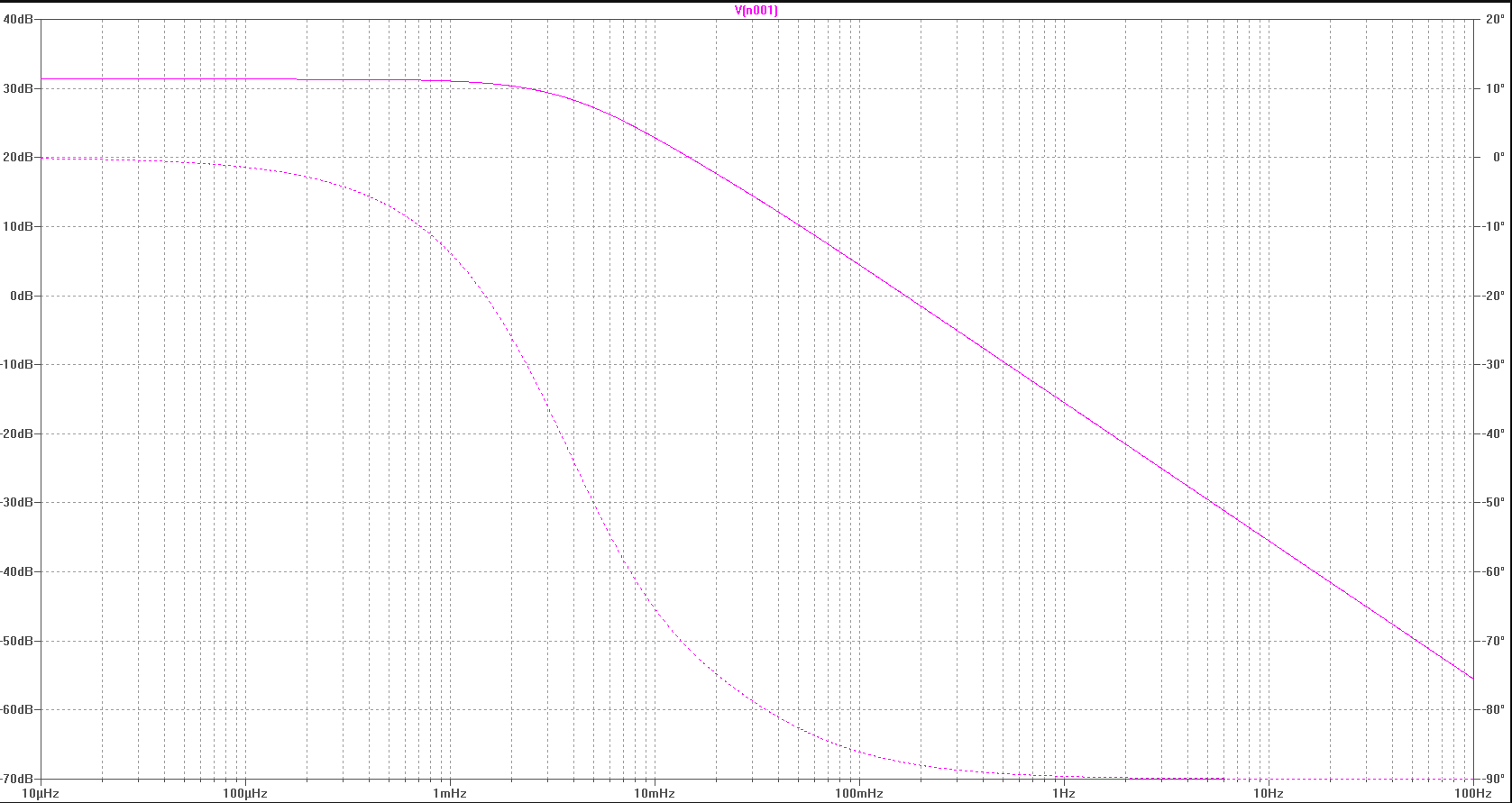
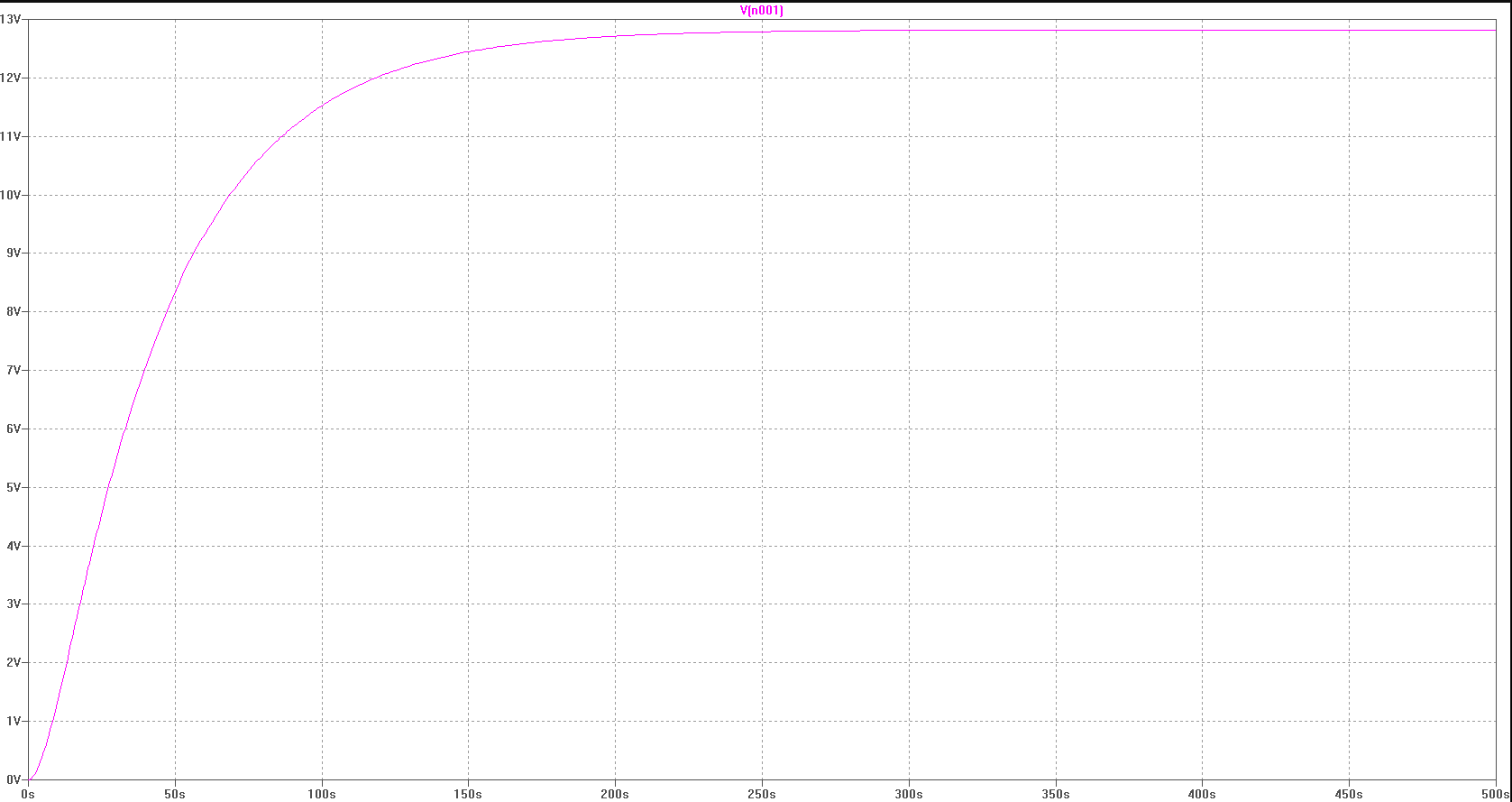
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Épaisseur** | **Surface** | **Résistance thermique** | **[K/W]** |
| FT232RL JC | n/a | n/a | RJ/c | 24.04 |
| Contact case-PCB  Estimated Rc ≈ 1mK⋅m2/W | n/a | 7.8mm x 10.2mm | RC/PCB | 12.6 |
| PCB | Cu, 4\*35 µm  FR4, 1\*150 µm | Cu, 35mmx83mm  FR4, 35mmx83mm | RPCB | 0.061 |
| Contact PCB-Spacecraft  Estimated Rc ≈ 1mK⋅m2/W | n/a | 35mm x 83mm x2+  (70mm+166mm)\*2mm | RPCB/SC | 0.16 |

Le calcul est le suivant : sse \* Thermal capacity

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Masse** | **Condensateur thermique** | **[J/K]** |
| FT232RL | 2.268 g (Estimation 0.85 J/(K**⋅**g)) | CJ/A et CJ/C | 1.9 |
| PCB | 2.4 g PCB+11 g components (Estimation for complete PCB with components: CPCB,total = 0.6 J/( K⋅g)) | CPCB | 8 |

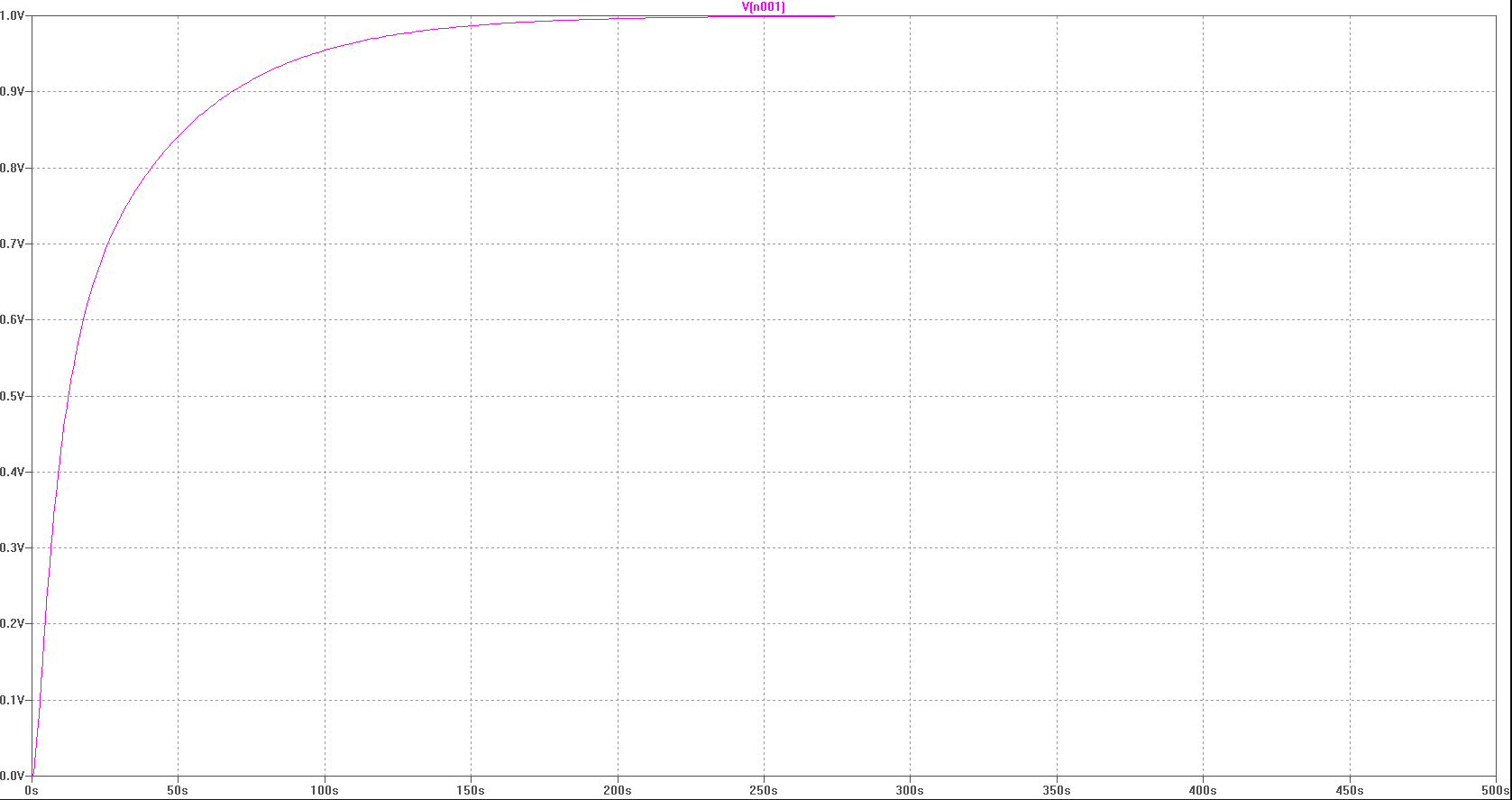
Nous obtenons le schéma équivalent suivant :

