

Filière Systèmes industriels

Orientation Infotronics

Travail de bachelor

Diplôme 2021

Jean Nanchen

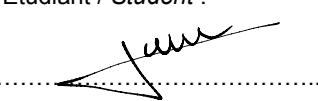
*Carte de développement FPGA pour une
électronique de puissance*

- Professeur
Prof. François Corthay
- Expert
André Rotzetta
- Date de la remise de rapport
20.08.2021



Filière / Studiengang SYND	Année académique / <i>Studienjahr</i> 2020/21	No TD / Nr. DA IT/2021/56
Mandant / <i>Auftraggeber</i> <input checked="" type="checkbox"/> HES—SO Valais <input type="checkbox"/> Industrie <input type="checkbox"/> Etablissement partenaire <i>Partnerinstitution</i>	Etudiant / <i>Student</i> Jean Nanchen	Lieu d'exécution / <i>Ausführungsort</i> <input checked="" type="checkbox"/> HES—SO Valais <input type="checkbox"/> Industrie <input type="checkbox"/> Etablissement partenaire <i>Partnerinstitution</i>
	Professeur / <i>Dozent</i> François Corthay	
Travail confidentiel / <i>vertrauliche Arbeit</i> <input type="checkbox"/> oui / ja ¹ <input checked="" type="checkbox"/> non / nein	Expert / <i>Experte</i> (<i>données complètes</i>) André Rotzetta HEIA-FR, Bd de Pérrolles 80, 1700 Fribourg	

Titre / <i>Titel</i> FPGA developing board for power electronics
Description / <i>Beschreibung</i> Le groupe "Electronique industrielle et entraînements" de la HEI a développé un système de prototypage pour des systèmes électroniques de puissance. Le but de ce travail est de développer une carte FPGA pour le contrôle de la partie de puissance et de réaliser un démonstrateur avec ce système.
Objectifs / <i>Ziele</i> <ul style="list-style-type: none"> — Adapter la carte existante "FPGA-Rack" pour pouvoir l'utiliser dans le système du groupe "Electronique industrielle et entraînements" — Mettre en place un système de développement FPGA pour le contrôle de l'électronique de puissance — Réaliser un démonstrateur pour le pilotage d'une charge de puissance.

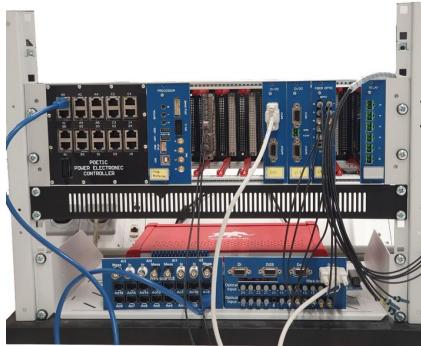
Signature ou visa / <i>Unterschrift oder Visum</i> Responsable de l'orientation / filière <i>Leiter der Vertiefungsrichtung / Studiengang:</i>  Etudiant / <i>Student</i> : 	Délais / <i>Termine</i> Attribution du thème / <i>Ausgabe des Auftrags</i> : 10.05.2021 Présentation intermédiaire / <i>Zwischenpräsentation</i> 07 – 08.06.2021 Remise du rapport / <i>Abgabe des Schlussberichts</i> : 20.08.2021, 12:00 Exposition / <i>Ausstellung der Diplomarbeiten</i> : 25 – 27.08.2021 (si autorisé / falls genehmigt) Défense orale / <i>Mündliche Verfechtung</i> : 30.08 – 09.09.2021
---	--

¹ Par sa signature, l'étudiant-e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de diplôme.

Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.

Carte de développement FPGA pour une électronique de puissance

Diplômant/e Jean Nanchen



Travail de diplôme | édition 2021 |

Filière
Système Industriel

Domaine d'application
Infotronique

Professeur responsable
François Corthay
francois.corthay@hevs.ch

Objectif du projet

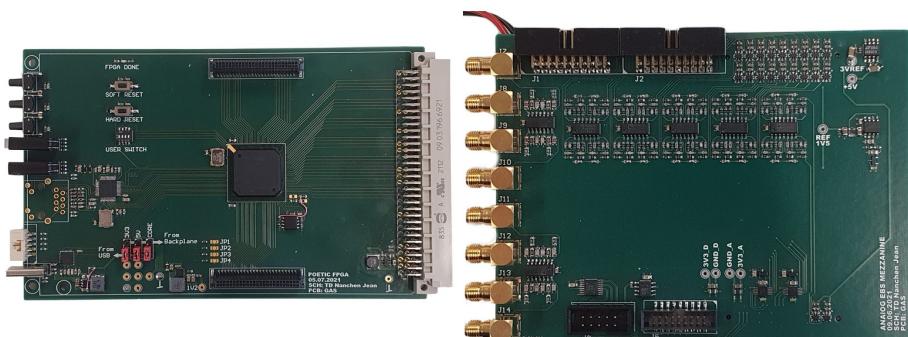
L'objectif de ce travail de Bachelor est de réaliser une carte électronique de développement FPGA. Elle doit s'intégrer au système de prototypage d'électronique de puissance *Poetic* développé par la HES-SO.

Méthodes | Expériences | Résultats

Un premier prototype de cette unité de contrôle a été produit durant ce travail de Bachelor. Cette unité est divisée en deux cartes électroniques. L'une d'elles contient une FPGA (Réseau de portes logiques programmables), un connecteur USB-C, un connecteur Ethernet, des boutons ainsi que des LED. L'autre PCB contient des convertisseurs analogiques numériques pouvant acquérir des signaux analogiques, des convertisseurs numériques analogiques permettant de convertir des signaux internes de la FPGA en signaux analogiques. Les deux PCB s'assemblent à l'aide de deux connecteurs mezzanine.

Une architecture software a été développée dans la FPGA. Celle-ci inclus des interfaces pouvant communiquer avec les composants des deux cartes électroniques. L'architecture est aussi composée d'un système de régulation à boucle fermée à l'aide d'un régulateur. Ce système permet typiquement de pouvoir réguler la vitesse d'un moteur DC ou Brushless DC. L'utilisateur peut envoyer la vitesse de consigne à travers le port série de son ordinateur.

Un démonstrateur a été réalisé grâce au simulateur Typhoon HIL et permet de réguler un moteur Brushless DC.



PCB FPGA V1.0
Vue du dessus du PCB
Dimensions 100mm x 160mm

PCB Mezzanine V1.0
Vue du dessus du PCB
Dimensions 100mm x 133.2mm

Information à propos de ce rapport

Information de contact

Auteur : Jean Nanchen
Etudiant Bachelor
HES-SO // Valais Wallis
Suisse
Email : jean.nanchen@students.hevs.ch

Déclaration d'honneur

Je, soussigné, Jean Nanchen, déclare que le travail présenté est le résultat d'un travail personnel. Je certifie que je n'ai pas eu recours au plagiat ou à d'autres formes de fraude. Toutes les sources d'information utilisées et les citations de l'auteur ont été clairement mentionnées.

Lieu et date : Sion, le 19 août 2021

Signature :



Validation

Accepté par la HES-SO // Valais Wallis (Suisse, Sion) sur une proposition de :

Prof. François Corthay, responsable du projet de thèse

André Rotzetta, HEIA-FR, Bd de Pérolles 80, 1700 Fribourg, expert principal

Lieu et date : _____

Table des matières

Table des matières	vi
Liste des figures	x
Liste des tableaux	xii
1 Introduction	1
1.1 Aperçu	1
1.2 But du projet	2
2 Analyse	3
2.1 Spécifications	3
2.1.1 Cahier des charges	3
2.2 Fonctionnement du rack Poetic	3
2.3 Carte POETIC Processor V2	4
3 Hardware	7
3.1 Schéma bloc	7
3.2 ADC	8
3.2.1 Caractéristiques des ADC internes de la carte POETIC Processeur	8
3.2.2 Fréquence d'échantillonnage minimale	9
3.2.3 Choix de l'ADC	9
3.2.4 Différentiel vers Signle-Ended	10
Simulation	10
3.2.5 Communication avec la FPGA (signal level)	12
3.2.6 Schématique	13
3.3 DAC	14
3.4 FPGA	15
3.4.1 Choix de la FPGA	15
3.4.2 Tailles & compatibilités	15
3.4.3 Bank 0-3	16
3.4.4 Découplage	16
3.4.5 Quartz	17
Fréquence maximale	18
3.4.6 Flash	18
3.4.7 Mode de configuration	20
3.4.8 HSWAPEN	20
3.4.9 INIT_B	21
3.4.10 Reset	21

Table des matières

3.4.11 Indicateur de configuration	22
3.5 Interface de communication	23
3.5.1 Communication série (USB to UART)	23
Connecteur USB-C	23
3.5.2 SPI LVDS	24
3.5.3 PHY Ethernet	24
3.6 Alimentations	25
3.6.1 Schéma bloc	25
3.6.2 1V2 Core	25
Choix de l'inductance	26
3.6.3 5V vers 3V3	26
Choix de l'inductance	27
3.6.4 3V3 vers 5V	27
Choix de l'inductance	27
3.6.5 Alimentation des ADCs	28
3.6.6 Points de tests	29
3.7 Interfaces utilisateurs	29
3.7.1 LED	29
3.7.2 Boutons	30
Boutons resets	30
3.7.3 DIP Switchs	30
3.7.4 Jumper Spare	31
3.8 Schématique	32
3.9 Coûts de production	32
3.10 PCB	32
3.10.1 FPGA Board	32
3.10.2 PCB Mezzanine	33
3.10.3 Spécificités du montage	33
3.10.4 Routage	34
3.10.5 Assemblage	34
3.11 Erreurs hardware et modifications effectuées	34
3.12 Face avant	35
4 Architecture software	37
4.1 Structure	37
4.2 ADC	37
4.2.1 Interface pour les ADC	38
4.2.2 Modèle des ADC	39
4.2.3 Simulation	39
4.3 DAC	40
4.3.1 Interface pour les DAC	41
4.3.2 Simulation	43
4.4 Régulation & modulateur PWM	44
4.5 Modèle moteur DC	44
4.5.1 Simulation	45
4.6 Générateur de clock	45
4.6.1 Simulation	45
4.7 Définir une consigne à travers l'UART	46

Table des matières

4.7.1	Interface UART série	46
4.7.2	Contrôleur	46
4.7.3	ASCII série vers bus parallèle	47
4.7.4	Simulation	48
4.8	BLDC Controller	48
4.8.1	Pilotage d'un moteur BLDC	48
4.8.2	Schéma trapézoïdal	49
4.9	Régulation d'un moteur DC	50
4.9.1	Simulation	50
5	Démonstrateur	53
5.1	Configuration du rack Poetic	53
5.2	Schématique dans le simulateur Typhoon HIL	54
5.3	Simulation	55
5.3.1	Réponse indicielle	55
5.3.2	Mesure de la PWM générée par la FPGA	56
6	Conclusion	57
6.1	Synthèse	57
6.2	Améliorations	58
6.3	Difficultés rencontrées	59
A	Analyse détaillée des convertisseurs A/D	61
A.1	Types d'architectures des ADCs	61
A.1.1	Sigma Delta	61
A.1.2	SAR	61
A.1.3	Pipeline	62
A.2	Latence	63
A.3	Types d'interfaces	63
B	Schéma bloc complet	65
C	Fichiers de commandes	67
D	Schématique PCB FPGA Rack V1.0	71
E	Schématique PCB Mezzanine	85
F	Schématique PCB FPGA	93
G	Interface de l'ADC ADS7886	101
H	Modèle de l'ADC ADS7886	103
I	Interface du DAC DACXX4S085	105
J	Régulateur PI	107
K	Modèle moteur DC	109
L	Générateur de clock	111

Table des matières

M Contrôleur UART	113
N Décodeur ASCII (UART)	117
O Contrôleur BLDC	119
P Fichier UCF	121
Q Face avant	123
R Mise en place du démonstrateur	125
R.1 Matériel requis	125
R.2 Câblage	126
R.3 Installations des fichiers	127
R.4 Flash BLDC POETIC sur la FPGA	128
R.5 Simulateur HIL Typhoon	132
R.6 Envoie de la consigne à travers l'UART	135
Bibliographie	137
Acronymes	139
Glossaire	141

Table des figures

1.1	Rack Poetic avec un simulateur Typhoon HIL 402	1
2.1	Régulation en boucle fermée d'un moteur à l'aide du rack Poetic	3
2.2	Carte POETIC Processeur v2 qui équipe les racks POETIC	4
2.3	Schéma bloc de la carte Poetic Processor V2	5
3.1	Schéma bloc du module FPGA dans l'environnement POETIC	7
3.2	Caractéristiques des ADC dans le processeur TMS320F28377SPTPT[4]	8
3.3	Comparatif des différentes architectures ADC[6]	9
3.4	ADC AD7886 qui équipe la carte FPGA Poetic[7]	10
3.5	Montage ampli-op différentiel vers asymétrique	10
3.6	Simulation du montage soustracteur passe-bas sur LTSpice	11
3.7	Analyse des signaux Uout, vinp et vimn	11
3.8	Réponse en fréquence du montage ampli op soustracteur	11
3.9	Spécifications des entrées sorties de la FPGA[8]	12
3.10	Spécifications des entrées sorties de l'ADC[7]	12
3.11	Niveaux de tensions des entrées sorties de la FPGA et de l'ADC	12
3.12	Convertisseur DAC 4CH	14
3.13	FPGA Spartan 6 LX150 en boîtier FGG484 (XC6SLX150-N3FGG484C)	15
3.14	Positionnement des banks sur le Spartan 6[10]	16
3.15	Condensateurs requis pour les différentes FPGA[11]	17
3.16	Caractéristiques des condensateurs de découplage[11]	17
3.17	Quartz de 100MHz qui cadence la FPGA	18
3.18	Fréquences maximales et minimales de clock sur le Spartan 6[8]	18
3.19	Flash SPI	19
3.20	Application type d'une FPGA Spartan 6 avec une flash SPI[12]	19
3.21	Mode de configuration de la FPGA Spartan 6[12]	20
3.22	Résistances de pull-up et de pull-down ordonnant la configuration SPI à la FPGA	20
3.23	Configuration de la pin HSWAPEN	21
3.24	Configuration de la pin INIT_B	21
3.25	Boutons hard reset et soft reset	22
3.26	LED affichant l'état de la configuration de la FPGA	22
3.27	USB vers UART, application type du FT232RL	23
3.28	Connecteur USB-C	23
3.29	LVDS Driver	24
3.30	LVDS Receiver	24
3.31	Schéma bloc de l'alimentation des PCB FPGA et Mezzanine	25
3.32	5V/3V3 vers 1V2 pour le FPGA Core	26
3.33	5V USB vers 3V3 USB	27

Table des figures

3.34	3V3 Backplane vers 5V Backplane	27
3.35	Inductance recommandée pour le convertisseur DC/DC RP402x[14]	27
3.36	Référence de tension 3V alimentant les ADC	28
3.37	Spécifications d'alimentation de l'ADC[7]	28
3.38	Tension dropout de la référence de tension 3V[15]	29
3.39	Points de tests sur le PCB FPGA	29
3.40	Deux indicateurs LED disposant de 4 LED bicolores superposées	30
3.41	Boutons poussoirs en façade	30
3.42	DIL Switch SMD	31
3.43	Jumper SMD de réserve	31
3.44	PCB FPGA	32
3.45	PCB Mezzanine	33
3.46	Assemblage du PCB FPGA et du PCB Mezzanine	34
3.47	Modification effectuée sur le buck (V1.0)	35
4.1	Schéma bloc de l'architecture	37
4.2	Diagramme de temps de l'interface série du ADS7886[7]	38
4.3	Interface pour l'ADS7886 & clock de 20MHz	38
4.4	Modèle simulant le convertisseur A/D ADS7886	39
4.5	Simulation de l'interface de l'ADC et du modèle de l'ADC	39
4.6	Diagramme de temps de l'interface série du DACXX4S085[16]	40
4.7	Paquet 16 bits envoyé au convertisseur D/A 12 bits DAC124S085[16]	40
4.8	Paquet 16 bits envoyé au convertisseur D/A 8 bits DAC084S085[17]	41
4.9	Interface DAC avec son clock de 40MHz et ses bits de configurations	42
4.10	Simulation de l'interface DAC (8 bits)	43
4.11	Regulateur PI, modulateur PWM et un contrôleur BLDC pour un moteur brushless DC (BLDC)	44
4.12	Modèle simulant un moteur DC	44
4.13	Saut indiciel sur le modèle du moteur DC	45
4.14	Générateur de clock à 50MHz	45
4.15	Simulation du générateur de clock	45
4.16	UART, avec son contrôleur et son décodeur	46
4.17	State machine du bloc «uartController»	47
4.18	Envoi de «S2730e» sur le port UART, pour définir la consigne à 2730	48
4.19	Trois paires de MOSFET pilotant un moteur brushless DC	49
4.20	Bloc logique «BLDCController» caractérisant le schéma de commutation trapézoïdale	49
4.21	Banc de test d'une régulation d'un moteur DC	50
4.22	Réponse indicielle avec le régulateur P, puis activation du I qui corrige l'erreur statique	51
4.23	Réponse indicielle avec le régulateur PI	51
5.1	Photo du rack Poetic avec les modules nécessaires au démonstrateur	53
5.2	Schéma bloc du rack Poetic connecté au simulateur HIL pour la régulation d'un moteur BLDC	54
5.3	Schéma de l'électronique de puissance dans le simulateur HIL Typhoon	54
5.4	Visualisation des PWM entrantes, des encodeurs et de la vitesse du moteur sur le simulateur HIL	55

5.5	Réponse indicielle de la vitesse, saut de consigne à 700 rad/s	56
5.6	PWM générée par le module FPGA	56
A.1	Convertisseur A/D à approximation successive[21]	61
A.2	Évolution de U(Z) / Z avec la méthode par pesée[21]	62
A.3	Architecture pipeline[21]	62
A.4	Exemple de latence type pour le Sigma Delta, le SAR et le Pipeline [22] .	63
R.1	Disposition du rack Poetic pour le démonstrateur	125
R.2	Câble plat reliant le module mesure et le module Poetic FPGA	126
R.3	Schéma bloc du câblage entre la FPGA et le simulateur Typhoon HIL .	127
R.4	Double clicker sur «Boundary Scan»	128
R.5	Click droit -> «Initialize Chain» ou CTRL+I	129
R.6	Clicker sur «Yes» pour assigner un fichier .bit de configuration	129
R.7	Aller dans le fichier /BLDC_BIT_FILE du repo et ouvrir le fichier «poetic_circuit.bit»	130
R.8	Clicker sur «No» pour flasher uniquement la FPGA	130
R.9	La FPGA a été reconnue. Clicker sur «Ok»	131
R.10	Pour programmer la FPGA, il effectuer un Click droit sur celle-ci et clicker sur «Program»	131
R.11	Programmation réussie	132
R.12	Ouvrir la schématique du simulateur avec le fichier «CmdUF_DC.tse» .	132
R.13	Compiler la schématique et lancer le simulateur	133
R.14	Ouvrir le panel «CmdUF_DC.cus» qui se trouve dans le dossier /HIL .	133
R.15	Charger les paramètres du modèle	134
R.16	Sélectionner le fichier «Model settings UF DC.runx» contenant les paramètres du modèle	134
R.17	Lancer le simulateur	135

Liste des tableaux

3.1	Compatibilité entre les FPGA de la série Spartan 6[10]	15
4.1	Paquet pour définir une consigne de 2730	46
4.2	Schéma de commutation trapézoïdale	49
6.1	Comparatif entre la carte POETIC PROCESSOR et la nouvelle carte POETIC FPGA	58
R.1	Paquet pour définir une consigne de 2730	135

1 | Introduction

1.1 Aperçu

Le groupe « Électronique industrielle et entraînements » de la HEI a développé un système de prototypage pour des systèmes électroniques de puissance. Ce système, visible sur la figure 1.1, se nomme «Poetic» et se présente sous la forme d'un rack.

Entièrement modulaire, il peut accueillir une multitude de modules propriétaire sur son connecteur **fond de panier** nommé «backplane». Sur figure 1.1, le rack Poetic contient différents modules. En l'occurrence, une unité de contrôle à processeur, un module mesure qui met en forme des signaux analogiques pour que l'unité de contrôle puisse les échantillonner, des interfaces digitales, des relais et une interface fibre optique (ces modules sont expliqués en détails au chapitre 2.2).

Cette structure permet donc la mise en place rapide d'un banc de test pour le pilotage d'une électronique de puissance. De plus, ce dispositif est compatible avec le simulateur d'électronique de puissance Typhoon HIL (visible sur la figure 1.1). Ce qui, dans un premier temps, permet de simuler entièrement l'électronique de puissance lors d'essais, sans utiliser de composants réels (alimentation, demi-pont à mosfet, moteurs, etc..).

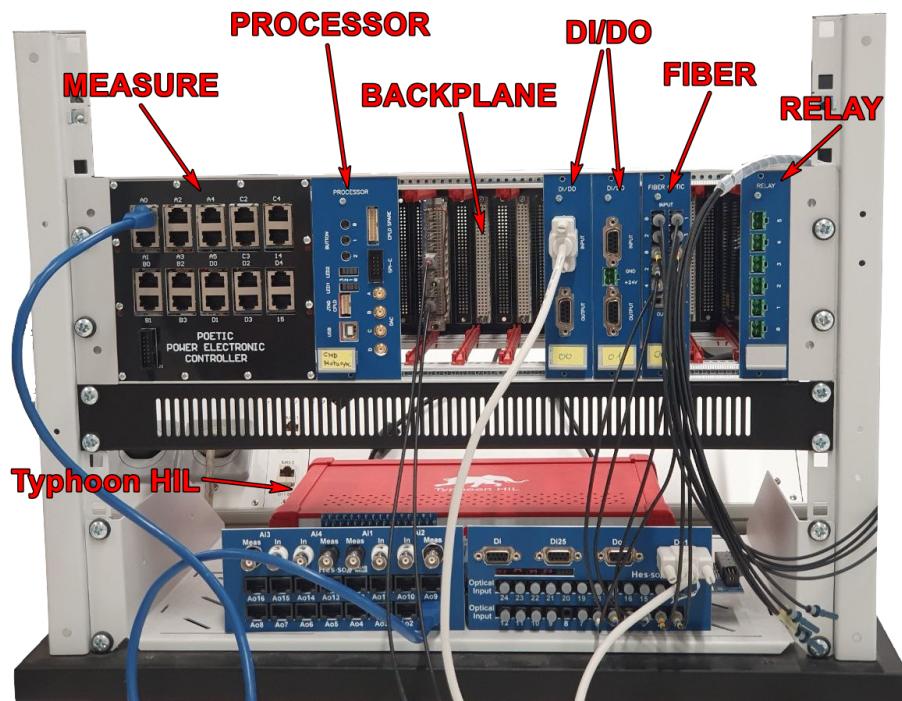


Figure 1.1 Rack Poetic avec un simulateur Typhoon HIL 402

1.2 But du projet

Actuellement l'unité de contrôle de ce système de prototypage est une carte à processeur. Le but de ce travail est de développer une carte autour d'une **FPGA**, similaire à la carte processeur, pour le contrôle de la partie puissance et de réaliser un démonstrateur avec ce système.

2 | Analyse

2.1 Spécifications

Le cahier des charges a été établi avec François Corthay, responsable du projet de thèse.

2.1.1 Cahier des charges

- Adapter la carte existante «FPGA-Rack¹» pour pouvoir l'utiliser dans le système groupe « Électronique industrielle et entraînements »
- Mettre en place un système de développement **FPGA** pour le contrôle de l'électronique de puissance
- Réaliser un démonstrateur pour le pilotage d'une charge de puissance
- Ajout d'un PHY Ethernet pour pouvoir transmettre plus d'informations
- Ajout d'un récepteur / driver **LVDS** pour pouvoir utiliser un **SPI** externe
- Le PCB Mezzanine doit-être compatible avec la carte de développement «FPGA-Rack¹»
- Doubler le nombre de **DAC** sur le nouveau module «FPGA Poetic»
- Le module doit pouvoir être alimenté avec le rack Poetic ou avec l'**USB**

2.2 Fonctionnement du rack Poetic

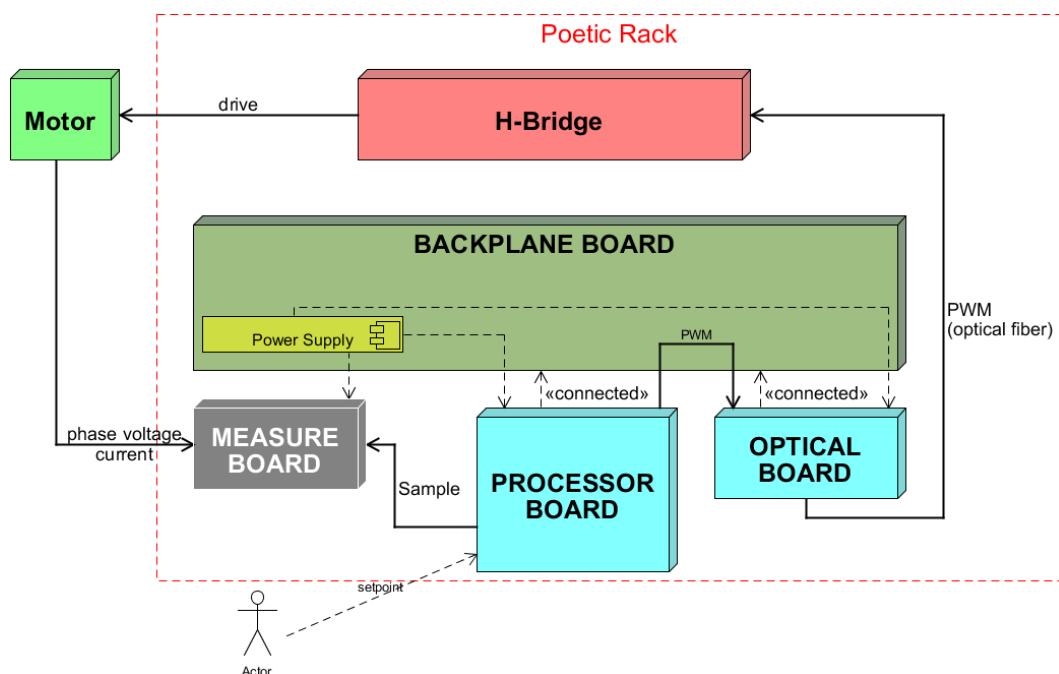


Figure 2.1 Régulation en boucle fermée d'un moteur à l'aide du rack Poetic

1. Kit de développement FPGA développé à la HES-SO : <http://wiki.hevs.ch/uit/index.php5/Hardware/FPGARack>, sa schématique se trouve à l'annexe D

Chapitre 2. Analyse

La figure 2.1, représente le rack Poetic (visible sur l'image 1.1) en train de réguler un moteur.

BACKPLANE BOARD

Il s'agit d'une carte [fond de panier](#). C'est à travers cette carte que les modules peuvent communiquer entre eux. Elle fournit l'alimentation de l'ensemble des modules numériques, analogiques, de commandes et d'interfaces.

PROCESSOR BOARD

Il s'agit d'un module physiquement connecté à la carte [fond de panier](#) «BACKPLANE BOARD». Il est composé d'un processeur effectuant la régulation du système. Il échantillonne des signaux fournis par le module «MEASURE BOARD». L'utilisateur peut communiquer avec ce module et lui fournir, dans cet exemple, la consigne du régulateur. Ce module envoie en sortie, un signal [PWM](#) qui est envoyé à l'électronique de puissance par l'intermédiaire du module optique

MEASURE BOARD

Module alimenté par la carte «BACKPLANE BOARD», il reçoit des signaux analogiques sur des paires différentielles. Ces signaux sont ensuite mis en forme par des convertisseurs et envoyés au module «PROCESSOR BOARD» pour qu'il puisse les échantillonner.

H-Bridge

Module de puissance contenant des MOSFET de puissance. Ce module drive le moteur et reçoit une [PWM](#) par l'intermédiaire d'une fibre optique. Ce module ainsi que le moteur peuvent être simulés à l'intérieur du simulateur Typhoon HIL, qui permet de simuler diverses électroniques de puissances.

OPTICAL BOARD

Module transformant un signal numérique électrique en signal numérique optique. Dans cet exemple, la [PWM](#) reçue par le régulateur est transmise à l'électronique de puissance par fibre optique.

2.3 Carte POETIC Processor V2



Figure 2.2 Carte POETIC Processeur v2 qui équipe les racks POETIC

Sur la figure 2.2, se trouve le module processeur qui équipe actuellement les racks Poetic. Son schéma bloc se trouve à la figure 2.3. Le module se compose de deux PCB. Un PCB contenant un processeur. Ce processeur reçoit sur ses [ADC](#) internes des signaux analogiques fournis par le module de mesure. Il est capable de discuter avec l'utilisateur en [USB](#). Presque toutes ses entrées sorties traversent une CPLD. Cette CPLD permet éventuellement, en cas de nécessité, d'effectuer un changement

2.3. Carte POETIC Processor V2

de pinning sur le connecteur VME. Il permet aussi de garantir une protection hardware. En effet, en cas de problème (arrêt d'urgence), un signal d'alarme est mis à 1 et bloque toutes les PWM et les GPIO sortantes sur le connecteur VME. Des boutons poussoirs ainsi que des LED permettent à l'utilisateur de pouvoir interagir avec le module. Le PCB du haut (mezzanine), est composé d'un convertisseur D/A qui permet à l'utilisateur de pouvoir voir des signaux internes au processeur ainsi que deux autres interface interface de communication non détaillée dans le schéma bloc.

Par mesure de confidentialité, la schématique de la carte «Poetic Processor V2» n'est pas partagée.

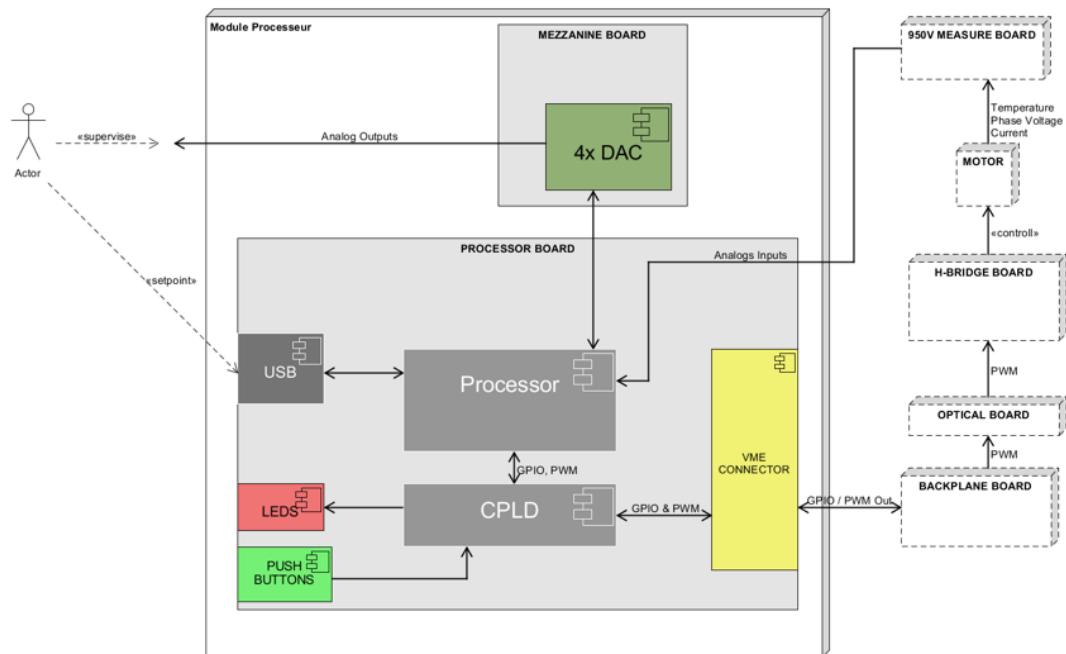


Figure 2.3 Schéma bloc de la carte Poetic Processor V2

3 | Hardware

3.1 Schéma bloc

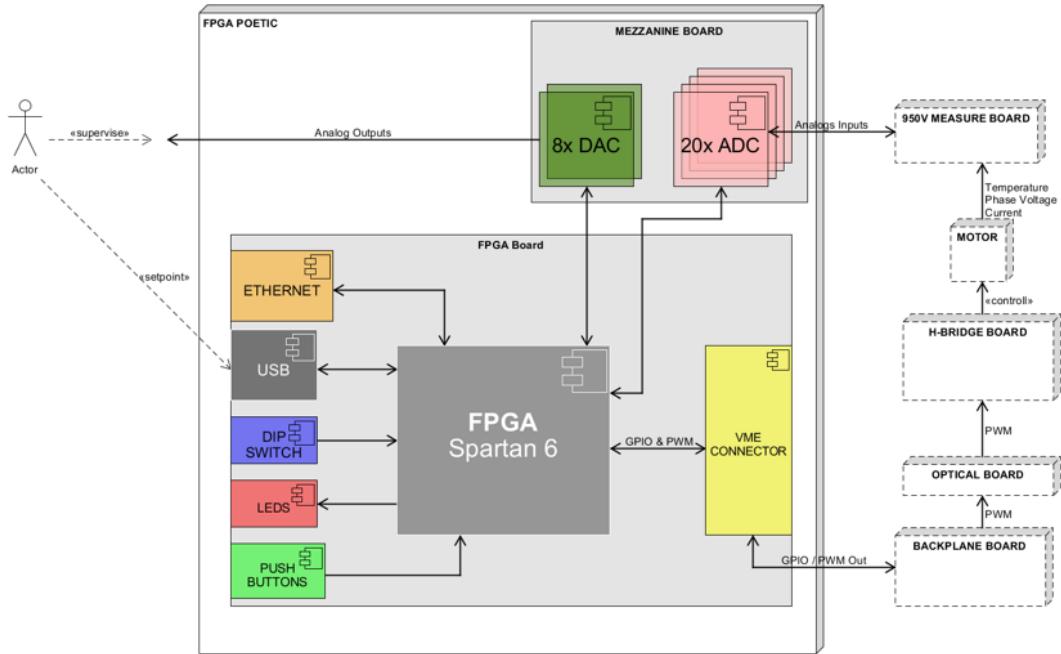


Figure 3.1 Schéma bloc du module FPGA dans l'environnement POETIC

L'électronique a été, comme pour l'électronique du module processeur (voir chapitre 2.3), séparée en deux parties distinctes. Ce qui permet de tester indépendamment les fonctionnalités ajoutées. (par exemple avec la carte de développement FPGA-Rack¹) ainsi que de limiter les sources d'erreurs. Le cœur de la « **FPGA BOARD** » est une **FPGA** Spartan 6 de la marque Xilinx. Cette **FPGA** peut interagir avec l'utilisateur avec des LED, des boutons poussoirs ainsi qu'un DIP Switch à 4 positions. Un Phy Ethernet ainsi qu'un convertisseur **USB** vers **UART** permet à la **FPGA** de communiquer avec le monde extérieur. Elle peut acquérir 20 signaux analogiques simultanément grâce à des convertisseurs analogiques digitaux (**ADC**) qui se trouvent sur le PCB Mezzanine. Sur cette même carte se trouvent 8 convertisseurs digitaux analogiques (**DAC**) commandés par la **FPGA**, ce qui permet à l'utilisateur de lire / afficher les signaux mesurés. La connexion entre les deux cartes est assurée par des connecteurs mezzanine. Dans son environnement, cette électronique permet donc de réguler par exemple un ou plusieurs moteurs. Grâce à ses interfaces de communications (Ethernet, **USB**), la board reçoit une consigne fournie par l'utilisateur. En fonction de celle-ci, la **FPGA** agit sur ses signaux de commande (**PWM**) qui contrôlent le moteur. La boucle de régulation est assurée par des mesures effectuées sur le moteur. Ces mesures sont effectuées par les convertisseurs analogiques digitaux. Les signaux sont en amont mis en forme par la carte « **950V MEASURE BOARD** ». Un schéma bloc complet se trouve à l'annexe B.

1. Kit de développement FPGA développé à la HES-SO : <http://wiki.hevs.ch/uit/index.php5/Hardware/FPGARack>, sa schématique se trouve à l'annexe D

3.2 ADC

Un convertisseur analogique digital est un dispositif électronique dont la fonction est de traduire une tension analogique en une valeur numérique codée sur plusieurs bits[1]. Lors de sa conversion, le signal est alors quantifié et échantillonné. L'échantillonnage consiste à relever à intervalle régulier la valeur d'un signal[2]. La quantification consiste à approcher la tension du signal d'un ensemble de valeurs discretes codées sur plusieurs bits[3]. Une quantification sur de nombreux bits ainsi qu'un échantillonnage rapide permet une reconstruction fidèle du signal mesuré. Contrairement au processeur TMS320F28377SPTPT sur la carte processeur (voir chapitre 2.3), les **FPGA** ne disposent, pas toutes, d'**ADC** dans leur architecture. Ce chapitre aborde le développement d'un système contenant une multitude de convertisseurs A/D 12 bits pouvant échantillonner 20 canaux simultanément à une fréquence d'environ 1MHz.

3.2.1 Caractéristiques des ADC internes de la carte POETIC Processeur

Pour déterminer les caractéristiques des **ADC** de la nouvelle électronique, il faut s'intéresser à la carte POETIC Processeur (voir la figure 2.2) qui équipe actuellement les racks POETIC. Selon le datasheet du processeur TMS320F28377SPTPT (figure 3.2) qui équipe actuellement la carte poetic processeur, il dispose dans son architecture 4 convertisseurs analogiques numériques 12 bits (entrées single-ended) ou 16 bits (entrées différentielles). Le processeur travaille avec des convertisseurs analogiques numériques 12 bits (single-ended) car l'électronique possède 20 canaux analogiques en entrées. Avec des **ADC** 16 bits, le nombre d'entrées analogiques serait limité à 12 canaux. La quantification du signal doit être de ce fait, au minimum de 12 bits, pour conserver les mêmes caractéristiques.

- Analog subsystem
 - Up to four Analog-to-Digital Converters (ADCs)
 - 16-bit mode
 - 1.1 MSPS each (up to 4.4-MSPS system throughput)
 - Differential inputs
 - Up to 12 external channels
 - 12-bit mode
 - 3.5 MSPS each (up to 14-MSPS system throughput)
 - Single-ended inputs
 - Up to 24 external channels

Figure 3.2 Caractéristiques des ADC dans le processeur TMS320F28377SPTPT[4]

La fréquence d'échantillonnage maximale pour 20 canaux de la carte «Poetic Processor» est calculée à l'équation 3.1.

$$F_{MaxEchantillonnageParCanal} = F_{chantillonnageADC} * \frac{N_{ADC}}{N_{canaux}} = 3.5MSPS * \frac{4}{20} = 700kSPS \quad (3.1)$$

Si le processeur décide de ne pas mesurer certains canaux, la fréquence d'échantillonnage sera bien entendu plus élevée. Dans la pratique, le processeur n'échantillonne jamais à pleine vitesse, il doit laisser des ressources pour effectuer d'autres opérations.

3.2.2 Fréquence d'échantillonnage minimale

La fréquence de l'échantillonnage est régie par le théorème de Shannon-Nyquist, qui définit la fréquence d'échantillonnage minimal pour représenter correctement un signal mesuré[5].

$$F_{EchantillonageMin} = 2 * F_{Max} \quad (3.2)$$

La largeur de bande en entrée de notre ADC est de 200kHz (en pratique, il s'agit de signaux de quelques kHz au maximum). De ce fait, il est établi que la fréquence d'échantillonnage minimale équivaut à

$$2 * 200KHz = 400KHz = 0.4MSPS \quad (3.3)$$

3.2.3 Choix de l'ADC

L'ADC doit répondre aux besoins ci-dessous :

- ≥ 12 bits
- ≥ 700 kSPS
- Single channel
- Single-Ended
- Peu de latence voir aucune (voir l'annexe A)
- Tension d'alimentation 3 – 5V

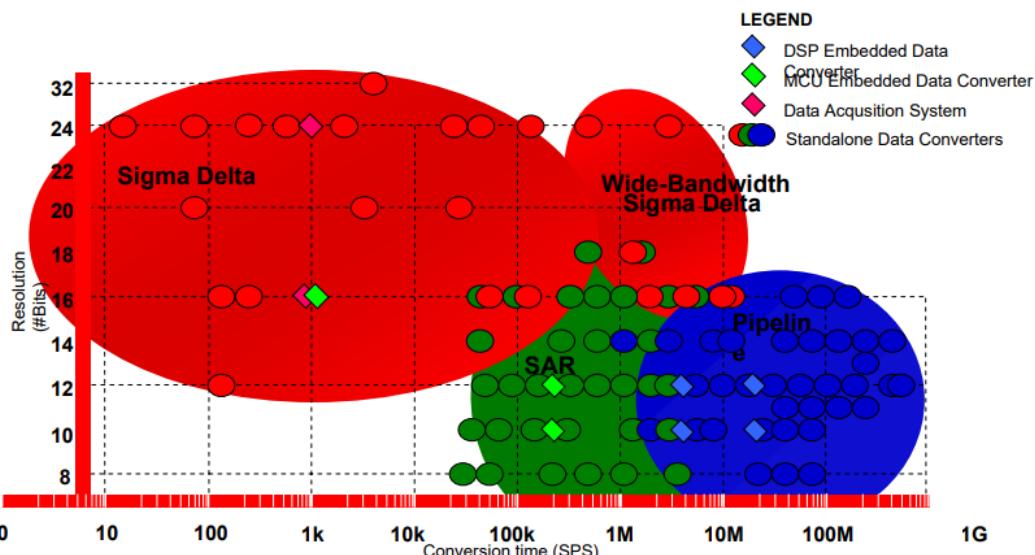


Figure 3.3 Comparatif des différentes architectures ADC[6]

Avec ces spécifications, le choix s'est porté pour la référence ADS7886 de Texas Instrument (figure 3.4) qui dispose d'une architecture SAR. Ce chip est un convertisseur analogique digital 12 bits 1 canal, avec une fréquence d'échantillonnage maximale de 1MSPS. Chaque canal (20 au total) sera échantillé par un convertisseur ADS7886. La FPGA travaillant en parallèle, elle recevra de ce fait, les 20 échantillons simultanément. La plage d'alimentation de ce convertisseur varie de 2.35V à 5.25V. L'alimentation est aussi sa référence de tension (voir le chapitre 3.6.5). Il dispose d'une interface

série qui lui permet de transmettre ses données.

Une analyse détaillée des différentes architectures des convertisseurs A/D sur la figure 3.3 se trouve à l'annexe A.

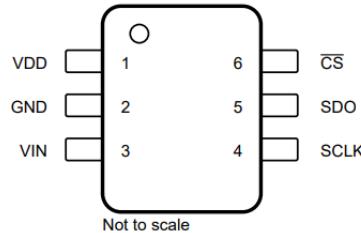


Figure 3.4 ADC AD7886 qui équipe la carte FPGA Poetic[7]

3.2.4 Differential vers Signle-Ended

Les signaux analogiques à échantillonner arrivant sur la carte Mezzanine sont des paires différentielles. Un simple montage soustracteur à l'aide d'un ampli-op (figure 3.5) permet de passer le signal différentiel en signal asymétrique. Une tension de 1.5V DC est ajoutée au signal de sortie, pour que le signal varie autour de la tension de référence 1.5V. Un jumper (JP1 sur la figure 3.5) a été rajouté pour supprimer l'offset. Un filtre antialiasing est ajouté dans ce montage pour couper toutes les fréquences de plus de 200kHz (voir la réponse en fréquence de la simulation à la figure 3.8). Ce montage à ampli-op a été (sauf pour le jumper) repris tel quel de la carte processeur.

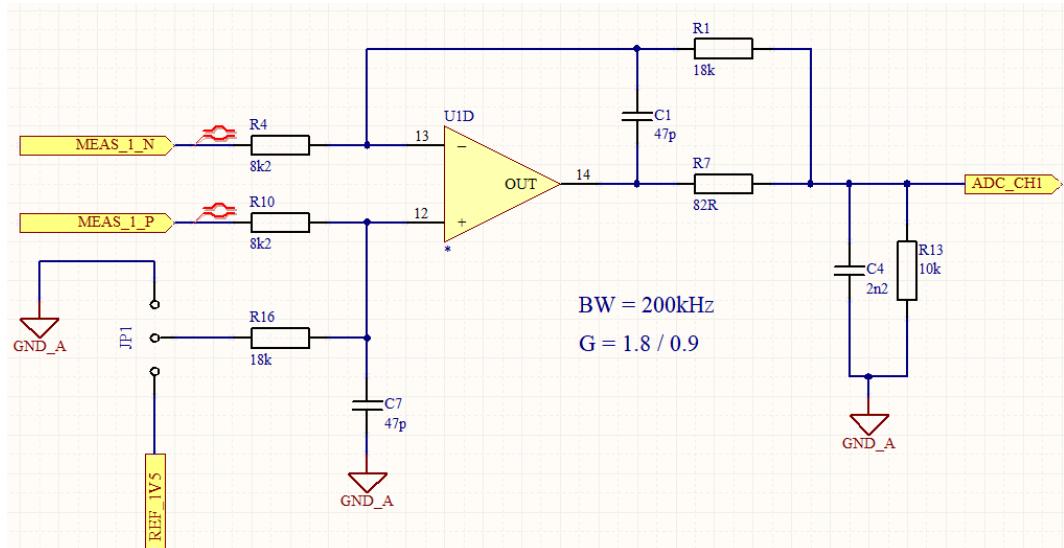


Figure 3.5 Montage ampli-op différentiel vers asymétrique

Simulation

Une simulation sur LTSpice du montage à ampli-op a été réalisée (voir la figure 3.6). La réponse en fréquence donne une fréquence de coupure d'environ 222kHz (visible à la figure 3.8). Deux résistances série de 1.8K ont été ajoutées qui proviennent du module de mesure «950V_Measure».

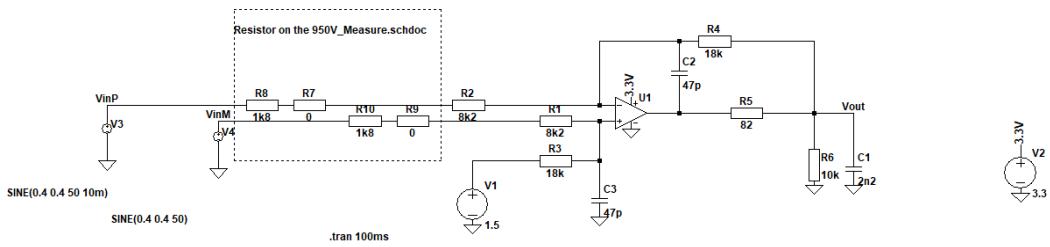


Figure 3.6 Simulation du montage soustracteur passe-bas sur LTSpice

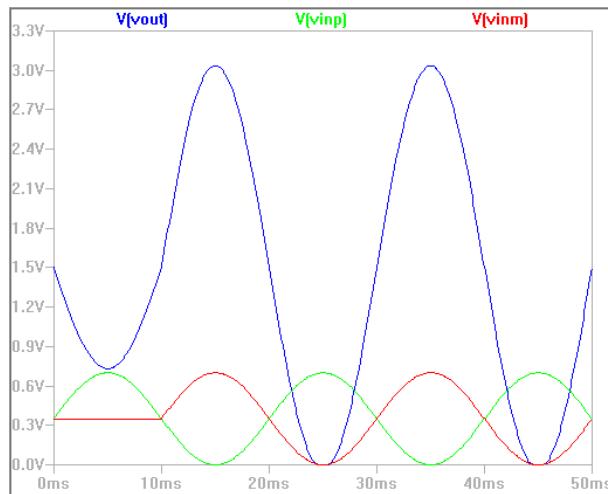


Figure 3.7 Analyse des signaux U_{out} , v_{inP} et v_{inM}

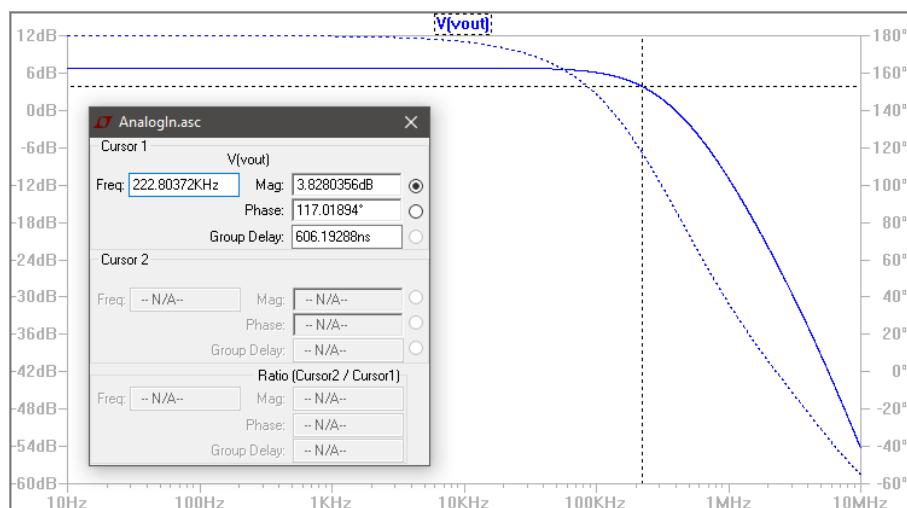


Figure 3.8 Réponse en fréquence du montage ampli op soustracteur

3.2.5 Communication avec la FPGA (signal level)

L'[ADC](#) et les I/O [FPGA](#) ne travaillent pas à la même tension (voir la chapitre [3.6.5](#)). Il est donc nécessaire de vérifier si l'[ADC](#) peut communiquer avec la [FPGA](#) et inversement malgré cette différence de tension. On peut observer sur la Figure 10 que la plage de sortie de l'[ADC](#) est incluse dans la plage d'entrée de la [FPGA](#) et inversement. Il n'est absolument pas nécessaire d'ajouter un composant qui adapte les deux niveaux de tensions.

I/O Standard	V _{IL}		V _{IH}		V _{OL}	V _{OH}	I _{OL}	I _{OH}
	V, Min	V, Max	V, Min	V, Max	V, Max	V, Min	mA	mA
LV TTL	-0.5	0.8	2.0	4.1	0.4	2.4	Note 2	Note 2
LVC MOS 33	-0.5	0.8	2.0	4.1	0.4	V _{CCO} - 0.4	Note 2	Note 2

Figure 3.9 Spécifications des entrées sorties de la [FPGA](#)[8]

DIGITAL INPUT/OUTPUT					
Logic family — CMOS					
V _{IH} High-level input voltage	V _{DD} = 2.35 V to 3.6 V		1.8	5.25	V
	V _{DD} = 3.6 V to 5.25 V		2.4	5.25	
V _{IL} Low-level input voltage	V _{DD} = 5 V			0.8	V
	V _{DD} = 3 V			0.4	
V _{OH} High-level output voltage	I _(source) = 200 µA		V _{DD} - 0.2		V
V _{OL} Low-level output voltage	I _(sink) = 200 µA			0.4	

Figure 3.10 Spécifications des entrées sorties de l'[ADC](#)[7]

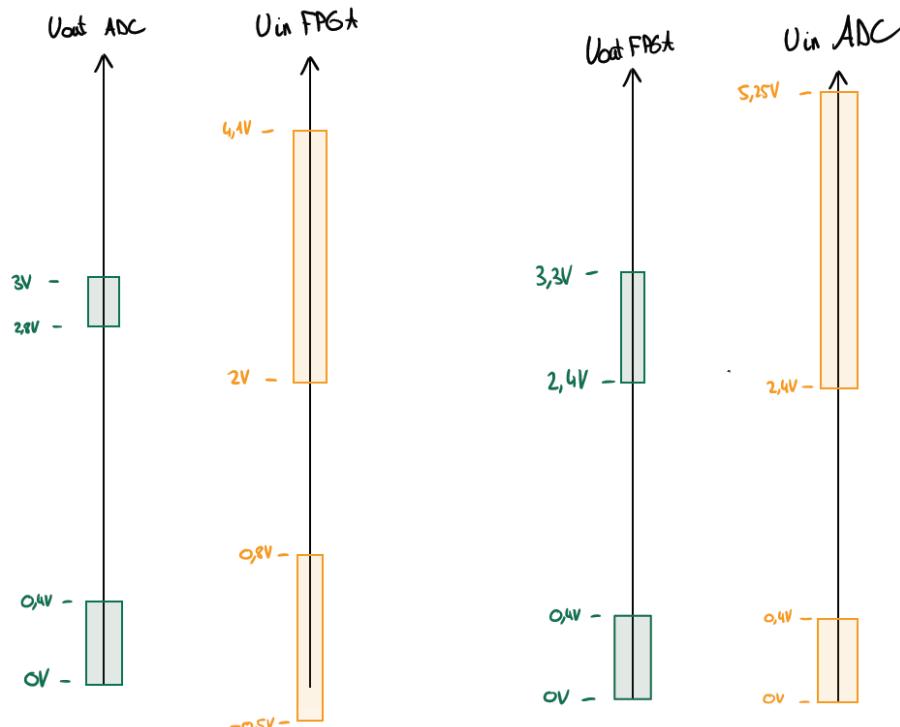


Figure 3.11 Niveaux de tensions des entrées sorties de la [FPGA](#) et de l'[ADC](#)

3.2.6 Schématique

La schématique des convertisseurs A/D se trouve à l'annexe E. Tous les convertisseurs A/D partagent le même clock, mais peuvent être indépendamment piloté grâce à leur chip select.

3.3 DAC

Des convertisseurs digitaux analogiques ont été implémentés sur la carte Mezzanine. Ces convertisseurs permettent à l'utilisateur d'observer des signaux internes de la **FPGA**. Le nombre de sorties analogiques étant trop faible sur l'ancienne carte à processeur, il a été doublé sur la carte **FPGA**. Il y a au total 8 sorties analogiques. Le convertisseur **DAC** est un DAC124S085CIMM/NOPB et dispose de 4 canaux. La tension de référence du convertisseur est générée par une référence de tension 3V REF3130. La plage de sortie du signal analogique de sortie varie de 0V à 3V. Un montage suiveur à ampli-op passe-bas se situe à la sortie du convertisseur D/A. Ce montage a été repris sur la nouvelle schématique telle quelle de la carte processeur.

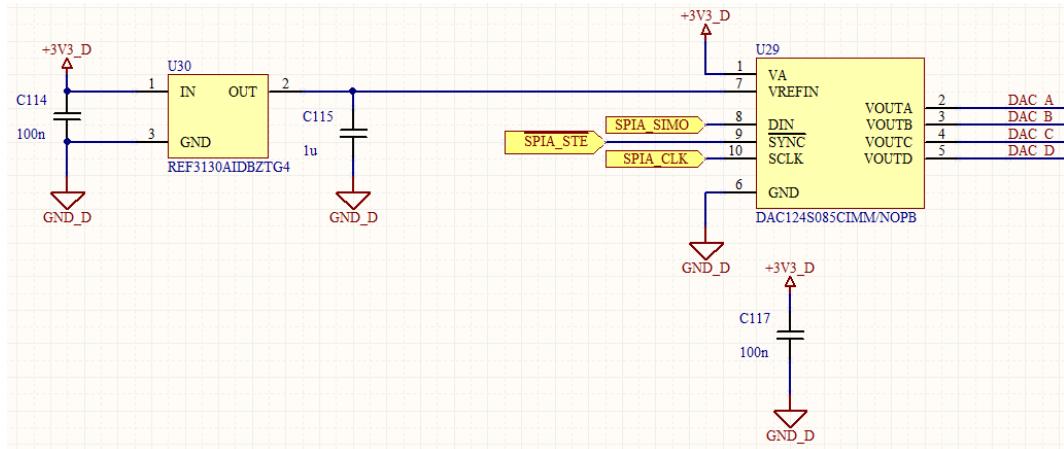


Figure 3.12 Convertisseur DAC 4CH

3.4 FPGA

3.4.1 Choix de la FPGA

Le choix de la [FPGA](#) s'est porté sur le Spartan 6 de Xilinx. Cette famille de [FPGA](#) est utilisée dans la board de développement « [FPGA-Rack](#) ». Néanmoins, il existe une nouvelle version de la gamme Spartan, le [Xilinx Spartan 7](#). Au moment de la conception de la carte [FPGA](#), aucune Spartan 7 n'était en stock.



Figure 3.13 *FPGA Spartan 6 LX150 en boîtier FGG484 (XC6SLX150-N3FGG484C)*

3.4.2 Tailles & compatibilités

La famille Spartan 6 est divisée en plusieurs [FPGA](#) de différentes tailles. Un comparatif entre les différentes tailles de cette famille se trouve sur le site officiel de Xilinx[9]. Elles sont généralement pin compatible. Sur le tableau 3.1, on observe qu'en effectuant la schématique avec une [FPGA](#) Spartan 6 LX75, il est possible d'y venir souder une [FPGA](#) LX100 ou une LX150 car toutes les pins de la LX75 sont disponibles sur les modèles plus grands. Ce qui n'est pas le cas pour par exemple le modèle LX45. Pour une implémentation plus rapide, la schématique à été désignée autour de la [FPGA](#) Spartan 6 LX75. Comme il s'agit d'une board de développement, la plus grande [FPGA](#) y sera implantée, soit la LX150.

Socket	Modèle	Compatibilité
FG(G)484	LX45	Not all pin compatible, but compatible : LX75, LX100, LX150
FG(G)484	LX75	All pin compatible : LX100, LX150 Not all pin compatible, but compatible : LX45
FG(G)484	LX100	All pin compatible : LX150 Not all pin compatible, but compatible : LX45, LX75

Table 3.1 *Compatibilité entre les [FPGA](#) de la série Spartan 6[10]*

3.4.3 Bank 0-3

La [FPGA](#) Spartan 6 au socket FG(G)484 sont toutes séparées en 5 parties : Bank 0-3 et Core (voir la figure 3.14). Tous les périphériques autour de la [FPGA](#) ont une alimentation de 3.3V, de ce fait, les banks ont toutes une tension d'alimentation de 3.3V. Les banks ont été néanmoins « séparées » par utilisation. La bank 0 est connectée à la partie analogique de la board Mezzanine. La bank 1 génère toutes les [PWM](#) et comporte les I/Os connectés connecteurs VME. La bank 2 est connectée à la partie digitale de la board Mezzanine ([DAC](#), [SPI LVDS](#), Spare). La bank 3 est connectée aux interfaces de communications (ethernet, [USB](#)) ainsi qu'aux boutons et LED.

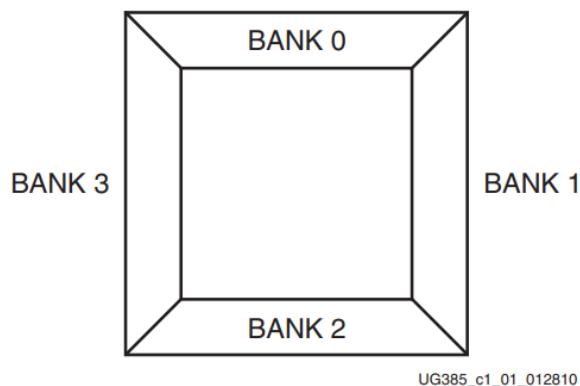


Figure 3.14 Positionnement des banks sur le Spartan 6[10]

3.4.4 Découplage

La [FPGA](#) Spartan 6 requiert de nombreux condensateurs de découplage. Selon le document «Xilinx UG393 Spartan-6 FPGA PCB Design Guide[11]», dont deux extraits sont disponibles en figures 3.15 et 3.16, le nombre de condensateurs de découplage varie en fonction du modèle de la [FPGA](#) utilisée, ainsi que du socket. Le PCB [FPGA](#) a été pensé pour venir accueillir au maximum la version la plus performante, soit la LX150. Il faut donc dimensionner les condensateurs de découplage en fonction de celle-ci. La caractéristique des condensateurs est décrite dans le même document, un extrait se trouve à la figure 3.16.

Il est important de respecter les consignes suivantes en choisissant les condensateurs de découplage :

- Les valeurs des condensateurs peuvent être plus grandes que spécifiées
- La taille (footprint) des condensateurs peut être plus petit que spécifiée
- La tension maximale peut être plus grande que spécifiée
- La résistance ESR doit être comprise dans la plage spécifiée

3.4. FPGA

Package	Device (XC6S)	V_{CCINT} in μF			V_{CCAUX} in μF			V_{CCO} Bank 0 in μF			V_{CCO} Bank 1 in μF			V_{CCO} Bank 2 in μF			V_{CCO} Bank 3 in μF			V_{CCO} Bank 4 in μF			Total (2)			
		100	4.7	0.47	100	4.7	0.47	100	4.7	0.47	100	4.7	0.47	100	4.7	0.47	100	4.7	0.47	100	4.7	0.47	100	4.7	0.47	
FG(G)484	LX75	1	2	3	1	2	4	1	1	2	1	1	3	1	1	4	1	1	3							33
FG(G)484	LX75T	1	2	3	1	2	4	1	1	2	1	1	3	1	1	3	1	1	3							32
FG(G)484	LX100	1	2	4	1	2	4	1	1	2	1	1	3	1	1	3	1	1	3							33
FG(G)484	LX100T	1	2	4	1	2	4	1	1	2	1	1	3	1	1	3	1	1	3							33
FG(G)484	LX150	2	3	6	1	2	4	1	1	2	1	1	3	1	1	3	1	1	3							37
FG(G)484	LX150T	2	3	6	1	2	4	1	1	2	1	1	3	1	1	3	1	1	3							37

Figure 3.15 Condensateurs requis pour les différentes FPGA[11]

Ideal Value	Value Range ⁽¹⁾	Body Size ⁽²⁾	Type	ESL Maximum	ESR Range ⁽³⁾	Voltage Rating ⁽⁴⁾	Suggested Part Number
100 μF	C > 100 μF	1210	2-Terminal Ceramic X7R or X5R	5 nH	10 m Ω < ESR < 60 m Ω	6.3V	GRM32ER60J107ME20L
4.7 μF	C > 4.7 μF	0805	2-Terminal Ceramic X7R or X5R	2 nH	10 m Ω < ESR < 60 m Ω	6.3V	
0.47 μF	C > 0.47 μF	0204 or 0402	2-Terminal Ceramic X7R or X5R	1.5 nH	10 m Ω < ESR < 60 m Ω	6.3V	

Figure 3.16 Caractéristiques des condensateurs de découplage[11]

3.4.5 Quartz

Le quartz cadence la [FPGA](#) à 100MHz (voir la figure 3.17). Sa fréquence doit être au minimum deux fois plus élevée que la plus grande fréquence à traiter, ce qui permet de pouvoir correctement acquérir tous les signaux. La fréquence du quartz ne doit pas être plus élevée que la fréquence maximale admise par la [FPGA](#) (voir le sous-chapitre 3.4.5). Le signal clock généré par l'oscillateur est envoyé dans une pin GCLK. Les pins GCLK ou «Global Clock» est une pin qui est préroulée dans la [FPGA](#) et qui distribue de manière «équilibrée» le clock à l'intérieur de la [FPGA](#).

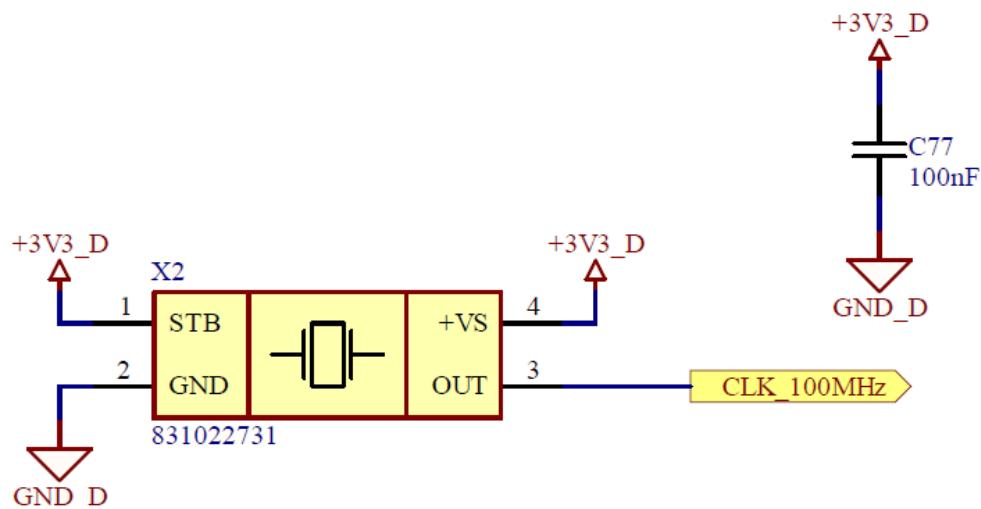


Figure 3.17 Quartz de 100MHz qui cadence la FPGA

Fréquence maximale

La vitesse maximale avec laquelle la **FPGA** peut travailler dépend du «Speed grade» soit un indice de vitesse. Cet indice peut être identifié grâce à la référence du fabricant ou avec une inscription sur son boîtier. Il est indépendant de la taille de la **FPGA**. Chaque grandeur de **FPGA** dispose en général de 4 indices de vitesses différentes. Les indices de vitesse sont (du plus lent au plus rapide) : -1L, -2, -3, -3N. Sur la figure 3.13, se trouve le Spartan 6 LX150 qui équipe le PCB FPGA avec un indice de vitesse -3N (inscription «3C» sur le boîtier et l'information «-N3» dans la référence du fabricant). Sur la figure 3.18, se trouvent les fréquences maximales de clock que peuvent recevoir en entrée les Spartan 6 en fonction de leur indice de vitesse.