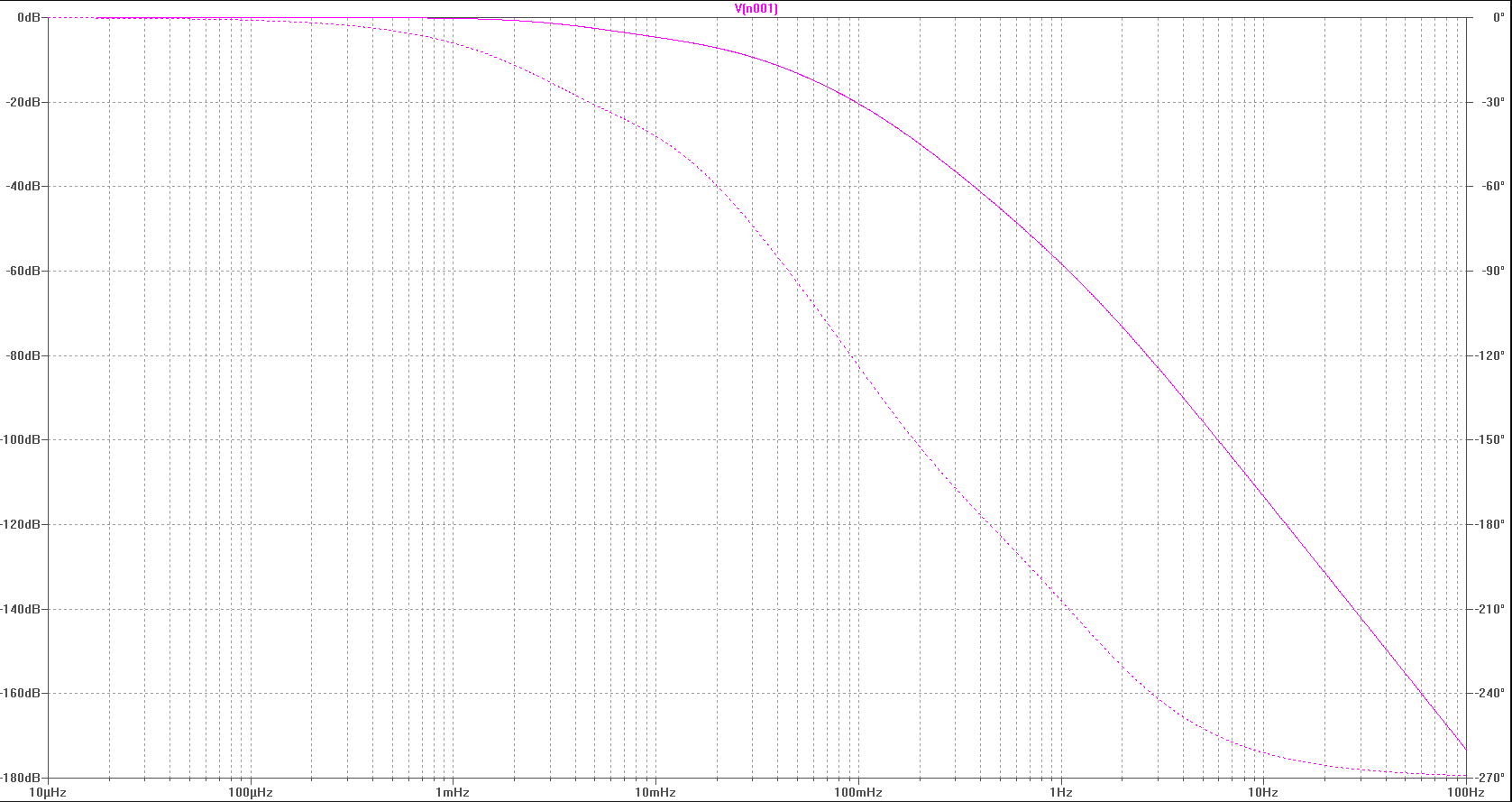
### Influence de PDiss sur TJ

Comportement de TJ lors d’un saut de sa dissipation PDiss de 1W :

Réponse en fréquence de TJ par rapport à sa dissipation PDiss :

### Influence de TSC sur TJ

Comportement de TJ lors d’un saut de température TSC de 1K  

Réponse en fréquence de TJ par rapport à la température TSC :

La réponse en fréquence et le comportement par rapport à un saut donnent les mêmes informations. On peut voir qu’il faut environ 300 secondes, soit 5 minutes pour élever la température de la jonction de 1 Kelvin. Et que suite à un saut de la puissance de 1 W, la température s’élève de 13 Kelvin en 5 minutes.

On remarque que la température du composant varie plus rapidement que l’exemple des slides, cela est dû au fait que le circuit présenté est beaucoup plus grand et est dans un boitier.

# Failure Mode, effects and criticality analysis (FMECA)

**Dossier de référence :**

# Part Stress Analysis (PSA)

**Dossier de référence :**

# Worst Case Analysis (WCA)

**Dossier de référence :** */WCA*

Nous avons choisi d’analyser le bloc de notre circuit qui nous semble le plus critique, celui de la mesure de courant.



## Détermination des paramètres à considérer

Les paramètres à considérer pour les résistances sont donnés par les slides du cours :

* Résistances :
* Temprérature : Dérive et valeur aléatoire de la résistance
* Humidité : Dérive de la résistance et composition du carbon
* Durrée de vie, alimenté : Dérive et valeur aléatoire de la résistance
* Durée de vie, non alimenté : Dérive et valeur aléatoire de la résistance
* Vide d’aire : Dérive de la résistance
* Mécanique : Dérive et valeur aléatoire de la résistance

## Détermination des performances à analyser

Nous allons analyser le gain du circuit avec la formule donnée plus haut.

## Analyse Monte Carlo

Le gain du circuit est défini par les résistances. Selon les paramètres à considérés que nous avons trouvés, il faut tenir compte de la valeur aléatoire des résistances dues aux tolérances données par les fabricants.

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

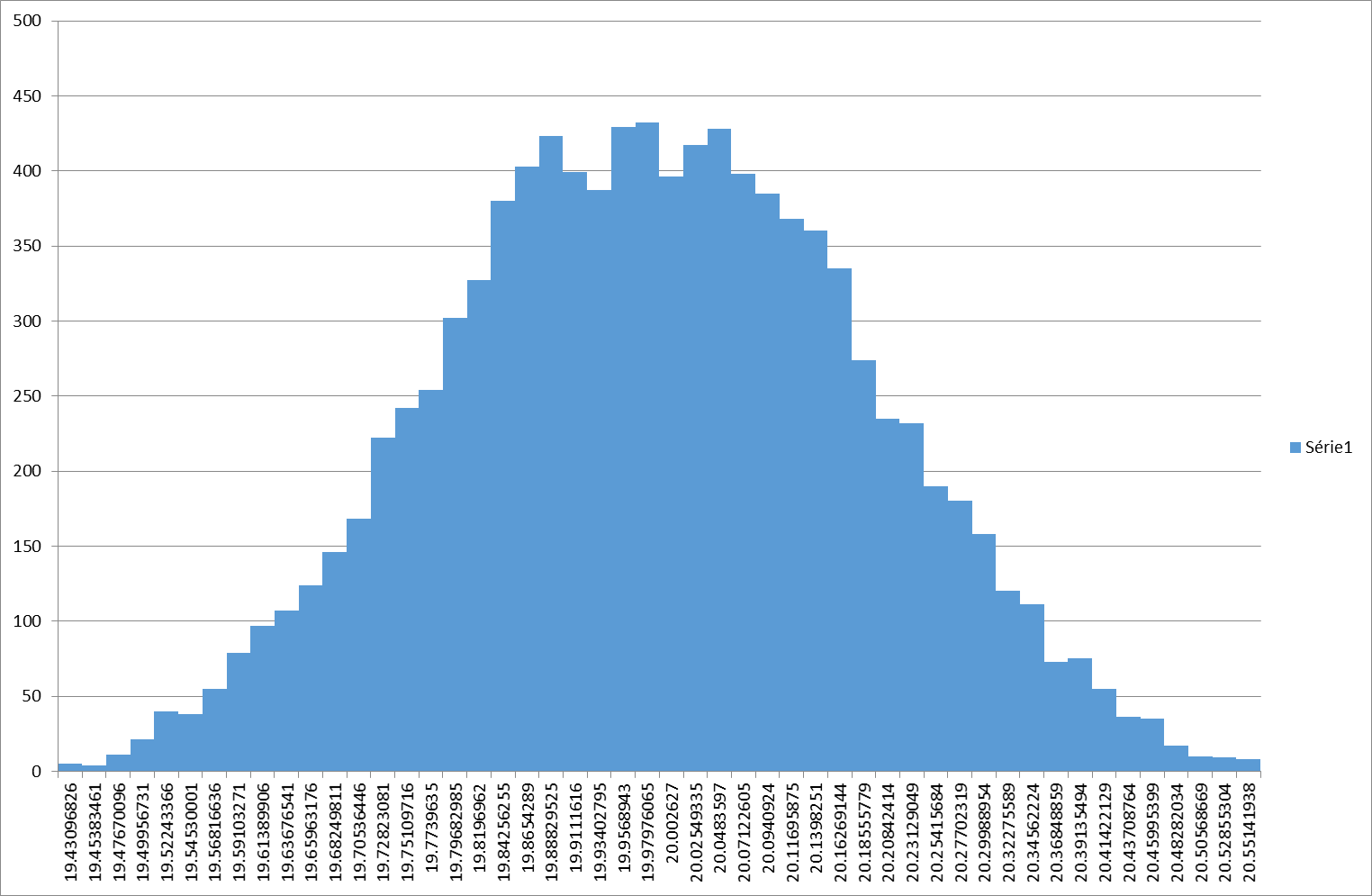
|  |  |  |  |
| --- | --- | --- | --- |
| Rsense | R3 | R1 | Gain |
| =0.1\*(1+0.02\*(ALEA()-0.5)) | =100\*(1+0.02\*(ALEA()-0.5)) | =20000\*(1+0.02\*(ALEA()-0.5)) | = Rsense \* R1/ R3 |

La valeur de gain espérée est de 20.