Symbol	Description	Device ⁽¹⁾	Speed Grade				Units
			-3	-3N	-2	-1L	
F_{INMAX}	Maximum Input Clock Frequency from I/O Clock (BUFI02)	LX devices	540	525	450	300	MHz
		LXT devices	540	525	450	N/A	MHz
F_{INMIN}	Maximum Input Clock Frequency from Global Clock Buffer (BUFGMUX)	LX devices	400	400	375	250	MHz
		LXT devices	400	400	375	N/A	MHz
F_{INMIN}	Minimum Input Clock Frequency	LX devices	19	19	19	19	MHz
		LXT devices	19	19	19	N/A	MHz

Figure 3.18 Fréquences maximales et minimales de clock sur le Spartan 6[8]

3.4.6 Flash

La **FPGA** ne conserve pas sa configuration. De ce fait, à chaque remise sous tension, la **FPGA** doit être reprogrammée. Pour pallier à ce problème, une flash **SPI** a été ajoutée. À chaque démarrage, la flash programme la **FPGA** avec son contenu. La schématique réalisée à la figure 3.19 a été inspirée d'une «typical application» (voir figure 3.20).

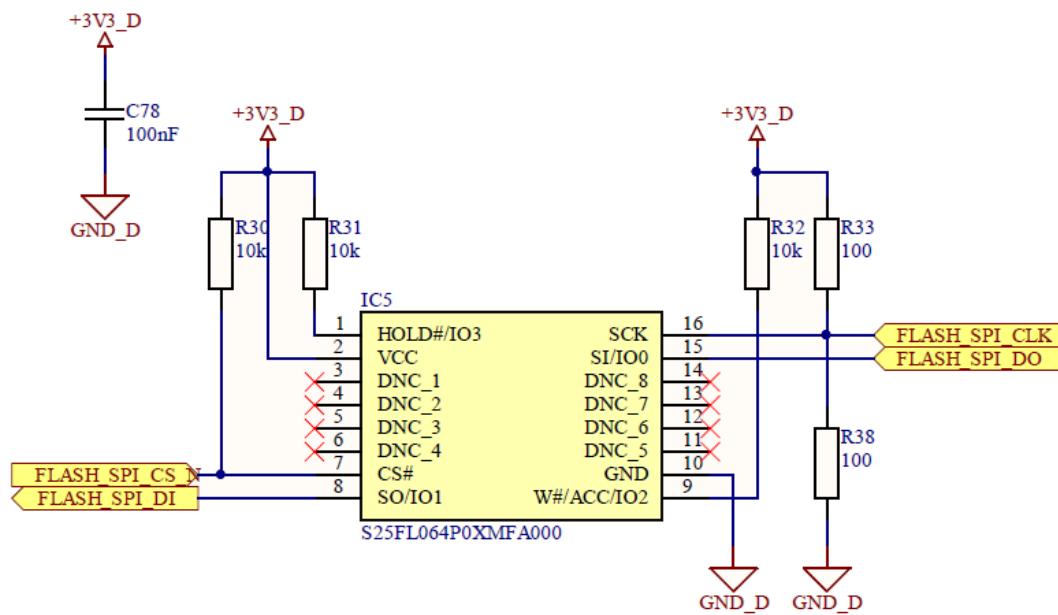


Figure 3.19 Flash SPI

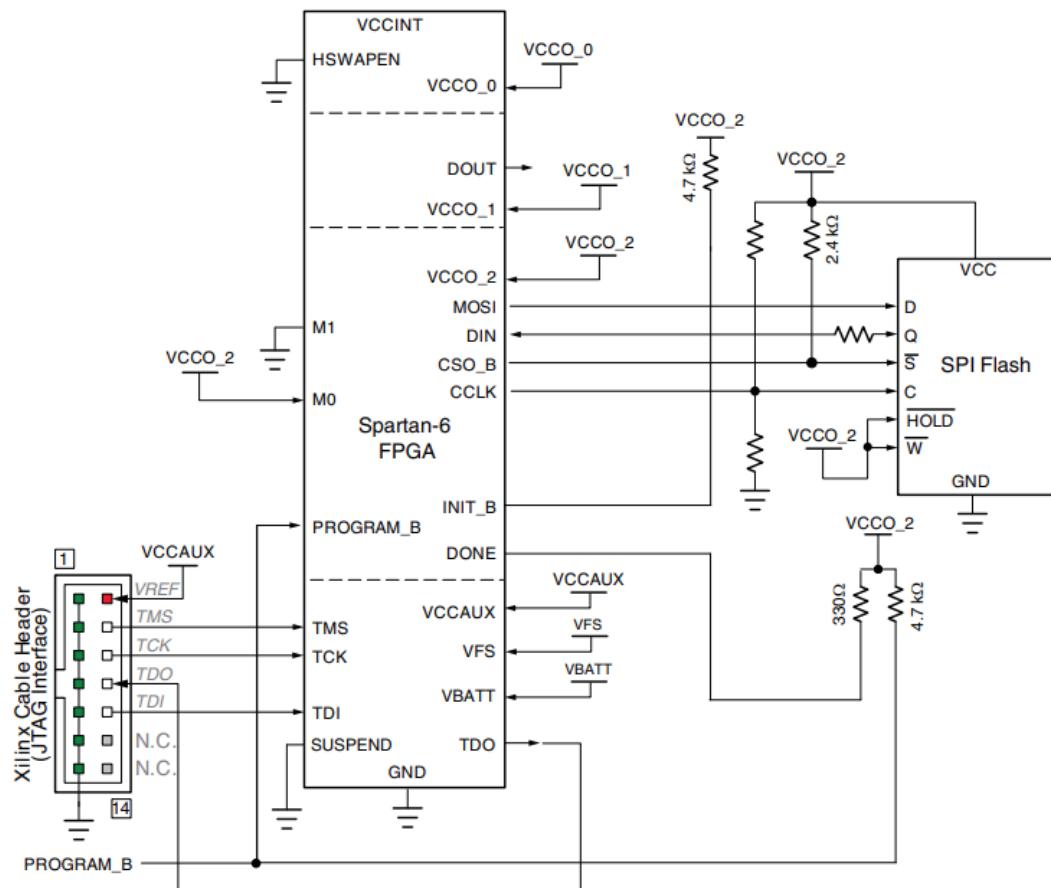


Figure 3.20 Application type d'une FPGA Spartan 6 avec une flash SPI[12]

3.4.7 Mode de configuration

La configuration de la [FPGA](#) se fait à l'aide de deux pins, M1 et M0. Sur la figure 3.21, se trouve la liste des configurations disponibles. La flash communiquant en [SPI](#), la configuration est mise à «01» avec une pull-down et une pull-up (voir la figure 3.22).

Configuration Mode	M[1:0]	Bus Width	CCLK Direction
Master Serial/SPI	01	1, 2, 4 ⁽¹⁾	Output
Master SelectMAP/BPI ⁽²⁾	00	8, 16	Output
JTAG ⁽³⁾	xx	1	Input (TCK)
Slave SelectMAP ⁽²⁾	10	8, 16	Input
Slave Serial ⁽⁴⁾	11	1	Input

Figure 3.21 Mode de configuration de la [FPGA](#) Spartan 6[12]

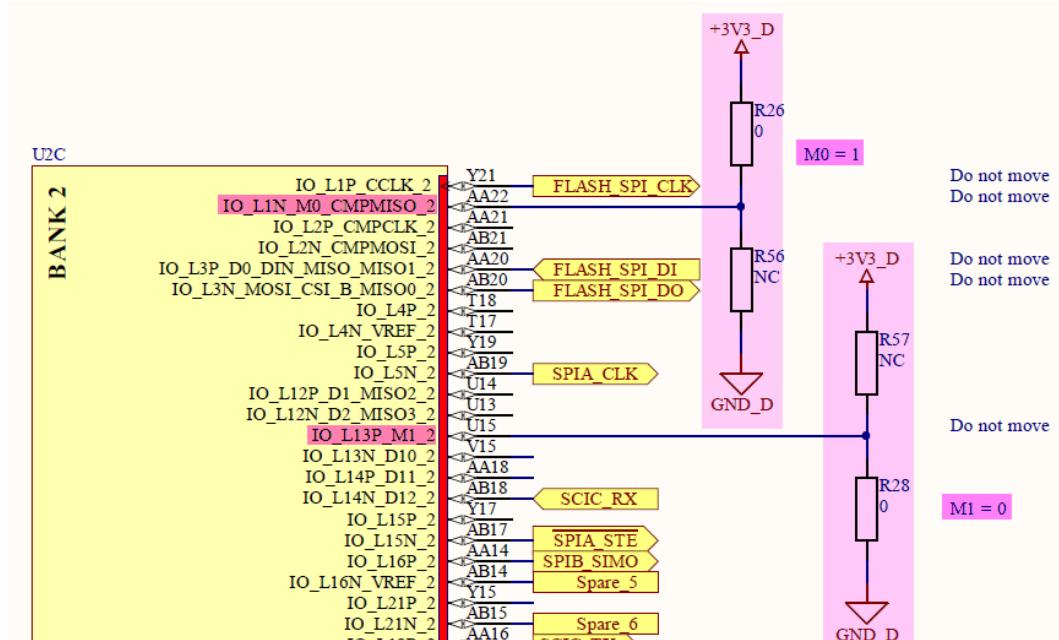


Figure 3.22 Résistances de pull-up et de pull-down ordonnant la configuration SPI à la [FPGA](#)

3.4.8 HSWAPEN

La pin HSWAPEN est mise à la masse à l'aide d'une résistance 0 Ohm. Cela permet d'activer des pull-up sur certaines entrées sorties pendant la configuration de la [FPGA](#). La liste des entrées sorties affectées par la pin «HSWAPEN» se trouve dans le document «Spartan-6 [FPGA Configuration User Guide](#)».

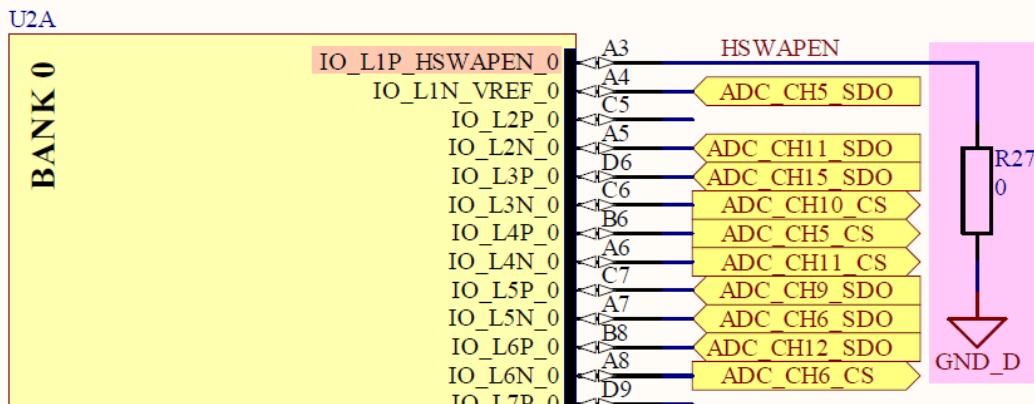


Figure 3.23 Configuration de la pin HSWAPEN

3.4.9 INIT_B

La pin INIT_B est mise à 1. Cette pin permet de retarder la configuration de la [FPGA](#) en la gardant à 0. Cette fonction n'est pas utilisée dans cette application.

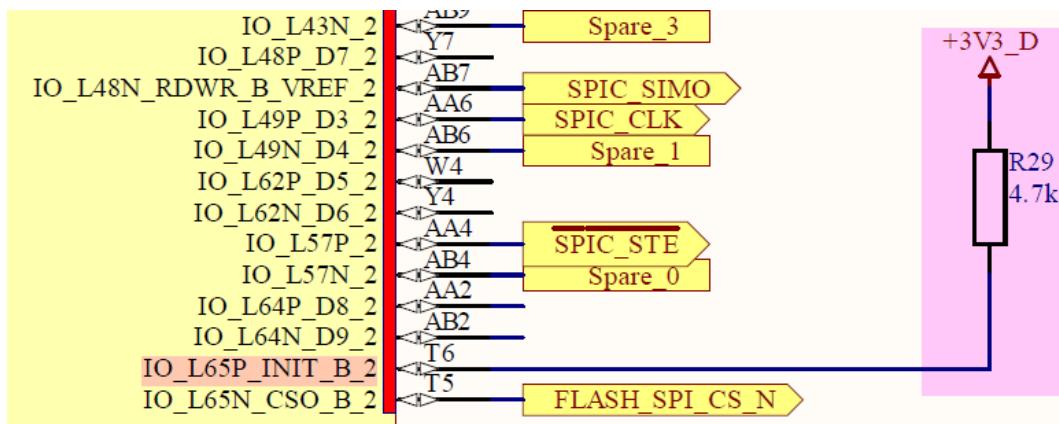


Figure 3.24 Configuration de la pin INIT_B

3.4.10 Reset

Deux boutons poussoirs permettent à l'utilisateur de pouvoir effectuer un soft reset ou un hard reset de la [FPGA](#). Le soft reset est connecté à une I/O de la [FPGA](#). Le hard reset quant à lui, est relié à la pin «PROGRAM_B_2» de la [FPGA](#) qui est défini selon le document «Spartan-6 FPGA Configuration User Guide» comme un reset complet actif bas.

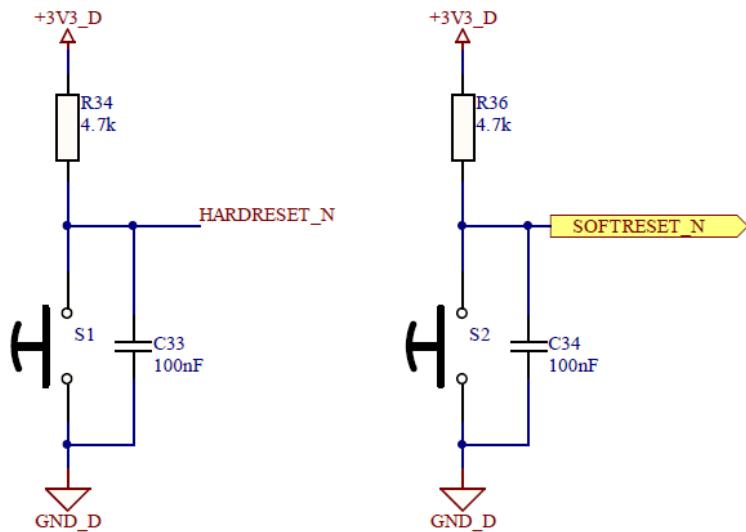


Figure 3.25 Boutons hard reset et soft reset

3.4.11 Indicateur de configuration

La **FPGA** Spartan 6 dispose d'une sortie nommée «**DONE_2**». Cette sortie reste à 0 tant que la **FPGA** n'est pas programmée. Lorsque la **FPGA** a fini d'être configurée, la sortie passe à 1. Sur la figure 3.26, une LED est reliée à cette sortie. Elle permet à l'utilisateur de connaître l'état de la configuration de la **FPGA**.

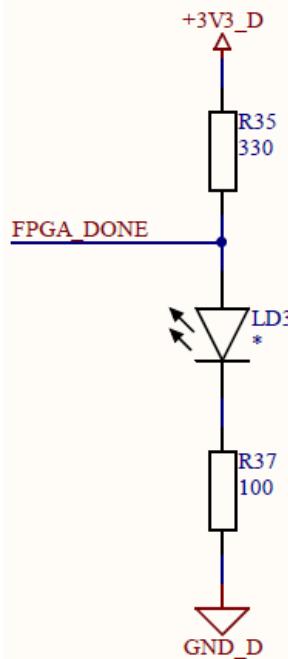


Figure 3.26 LED affichant l'état de la configuration de la **FPGA**

3.5 Interface de communication

3.5.1 Communication série (USB to UART)

Un convertisseur **UART** vers **USB** a été ajouté sur la board. Bien que peu utilisées, les pins RTS et CTS sont tout de même connectées à la **FPGA**. Deux LED TX et RX indiquent tout échange d'informations entre l'hôte et la **FPGA**. La schématique a été tirée de la « typical application 3.27 » du FT232RL.

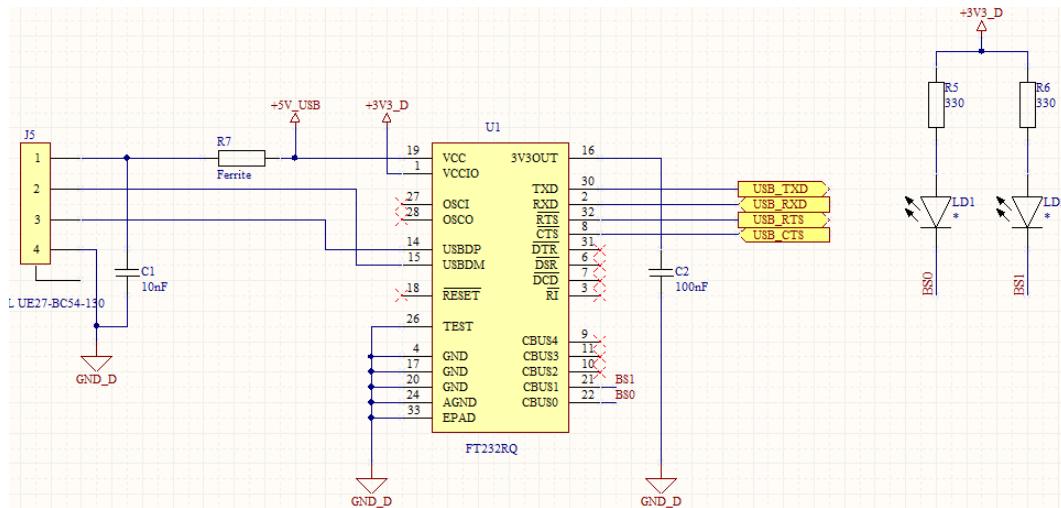


Figure 3.27 USB vers UART, application type du FT232RL

Connecteur USB-C

Par manque de place, un connecteur **USB** type C coudé a été utilisé. Ce connecteur (visible à la figure 3.28), est relié au convertisseur **USB/UART** FT232RL. Il dispose de deux résistances de terminaisons de 5.1KOhm, qui permet à l'ordinateur de savoir si le périphérique est connecté, la puissance qu'il peut délivrer ainsi que de détecter le sens du câble[13].

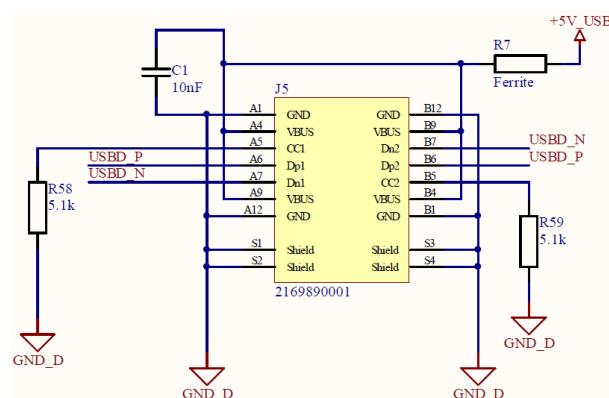


Figure 3.28 Connecteur USB-C

3.5.2 SPI LVDS

Sur la carte Mezzanine se trouvent un driver (figure 3.29) et un récepteur (figure 3.30) LVDS (« Low Voltage Differential Signal ») permettant de communiquer via SPI avec une autre carte. Cette fonctionnalité, reprise de la carte processeur, n'est actuellement toujours pas utilisée, il s'agit d'une réserve.

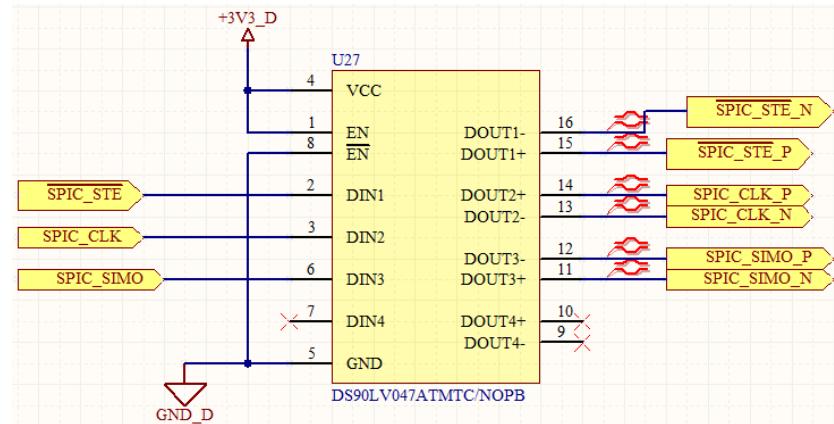


Figure 3.29 LVDS Driver

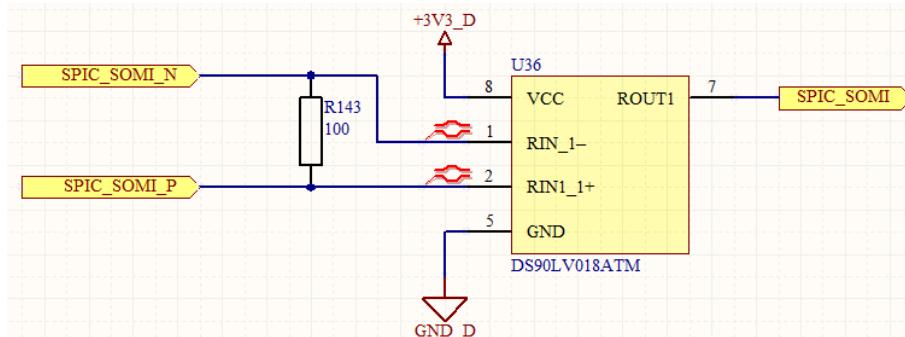


Figure 3.30 LVDS Receiver

3.5.3 PHY Ethernet

Un PHY Ethernet a été désigné sur le PCB **FPGA**. Ce bout de schématique a été repris presque tel quel de la schématique de la carte de développement FPGA-Rack². Ce PHY Ethernet permet de transmettre une grande quantité d'informations à haute vitesse. L'**USB** ne peut pas répondre à cette tâche, car il est limité par l'interface **USB** / **UART**. La schématique du PHY Ethernet se trouve à l'annexe F.

². Kit de développement FPGA développé à la HES-SO : <http://wiki.hevs.ch/uit/index.php5/Hardware/FPGARack>, sa schématique se trouve à l'annexe D

3.6 Alimentations

3.6.1 Schéma bloc

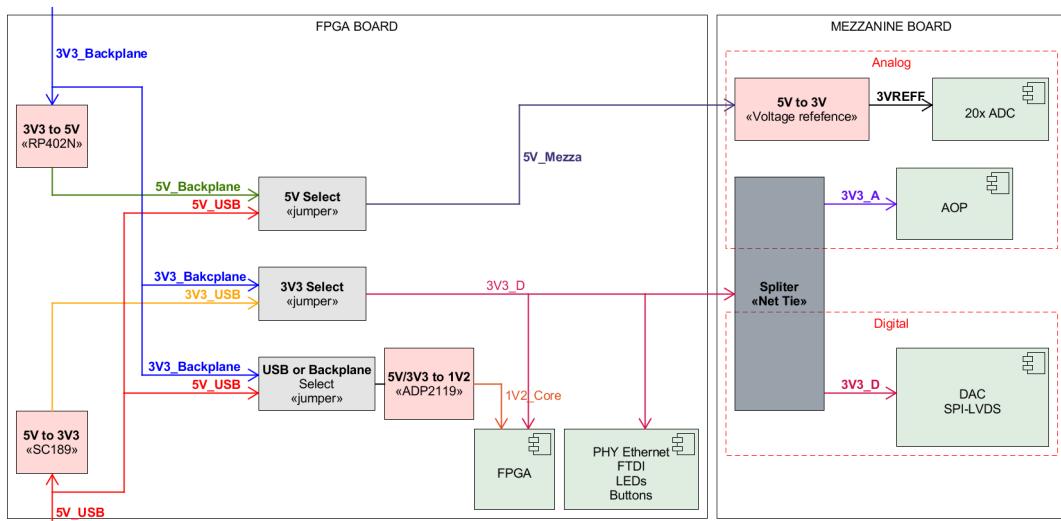


Figure 3.31 Schéma bloc de l'alimentation des PCB FPGA et Mezzanine

Sur la figure 3.31 se trouve le schéma bloc complet de l'alimentation du PCB Mezzanine et du PCB FPGA. L'ensemble peut-être alimenté par deux sources de tensions différentes. Le PCB FPGA reçoit sur son connecteur USB-C une alimentation 5V et sur son connecteur VME alimentation 3.3V (qui provient de la carte backplane). C'est lui qui s'occupe de fournir les tension nécessaire au PCB Mezzanine (par l'intermédiaire des connecteurs mezzanines). Le PCB mezzanine requiert une tension de 5V pour alimenter la référence de tension qui alimente les ADC ainsi que d'une tension de 3.3V pour le reste des ses composants. Le PCB FPGA requiert quant à lui une tension de 1V2 pour le cœur de la FPGA ainsi qu'une tension de 3.3V pour le reste de ses composants. Sur le PCB FPGA, se trouve 2 convertisseurs buck DC/DC et un convertisseur boost DC/DC. La source d'alimentation (USB ou Backplane) est choisie à l'aide des 3 jumpers.

3.6.2 1V2 Core

La **FPGA** Spartan 6 a besoin d'une tension de 1V2 pour alimenter son cœur. Pour générer cette tension, un convertisseur buck à été dimensionné à l'aide du convertisseur buck ADP2119ACPZ-1.2-R7 (visible à la figure 3.32). Sa tension de sortie est fixe. Il existe d'autres versions de ce convertisseur buck avec une tension de sortie réglable. Grâce au jumper J9, la source d'alimentation du convertisseur (sur VIN) peut être choisie facilement. En court-circuitant J9-1 et J9-2, la **FPGA** sera alimentée par l'**USB**. Si ce sont J9-2 et J9-3 qui sont court-circuités, la **FPGA** sera alimentée par le 3V3 du connecteur VME. Le courant maximal de sortie est de 2A.

Chapitre 3. Hardware

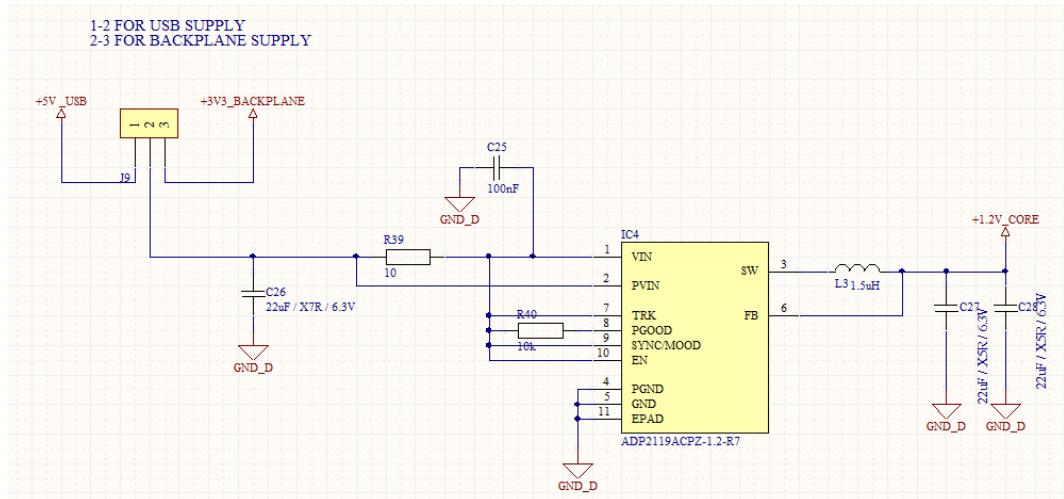


Figure 3.32 5V/3V3 vers 1V2 pour le FPGA Core

Choix de l'inductance

Selon le datasheet du ADP2119ACPZ-1.2-R7, l'inductance dépend de la fréquence de commutation du convertisseur, la tension d'entrée, la tension de sortie et la variation du courant dans la bobine. De plus, il est ajouté qu'une petite bobine augmente la variation du courant dans celle-ci et sa réponse transitoire est plus rapide. Mais son efficacité est diminuée. Pour obtenir le meilleur compromis entre efficacité et réponse transitoire, le courant dans la bobine est généralement égal aux 1/3 du courant maximum de sortie. La tension de sortie maximale est définie à 1.5A. Dans les équations 3.4 et 3.5 (fournies par le datasheet du ADP2119), sont calculées les deux inductances en fonction des deux tensions entrées possibles. Pour conserver une réponse transitoire intéressante, une inductance de 1.5uH a été choisie.

$$L_{3V3} = \frac{(V_{in} - V_{out}) * \frac{V_{out}}{V_{in}}}{\frac{I_{outMax}}{3} * f_s} = \frac{(3.3 - 1.2) * \frac{1.2}{3.3}}{\frac{1.5}{3} * 10^6} = 1.82\mu H \quad (3.4)$$

$$L_{5V} = \frac{(V_{in} - V_{out}) * \frac{V_{out}}{V_{in}}}{\frac{I_{outMax}}{3} * f_s} = \frac{(5 - 1.2) * \frac{1.2}{5}}{\frac{1.5}{3} * 10^6} = 1.52\mu H \quad (3.5)$$

3.6.3 5V vers 3V3

Un 3V3 doit être généré par un convertisseur DC/DC si l'alimentation de la carte se fait par le biais de l'USB. Pour ce faire, un convertisseur buck SC189ZSKTRT a été implémenté (visible à la figure 3.33). Il dispose d'une sortie fixe de 3V3. Un simple jumper permet de sélectionner entre l'alimentation 3V3 du connecteur VME et le 3V3 généré par ce convertisseur à l'aide du 5V de l'USB.

3.6. Alimentations

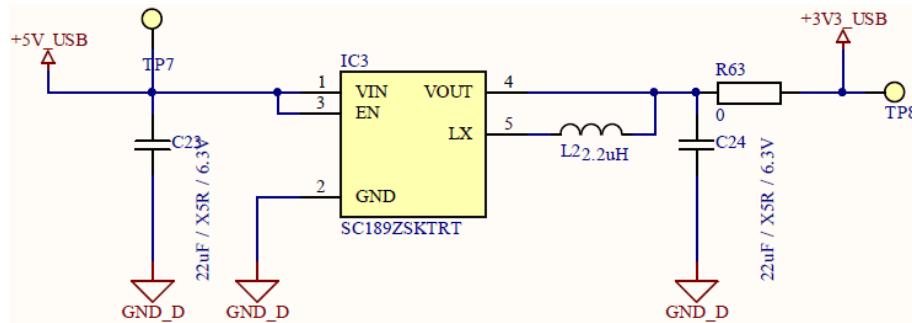


Figure 3.33 5V USB vers 3V3 USB

Choix de l'inductance

Selon les recommandations du datasheet du convertisseur buck SC189, une inductance de 2.2uH a été sélectionnée au format 5020.

3.6.4 3V3 vers 5V

Si l'alimentation de la carte FPGA se fait par le biais du connecteur VME (Backplane Board), une tension de 5V doit être générée. Un convertisseur boost, visible sur la figure 3.34, a été désigné autour du convertisseur boost DC/DC RP402N501F

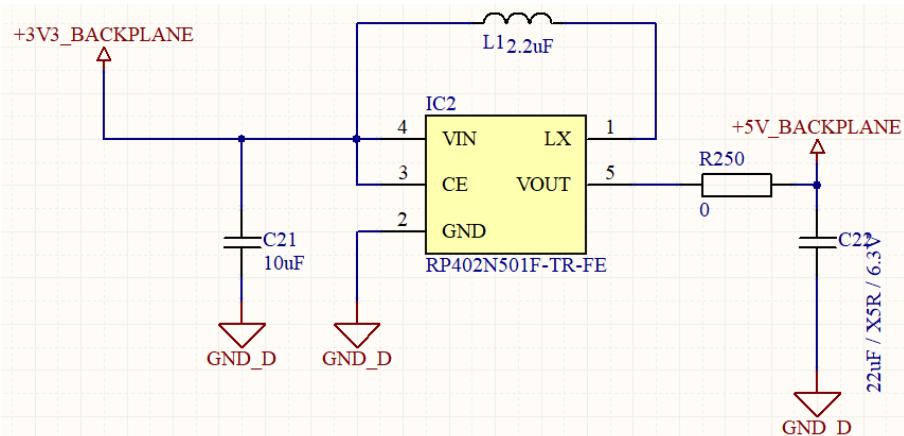


Figure 3.34 3V3 Backplane vers 5V Backplane

Choix de l'inductance

Le datasheet du convertisseur RP402N501F recommande l'utilisation de certaines références de bobines ainsi que de condensateurs (voir l'extrait à la figure 3.35). Dans cette application, la bobine NRS5020T2R2NMGJ à été choisie par facilité d'implémentation.

Recommended Components

Symbol	Descriptions
L	VLF403215MT-2R2M, 2.2 μ H, TDK VLS3012HBX-2R2M, TDK NRS5020T2R2NMGJ, TAIYO YUDEN

Figure 3.35 Inductance recommandée pour le convertisseur DC/DC RP402x[14]

3.6.5 Alimentation des ADCs

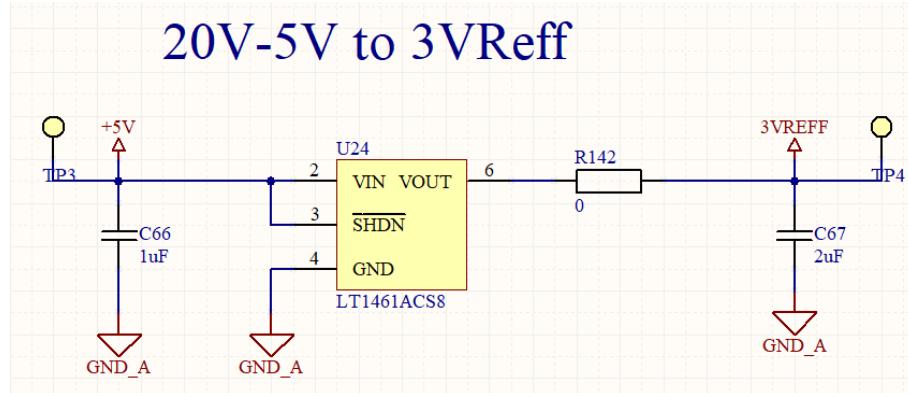


Figure 3.36 Référence de tension 3V alimentant les ADC

Pour alimenter l'ADS7886, il faut appliquer une tension sur sa pin Vdd. Cette tension d'alimentation est aussi la tension de référence de notre [ADC](#). De ce fait, la tension d'alimentation doit équivaloir à 3V (le signal d'entrée varie de 0V à 3V). La tension doit être stable pour ne pas fausser notre mesure. Une référence de tension doit alors être privilégiée. Selon le datasheet de l'ADS7886, le convertisseur consomme 1.3mA avec une alimentation de 3V (voir la figure 3.37). La consommation totale pour 20 [ADC](#) est de 26mA.

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
POWER SUPPLY REQUIREMENTS					
+V _{DD} Supply voltage		2.35	3.3	5.25	V
Supply current (normal mode)	V _{DD} = 2.35 V to 3.6 V, 1-MHz throughput		1.3	1.5	mA
	V _{DD} = 4.75 V to 5.25 V, 1-MHz throughput		1.5	2	
	V _{DD} = 2.35 V to 3.6 V, static state			1.1	
	V _{DD} = 4.75 V to 5.25 V, static state			1.5	

Figure 3.37 Spécifications d'alimentation de l'ADC[7]

Certaines références de tension peuvent fournir un courant d'environ 50mA. C'est le cas de la référence de tension 3V le LT1461CCS8-3PBF. Son datasheet indique qu'il est nécessaire d'avoir une tension de « dropout » (Vout – Vin) de 1.5V pour fournir un courant de 50mA (voir figure 3.38). Pour un courant plus faible, la tension de « dropout » (Vout – Vin) est moins élevée. Par mesure de sécurité la tension aux bornes de l'alimentation de cette référence de tension a été fixée à 5V. Cette référence de tension alimente les 20 convertisseurs A/D (voir la figure 3.36). Des condensateurs de découplage sont ajoutés proche de l'entrée Vin et de la sortie Vout.

3.7. Interfaces utilisateurs

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
Line Regulation	($V_{OUT} + 0.5V \leq V_{IN} \leq 20V$)	●	2 15	8 50	ppm/V ppm/V
	LT1461DHS8				
Load Regulation Sourcing (Note 6)	$V_{IN} = V_{OUT} + 2.5V$ $0 \leq I_{OUT} \leq 50mA$	●	12 40	30 40	ppm/mA ppm/mA
	LT1461DHS8, $0 \leq I_{OUT} \leq 10mA$				
Dropout Voltage	$V_{IN} - V_{OUT}$, V_{OUT} Error = 0.1% $I_{OUT} = 0mA$ $I_{OUT} = 1mA$ $I_{OUT} = 10mA$ $I_{OUT} = 50mA$, I and C Grades Only		0.06 0.13 0.20 1.50	0.3 0.4 2.0	V
Output Current	Short V_{OUT} to GND		100		mA

Figure 3.38 Tension dropout de la référence de tension 3V[15]

3.6.6 Points de tests

Sur le PCB FPGA et le PCB Mezzanine, se trouve une multitude de points de tests (voir figure 3.39). Il s'agit principalement de tensions d'alimentation, de références de tension et de masses (GND).

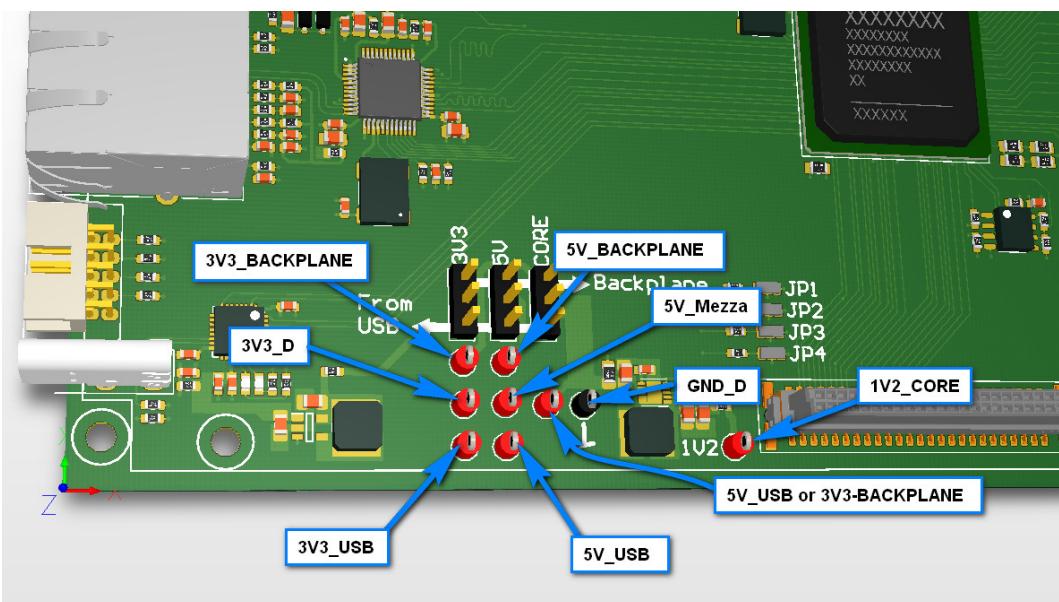


Figure 3.39 Points de tests sur le PCB FPGA

3.7 Interfaces utilisateurs

Comme sur la carte Poetic Processeur, l'utilisateur peut interagir avec la carte à l'aide de boutons, de LED qui se situent sur la face avant l'électronique.

3.7.1 LED

Comme sur la carte Poetic processeur, le PCB FPGA dispose de deux indicateurs LED³. Ces indicateurs LED contiennent chacun 4 LED bicolores superposées. Sur la figure 3.40, le sens des LED rouges est affiché. Pour faire briller une LED en vert, il suffit d'appliquer un courant inverse.

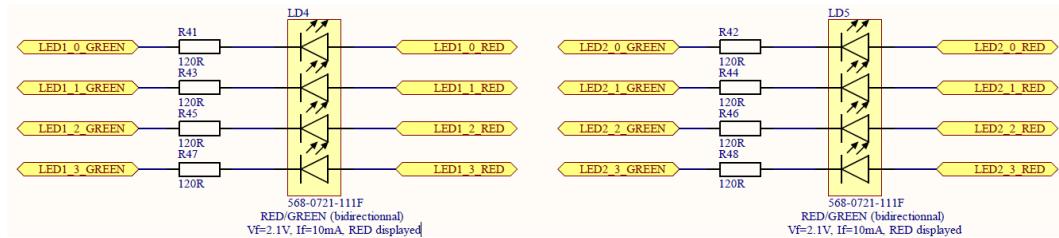


Figure 3.40 Deux indicateurs LED disposant de 4 LED bicolores superposées

3.7.2 Boutons

Le PCB FPGA dispose des mêmes boutons-poussoirs⁴ en façade que la carte Poetic processeur (voir la schématique à la figure 3.41). Des condensateurs ont été mis en parallèle pour filtrer les rebonds des boutons poussoirs.

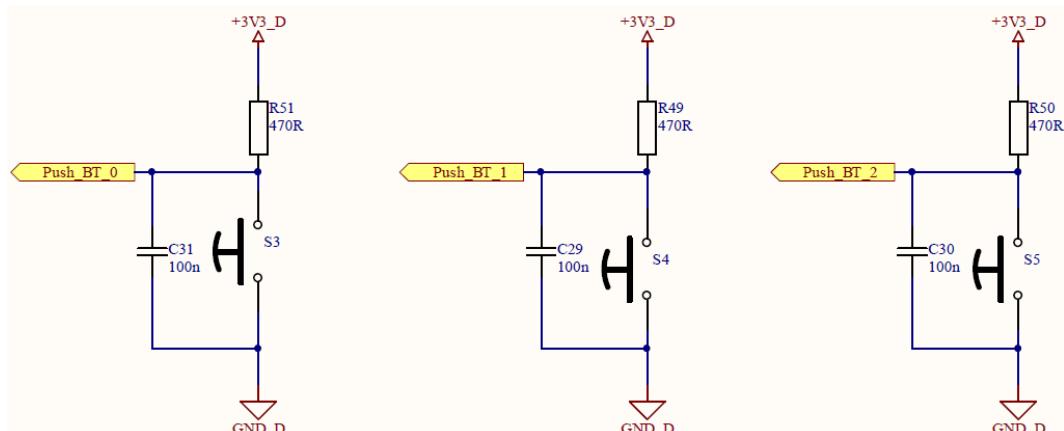


Figure 3.41 Boutons poussoirs en façade

Boutons resets

3.7.3 DIP Switchs

Un DIP switch 4 positions a été rajouté pour permettre facilement à l'utilisateur de pouvoir changer entre différents mode de configurations. Le DIP switch n'est connecté à aucun réseau de pull-up. Il est donc nécessaire d'activer des pull-up internes dans la [FPGA](#).

3. Référence fabricant de l'afficheur LED : 568-0721-111F

4. Référence fabricant des boutons-poussoirs : FSMRA4JH04

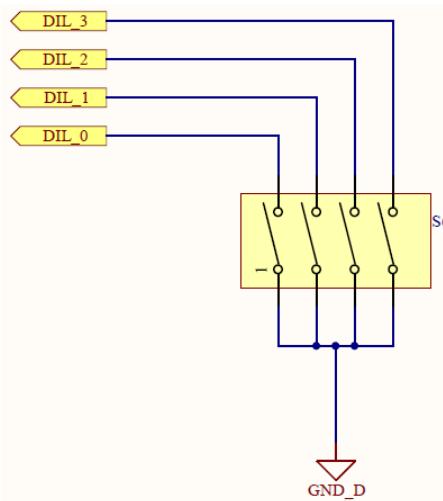


Figure 3.42 DIL Switch SMD

3.7.4 Jumper Spare

Quatre jumpers SMD supplémentaires ont été rajoutés (voir le schéma à la figure 3.43). Ces jumpers permettent tout comme les DIP switch, de définir éventuellement un mode de configurations qui, ne pourrait pas être facilement modifié par l'utilisateur. Ces jumpers SMD disposent de résistances de pull-up.

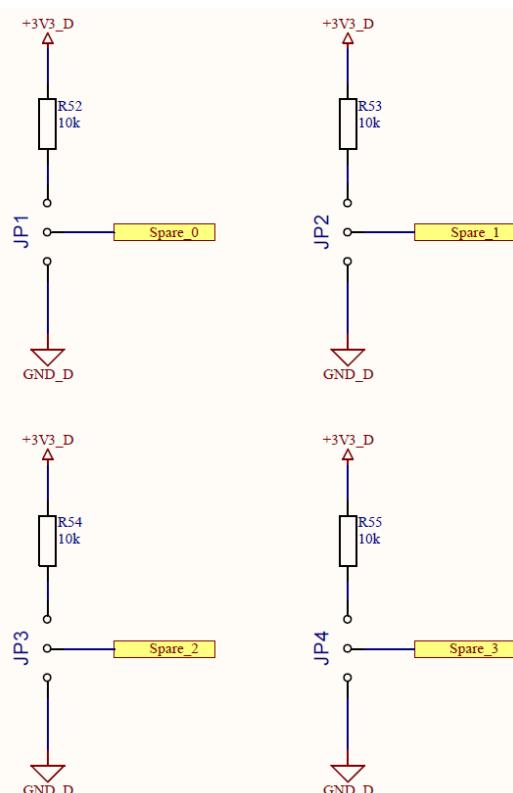


Figure 3.43 Jumper SMD de réserve

3.8 Schématique

La schématique complète se trouve à l'annexe F pour PCB FPGA et à l'annexe E pour le PCB Mezzanine.

3.9 Coûts de production

PCB FPGA

Prix de fabrication : **460 CHF** par PCB

PCB Mezzanine

Prix de fabrication : **285.10 CHF** par PCB

Les fichiers de commande des deux PCB se trouvent à l'annexe C.

3.10 PCB

3.10.1 FPGA Board

Sur la figure 3.44, se trouve le PCB FPGA. En facade (à gauche de la figure), sont placés : un connecteur Ethernet, un connecteur de programmation pour la [FPGA JTAG](#) ainsi qu'un connecteur USB-C assurant la communication série avec la [FPGA](#). Au centre du PCB, sur les extrémités, se trouve deux connecteurs Mezzanine 2x25 pins. Ces connecteurs permettent d'y «plugger» le PCB Mezzanine. Au fond du PCB (à droite), se situe le connecteur VME qui permet de se connecter sur le POETIC.

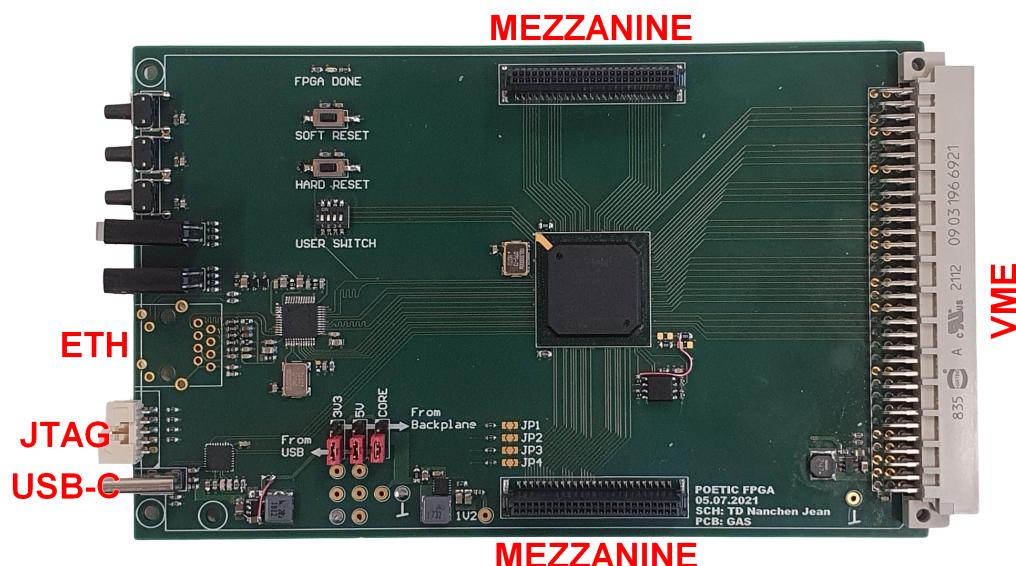


Figure 3.44 PCB FPGA

3.10.2 PCB Mezzanine

Sur la figure 3.45, se trouve le PCB Mezzanine. Sur la gauche de cette figure, sont disposés 8 connecteurs SMA pouvant générer une tension analogique. Deux connecteurs 2x10 pins situés au sommet de la carte accueillent les paires différentielles qui sont échantillonnées et quantifiées par les ADC. En bas du PCB, sont situés deux connecteurs, un 2x5 pins pour le SPI LVDS et un connecteur 2x9 pins pour les entrées-sorties SPARE. À l'arrière de cette carte se trouve deux connecteurs mezzanine 2x25 pins. Ces derniers permettent à la carte Mezzanine de venir se «plugger» sur le PCB FPGA.

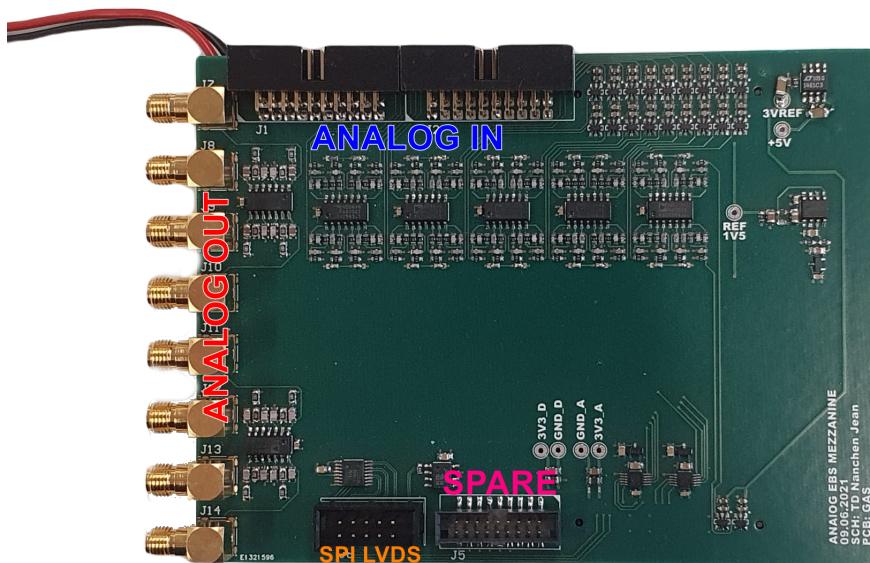


Figure 3.45 PCB Mezzanine

3.10.3 Spécificités du montage

PCB Mezzanine

- Lors du montage du PCB Mezzanine, aucun convertisseur D/A 12 bits (DAC124S085) n'étaient disponibles. Il a été temporairement remplacé par un convertisseur de même famille, mais avec une quantification de 8 bits (DAC084S085).
- Les convertisseurs A/D sur le PCB Mezzanine ont été commandés au mauvais format. Ils ont tout de même pu être soudés.

PCB FPGA

- Les condensateurs de découplages de la [FPGA](#) ne sont pas des X7R. Les 100nF ont été remplacés par des 47nF.
- Les ferrites ont été remplacées par des résistances de 0 Ohm.

3.10.4 Routage

Le routage a été réalisé par Steve Gallay.

3.10.5 Assemblage

Sur la figure 3.46, se trouve le module FPGA Poetic, assemblé. En haut se trouve le PCB Mezzanine, en bas, se trouve le PCB FPGA. La liaison entre les deux PCB est assurée par deux connecteurs SMC 50 pôles de la marque Erni.

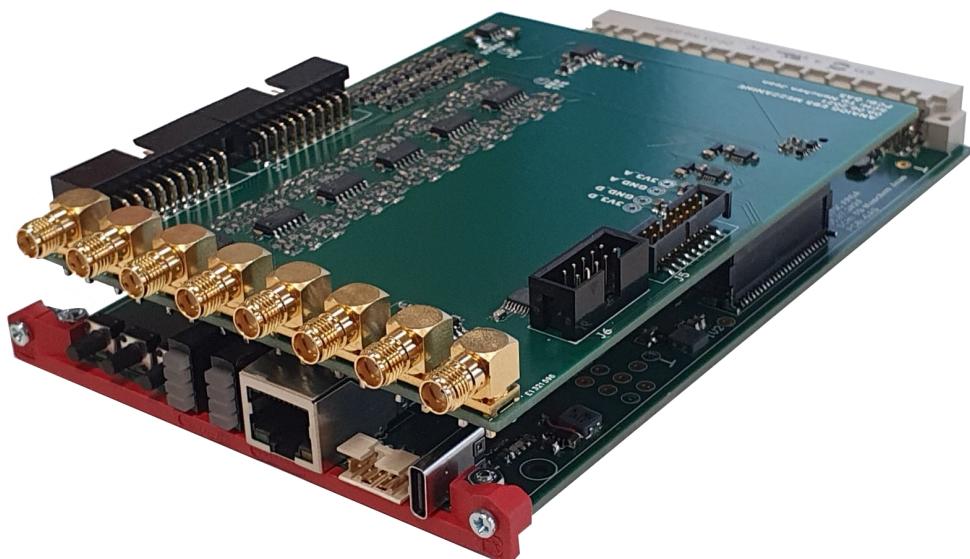


Figure 3.46 Assemblage du PCB FPGA et du PCB Mezzanine

3.11 Erreurs hardware et modifications effectuées

Lors de la mise en service de la carte PCB FPGA, plusieurs erreurs ont été observées.

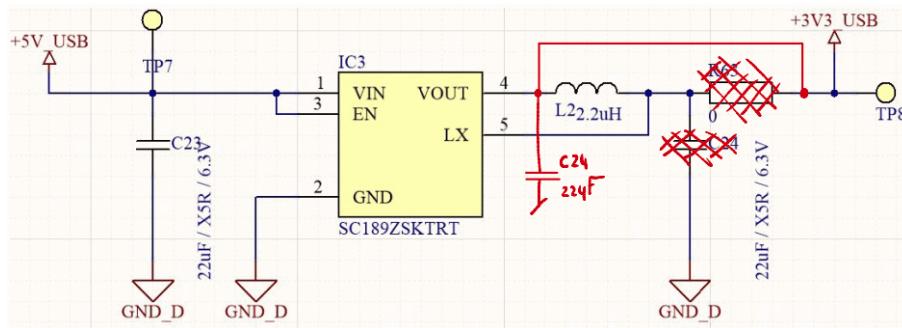
FLASH

La flash sur la première version du PCB n'était pas correctement reliée à la [FPGA](#). Il est donc impossible de l'utiliser pour stocker le programme de la [FPGA](#). Le signal de clock ainsi que le chip select ont été déplacés lors du routage. Ce problème a été corrigé sur la deuxième version du PCB, mais n'a pas pu être résolu à l'aide de fils de wrapping.

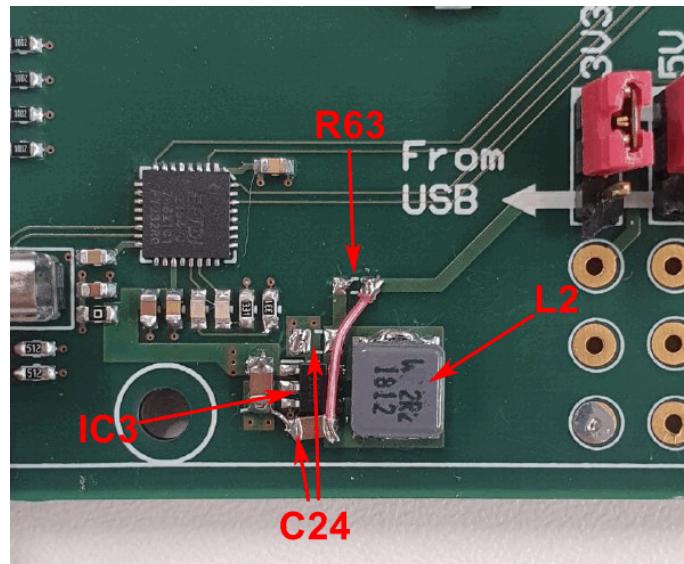
Convertisseur buck 3V3 USB

Deux pins du convertisseur buck abaissant la tension de l'[USB](#) (IC3) ont été inversées. Cette erreur a été corrigée sur le PCB et la schématique a été mise à jour sur la deuxième version.

3.12. Face avant



(a) Modifications sur la schématique



(b) Modifications sur le PCB

Figure 3.47 Modification effectuée sur le buck (V1.0)

3.12 Face avant

Une face avant a été dessiné pour le module Poetic FPGA. Elle est visible à l'annexe Q. Cette face avant n'a pas pu être gravée dû au déménagement de la HES-SO au Campus Energopolis.

4 | Architecture software

4.1 Structure

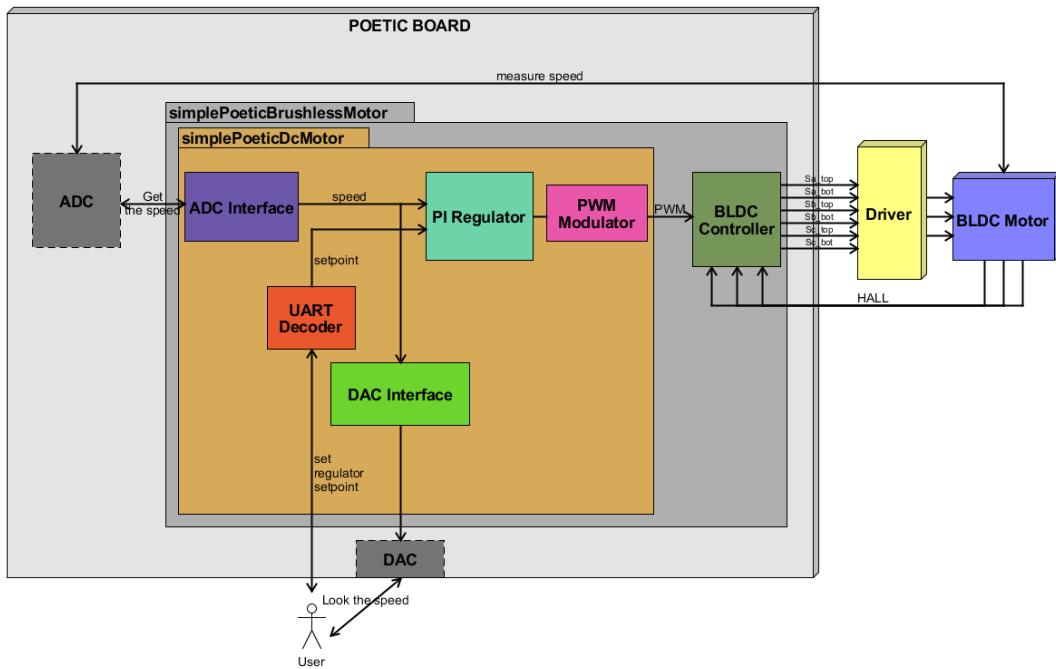


Figure 4.1 Schéma bloc de l'architecture

Une architecture, pouvant réguler deux types de moteurs, a été réalisée durant ce travail de diplôme. Un schéma bloc est disponible à la figure 4.1. À l'intérieur de cette architecture, sont situées, plusieurs interfaces afin de pouvoir communiquer avec les différents périphériques du PCB Mezzanine et du PCB FPGA. Un régulateur PI permet d'effectuer une régulation en boucle fermée, ici d'une vitesse. Un modulateur effectue une modulation PWM, celle-ci est envoyée sur le moteur par l'intermédiaire d'un «BLDC Controller». Ce bloc dispose d'une logique qui alimente les bonnes bobines au bon moment en fonction de la position du rotor (fourni par les capteurs HALL). Cette architecture a été implémentée en VHDL à l'aide de HDL designer.

4.2 ADC

Dans cette section, est développée la partie communication entre les convertisseurs analogiques numériques et la [FPGA](#). Une interface pour le convertisseur ADS7886 ainsi qu'un modèle du convertisseur ADS7886 a été implanté. Ce qui, permet de simuler la conversion A/D en simulation. Le détail sur la communication série entre le convertisseur et son maître se trouve sur le diagramme de temps de la figure 4.2. Lorsque le chip select (CS_n) est mis à 0, le convertisseur analogique digital lance la conversion. Quatre flancs montants de clock SCLK plus tard, le convertisseur envoie les 12 bits de la conversion en série sur le signal SDO.

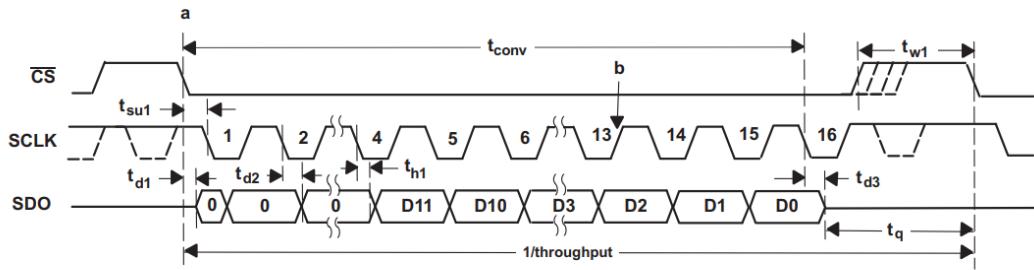


Figure 1. Interface Timing Diagram

Figure 4.2 Diagramme de temps de l'interface série du ADS7886[7]

4.2.1 Interface pour les ADC

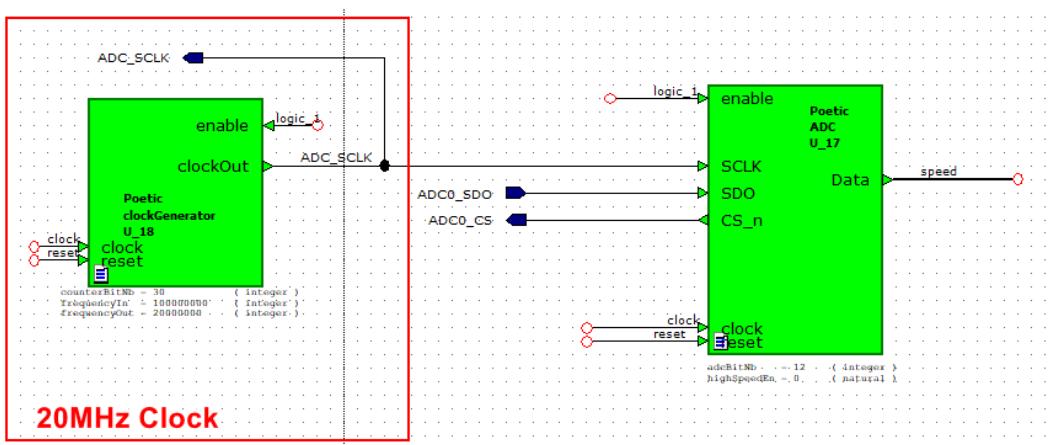


Figure 4.3 Interface pour l'ADS7886 & clock de 20MHz

Un bloc «ADC» a été implémenté dans le design (visible sur la figure 4.3). Il a permis de communiquer avec un convertisseur analogique digital qui se situe sur le PCB Mezzanine ou avec son modèle. Il fonctionne à l'aide d'une machine d'état qui progresse grâce au clock de la [FPGA](#). Elle est synchronisée au signal de clock «SCLK» de 20MHz (fourni par un clock divider) qui est aussi envoyé sur tous les convertisseurs A/D. Les données série sont reçues sur l'entrée «SDO». Elles sont ensuite déserialisées et mises sur le bus de sortie 12 bits, qui se nomme «Data». Une entrée «enable» permet de lancer les conversions sur le convertisseur A/D. Le code [VHDL](#) se trouve à l'annexe G.