Nous n’avons pas fait l’analyse avec PSPICE, mais avec le fichier Excel fourni. On pourrait faire l’analyse avec PSPICE ou LTSPice, cela nous fournira les mêmes résultats. L’avantage d’utiliser LTSpice est que l’on n’a pas besoin d’entrer les formules, il suffit de donner les précisions de résistances. Cela peut être très utile avec un grand circuit à simuler. Le désavantage par rapport au fichier Excel est que les résultats ne sont pas donnés sous forme graphique. Comme les équations à simuler sont relativement simples, nous avons choisi d’utiliser la feuille Excel.

## Evaluation des résultats

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

On peut constater que sur les 10'000 répétitions, on obtient un gain très proche des 20 espérés. La figure ci-dessous présente ces mêmes résultats de manière graphique.

C:\Users\Emilie\Documents\Git Hub\HiRel\Rapport\images\image_large.pngLe résultat de l’analyse Monte Carlo valide le choix des résistances pour le gain de l’amplificateur. Leur imprécision donne une plage 3σ de : 19.43 ≤ Gain ≤ 20.55, ce qui correspond à ±2.75% du gain espéré pour le pire des cas. Cela nous convient, on est en dessous des 3% et le graphique est correctement centré autour de 20.

# Analyse de fiabilité (FRA)

**Dossier de référence :**

# Systèmes numériques

**Dossier de référence :** */VHDL/CanSat*

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.

Des captures d’écran de toutes les simulations sont présentes dans ce document au cas où le code ne serait pas simulable avec un autre logiciel.

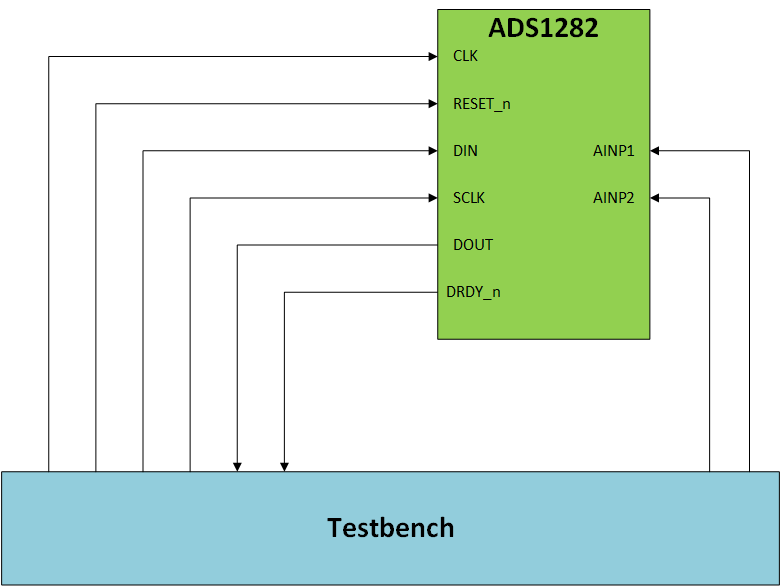
L’exécutable compilant le projet, *modelsim.bat*, a dû être modifié pour utiliser questasim et non modelsim.

# Périphérique de contrôle

## Convertisseur A/D

**Dossier de référence :** */VHDL/CanSat*

### 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.

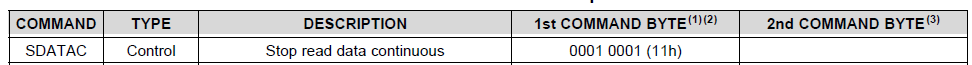
Nous avons enlevé certains signaux tels que SYNC ou PWD\_n pour refléter au mieux la réalité de notre circuit. Dans notre schématique, nous ne disposions que de six pins pour communiquer entre le convertisseur et la FPGA. Nous n’avons donc pas pu relier absolument tous les signaux de contrôle du convertisseur.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

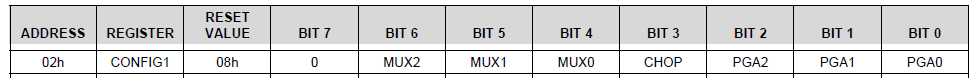
1. Stopper la lecture en continu[[7]](#footnote-8)

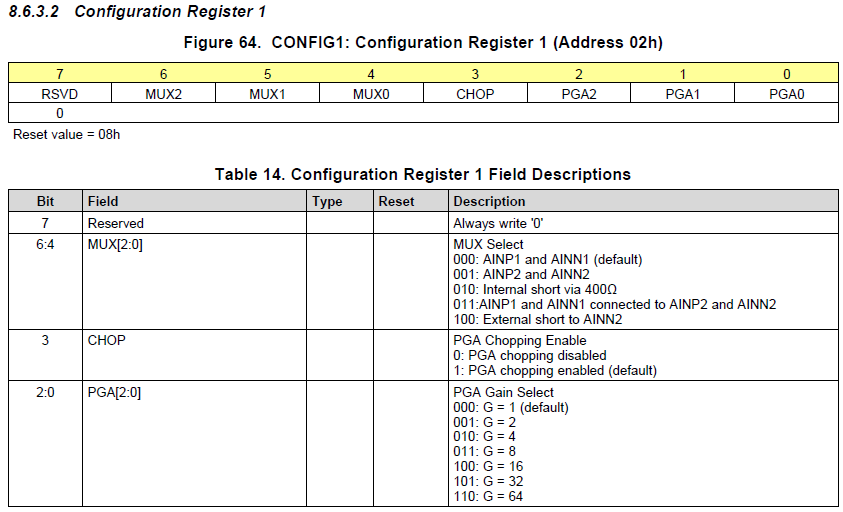


1. Configurer le canal à utiliser :



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

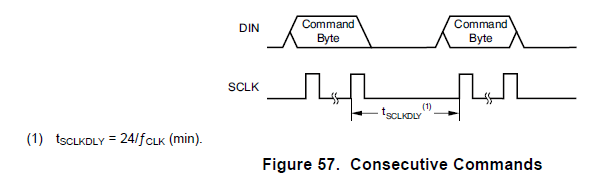


Donc pour configurer la conversion sur le canal 1, il faut envoyer la suite de bytes suivante : 0h420008. Et pour le canal 2, la séquence 0h420018.