4.2.2 Modèle des ADC

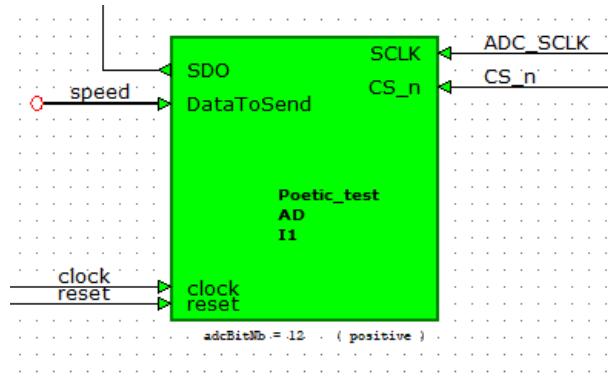


Figure 4.4 Modèle simulant le convertisseur A/D ADS7886

Sur la figure 4.4, se trouve le modèle du convertisseur analogique digital ADS7886, implémenté sur le PCB Mezzanine. Il permet de reproduire à l'identique l'ADS7886. Il transmet la valeur de son bus d'entrée 12 bits en série sur sa sortie SDO. Le code [VHDL](#) se trouve à l'annexe H.

4.2.3 Simulation

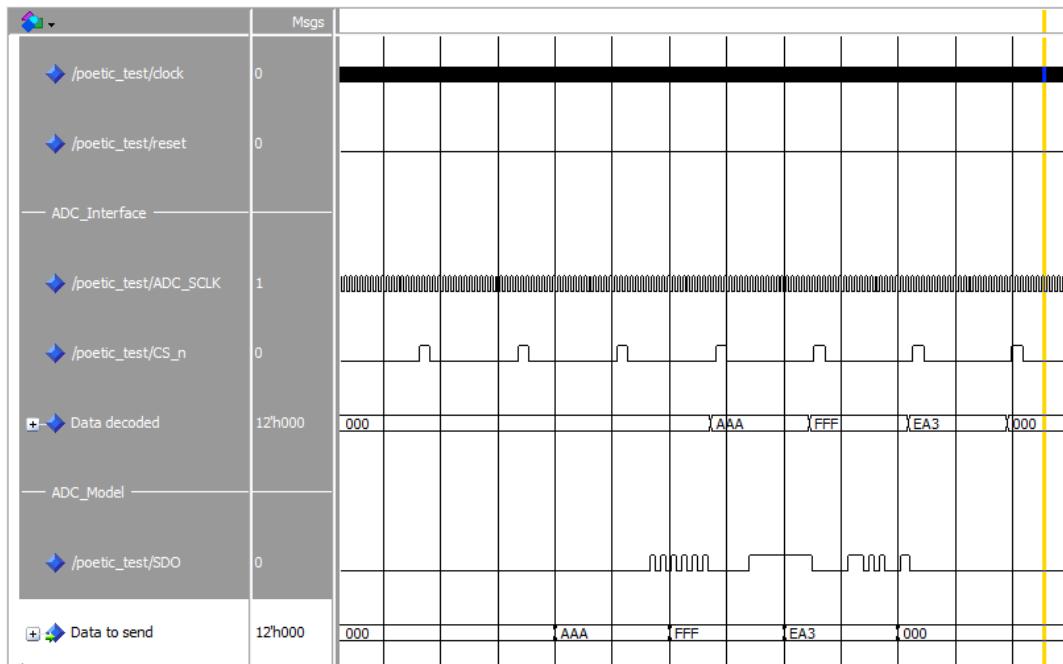


Figure 4.5 Simulation de l'interface de l'ADC et du modèle de l'ADC

Sur la simulation qui se trouve à la figure 4.5, la communication entre le modèle et l'interface du convertisseur A/D ADS7886. Le modèle qui envoie successivement 0xAAA, 0xFFFF, 0xEA3 ainsi que 0x000 sur sa sortie SDO (valeurs qui se trouvent sur le bus «Data to send»). Les données sont envoyées lorsque l'interface lui envoie un flanc

descendant sur son entrée chip select (CS_n). L'interface quant à elle, dé-sérialise les données transmises sur le signal série SDO et l'affiche sur la sortie qui se nomme «Data decoded».

4.3 DAC

Cette section introduit la communication avec les convertisseurs numériques analogiques DAC124S085 et DAC084S085 (voir section 3.10.3). Sur la figure 4.6, se trouve un diagramme de temps qui montre la communication série avec la famille de convertisseur D/A DACXX4S085 et son maître. Le maître envoie un flanc descendant sur la ligne «SYNC_n», et ensuite transmet en série un paquet 16 bits sur l'entrée «DIN» du convertisseur D/A. Le contenu de ce paquet à transmettre se trouve sur la figure 4.7 pour le convertisseur 12 bits et sur la figure 4.8 pour le convertisseur 8 bits. Le paquet se compose donc de 4 bits de configurations, ainsi que N bits à transmettre (dépendant du modèle de convertisseur). La configuration A1-A0 permet de sélectionner une des quatre sorties du convertisseur digital analogique. L'autre configuration, OP1-OP2, sur deux bits, permet de sélectionner le mode d'écriture. Par exemple, il est possible d'écrire dans les quatre registres de sorties, et d'actualiser les sorties à la fin de ces écritures. Cela permet de synchroniser les quatre canaux. Une fois cette trame de 16 bits envoyée, le maître doit remettre à 1 le signal «SYNC_n». 4.7.

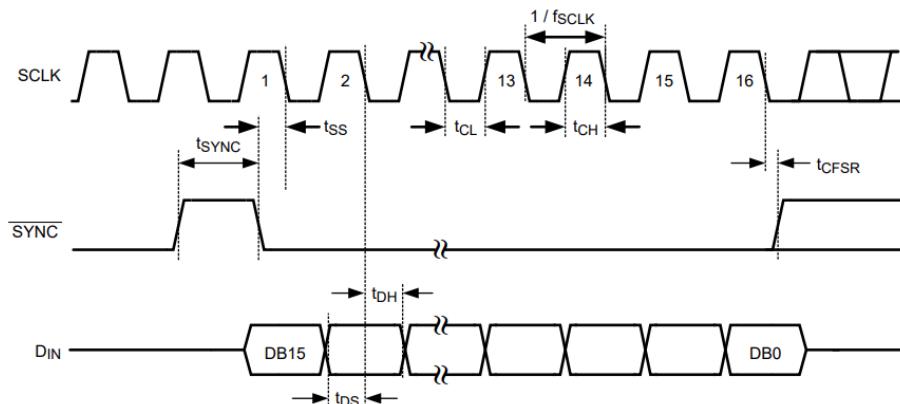


Figure 4.6 Diagramme de temps de l'interface série du DACXX4S085[16]

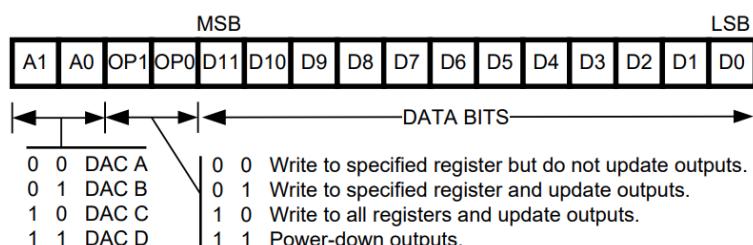


Figure 4.7 Paquet 16 bits envoyé au convertisseur D/A 12 bits DAC124S085[16]

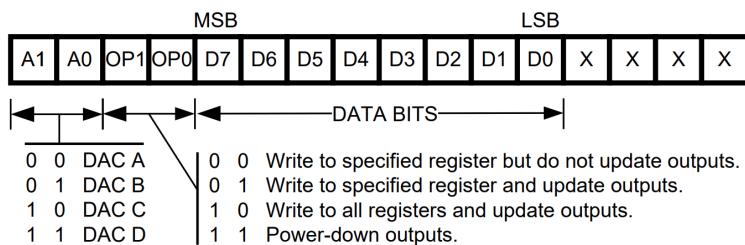


Figure 4.8 Paquet 16 bits envoyé au convertisseur D/A 8 bits DAC084S085[17]

4.3.1 Interface pour les DAC

Sur la figure 4.9, se trouve l'interface du convertisseur digital analogique avec son générateur de clock 40MHz. Il est important de noter que l'interface **DAC** est configurée pour discuter avec un **DAC** 8 bits (voir chapitre 3.10.3). Si le **DAC** à une quantification de 12 bits, il suffit de modifier sa constante «dacBitNb». Dans le carré bleu se trouvent les quatre bits de configurations. Des signaux logiques 0 et 1 configurent le convertisseur pour envoyer sur toutes les sorties et rafraîchir les sorties à chaque envoi, voir la figure 4.7 qui parle du paquet 16 bits contenant les quatre bits de configurations. Une entrée «send» permet d'activer l'envoi ou non. Cette interface concatène donc l'entrée «data» qui est un bus 8 bits qui contient les données à envoyer, «dacSel» qui est un bus 2 bits contenant les bits de configurations A1-A0 ainsi que «mode» qui est un bus 2 bits contenant le bits de configurations OP1-OP0. Cette trame 16 bits est envoyée en série sur la sortie «Dout» cadencée par le clock «SCLK». L'interface génère aussi le flanc descendant sur le signal «Sync_n» avant l'envoi de la trame 16 bits. Le code **VHDL** se trouve à l'annexe I.

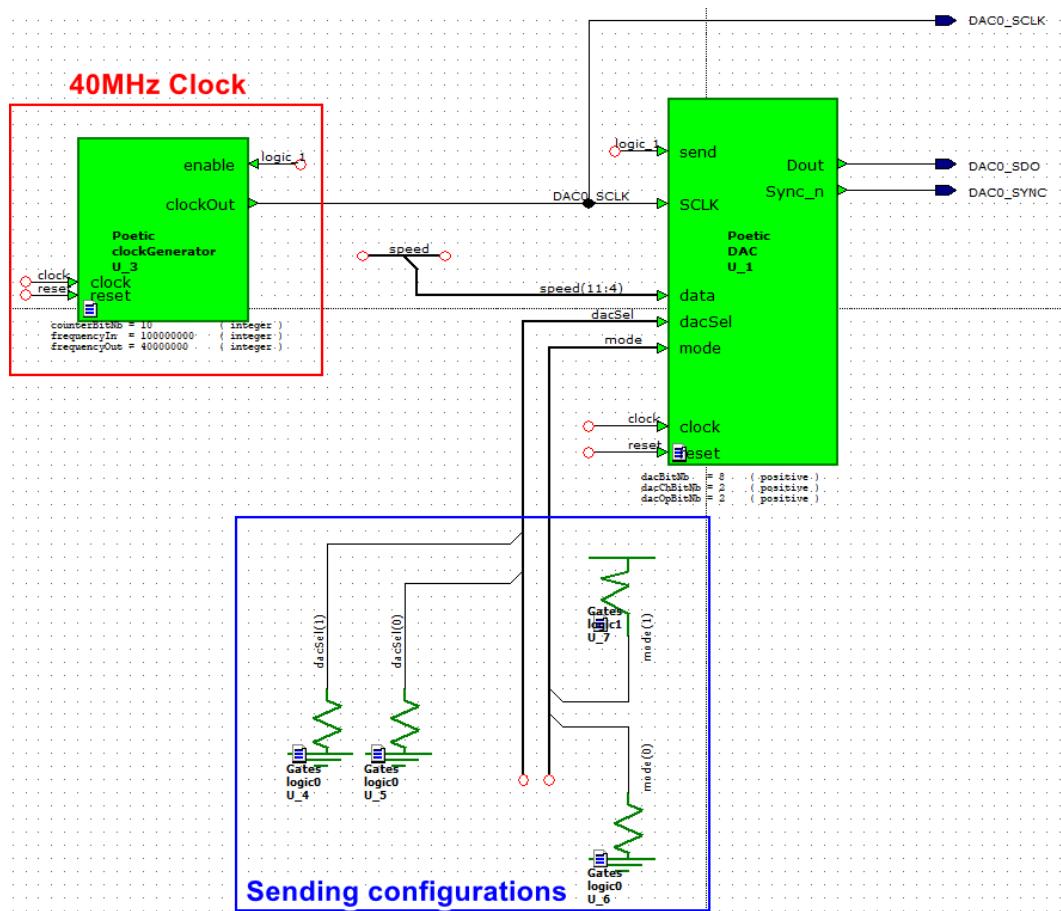


Figure 4.9 Interface DAC avec son clock de 40MHz et ses bits de configurations

4.3.2 Simulation

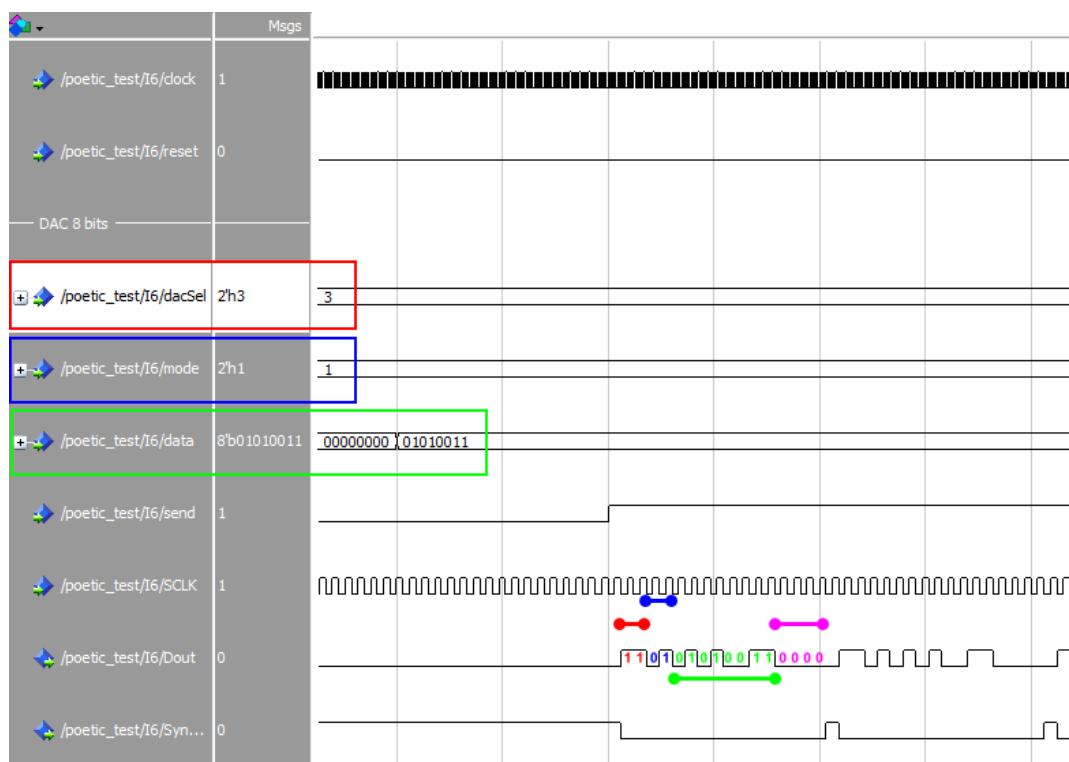


Figure 4.10 Simulation de l'interface DAC (8 bits)

Sur la figure 4.10, est simulé le bloc d'interface DAC 8 bits. On y observe l'envoi successif sur la sortie «Dout» des quatre bits de configurations «dacSel» et «mode» suivis des 8 bits de données, puis pour finir de quatre zéros pour remplir le paquet 16 bits (car il s'agit d'une interface pour un DAC 8 bits, voir la figure 4.8 et le chapitre 3.10.3).

4.4 Régulation & modulateur PWM

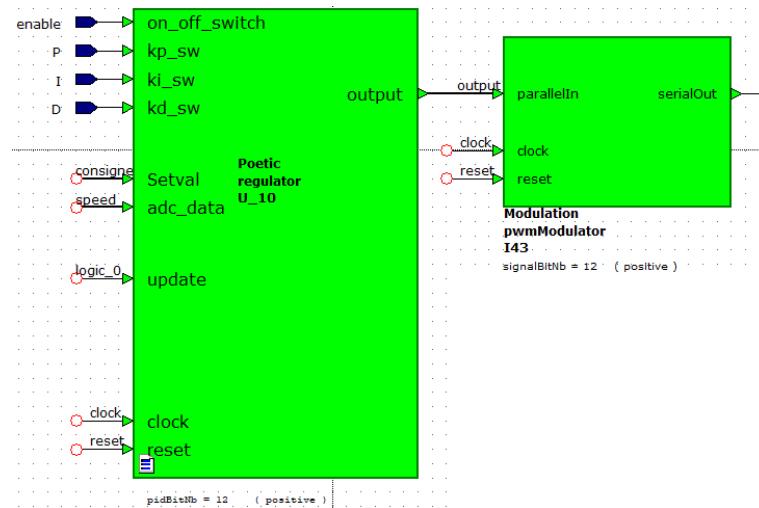


Figure 4.11 Régulateur PI, modulateur PWM et un contrôleur BLDC pour un moteur brushless DC (BLDC)

Un régulateur PI a été implémenté dans ce design. Celui-ci, visible à la figure 4.11, permet l'asservissement d'une boucle de régulation. Il reçoit en entrée une consigne de vitesse sur son bus «Setval» ainsi que la vitesse mesurée sur le bus d'entrée «adc_data». Le signal de commande est ensuite envoyé sur la sortie «output» et est transformé en PWM par un modulateur PWM. La PWM générée par le modulateur à une résolution de 12 bits. Avec un quartz cadençant la FPGA à 100MHz ainsi qu'une résolution de 12 bits, la fréquence de la PWM équivaut à 24.4KHz. Le code du régulateur PI se trouve à l'annexe J.

4.5 Modèle moteur DC

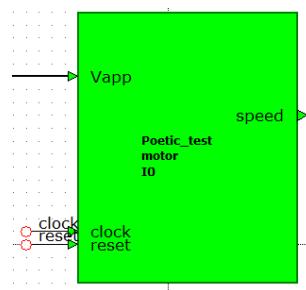


Figure 4.12 Modèle simulant un moteur DC

Pour pouvoir simuler une régulation en boucle fermée d'un moteur, un modèle d'un moteur DC a été implémenté (voir la figure 4.12). Ce bloc reçoit en entrée, sur un bus 12 bits, une «tension». En sortie se trouve la vitesse du moteur encodée sur un bus 12 bits.

4.5.1 Simulation

Sur la figure 4.13, un saut indiciel sur l'entrée «Vapp» du moteur a été simulé. Il est possible d'observer la courbe d'accélération du modèle ainsi que la courbe de décélération.

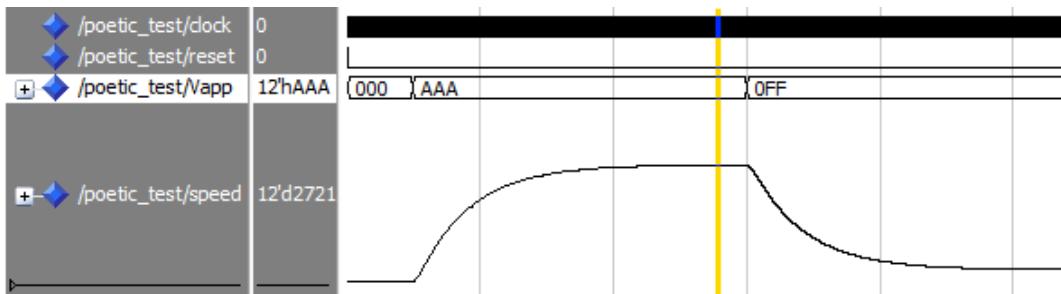


Figure 4.13 Saut indiciel sur le modèle du moteur DC

4.6 Générateur de clock

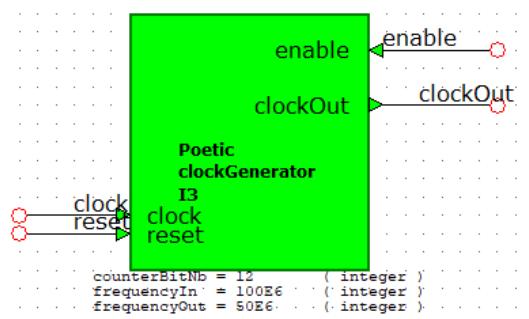


Figure 4.14 Générateur de clock à 50MHz

Le bloc visible à la figure 4.14, permet de générer un signal clock à une fréquence voulue à partir du clock de 100MHz qui cadence la **FPGA**. La fréquence de sortie est définie grâce aux paramètres génériques du bloc. Ce bloc permet de fournir un signal clock à 40MHz aux **DAC** ainsi qu'un clock de 20MHz aux **ADC**. Le code **VHDL** se trouve à l'annexe L.

4.6.1 Simulation

Dans la simulation visible à la figure 4.15, le clock qui cadence la **FPGA** a été divisé par deux. Le signal «clockOut» a donc bien une fréquence de 50MHz.

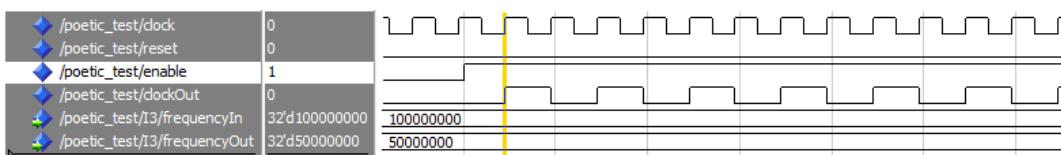


Figure 4.15 Simulation du générateur de clock

4.7 Définir une consigne à travers l'UART

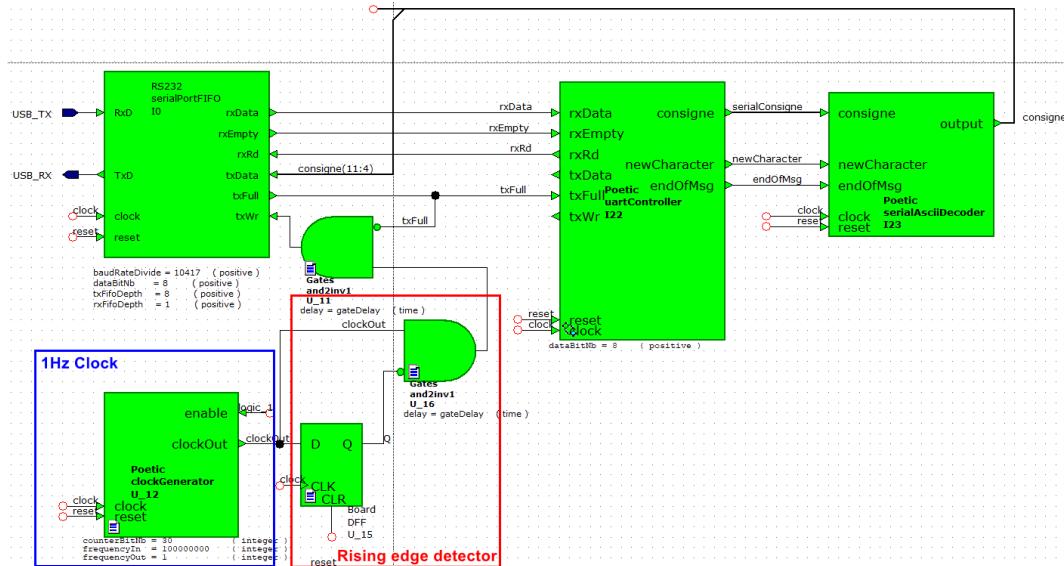


Figure 4.16 UART, avec son contrôleur et son décodeur

Grâce à l'architecture visible sur la figure 4.16, l'utilisateur peut changer / définir la consigne du régulateur. La consigne doit être empaquetée dans un paquet visible sur le tableau 4.1. Le caractère «S» au début du paquet indique l'utilisateur souhaite changer de consigne, les données sont ensuite envoyées à la suite. Un «e» indique que la transmission est terminée. Une fois le paquet envoyé via [UART](#), la consigne est automatiquement mise à jour en sortie. Les 8 bits de poids fort de la consigne sont envoyés sur l'[UART](#).

Commande	Données				Fin de transmission
S	2	7	3	0	e

Table 4.1 Paquet pour définir une consigne de 2730

4.7.1 Interface UART série

Un bloc «serialPortFIFO» de la librairie RS232 a été utilisé. Il permet de dé-sérialiser les données série arrivant sur «RxData» sur le bus «rxData». Il permet aussi de sérialiser les données sur «txData» à envoyer sur «TxD». «rxData» et «txData» pointent tous deux sur une [FIFO](#). Des signaux logiques indiquent l'état des [FIFO](#). Si une donnée est disponible dans la [FIFO](#) «rxData», «rxEmpty» passe à 0. Pour vider la [FIFO](#), il suffit de mettre à 1 le signal «rxRd». De même pour la [FIFO](#) «txData», une pulsation sur le signal logique «txWr» permet l'envoie du contenu se trouvant sur le bus «txData». Le signal logique «txFull» passe à 1 lorsque la [FIFO](#) d'envoi est pleine. 4.17.

4.7.2 Contrôleur

Le contrôleur «uartController» lit le contenu de la [FIFO](#) «serialPortFIFO». Quand un caractère de commande est envoyé, par exemple le caractère «S» (voir le tableau 4.1), le contrôleur envoie les futures données au bloc «serialAsciiDecoder». À chaque nou-

4.7. Définir une consigne à travers l'UART

veau caractère, une pulsation d'une largeur de clock est envoyée sur le signal logique «newCharactere». Le contrôleur attend la réception du caractère de fin de transmission, en l'occurrence le caractère «e», pour envoyer une pulsation sur le signal logique «endOfMsg». Le code [VHDL](#) du contrôleur se trouve à l'annexe [M](#),

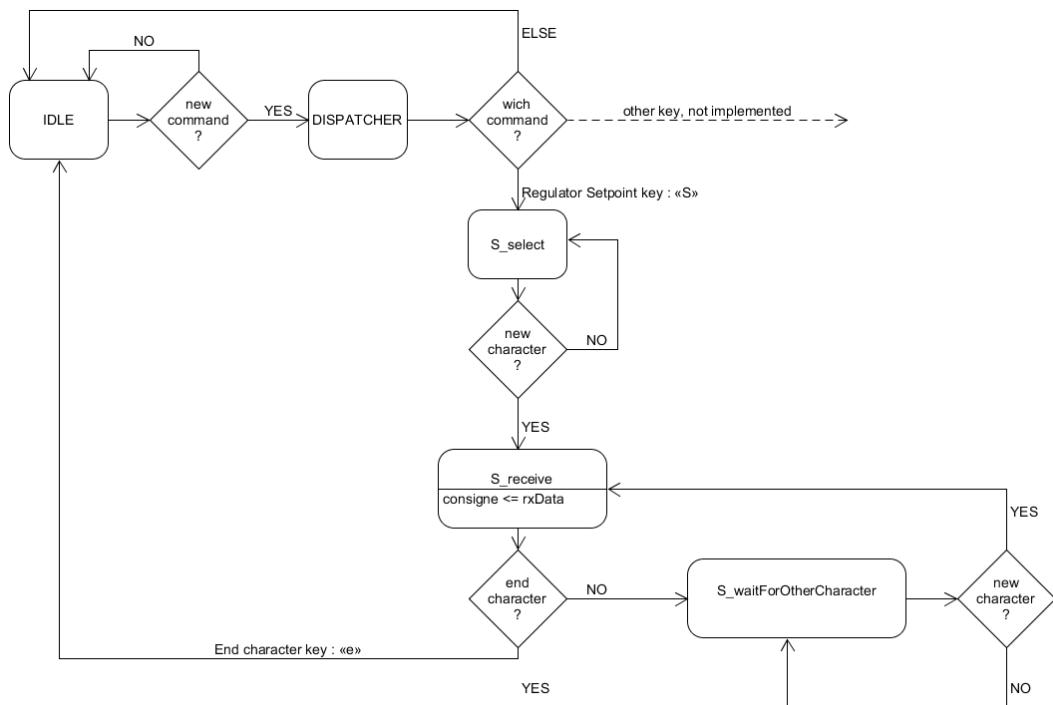


Figure 4.17 State machine du bloc «uartController»

4.7.3 ASCII série vers bus parallèle

Le bloc «serialAsciiDecoder», dé-sérialise et décode une succession de chiffres ASCII lui étant envoyé. Il a besoin de savoir la présence d'un nouveau caractère avec l'entrée «newCharacter» ainsi que de la fin du message avec l'entrée logique «endOfMsg». Ces deux signaux de contrôles sont fournis par le contrôleur. Le code [VHDL](#) du décodeur se trouve à l'annexe [N](#).

4.7.4 Simulation

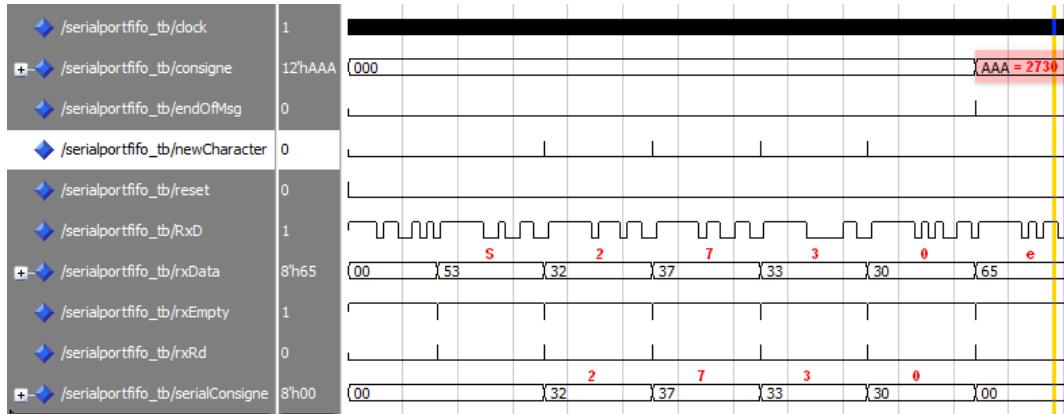


Figure 4.18 Envoi de «S2730e» sur le port UART, pour définir la consigne à 2730

Un simulation à la figure 4.18 permet d'illustrer l'envoi de «S2370e» à travers le port [UART](#) (depuis le PC). À la sortie du bloc «serialAsciiDecoder», sur le bus «consigne», se trouve 0xAAA (hexadécimal), soit 2730 en décimal.

4.8 BLDC Controller

4.8.1 Pilotage d'un moteur BLDC

En règle générale, le pilotage d'un moteur brushless DC s'effectue à l'aide trois ponts de MOSFET (voir figure 4.19). Les ponts sont pilotés par des [PWM](#) car cette modulation offre un contrôle précis de la vitesse et du couple du moteur. Il existe 3 schémas de commutation, la commutation trapézoïdale, la commutation sinusoïdale et la commutation «field-oriented».

La commutation trapézoïdale est la plus simple d'utilisation, car il s'agit, à chaque étape, d'alimenter deux bobines et laisser l'autre «flotter». Cette méthode a pour inconvénient de produire des ondulations de couple à basse vitesse[18].

La commutation sinusoïdale consiste à alimenter les trois bobines continuellement, avec des sinus (modulés par des [PWM](#)) déphasés à 120°. Cette commutation est bien plus «douce» et «ronde» car elle ne produit presque aucune ondulation de couple[18].

La commutation «field-oriented» est la plus complexe de toutes. Selon l'article «Field Oriented Control[19]», ce schéma de commutation consiste à faire en sorte que l'aimant du rotor soit toujours à 90° du champ magnétique en mesurant le courant qui passe dans ses bobines. En effet, le champ magnétique induit dépend du courant qui traverse la bobine, et non de la tension. Il y a donc un petit déphasage entre la tension et le courant. C'est ce phénomène qui est corrigé avec ce style de commutation.