1. Lire par commande :



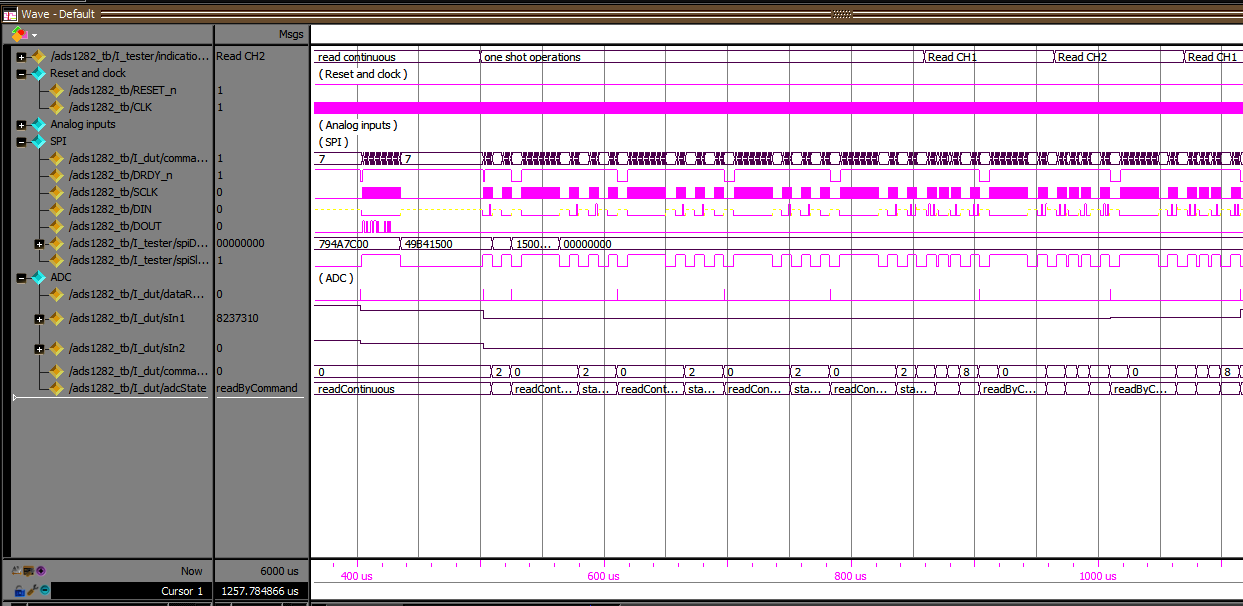
### Timing à respecter

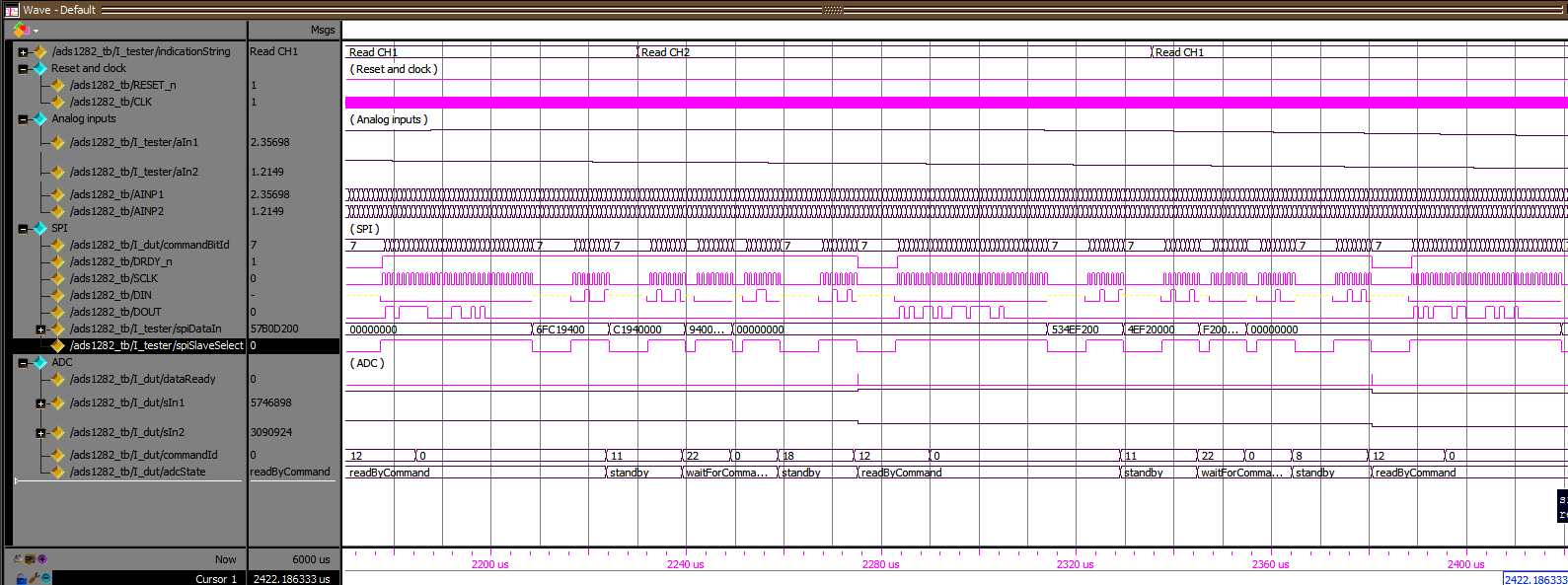
Pour envoyer plusieurs commandes à la suite, il faut respecter 24 coups de clock entre deux commandes :

### 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.

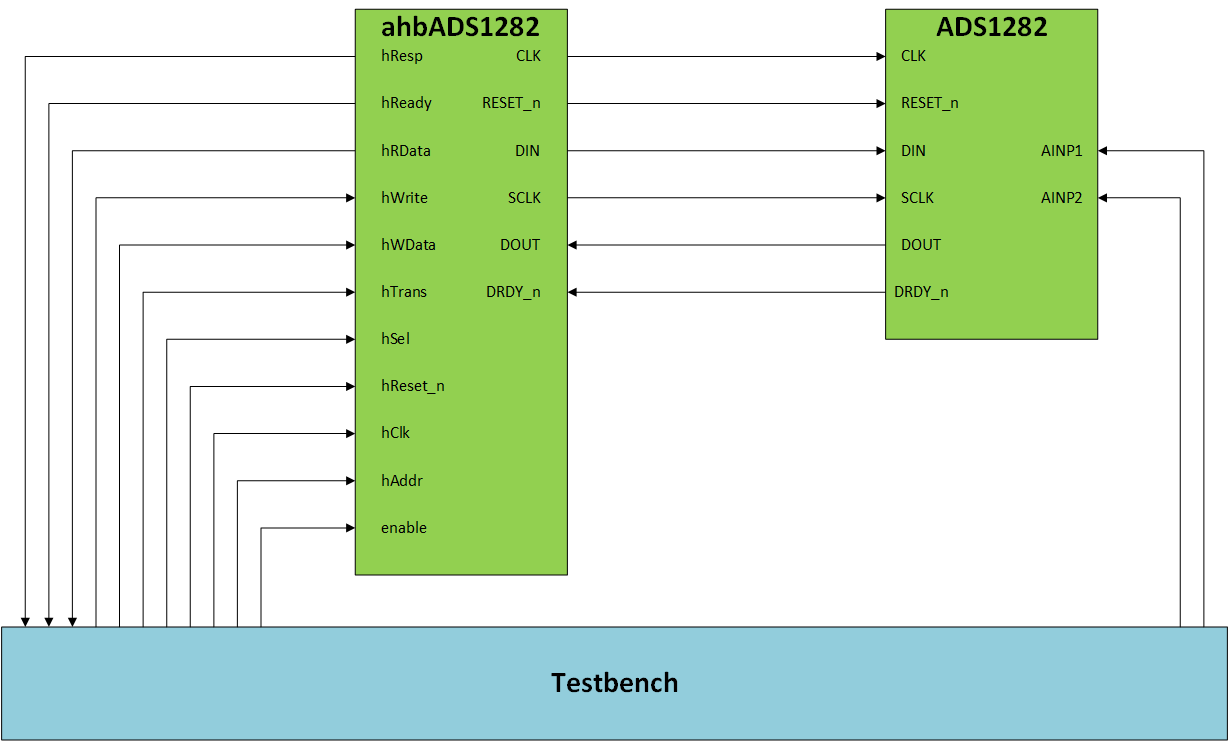
C:\Users\Emilie\Documents\Git Hub\HiRel\Rapport\images\image_large.pngEn regardant de plus près la lecture des canaux, on observe que l’on a bien une valeur qui est envoyé sur la sortie DOUT lors d’une demande de lecture. Nous pouvons donc à ce stade valider la bonne implémentation des commandes de contrôle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

## Périphérique AMBA

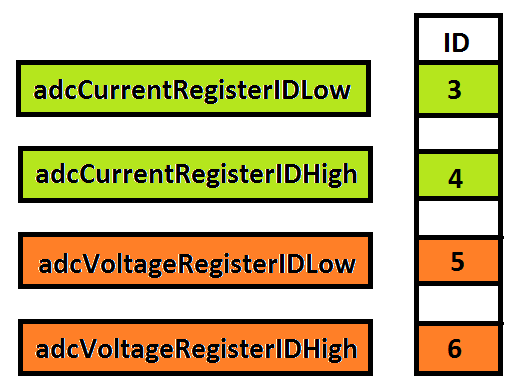
**Dossier de référence :** */VHDL/CanSat*

### 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 quatre registres pour permettre de récupérer la valeur du courant et de la tension. Chacune de ces deux valeurs étant partagée dans un registre contenant son MSB et l’autre son LSB.

Ces 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.

## Machine d’état du contrôleur ADC

**Dossier de référence :** */VHDL/MachineEtat*

La machine d’état comporte maintenant huit 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.

## Codage de la machine d’états

**Dossier de référence :** */VHDL/CanSat*

### Signaux de contrôle

Comme le code n’était pas pensé pour envoyer plusieurs commandes à la suite, ni des commandes plusieurs bytes. Trois 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' and adcCmdTimeWait = '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,adcSendCommand)

begin

if reset = '1' then

adcCommandWait <='0';

elsif rising\_edge(adcSendCommand) then

adcCommandWait <='1';

elsif rising\_edge(clock) then

adcCommandWait <='0';

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

adcCommandWait <='1';

end if;

end if;

end process;

Finalement, celui-ci attend les 24 coups d’horloge entre les commandes :

adcCmdTimeWait24: process(reset, modulatorClock, adcSending, adcSendCommand, adcState, adcConfigByteNbr)

begin

if reset = '1' then

adcTimecounter <= 0;

adcCmdTimeWait <= '0';

adcCounterEnable <= '0';

elsif adcSendCommand = '1' then

if adcState /= sendConfigCH or (adcState=sendConfigCH and adcConfigByteNbr >= 2) then

if rising\_edge(adcSending) then

adcCmdTimeWait <= '1';

elsif falling\_edge(adcSending) then

adcTimecounter <= 0;

adcCounterEnable <= '1';

adcCmdTimeWait <= '1';

elsif rising\_edge(modulatorClock) and adcCounterEnable = '1' then

adcCmdTimeWait <= '1';

adcTimecounter <= adcTimecounter + 1;

if adcTimecounter >= 24 then

adcTimecounter <= 0;

adcCounterEnable <= '0';

adcCmdTimeWait <= '0';

end if;

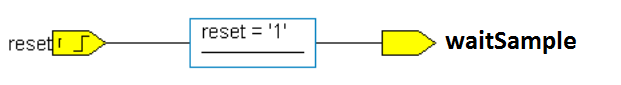
end if;

end if;

end if;

end process adcCmdTimeWait24;

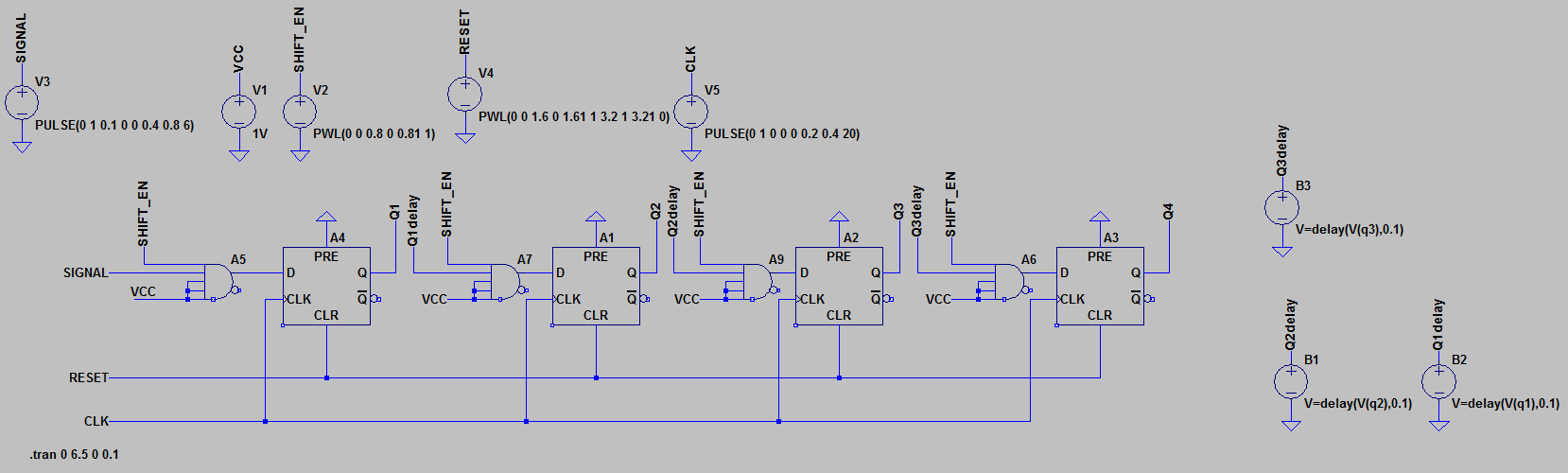
### Fiabilisation

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.