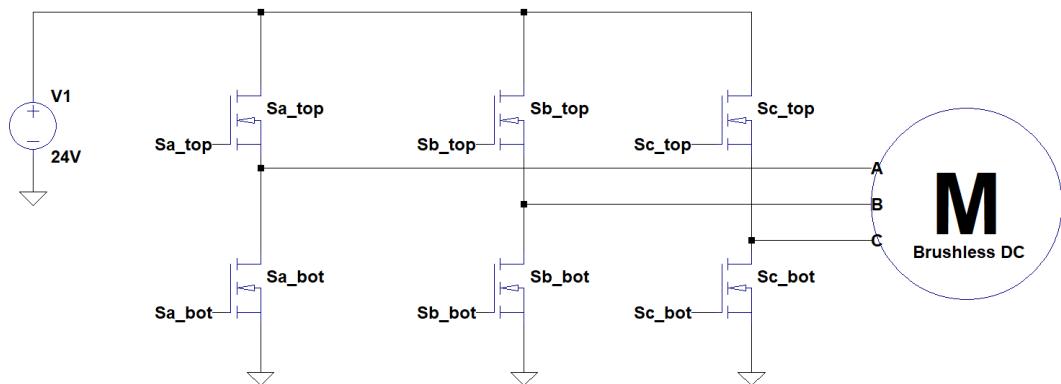


Figure 4.19 Trois paires de MOSFET pilotant un moteur brushless DC

4.8.2 Schéma trapézoïdal

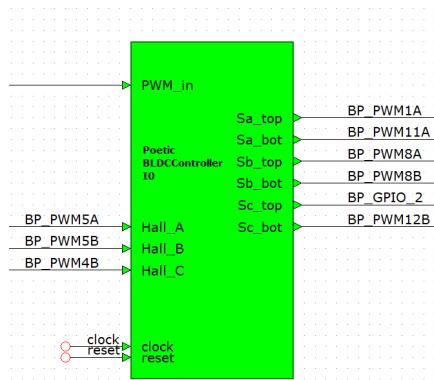


Figure 4.20 Bloc logique «BLDCCController» caractérisant le schéma de commutation trapézoïdale

Dans ce travail de diplôme, une commutation trapézoïdale a été implémentée (la commutation sinusoïdale a été implémentée, mais n'a pas été testée). Avec l'aide de l'article «How to Power and Control Brushless DC Motors[18]», le schéma de la commutation trapézoïdale a pu être défini (voir le tableau 4.2). Le bloc «BLDCCController», visible à la figure 4.20, est un bloc purement combinatoire. En fonction du code hall fourni par le moteur, la PWM est dirigée sur le bon MOSFET. Son code VHDL se trouve en annexe O.

Hall Code (ABC)	001	101	100	110	010	011
Sa_top	1	1	0	0	0	0
Sa_bot	0	0	0	1	1	0
Sb_top	0	0	1	1	0	0
Sb_bot	1	0	0	0	0	1
Sc_top	0	0	0	0	1	1
Sc_bot	0	1	1	0	0	0

Table 4.2 Schéma de commutation trapézoïdale

4.9 Régulation d'un moteur DC

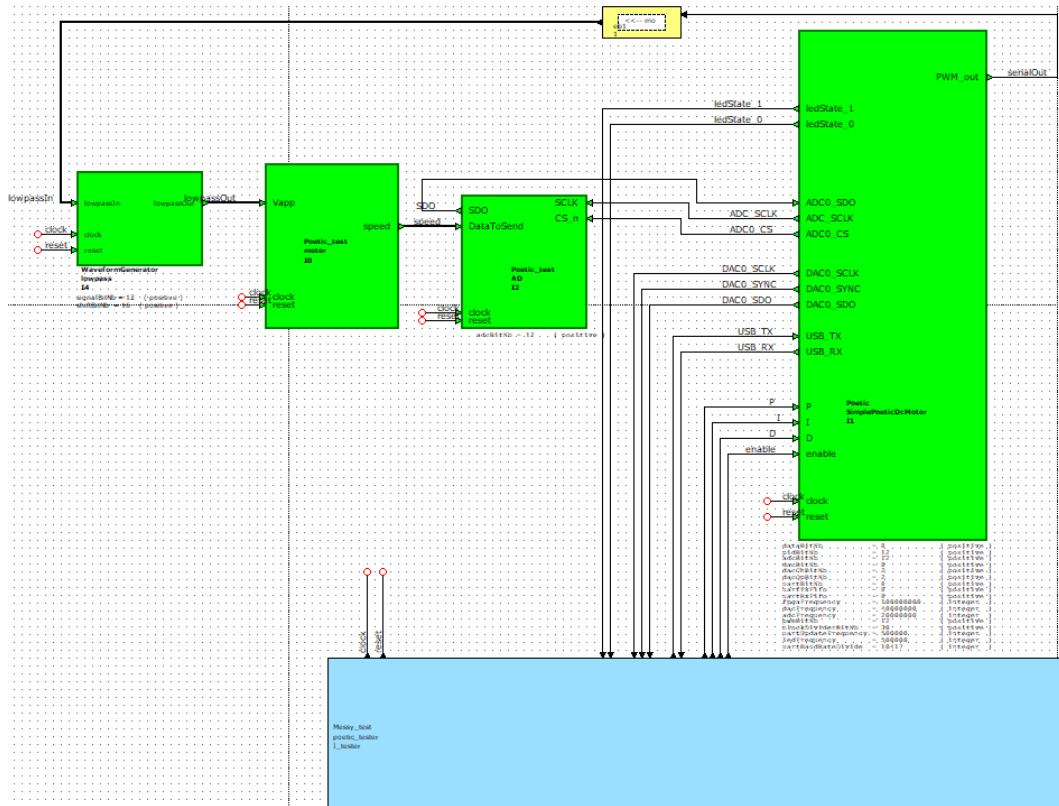


Figure 4.21 Banc de test d'une régulation d'un moteur DC

Dans le banc de test visible à la figure 4.21, se trouve une régulation de vitesse à boucle fermée d'un moteur DC. La **PWM** en sortie du bloc régulateur «simplePoeticDcMotor» est filtrée par le bloc «lowPassFilter». Il en résulte, une valeur «analogique» codée sur un bus 12 bits. Cette valeur est donc appliquée au modèle «motor» (correspond par exemple à la tension appliquée sur le moteur). La vitesse (valeur sur 12 bits) est ensuite transmise au bloc de régulation «simplePoeticDcMotor» par l'intermédiaire du modèle du convertisseur A/D.

4.9.1 Simulation

Deux réponses indicielles ont été testées en simulation pour prouver le bon fonctionnement du régulateur.

Sur la figure 4.22, se trouve une réponse indicielle. La consigne est mise à 0xF00. Dans un premier temps, un régulateur P asservit le système et celui-ci se stabilise avec une certaine erreur statique. Puis le facteur I est ajouté pour corriger l'erreur statique. On peut observer qu'au curseur jaune, le système a bien atteint la valeur de consigne qui est de 0xF00.

Sur la simulation se trouvant à la figure 4.23, un autre saut indiciel a été simulé, cette fois-ci avec un régulateur PI. La consigne a été fixée à 0xB00. On peut y voir le moteur se stabiliser à la valeur de consigne 0xB00.

4.9. Régulation d'un moteur DC

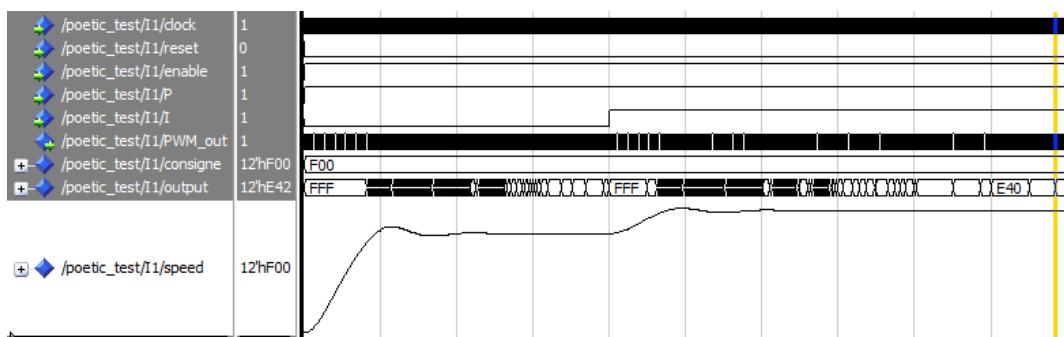


Figure 4.22 Réponse indicielle avec le régulateur P , puis activation du I qui corrige l'erreur statique

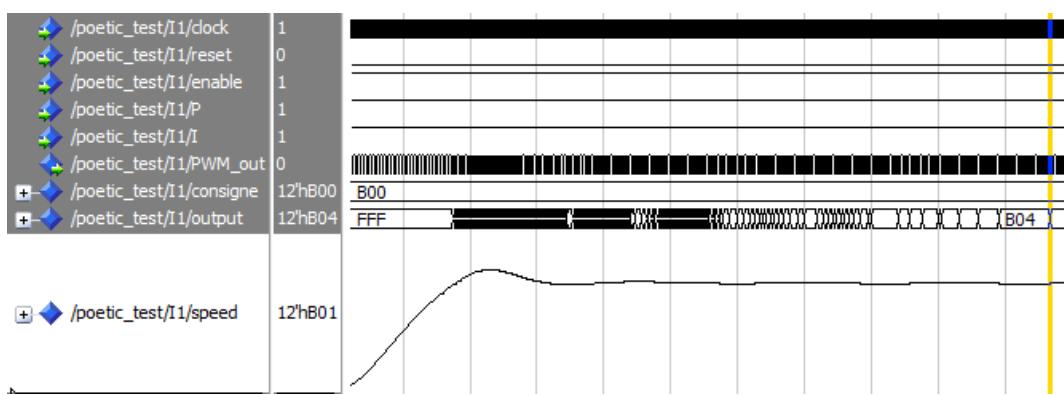


Figure 4.23 Réponse indicielle avec le régulateur PI

5 | Démonstrateur

5.1 Configuration du rack Poetic

La figure 5.1 illustre la structure mise en place pour effectuer la régulation d'un moteur BLDC à l'aide du simulateur HIL Typhoon. Le rack dispose d'au total 4 modules. Le module Poetic FPGA, le module fibre, le module DI/DO ainsi que le module mesure. Le simulateur Typhoon lui, est connecté à deux modules. Le module TYPHOON CIA pour les entrées sorties analogiques et le module TYPHOON CID pour les entrées sorties digitales. Ces deux modules permettent de relier le rack Poetic au simulateur HIL.

Les connections entre le rack Poetic ainsi que le simulateur sont détaillées dans le schéma bloc qui se trouve à la figure 5.2. Les PWM sont envoyées au simulateur avec des fibres optiques. Les signaux des trois capteurs hall sont reçus par un câble interface 9 pins. La vitesse, analogique, est transmise au rack Poetic par un câble RJ-45.

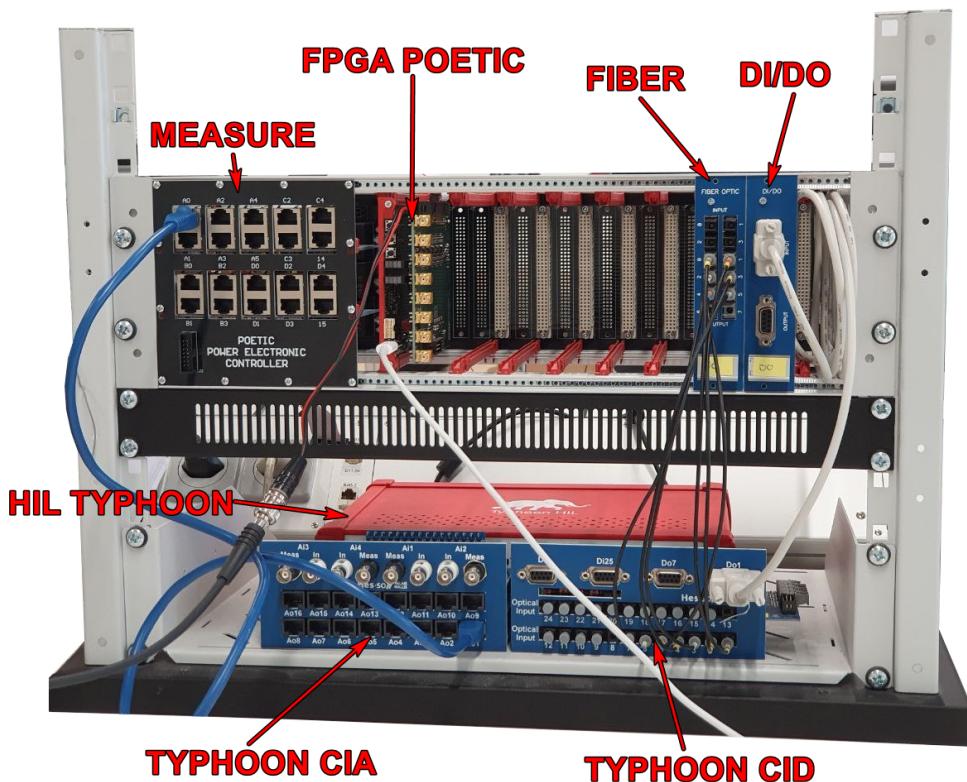


Figure 5.1 Photo du rack Poetic avec les modules nécessaires au démonstrateur

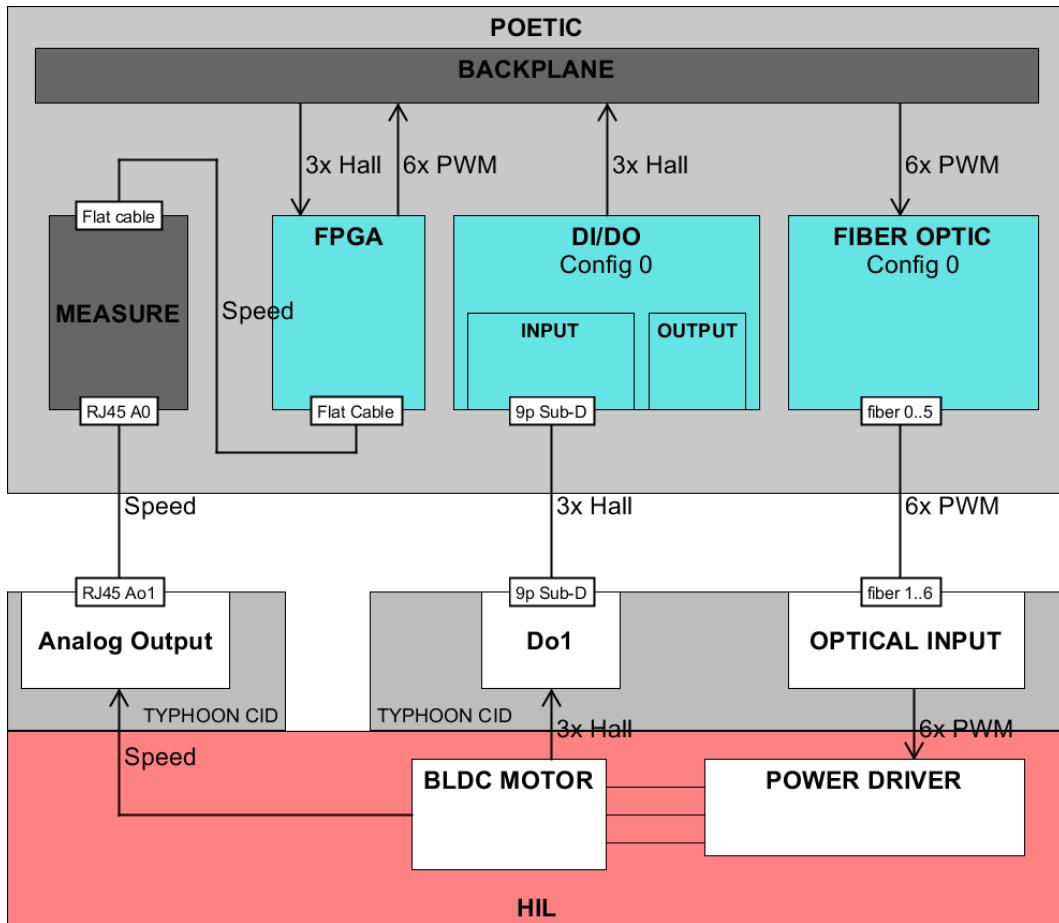


Figure 5.2 Schéma bloc du rack Poetic connecté au simulateur HIL pour la régulation d'un moteur BLDC

5.2 Schématique dans le simulateur Typhoon HIL

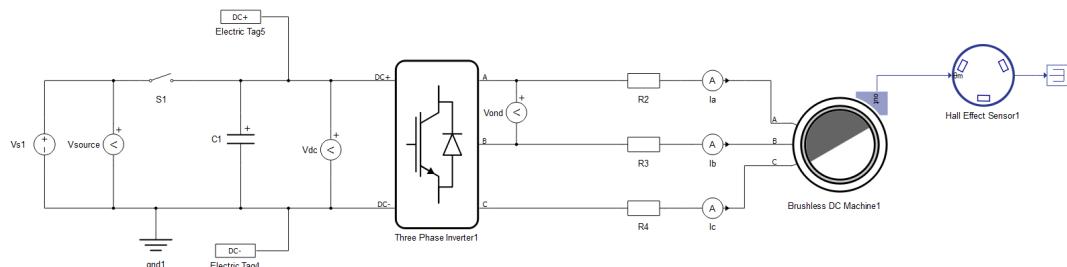


Figure 5.3 Schéma de l'électronique de puissance dans le simulateur HIL Typhoon

Sur la figure 5.3, se trouve le schéma de l'électronique de puissance à l'intérieur du simulateur HIL. Cette schématique est composée d'une alimentation, d'un driver contenant 3 pont de MOSFET pour piloter la puissance, un moteur brushless dc ainsi qu'un capteur Hall.

5.3 Simulation

Sur la figure 5.4, se trouvent trois oscilloscopes. En haut à gauche se trouve les 6 signaux PWM envoyés par le module FPGA polarisant les MSOFET de l'étage de puissance. Ces PWM ont une fréquence de 25KHz (voir la section PWM en 5.3.2). Sur l'oscilloscope en bas à gauche, se trouve les signaux sortant du capteur Hall. Sur l'oscilloscope en haut à droite, se trouve la vitesse. Elle est négative, car le schéma de commutation réalisé fait tourner le moteur dans le sens antihoraire. Pour palier à ce problème, il faudrait implémenter en plus, un schéma de commutation horaire, dans le bloc BLDC Controller (voir le chapitre 4.8). Le moteur tourne à la vitesse de 786.4 rad/s, ce qui correspond environ à la consigne de 790 rad/s envoyée au régulateur. Sur la simulation, il est possible d'ajouter un couple de charge en sortie du moteur pour pouvoir le «freiner».

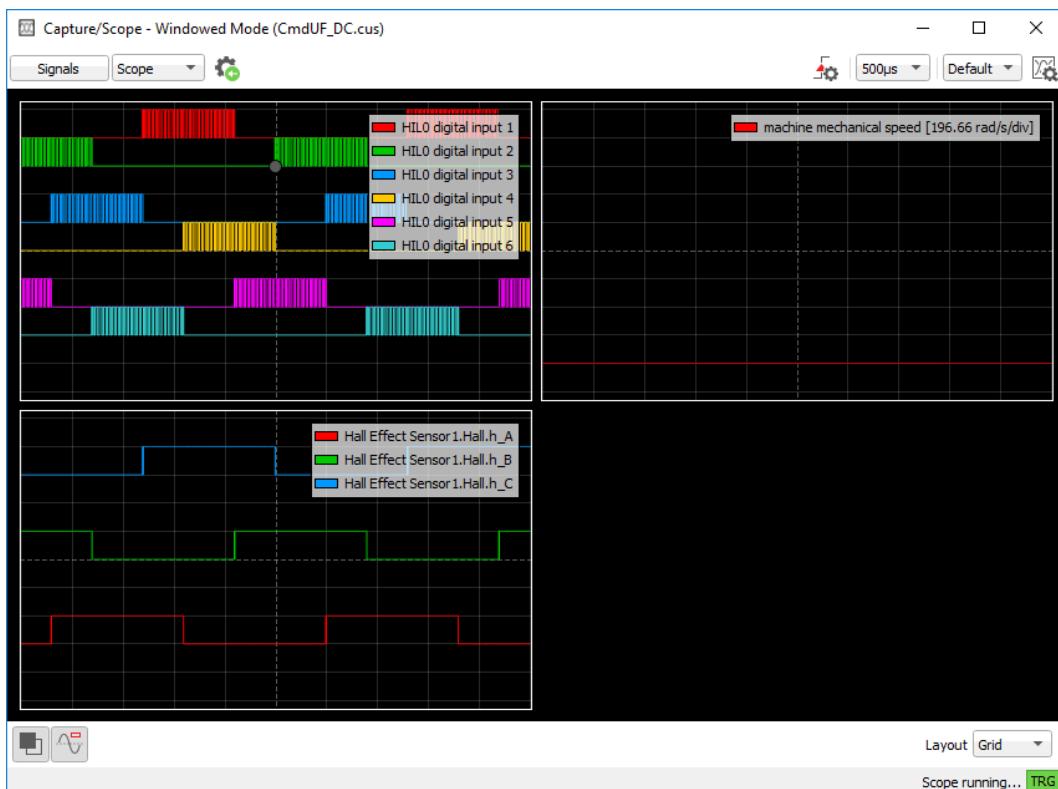


Figure 5.4 Visualisation des PWM entrantes, des encodeurs et de la vitesse du moteur sur le simulateur HIL

5.3.1 Réponse indicelle

Une réponse indicelle (saut de consigne) se trouve à la figure 5.5. On y observe l'oscilloscope représentant la vitesse du moteur en fonction du temps. La vitesse se stabilise à la valeur de consigne choisie par l'utilisateur (700 rad/s).

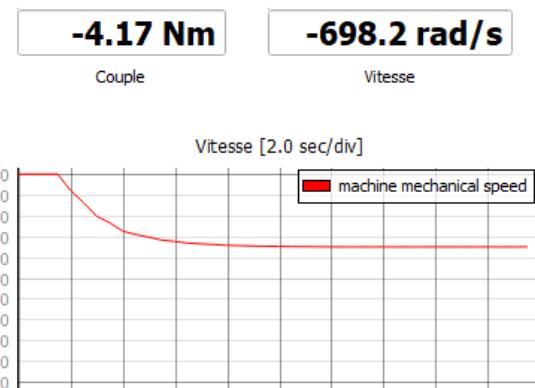


Figure 5.5 Réponse indicielle de la vitesse, saut de consigne à 700 rad/s

5.3.2 Mesure de la PWM générée par la FPGA

La **PWM** en sortie du régulateur a été mesurée à l'aide d'un oscilloscope (voir la trace est à la figure 5.6). La fréquence de la **PWM** est bien de 25KHz (corresponds à la fréquence du quartz divisé par la résolution du modulateur).

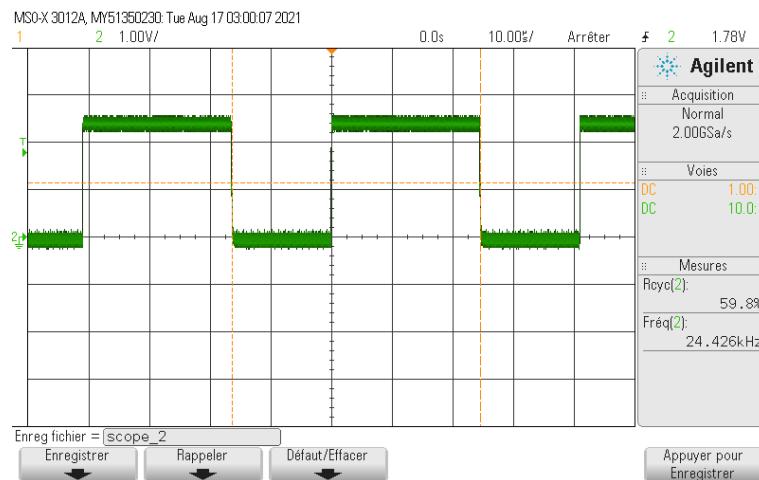


Figure 5.6 PWM générée par le module FPGA

6 | Conclusion

6.1 Synthèse

Durant ce travail de diplôme, tous les objectifs du cahier des charges ont été remplis. Une unité de contrôle **FPGA** a été développée pour l'environnement Poetic. L'unité est composée de deux PCB, le PCB FPGA et Mezzanine. Le PCB Mezzanine a été désigné pour pouvoir se brancher à la fois sur le PCB FPGA ainsi que sur la carte de développement FPGA-Rack¹.

Quelques problèmes hardware ont été constatés sur le PCB FPGA. Deux pins d'une alimentation à découpage ont été inversées ainsi qu'un mauvais pinning pour la flash. Ces deux erreurs ont été corrigées sur la deuxième version du PCB.

Un système de développement **FPGA** a été mis en place, pour effectuer une régulation de vitesse pour des moteurs brushless DC ainsi que des moteurs DC à l'aide d'un régulateur PI.

Un démonstrateur sur simulateur HIL a été implémenté et permet d'effectuer une régulation de vitesse sur un moteur brushless dc. Le démonstrateur totalement est fonctionnel.

Un comparatif, entre l'ancienne unité de contrôle et la nouvelle, est illustré sur le tableau 6.1.

1. Kit de développement FPGA développé à la HES-SO : <http://wiki.hevs.ch/uit/index.php5/Hardware/FPGARack>, sa schématique se trouve à l'annexe D

Chapitre 6. Conclusion

Fonctionnalités	POETIC PROCESSOR	POETIC FPGA	Remarques
Mode de configurations (jumper, dip-switch)	4 Spare & 2 GPIO	4 Spares & 1 Dip Switch 4P	192 configurations supplémentaires possibles
Vitesse d'échantillonnage (par canal, 20 canaux)	<700kSPS	1000kSPS	42% plus rapide
Nombre d'entrées analogiques	20	20	Même nombres d'entrées
Vitesse de reconstruction des D/A	min. 117.6kSPS	min 117.6kSPS	Même vitesse
Nombre de sorties analogiques	4	8	100% de sorties en plus
SPI LVDS	MOSI	MOSI & MISO	Un récepteur LVDS à été rajouté
Ethernet	-	100Mb/s	Ajout d'un PHY Ethernet
USB	USB type A	USB type C	
Sources d'alimentations	Poetic	Poetic & USB	Peut être alimenté par deux sources d'alimentations

Table 6.1 Comparatif entre la carte POETIC PROCESSOR et la nouvelle carte POETIC FPGA

6.2 Améliorations

Hardware

- Pour de futures versions, la partie acquisition de signaux (convertisseurs A/D, ampli-op) devrait être déplacée sur le dos du PCB FPGA. En effet, le câble plat provenant du module de mesure doit enjamber la glissière à carte, ce qui n'est pas idéal, en comparaison au module processeur. Au vu du temps à disposition, la séparation de l'électronique était indispensable. En effet, tout le système a pu être testé avec la carte «FPGA-Rack²» avant que le PCB FPGA n'ait été produit.
- Utiliser un Quartz plus rapide que 100MHz pour booster les performances. Éventuellement changer de footprint si la fréquence voulue doit dépasser les 150MHz.

Architecture software

- Le schéma de commutation pour le moteur **BLDC** est actuellement un schéma trapézoïdal. Ce qui, à basse vitesse, génère des ondulations de couple. Il serait judicieux de réaliser une commutation sinusoïdale ainsi que field-oriented (la commutation sinusoïdale a été implémentée, mais non testée).
- L'implémentation d'un modèle pour les **DAC** d'effectuer des simulations plus «poussées».

2. Kit de développement FPGA développé à la HES-SO : <http://wiki.hevs.ch/uit/index.php5/Hardware/FPGARack>, sa schématique se trouve à l'annexe D

6.3. Difficultés rencontrées

- Actuellement, la lecture de la vitesse du moteur s'effectue avec une tension analogique fournie par le simulateur HIL. Sans simulateur, cette vitesse devrait être estimée à l'aide des capteurs Hall fournis par le moteur.
- Ajouter plus de commandes via l'[UART](#), constante P et I du régulateur, compatibilité avec le programme Poetic...

Simulateur

6.3 Difficultés rencontrées

Dû à la pandémie de la COVID-19, beaucoup de composants n'étaient plus disponibles lors de la commande des composants (stocks raflés) . Par conséquent, des composants similaires ont été soudés à la place des originaux quand cela était possible. Pour d'autre, notamment les connecteurs femelles erni reliant les deux PCB n'étaient tout simplement plus disponible chez les fournisseurs européens. Par chance, deux connecteurs femelles ont pu être livrés avant la fin du travail de Bachelor à une adresse privée en France.

A | Analyse détaillée des convertisseurs A/D

A.1 Types d'architectures des ADCs

A.1.1 Sigma Delta

Le convertisseur analogique digital sigma delta peut atteindre une résolution de quantification très importante (jusqu'à 24 bits) en dépit de la bande passante (inférieure au MHz)[20]. Cette architecture a tendance à rejeter le bruit ce qui le rend très stable (pas besoin de filtre antialiasing en entrée). Ce type de convertisseur a pour la plupart une latence non négligeable, ce qui pour notre application peut poser certains soucis de synchronisation.

A.1.2 SAR

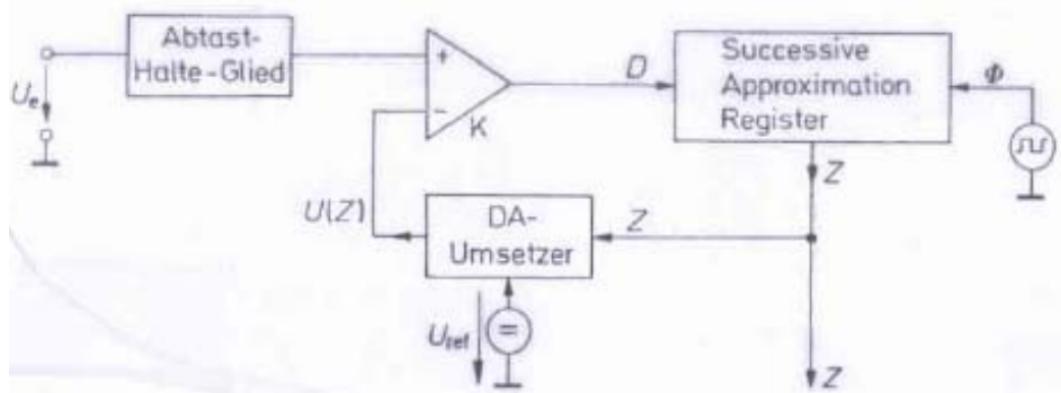


Figure A.1 Convertisseur A/D à approximation successive[21]

Le convertisseur analogique digital par approximation successive utilise la méthode par pesée. Cette technique utilise un comparateur qui compare la valeur mesurée mémorisée avec la tension de sortie du convertisseur D/A. Au début de la conversion, le nombre Z est mis à 0. Ensuite le bit de poids fort (MSB) est mis à 1, et on effectue un test avec le comparateur. Si la tension d'entrée U_e est supérieure à la tension de sortie $U(Z)$, on laisse le bit à 1. Si ce n'est pas le cas, on le met à 0. Cette procédure est ensuite répétée pour tous les autres bits. Les bits sont ensuite stockés dans un registre à approximation successive[21].

Annexe A. Analyse détaillée des convertisseurs A/D

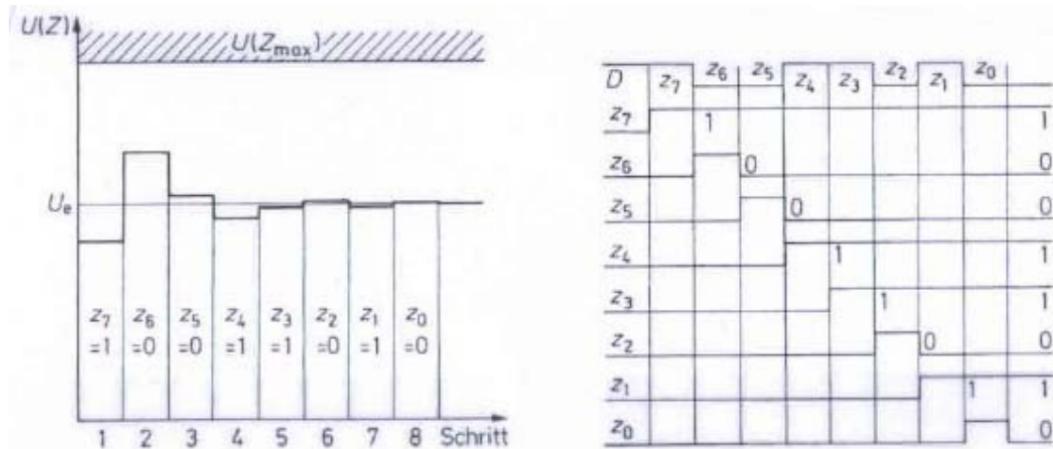


Figure A.2 Évolution de $U(Z)$ / Z avec la méthode par pesée[21]

Ce type de convertisseurs offrent le « meilleur des deux mondes ». À la fois un ratio vitesse d'échantillonnage / quantification élevé, une grande facilité de mise en œuvre, une très bonne précision ainsi que 0 cycle de latence.

A.1.3 Pipeline

Le convertisseur A/D pipeline est un mix entre la méthode parallèle et la méthode par pesée. Sur la FIGURE, une structure pipeline 10 bits est réalisée à l'aide de deux convertisseurs 5 bits. Dans un premier temps, le premier ADC effectue une conversion grossière du signal. Cette première conversion correspond aux bits de poids fort et est ensuite retransformée en signal analogique à l'aide d'un D/A. Une soustraction est effectuée entre le signal d'entrée U_e et le signal de sortie du D/A, le tout est multiplié par 2^{N-1} . Le deuxième convertisseur A/D 5 bits converti ce signal, et sa sortie correspond aux bits de poids faible[21]. Ce type de convertisseur peut travailler à de grandes vitesses, mais sa résolution est limitée. De plus, ce type de convertisseur dispose d'un délai de propagation des données non négligeables.

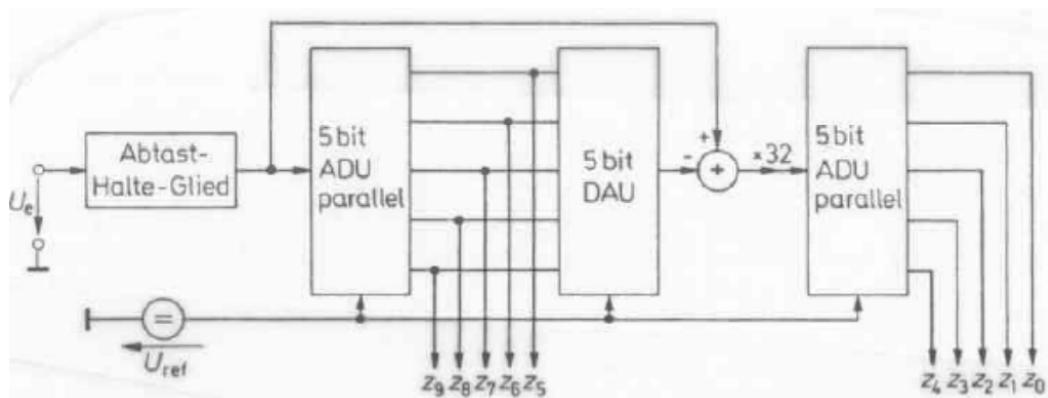


Figure A.3 Architecture pipeline[21]

A.2 Latence

L'algorithme de moyennage et de sur-échantillonnage du Sigma Delta provoque un retard sur le données de sorties. La latence du convertisseur à approximation successive est nulle. Le retard des convertisseurs pipeline crée une latence non nulle. Pour l'application de régulation de moteur, il est important de ne pas négliger cette latence.

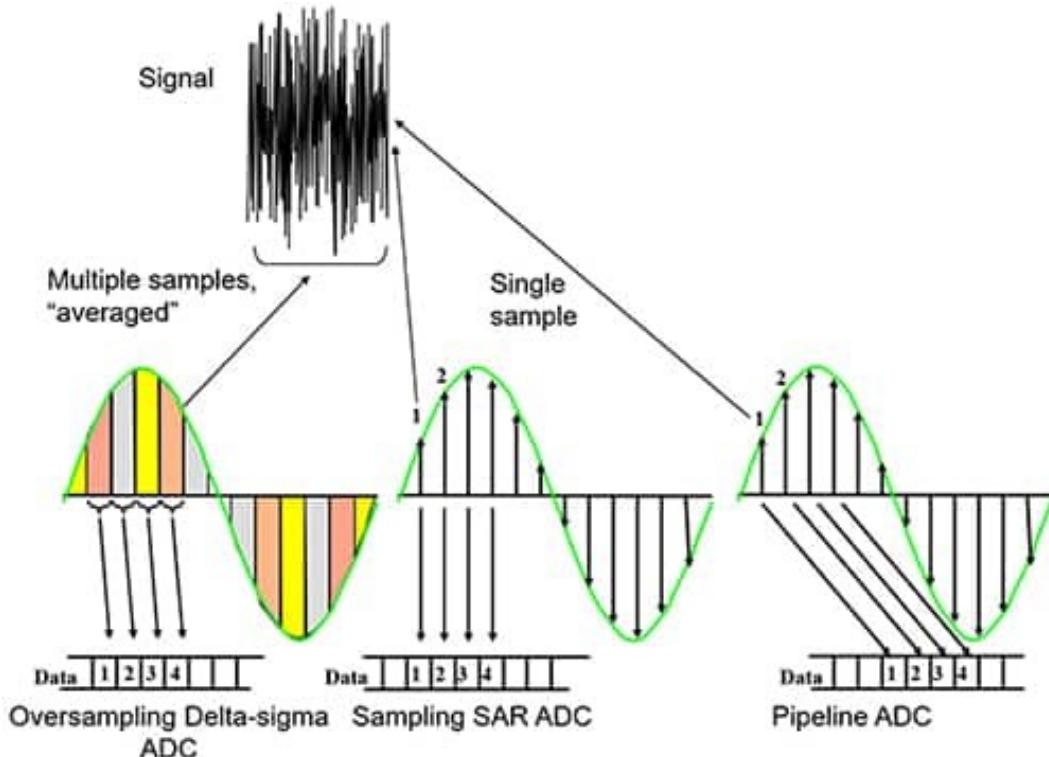


Figure A.4 Exemple de latence type pour le Sigma Delta, le SAR et le Pipeline [22]

A.3 Types d'interfaces

Les principales interfaces pour lire les données fournies par les convertisseurs analogiques digitaux sont :

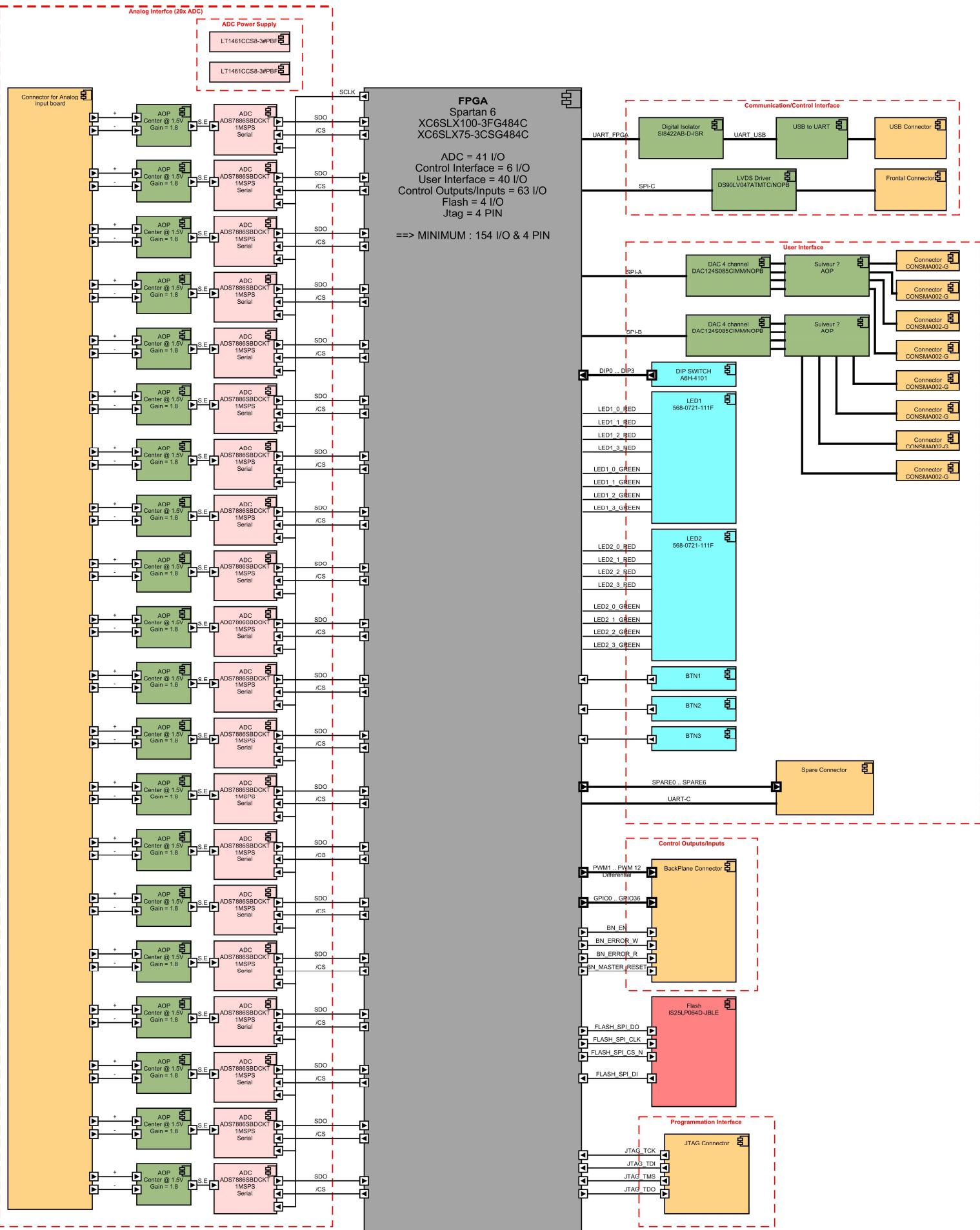
- I2C
 - Faible vitesse
 - 1 signal de données et 1 signal de clock
- SPI
 - Vitesse élevée
 - 1 clock, 1 chip select, 1 signal de données entrantes et 1 signal de données sortantes
- Série
 - Vitesse élevée
 - TX Single-Ended

Annexe A. Analyse détaillée des convertisseurs A/D

- LVDS Série
 - Vitesse très élevée (1Gb/s)
 - TX Différentielle

Chaque interface à ses avantages ainsi que ses inconvénients. Pour cette électronique, l'interface parallèle a été évitée au vu du nombre de connexions que cette interface engendre pour 20 ADC. Il existe un moyen de multiplexer le bus parallèle à l'aide de tri-state. Mais avec des vitesses élevées et à cause du retard engendré par les tri-state, il en est presque impossible. Une interface série à donc été privilégiée.

B | Schéma bloc complet



C | Fichiers de commandes

Personne concernée:

Jean Nanchen

Mandat no.:

TD Nanchen Jean

Chef de projet / professeur:

Salle:

321

Délai désiré

Mouser

Quantity	Reference	Designation
1	217-C6SLX150-3FG484C	XC6SLX150-3FG484C
1	710-150060VS75000	LED Verte FPGA DONE 2V 20mA
2	653-B3U-1000P-B	Push buttons Reset
1	710-831022731	Quartz 100MHz FPGA
1	998-KSZ8041TL	Ethernet PHY
2	78-1N4148WS-HG3-08	1N4141 SOD323-2
1	710-831019170	Quartz 25MHz ETH
1	673-J0011D01BNL	Ethernet RJ45 Connector without EI
1	727-S25FL064LABMA000	64Mb flash
3	506-FSMRA4JH04	Right angle button FSMRA4JH04
1	653-A6H-4101	Dip Switch
1	848-RP402N501F-TR-FE	DC/DC 5V 800mA Boost
1	963-NRS5020T2R2NMGJ	Bobine 2.2uH 5020
1	584-ADP2119ACPZ1.2R7	DC/DC 1.2V Buck
1	710-74437334015	Bobine 1.5uH 5020
1	947-SC189ZSKTRT	DC/DC 3.3V 1.5A
1	710-74437334022	Bobine 2.2uH 5020 3.5A
2	305-234199	MEZZANINE 13.65mm FEMALE
10	81-GRM21BZ70J226ME4L	Capa X7R 22uF 6.3V
2	645-568-0721-111F	4 LEDs Bipolar Red-Green Stacked
1	617-09-03-196-6921	Connecteur DIN 41612 Backplane
2	604-AA1608SURSK	LED RED 1.95V
1	649-98464-G6110ULF	JTAG Right angle Connector
1	523-12402054M611A	USB C Right angle up
1	895-FT232RQ	USB Interface to UART FT232RL
10	81-GRM31CR60J107ME9K	Capa X5R 100uF 3216
15	80-C0805X475K9R	Capa X7R 4.7uF 6.3V 0805
30	80-C0402C474K9RACTU	Capa X7R 470nF 0402
2	754-RG1608P-6491-BT5	6.49kOhm 1608
5	754-RG1608P-49R9-BT5	49.9 Ohm 1608

Eurocircuit

Quantity	Reference	Designation
3		PCB

Personne concernée:

Jean Nanchen

Mandat no.:

TD Nanchen Jean

Salle:

321

Chef de projet / professeur:

Délai désiré

Mouser

Quantity	Reference	Designation
20	595-ADS7886SBDVBR	AD 1MSPS Serial 1CH 12bits
1	584-LT1461CCS8-3#PBF	3V Reference 100mA
2	926-DAC124S085CIMMNO	DAC 4CH 12bits SPI
2	200-TSS11004LDRA	Connecteur Cable Plat Signaux
1	584-AD8646ARZ	AOP 2 CH Rail-to-Rail AD8646ARZ
7	584-AD8648ARZ-R	AOP 4 CH Rail-to-Rail AD8648ARZ
8	712-CONSMA002-G	SMA Right Angle
1	200-TST10501GD	SPIC LVDS Connector 2.54mm
1	200-STMM10902GDSM	Connecteur SPARE
3	595-REF3130AIDBZR	Référence tension 3V
2	305-244856	Mezzanine MALE 3.65mm
1	926-DS90LV018ATMNOPB	Receiver LVDS 1 CH
1	926-DS90LV047ATMTCNO	Interface LVDS DS90LV047ATMTC

Eurocircuit

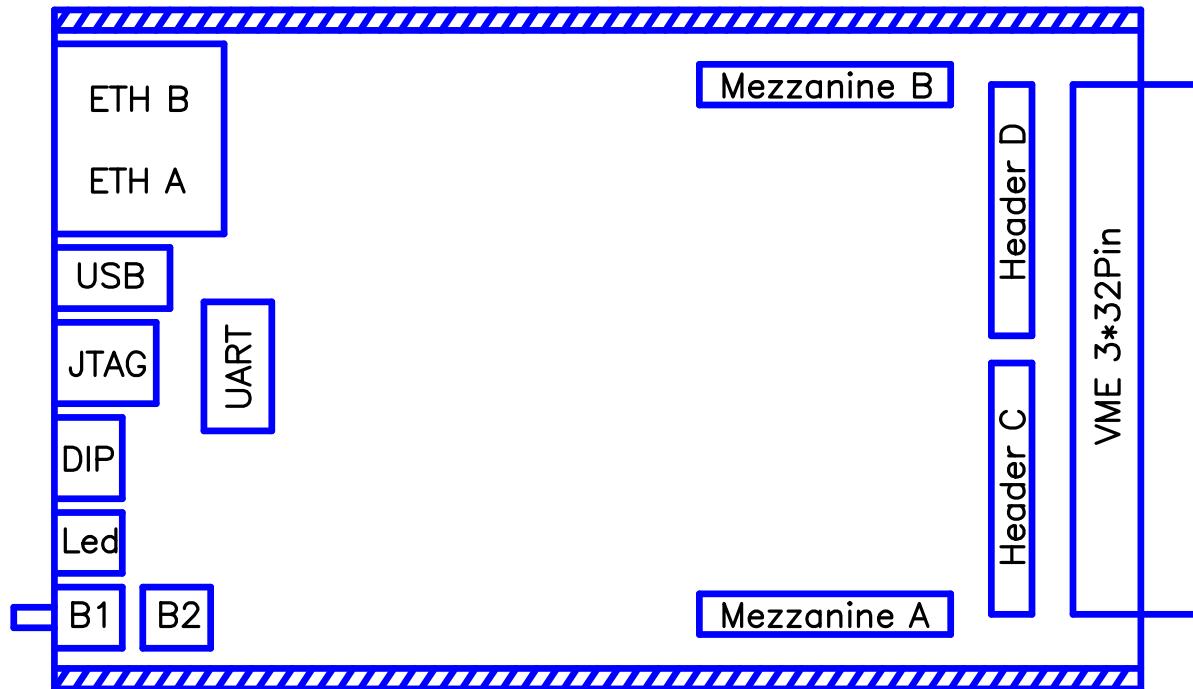
Quantity	Reference	Designation
3		PCB

D | Schématique PCB FPGA Rack V1.0

	1	2	3	4	5	6	7	8	9	10	
A	HEVs Route du Rawyl 47 1950 Sion 2 www.hevs.ch										A
B	Designer: Zahno Silvan silvan.zahno@hevs.ch or zahno.silvan@gmail.com										B
	FPGA_Rack Board : Rack-mounted Development FPGA platform										
c	Page	Title	Description								c
	1	Cover	This first page								
	2	Board	Shows Board connectivity								
	3	Power	3.3V 2.5V 1.8V 1.2V Regulation								
D	4	FPGA Power	Xilinx Spartan 6 Power and Decoupling								D
	5	FPGA 1	Xilinx Spartan 6 Bank 0 and 2 !!!!! Pay Attention on differences between LX45 and LX100								
	6	FPGA 2	Xilinx Spartan 6 Bank 1 and 3								
	7	FPGA Config	JTAG Connector / Reset Circuit / Done / Button / Leds								
E	8	Memory	FLASH / SDRAM / Decoupling capacitors								E
	9	Mezzanine	Mezza A and Mezza B Connectors (mezzanine support)								
	10	VME	VME Compatible Connector 96Pin								
	11	Ethernet	2 Ethernet Ports								
	12	UART	FTDI USB UART / UART								
F											F
	FPGA Rack HES-SO // Valais Wallis Cover HAUTE ECOLE VALAISANNE										
	DES {Date} zas REV V1.0 1/25 {Path} FPGA_Rack_v1_0.sch										
	1	2	3	4	5	6	7	8	9	10	

A FPGA Rack Connectivity

 = min. 3mm with no components for Rack insertion



FPGA Rack HES-SO // Valais Wallis HAUTE ECOLE VALAISANNE	Board	DES	{Date}	zas
		REV	V1.0	
		2/25	{Path}	FPGA_Rack_v1_0.sch

1 2 3 4 5 6 7 8 9 10

Power supply

A
 5V -> From VME or USB
 3.3V -> From LTM4615 (1)
 1.2V -> From LTM4615 (2)
 1.8V -> From LTM4615 LDO
 2.5V -> From LDO

Layout Hints
 - THERMAL_SW1 and THERMAL_SW2
 Floating on separate copper planes
 Decoupling capacitors close to pins

LDO VOUT
 $V_{LDO_OUT} = 0.4V * (4.99k + RFB_LDO) / RFB_LDO$
 -> $RFB_LDO = 1.996k / (V_{LDO_OUT} - 0.4V)$
 $RFB_LDO (Vout = 1.8V) = 1.996k / (1.8 - 0.4) = 1.42571k \rightarrow 1.43k \text{ 1\%} \rightarrow 1.80563V$

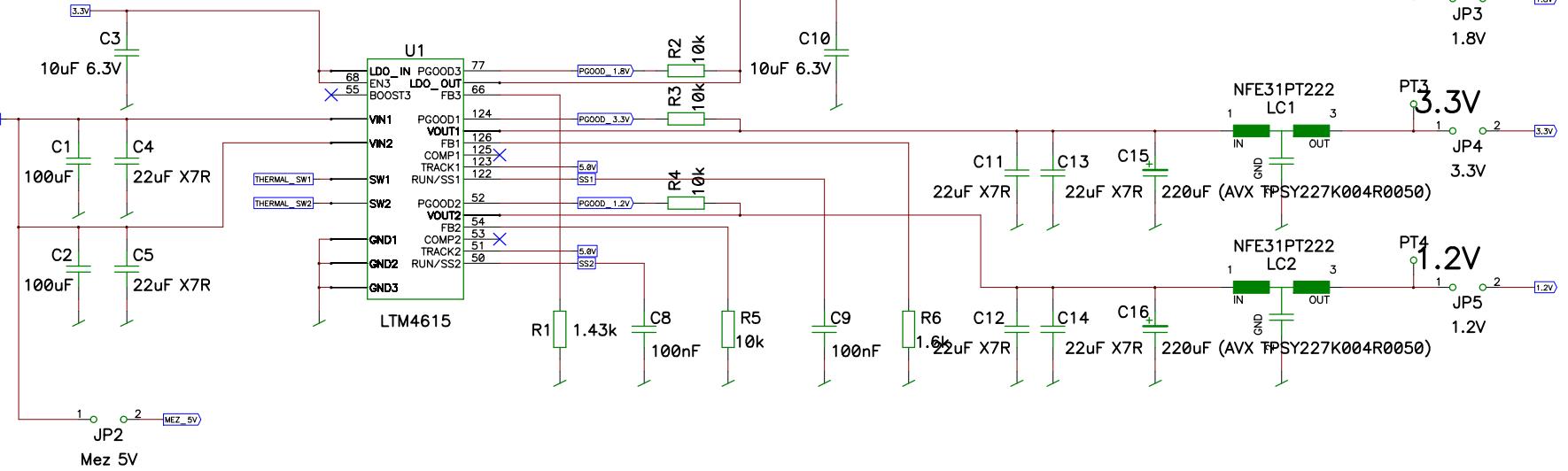
VOUT = $0.8V * (4.99k + RFB) / RFB$
 -> $RFB = 3.992k / (Vout - 0.8)$

RFB ($Vout = 3.3V$) = $3.992k / (3.3 - 0.8) = 1.5968k \rightarrow 1.6k \text{ 1\%} \rightarrow 3.295 V$

RFB ($Vout = 1.2V$) = $3.992k / (1.2 - 0.8) = 9.98k \rightarrow 10k \text{ 1\%} \rightarrow 1.1992 V$

3.3V @ 4A / 1.2V @ 4A / 1.8V @ 1.5A

B



D

Mez 5V

2.5V @950mA

E



F

FPGA Rack	DES	{Date}	zas
HES-SO // Valais Wallis	Power	V1.0	
HAUTE ECOLE VALAISANNE	REV	{Path}	3/25 {Path}
			FPGA_Rack_v1_0.sch

1 2 3 4 5 6 7 8 9 10

1

2

3

4

5

6

7

8

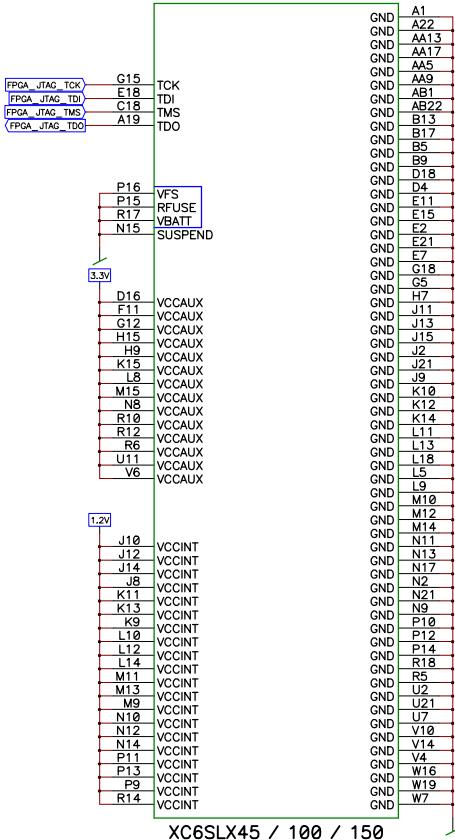
9

10

FPGA Power

PINS NOT AVAILABLE ON LX45
ONLY AVAILABLE ON LX100 / 150
 - VFS
 - RFUSE
 - VBATT

U3:E



A

B

C

D

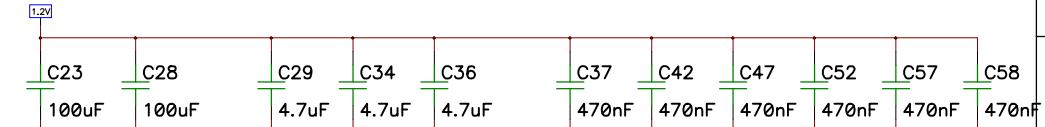
E

F

FPGA Decoupling

VCCINT Decoupling

LX45 → 1x100uF / 1x4.7uF / 2x0.47uF
 LX100 → 1x100uF / 2x4.7uF / 4x0.47uF
 LX150 → 2x100uF / 3x4.7uF / 6x0.47uF