# Effets des radiations

## Registre à décalage pour réception d’un signal série

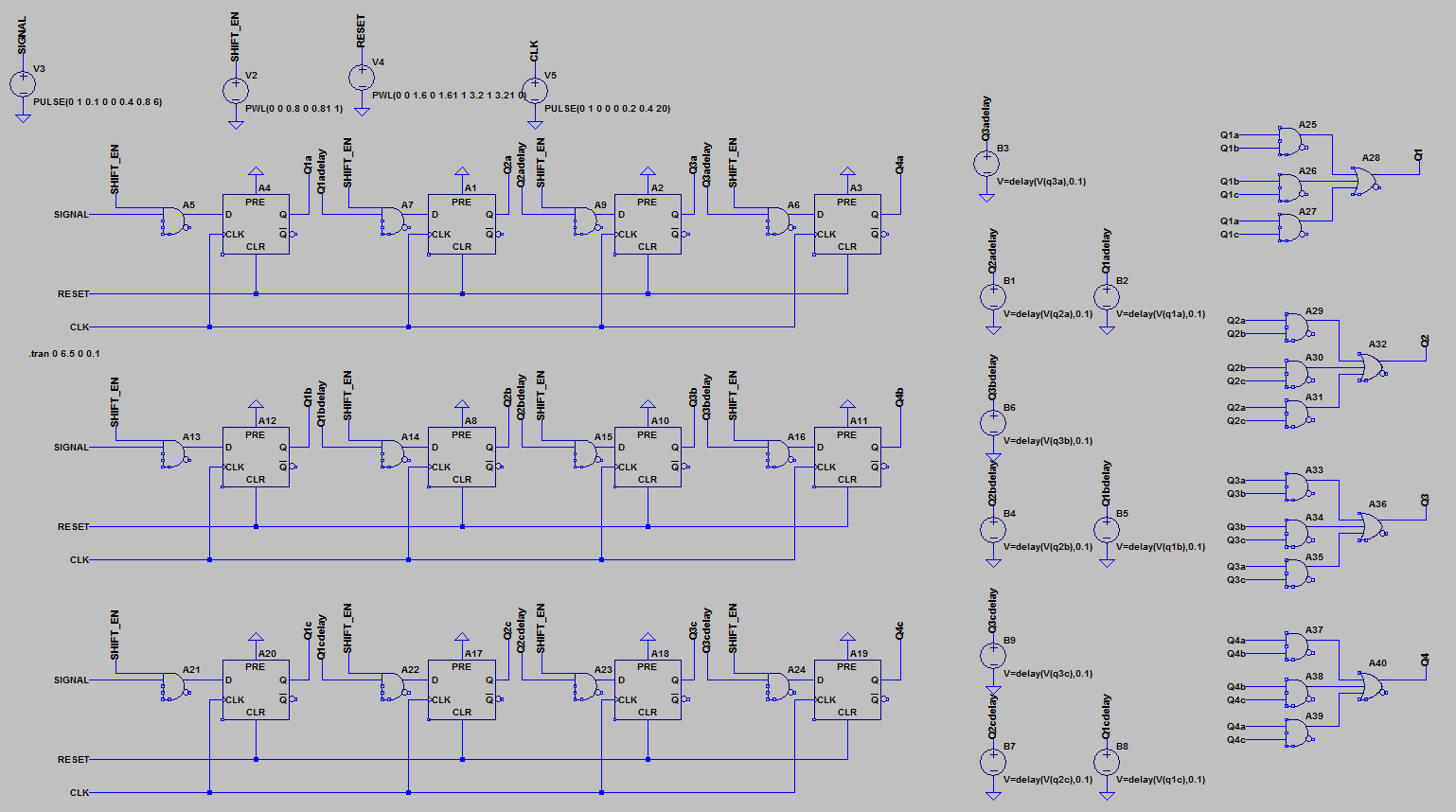
**Dossier de référence :** */EffetRadiations/RegistreADecalage*

Nous avons utilisé LTSpice pour simuler un registre à décalage 4 bits. Voici le circuit de base :

Nous avons constaté un problème avec le composant de la bascule D, il n’y a pas de temps de propagation entre l’entrée et la sortie, cela induit que le signal est propagé en un seul coup de clock jusqu’à la sortie Q4.

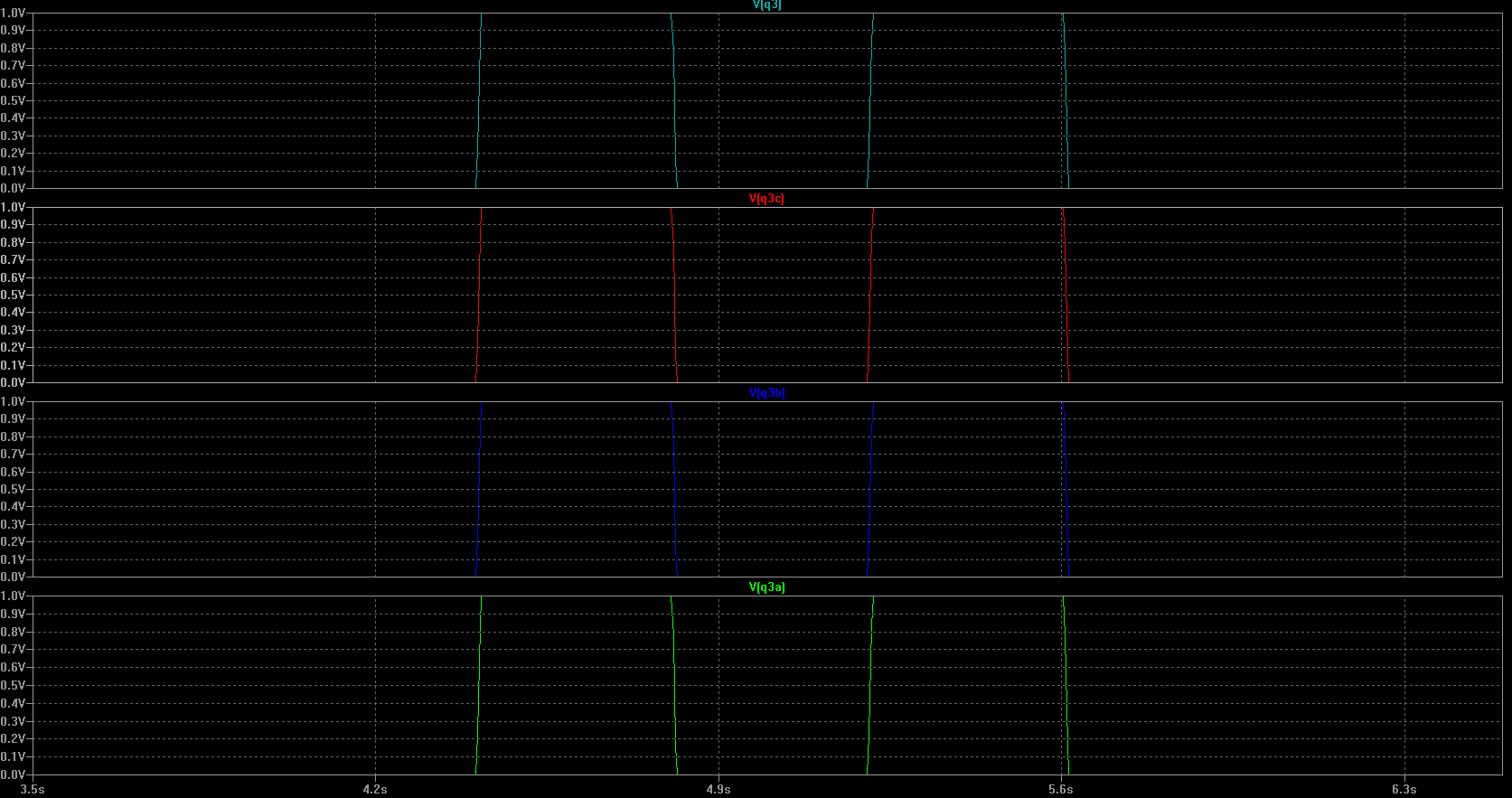
Pour résoudre ce problème, nous avons ajouté des délais de propagation à l’aide de sources BV.

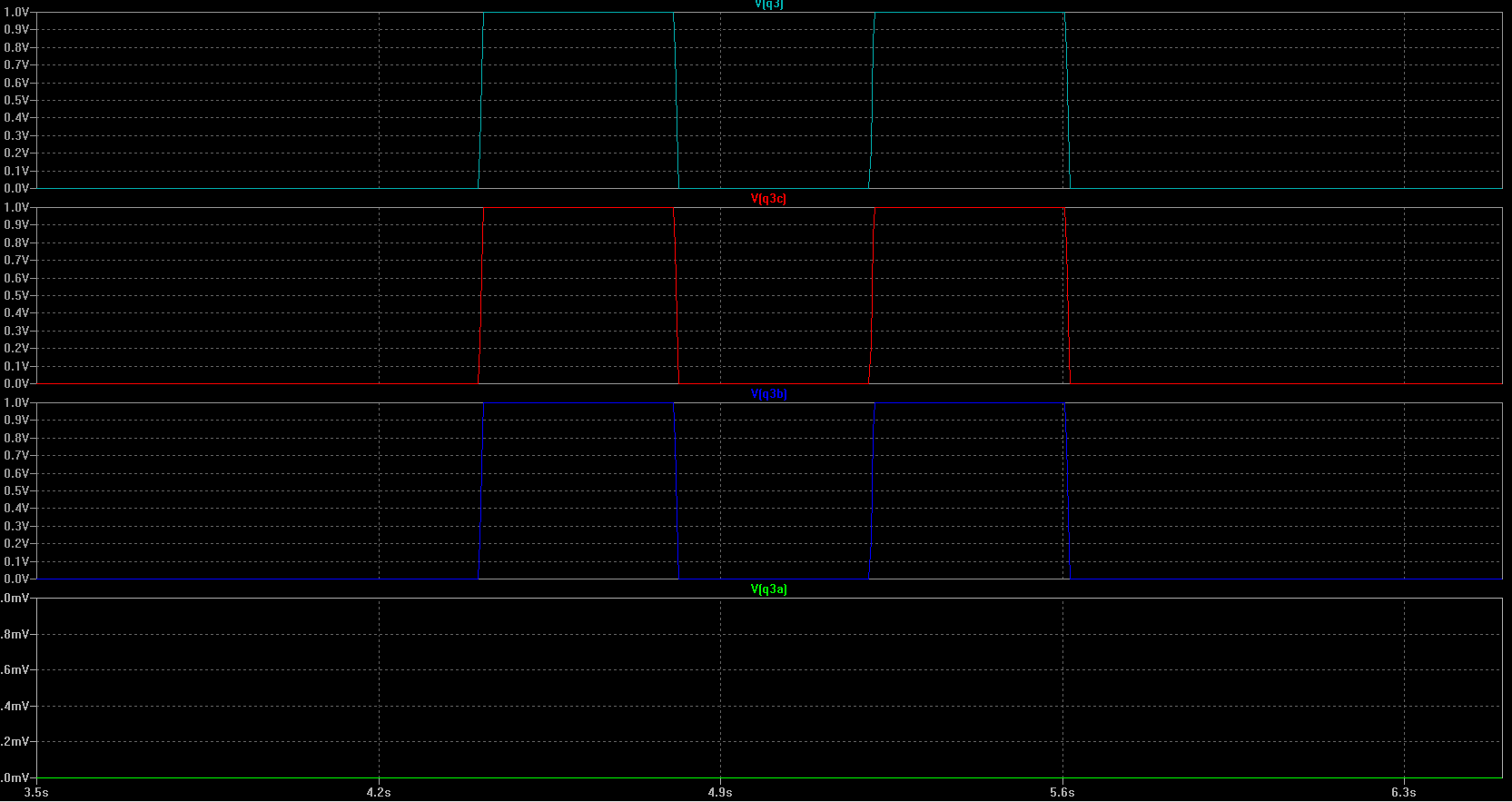
Voici une image validant le fonctionnement de notre registre à décalage. On voit que le signal n’est pas propagé en cas de reset ou lorsque le signal SHIFT\_ENABLE n’est pas actif.

Nous avons ensuite dupliqué notre circuit trois fois. Les signaux de contrôles sont les mêmes pour chacun des circuits, les sorties Q1, Q2 et Q3 de chaque circuit passent par une logique de vote à la majorité séparée pour définir le signal de sortie final.

Voici la table de Karnaugh utilisée pour définir le circuit logique pour le vote à la majorité.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | Q1ab | | | |
| 00 | 01 | 11 | 10 |
| Q1c | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |

Voici le résultat de la simulation sur la sortie Q3 lorsque tous les composants fonctionnent correctement

Si on simule une panne sur l’entrée Q3a, on peut constater que notre circuit fourni quand même la bonne valeur de sortie sur Q3.

Bien entendu, une panne de la bascule Q3 implique que la bascule Q4 est également affectée. La valeur de Q4 est également correctement corrigée.

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

Avec la triple redondance, on arrive à conserver des signaux corrects même si un des trois circuits ne fonctionne plus du tout.

C:\Users\Emilie\Documents\Git Hub\HiRel\Rapport\images\croix-rouge.pngPar contre, dans le cas de ce registre à décalage, si une ou plusieurs bascules ne fonctionnent plus dans deux registres à décalage différents, les sorties ne sont plus correctes.

## Machine d’état du contrôleur ADC avec contraintes SEU

**Dossier de référence :** */VHDL/CanSat*

Pour rendre notre machine d’état un peu plus robuste nous avons codé les états avec un codage 1 parmi m un seul bit est à 1 et se décale vers la gauche en fonction de la séquence suivante :

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

constant waitSample : fsm\_stateType := "00000001";

constant sendSDATAC : fsm\_stateType := "00000010";

constant sendConfigCH : fsm\_stateType := "00000100";

constant sendReadByCmd : fsm\_stateType := "00001000";

constant waitDataReady : fsm\_stateType := "00010000";

constant startRead : fsm\_stateType := "00100000";

constant waitRead : fsm\_stateType := "01000000";

constant reading : fsm\_stateType := "10000000";

Comme nous avons huit états, il nous faut huit bits pour coder les états. La correction de la séquence en cas d’erreur est représentée dans la figure suivante :

Le processus sequenceCorrection a été implémenté et contrôle que les états se suivent correctement. En cas d’erreur dans la séquence, la machine d’état est remise dans l’état waitSample.

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

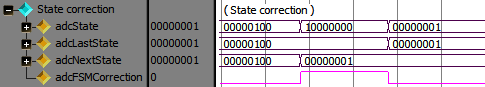
when sendConfigCH =>

if adcConfigured = '1' then

--adcState <= sendReadByCmd; //Correct state "00001000";

adcState <= "10000000"; // Wrong state, reading

end if;

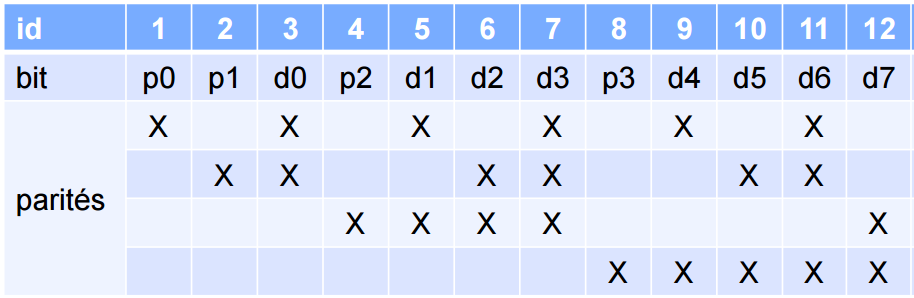
Voici le résultat de la simulation, on voit que l’état est corrigé correctement :

C:\Users\Emilie\Documents\Git Hub\HiRel\Rapport\images\image_large.pngLa machine d’état se remet dans un état connu en cas de modification inopinée d’un bit du vecteur d’état. Lors de la détection de l’erreur indiquée par le signal adcFSMCorrection, le signal adcState pilotant la machine d’état est remis à l’état waitSample au cycle d’horloge suivant.

## Décodeur de Hamming

**Dossier de référence :** */VHDL/CanSat*

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 huit bits de données. Selon le tableau ci-dessous, nous devons donc ajouter 4 bits de parité au codage des états. On dit que c’est un code (12,8). Les croix du tableau indiquent où appliquer les XOR pour calculer les bits de parité.

Voici le codage des états en 1 parmi m avec en plus les bits de parité du codage de Hamming

constant waitSample : hamming\_stateType := "000000010011";

constant sendSDATAC : hamming\_stateType := "000000100101";

constant sendConfigCH : hamming\_stateType := "000001000110";

constant sendReadByCmd : hamming\_stateType := "000010000111";

constant waitDataReady : hamming\_stateType := "000100001001";

constant startRead : hamming\_stateType := "001000001010";

constant waitRead : hamming\_stateType := "010000001011";

constant reading : hamming\_stateType := "100000001100";

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 donné 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.

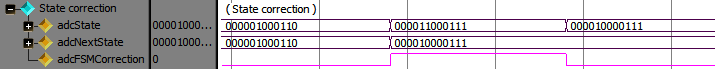
when sendConfigCH =>

if adcConfigured = '1' then

--adcState <= sendReadByCmd; //Correct state "000010000111";

adcState <= "000011000111"; //Wrong state, modify one bit

end if;

Voici le résultat de la simulation, on voit que l’état est corrigé correctement, la machine d’état recharge l’état correct au flanc montant d’horloge suivant :

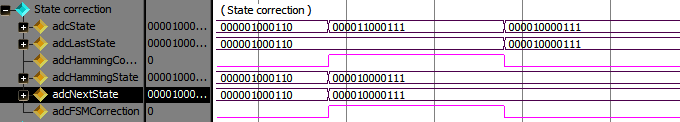
### *Réponse aux questions des slides*

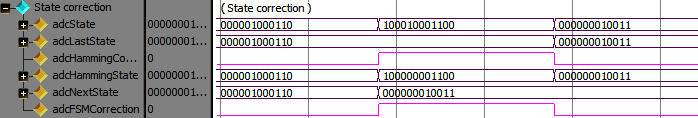
Avec le codage de Hamming, on peut corriger **une** erreur sur le vecteur d’état.

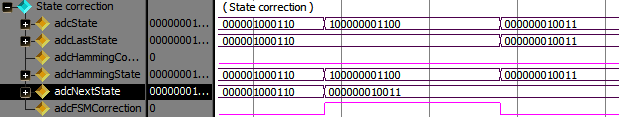
On peut détecter dans notre cas, 12 erreurs sans pour autant les corriger. On peut donc dire qu’on peut détecter autant d’erreurs qu’il y a de bits dans le vecteur d’état (parité + donnée).

### Mise en commun du codage de Hamming avec correction de séquence

Nous avons ensuite modifié les processus développés précédemment pour les mettre en commun. C’est-à-dire pouvoir corriger un bit du vecteur d’état et revenir dans un état connu si la séquence n’est pas bonne

Cette simulation montre que l’état est correctement modifié lors qu’un bit du vecteur d’état change mais que la séquence est correctement suivie :

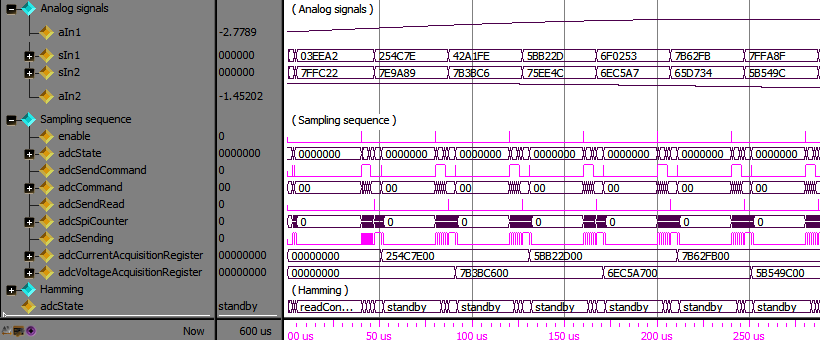
Cette simulation montre que l’on revient dans l’état waitSample lorsqu’un bit du vecteur d’état change mais que la séquence n’est pas correctement suivie :

On peut également encore faire le test si aucun bit du vecteur d’état ne change, mais que la séquence n’est pas suivie correctement.

### Transmission de l’erreur au registre de status

Dans l’exemple de code donné, un registre de status est implémenté. Nous avons fait en sorte que l’état du signal adcFSMCorrection se retrouve sur le bit 1 de ce registre pour pouvoir transmettre sur le bus AMBA s’il y a eu une erreur dans les états.

## Simulation de l’ensemble 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*.

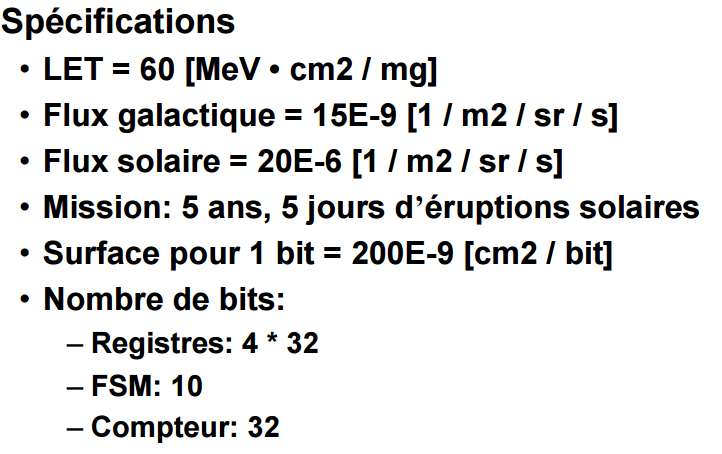
À ce stade on a donc :

C:\Users\Emilie\Documents\Git Hub\HiRel\Rapport\images\image_large.pngUne machine d’état robuste dont l’état peut être corrigé en cas de perturbation extérieure et qui est résistante aux SEU.

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.

## Estimation du taux de SEU par jour

**Dossier de référence :** */EffetRadiations*

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 la suivante :

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.

# Conception de circuit numériques

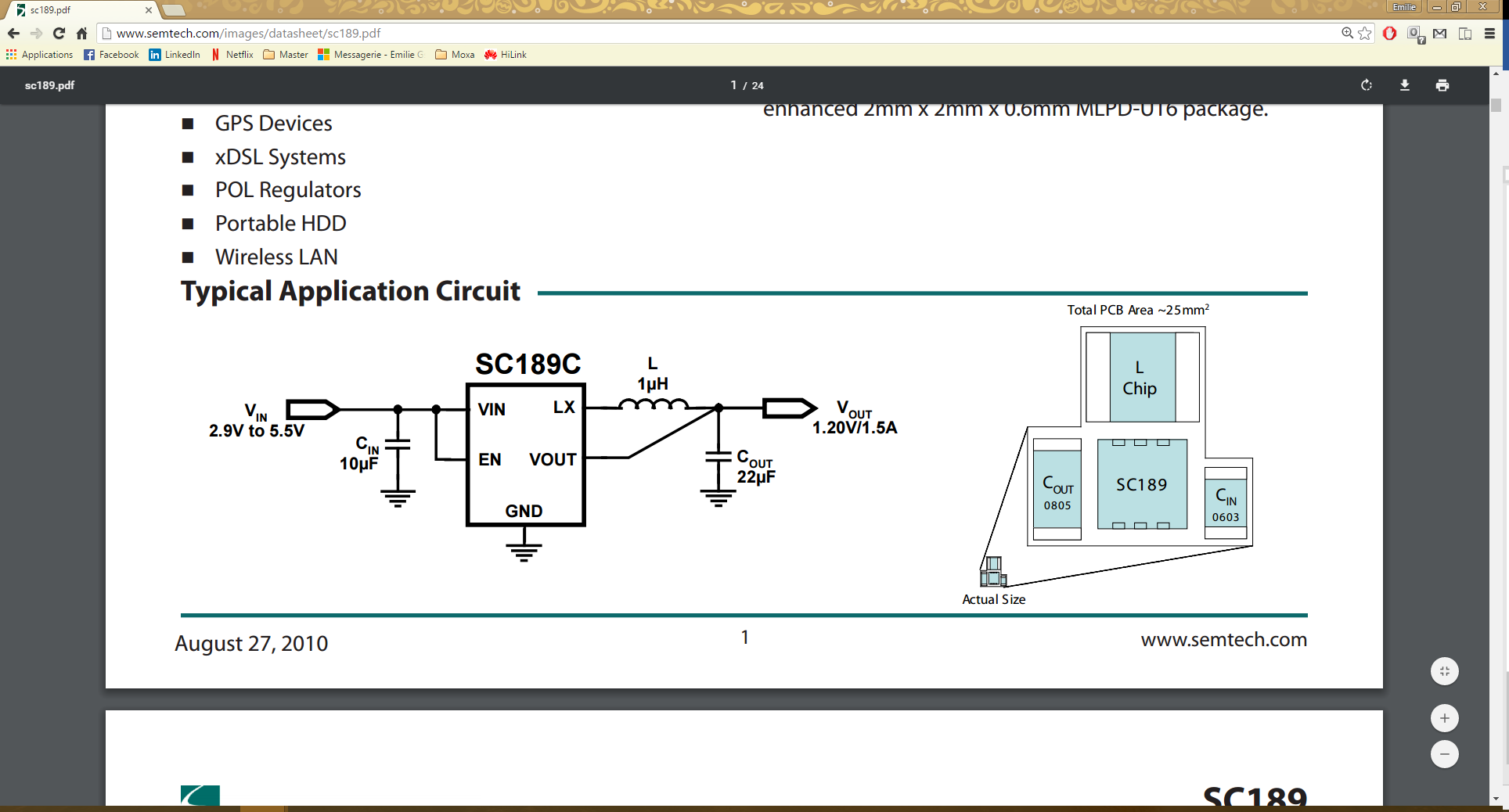
## Alimentation FPGA

**Dossier de référence :** */ConceptionCircuitNumerique*

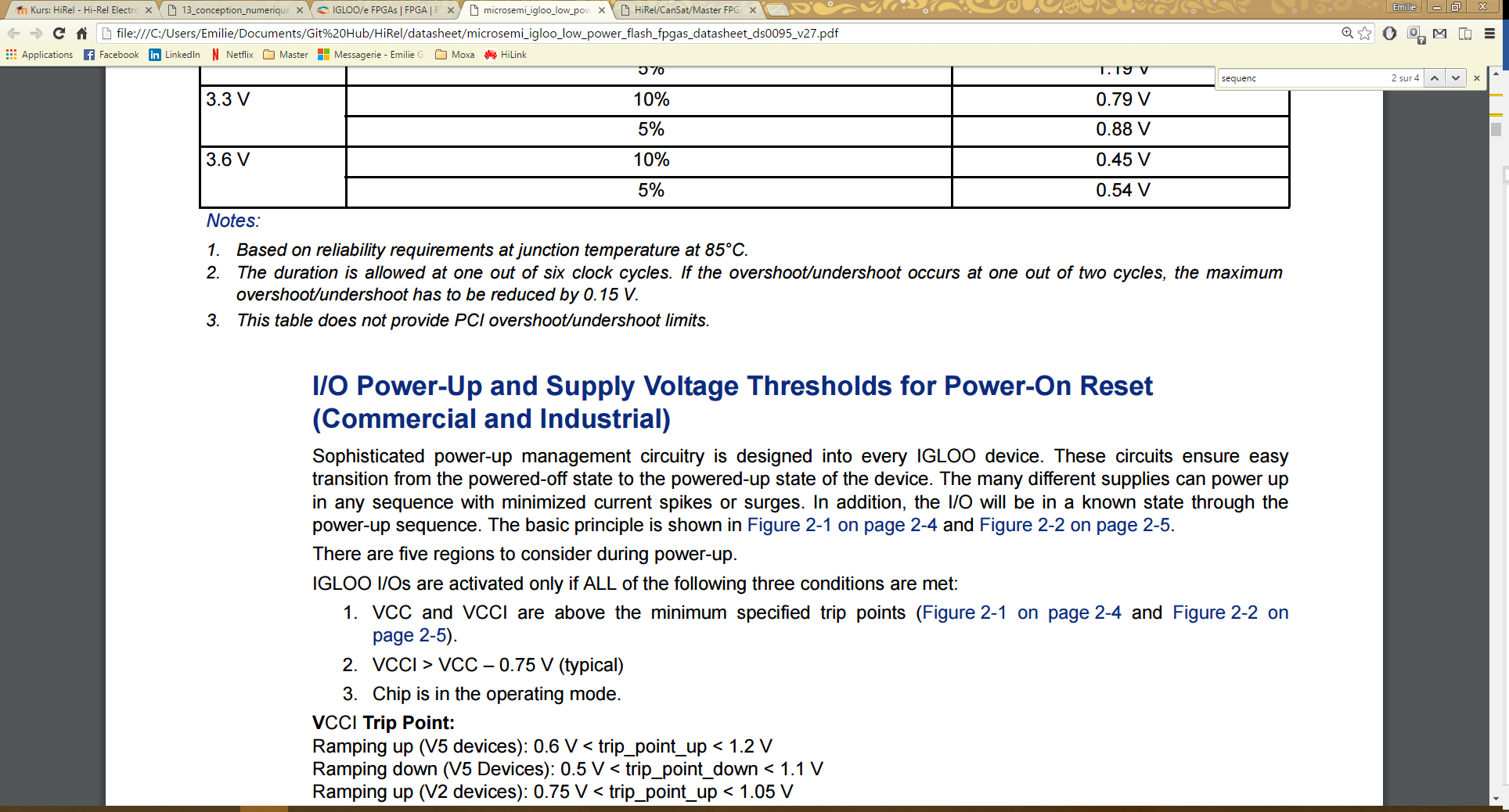
Voici la liste des tensions d’alimentation et courants nécessaire pour la FPGA Microsemi IGLOO Low Power Flash. Les courants ont été pris pour le modèle AGL015.

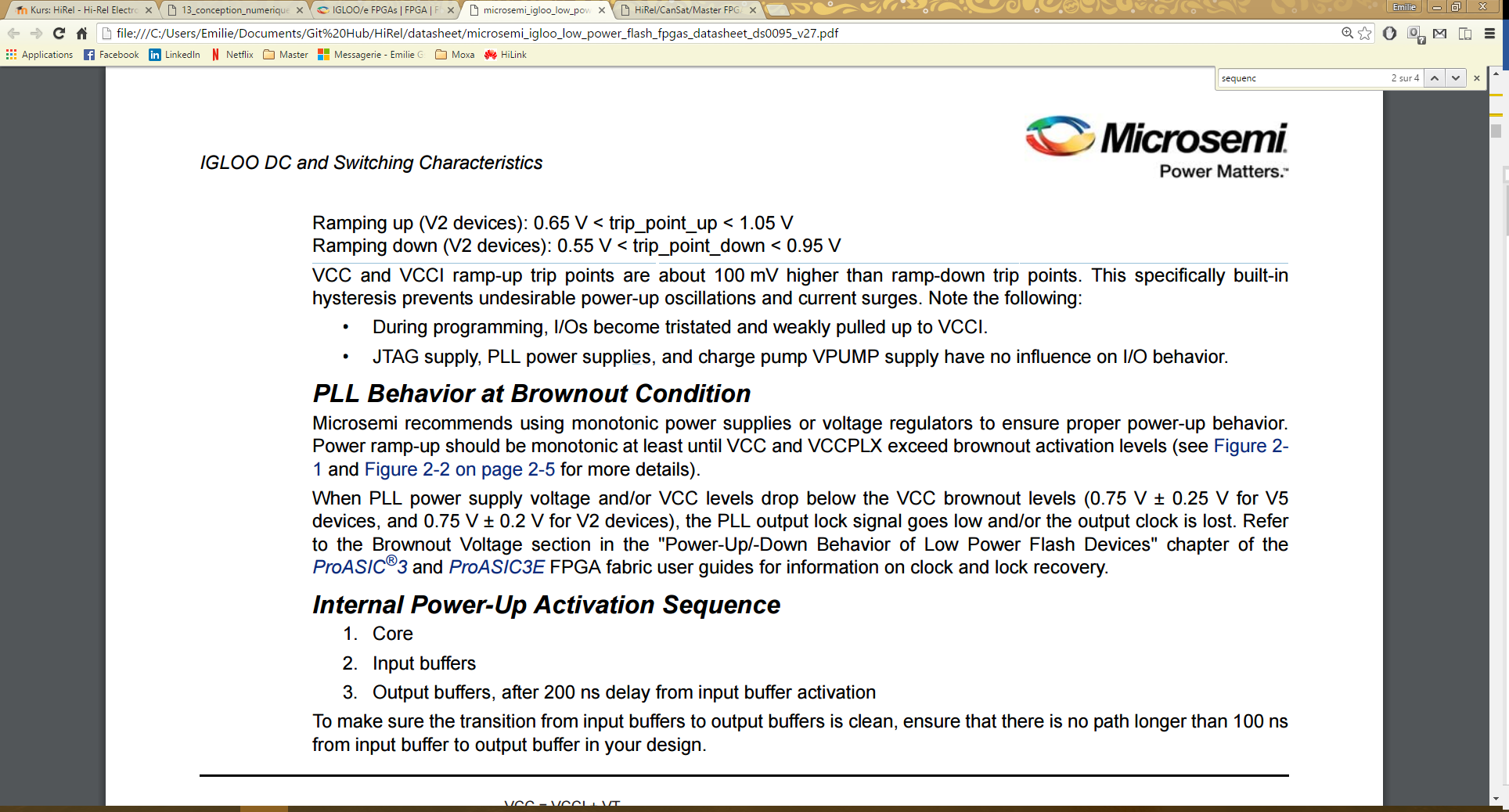
### Circuit d’alimentation

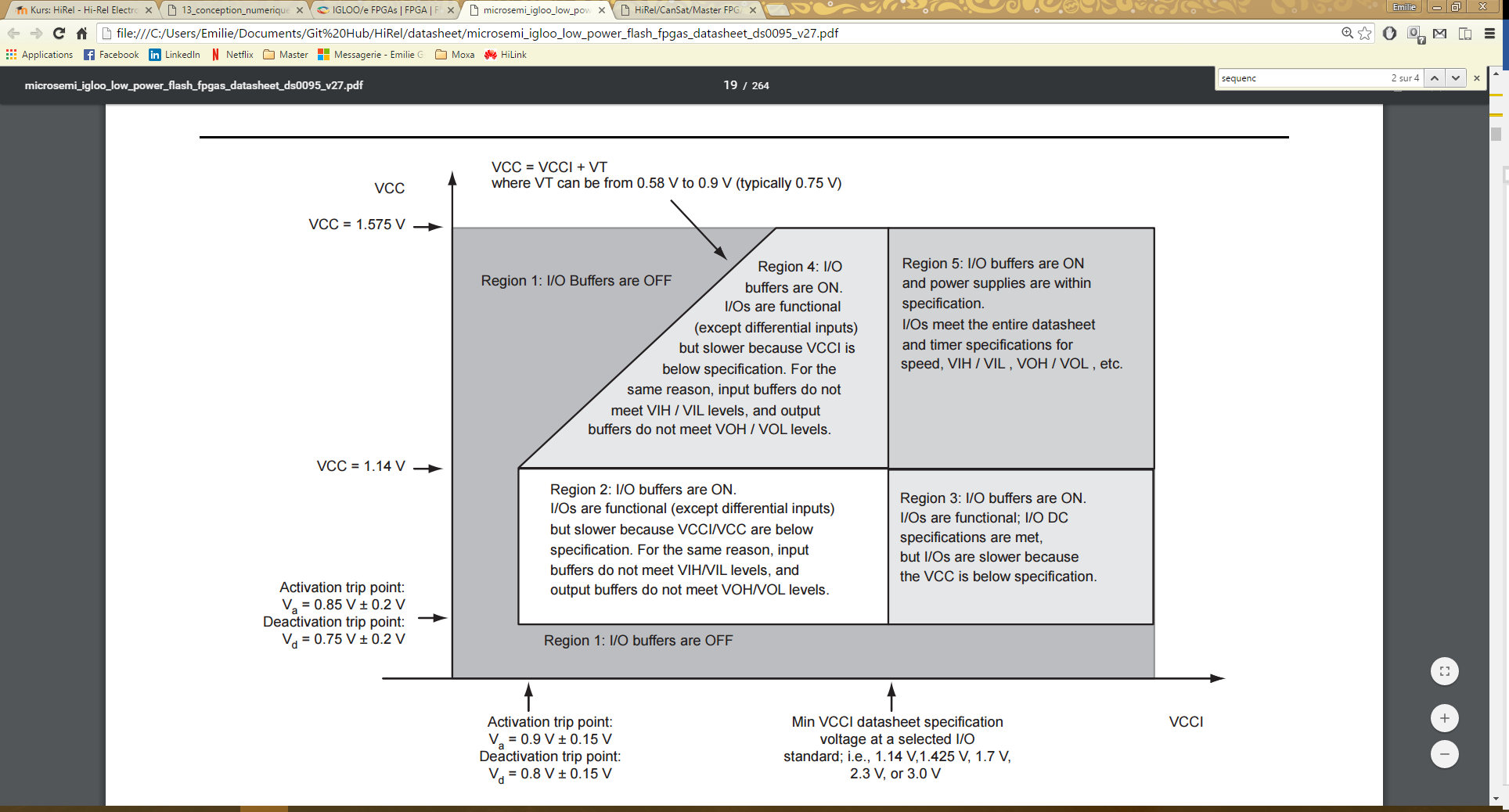
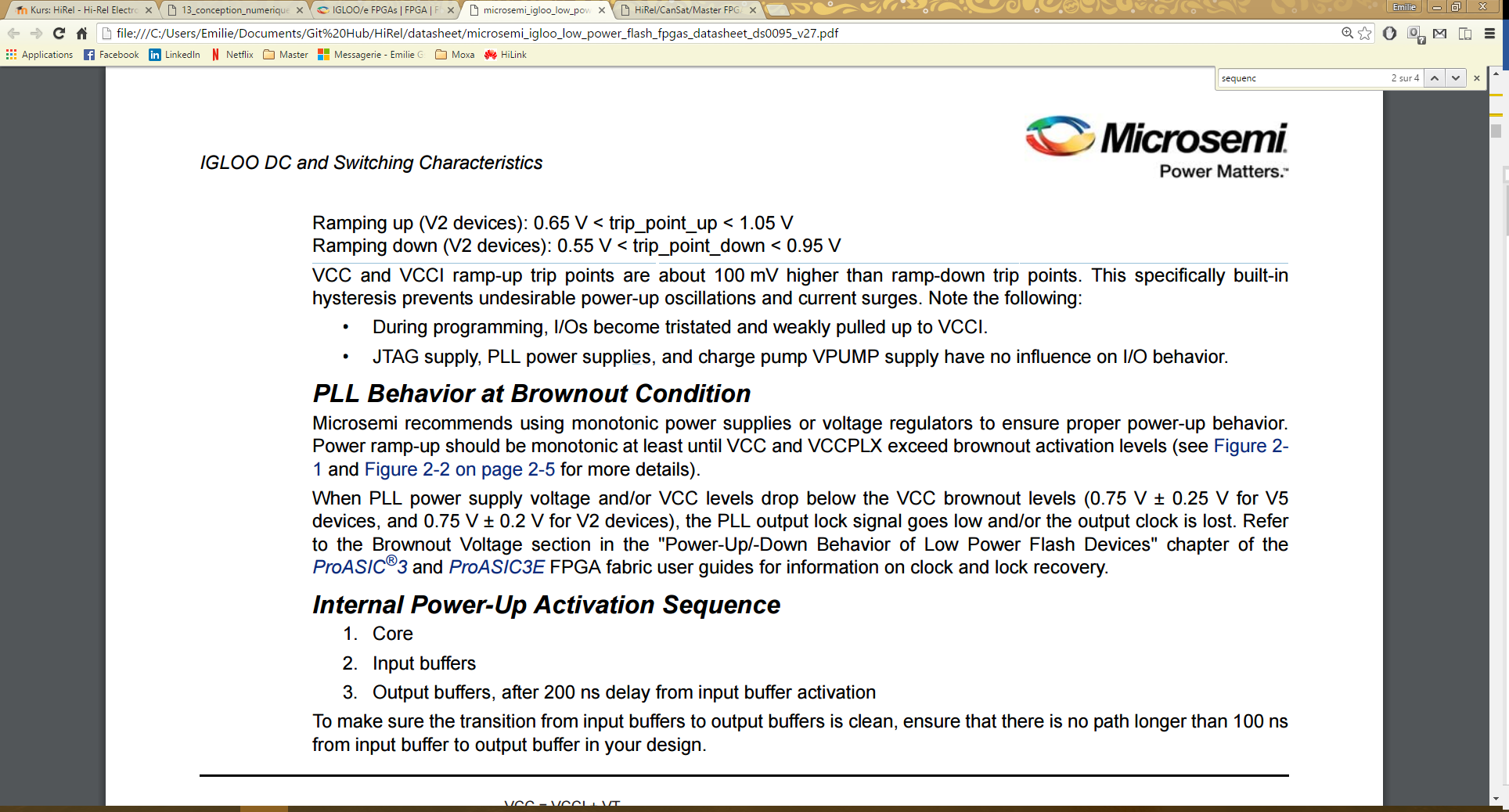
À l’aide du tableau ci-dessus, on observe qu’une seule tension d’alimentation peut suffire à alimenter tout le circuit. On pourrait alimenter l’ensemble à 1.2V. Cela donnerait le tableau de courant plus précis suivant :

Avec les 1.2V, on peut reprendre le circuit d’alimentation déjà existant dans la schématique de la FPGA du CanSat. Le DC/DC utilisé fournit 1.2V et 1.5A max, ce qui est suffisant pour alimenter notre circuit.

### Séquencement

Voici les informations sur la séquence de démarrage trouvée dans le datasheet de la FPGA.





En résumé, toutes les entrées-sorties sont dans un état connu durant toute la séquence de démarrage du circuit. Lors de la programmation de la FPGA, toutes les entrées-sorties passent dans l’état tri-state.

## Entrées / Sorties du circuit

Voici une représentation du top-level de notre circuit pilotant le convertisseur A/D. Si nous avions eu HDL Designer, on aurait pu visualiser l’intérieur de ce bloc et voir ainsi comment nos différents processus sont reliés entre eux. On ne peut malheureusement pas le montrer.

Dans notre développement, nous avons fait en sorte de synchroniser un maximum de processus sur les signaux de reset et de clock. Toutes nos entrées ont un état stable après un reset. Le clock est le même dans tous nos processus.

# Vérification de circuits numériques

## Matrice de conformité

**Dossier de référence :** */VerificationCircuitNumerique* et */VHDL/CanSat*

Le document contient dans sa première feuille la matrice de conformité de notre périphérique de contrôle. Les feuilles suivantes contiennent des captures d’écran ainsi que des remarques sur les différents tests réalisés.

Le code de notre périphérique a dû être adapté, car certains tests ont montré des disfonctionnements. Le banc de test *ahbAds1282\_tester\_test.vhd* a également été modifié pour permettre d’effectuer nos différents tests.

# Tests fonctionnels

## Environnement de test

**Dossier de référence :** */*

## Plan de test

**Dossier de référence :** */*

## Procédure de test

**Dossier de référence :** */*

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. Source de l’image : <http://images.all-free-download.com/images/graphicthumb/sign_alert_98028.jpg> [↑](#footnote-ref-4)
5. Lien : <http://www.expertmultimedia.ch/ressources/graphisme-symboles-logos/symboles-1/symbole-vu/image_large> [↑](#footnote-ref-5)
6. Lien : <https://esgmsc.files.wordpress.com/2014/04/croix-rouge.png> [↑](#footnote-ref-6)
7. Source des images : Datasheet du ADS1282 [↑](#footnote-ref-8)