VCCAUX Decoupling

LX45, LX100, LX150 → 1x100uF / 2x4.7uF / 4x0.47uF



VCC0 Bank 0 Decoupling

LX45, LX100, LX150 → 1x100uF / 1x4.7uF / 2x0.47uF



VCC0 Bank 1 Decoupling

LX45, LX100, LX150 → 1x100uF / 1x4.7uF / 3x0.47uF



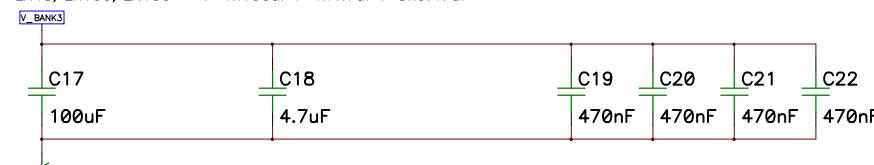
VCC0 Bank 2 Decoupling

LX45 → 1x100uF / 1x4.7uF / 4x0.47uF
 LX100, LX150 → 1x100uF / 1x4.7uF / 3x0.47uF



VCC0 Bank 3 Decoupling

LX45, LX100, LX150 → 1x100uF / 1x4.7uF / 3x0.47uF



FPGA Rack

HES-SO // Valais Wallis

FPGA Power

HAUTE ECOLE VALAISANNE

DES

{Date}

zas

REV

V1.0

4/25

{Path}
 FPGA_Rack_v1_0.sch

1

2

3

4

5

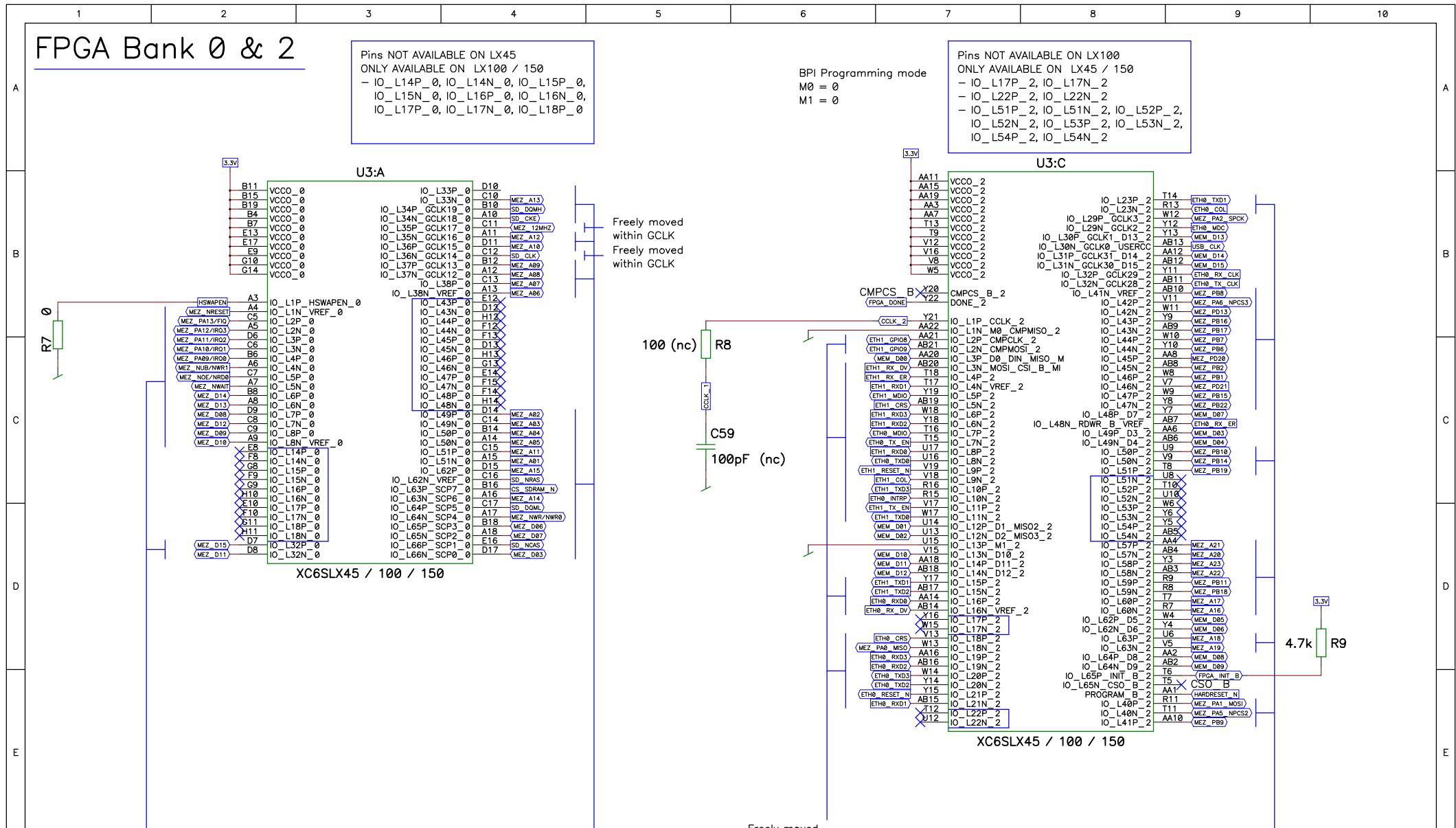
6

7

8

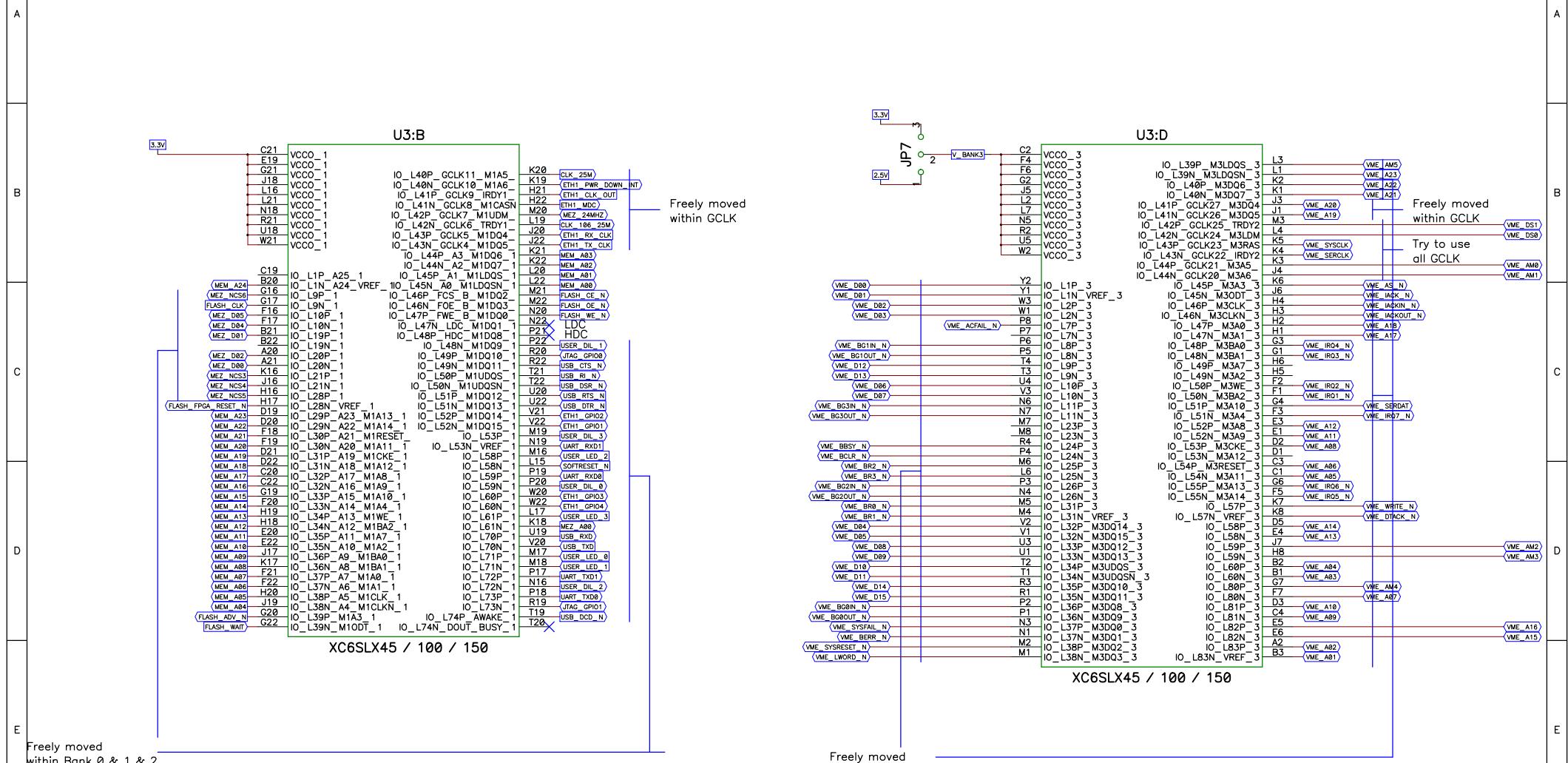
9

10

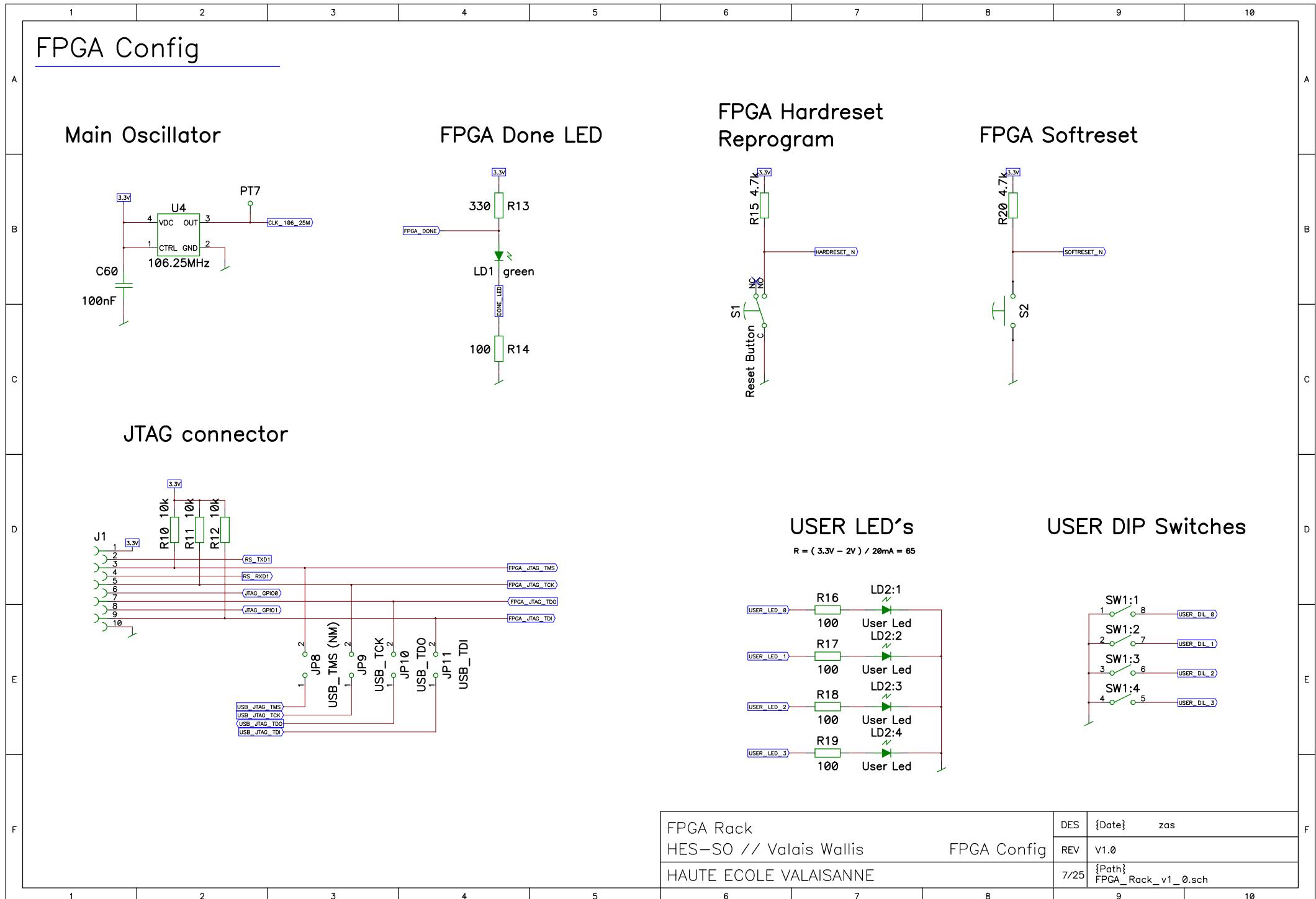


FPGA Rack	DES	{Date}	zos
HES-SO // Valais Wallis	FPGA 1	REV	V1.0
HAUTE ECOLE VALAISANNE	5/25	{Path}	FPGA_Rack_v1_0.sch

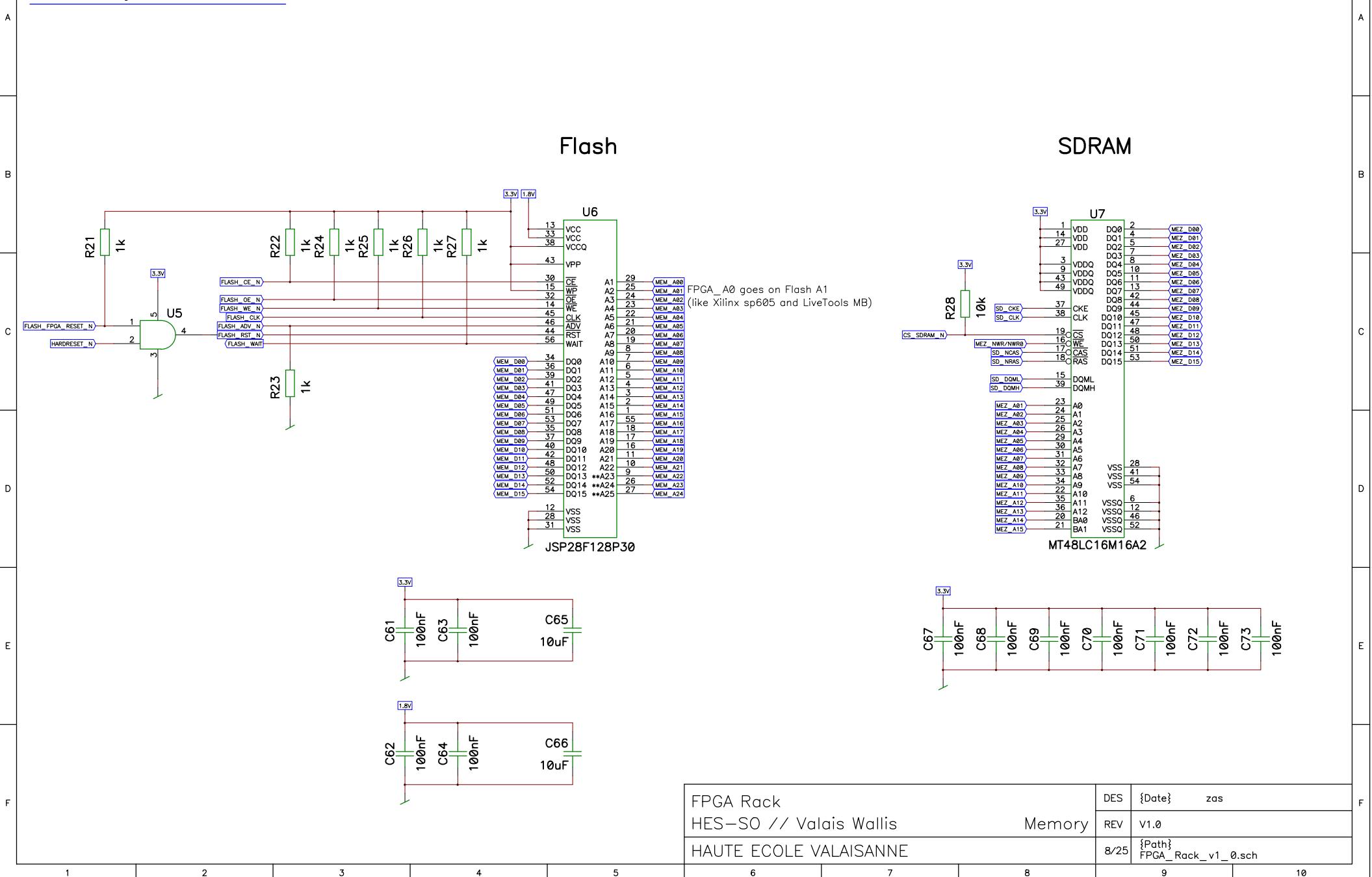
FPGA Bank 1 & 3



FPGA Rack HES-SO // Valais Wallis	FPGA 2	DES	{Date}	zas
		REV	V1.0	
HAUTE ECOLE VALAISANNE		6/25	{Path} FPGA_Rack_v1_0.sch	



Memory Flash RAM



Mezzanine

A

A

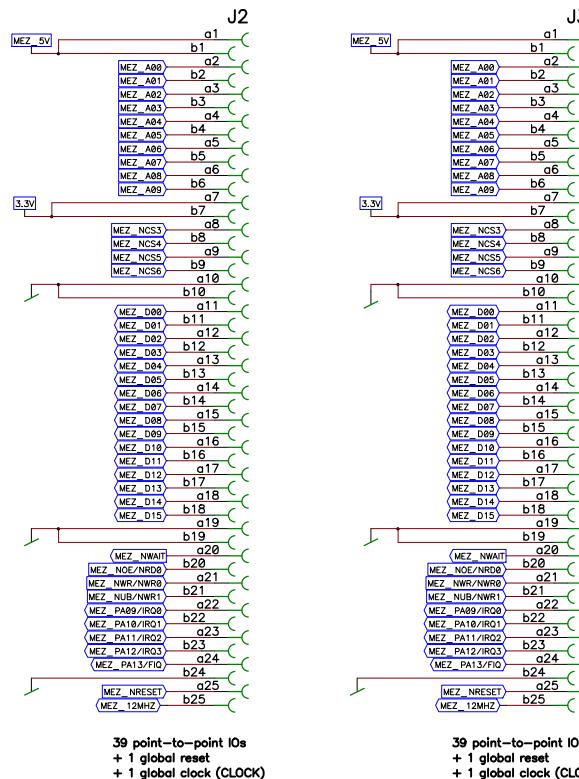
Mezza A

B

B

TOP SIDE

BOTTOM SIDE



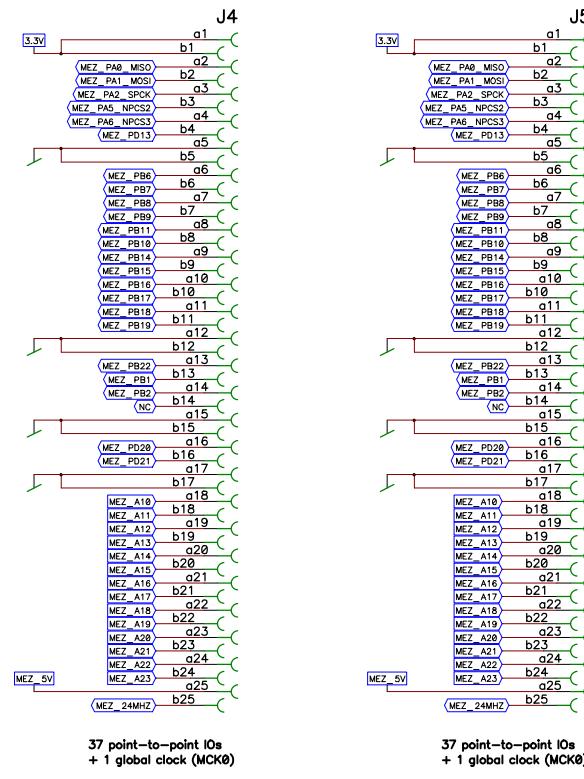
Mezza B

B

B

TOP SIDE

BOTTOM SIDE



FPGA Rack
HES-SO // Valais Wallis
HAUTE ECOLE VALAISANNE

Mezzanine
REV: V1.0
9/25 {Path}
FPGA_Rack_v1_0.sch

DES {Date} zas
REV V1.0
9/25 {Path}
FPGA_Rack_v1_0.sch

VME 96Pin

A

A

B

B

C

C

D

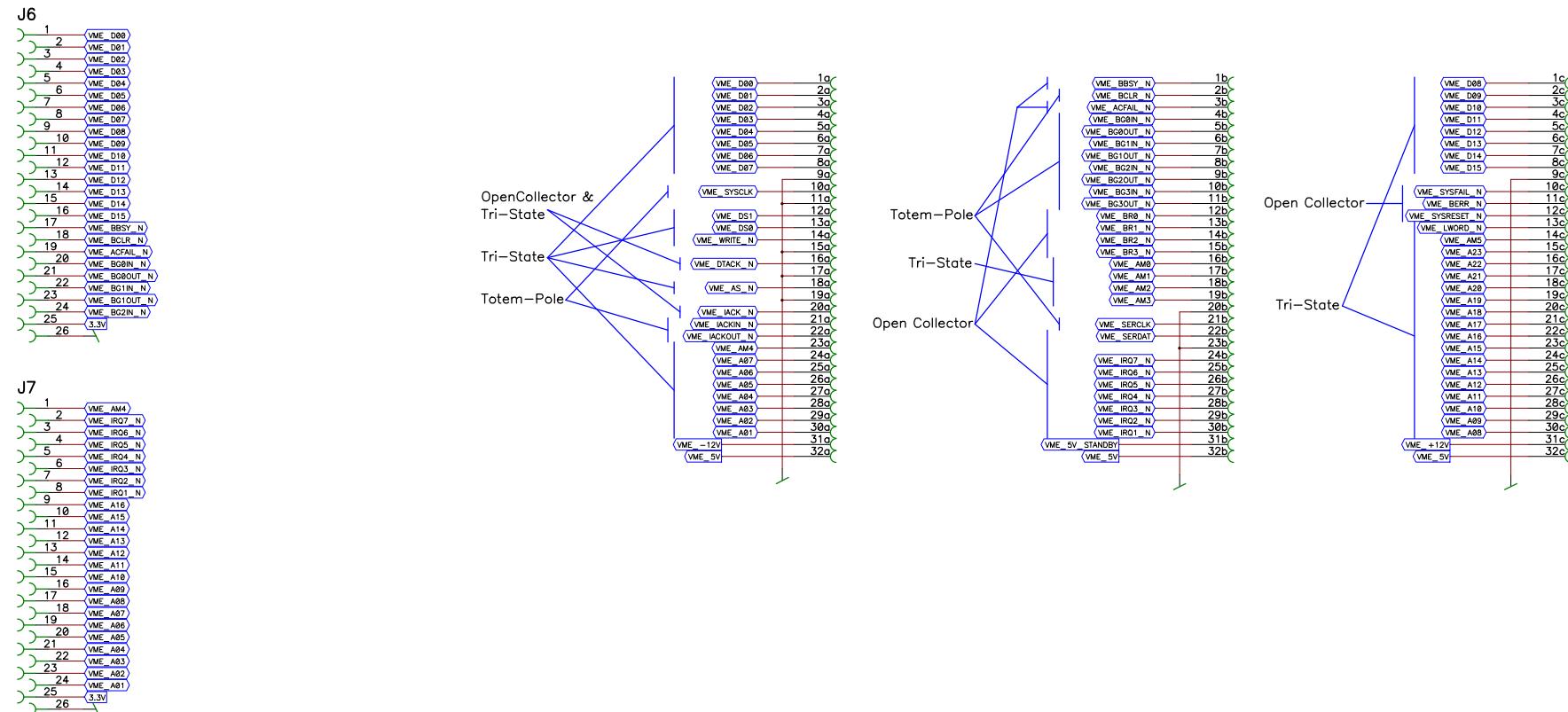
D

E

E

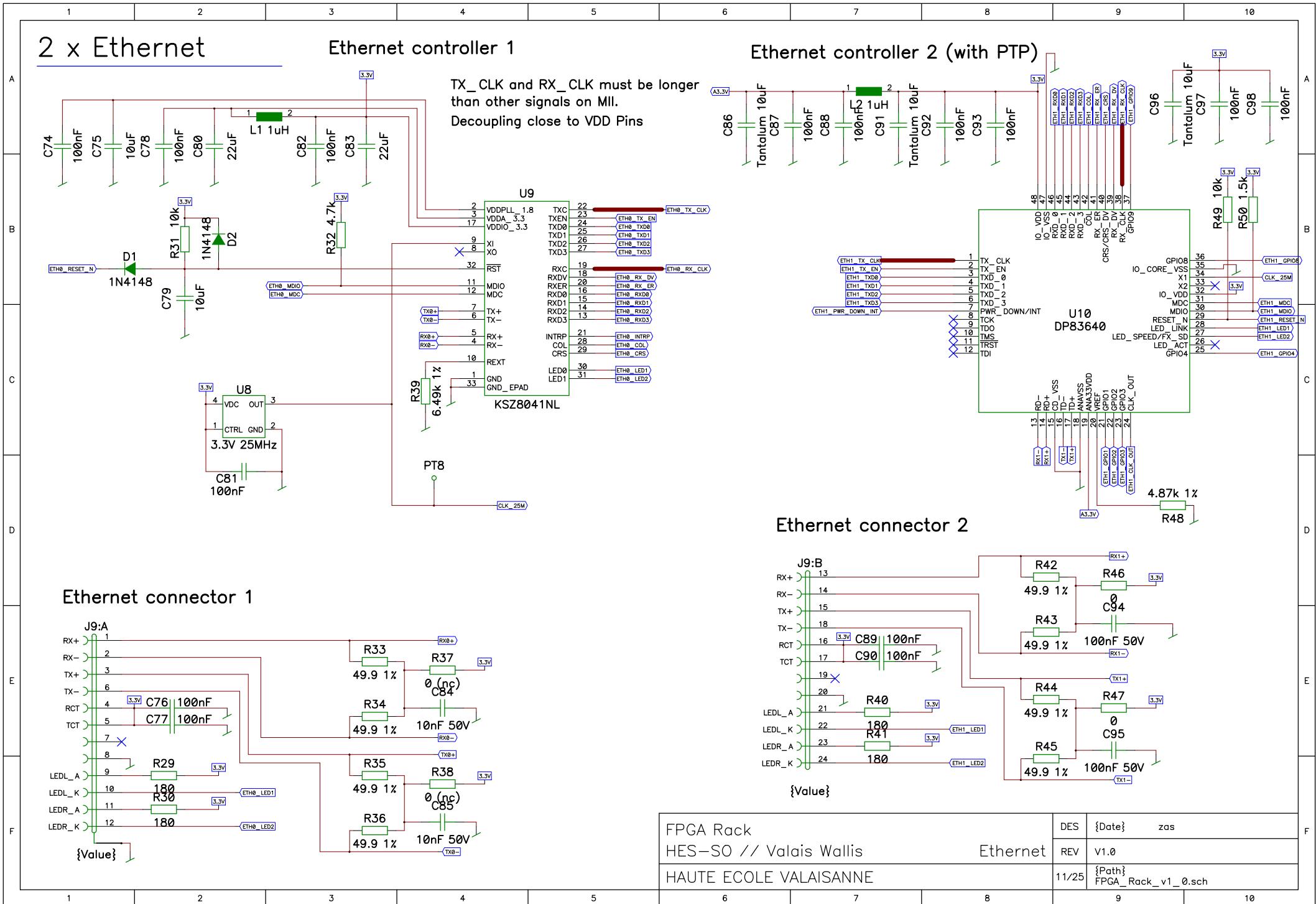
F

F

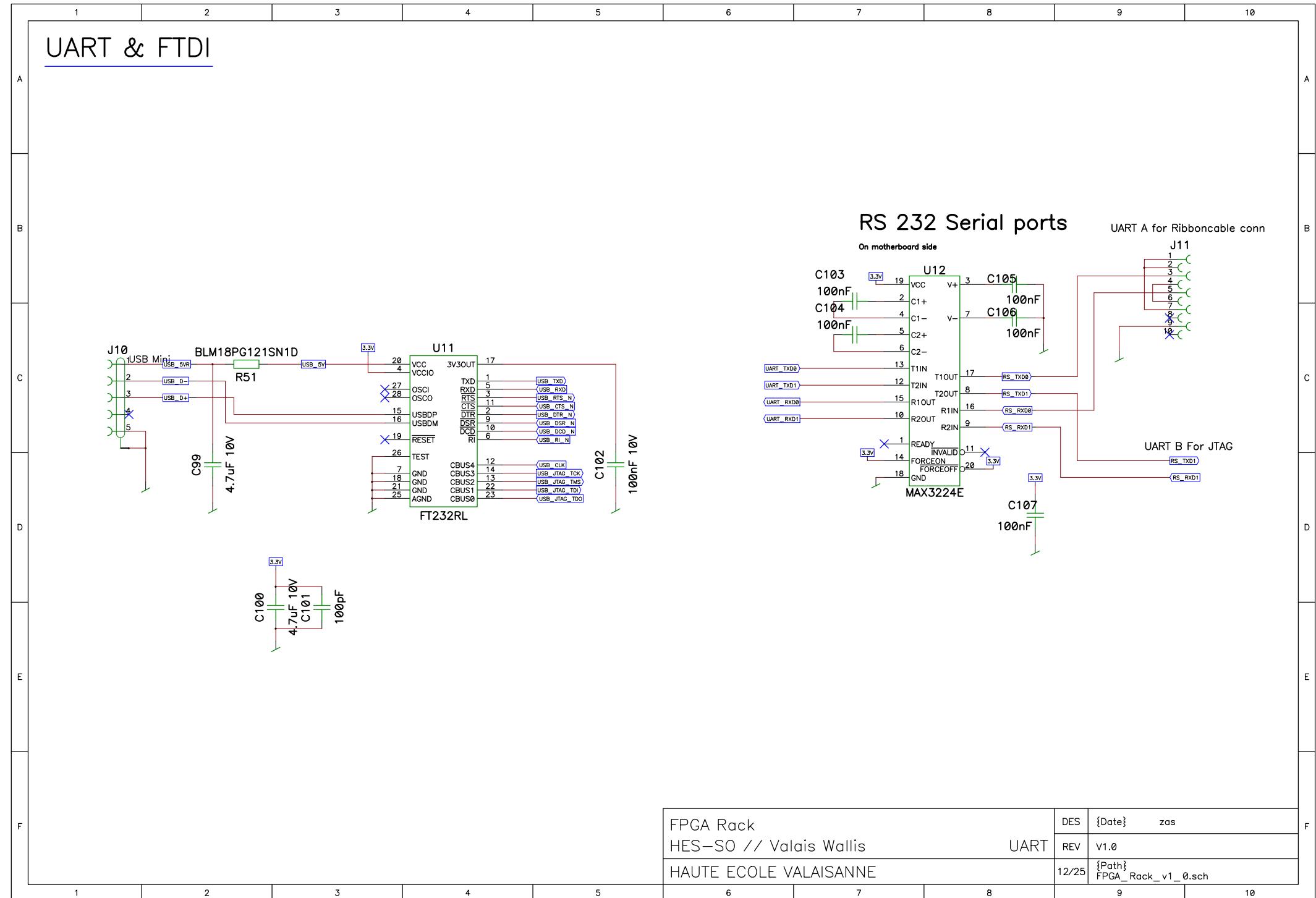


Pins on J10 and J11
can be freely moved except power

FPGA Rack HES-SO // Valais Wallis HAUTE ECOLE VALAISANNE	DES	{Date}	zas
	REV	V1.0	
	10/25	{Path}	FPGA_Rack_v1_0.sch

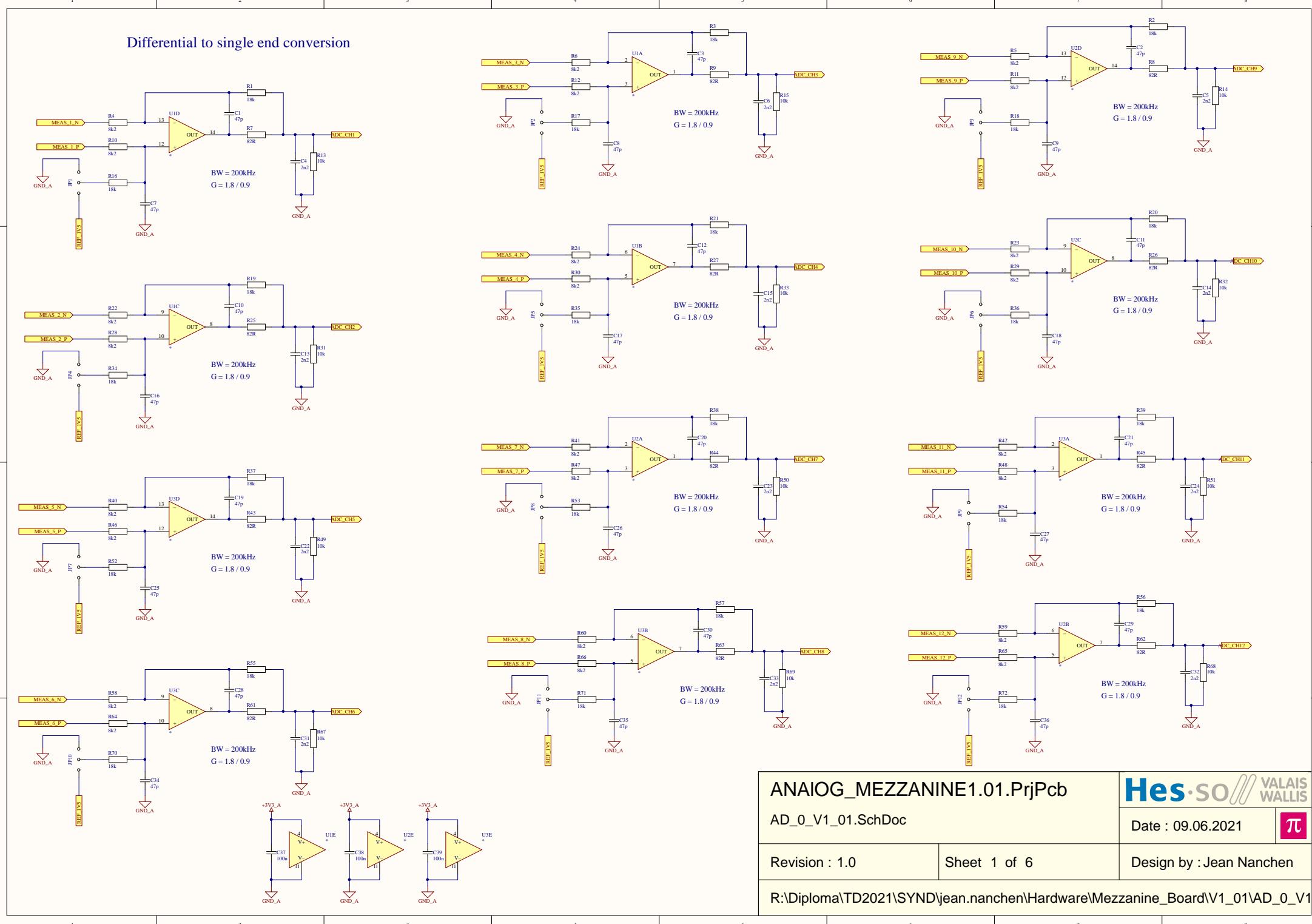


UART & FTDI



E | Schématique PCB Mezzanine

Differential to single end conversion



ANAI0G_MEZZANINE1.01.PrjPcb

AD_0_V1_01.SchDoc

Hes-SO VALAIS WALLIS

Date : 09.06.2021



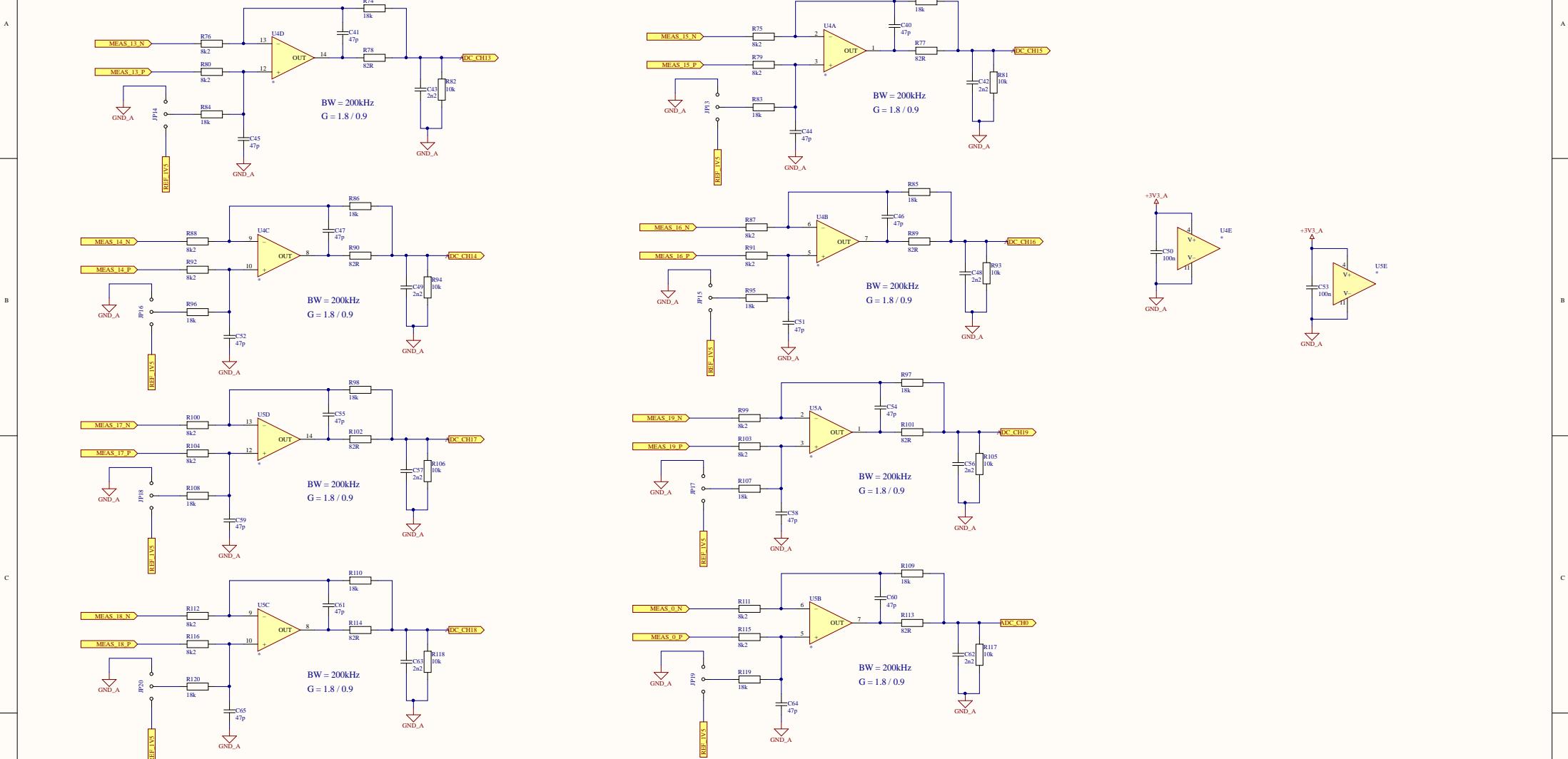
Revision : 1.0

Sheet 1 of 6

Design by : Jean Nanchen

R:\Diploma\TD2021\SYND\jean.nanchen\Hardware\Mezzanine_Board\V1_01\AD_0_V1_01.

Differential to single end conversion



ANALOG MEZZANINE1.01.PriPcb

AD_1_V1_01.SchDoc

Hes•so // VALAIS WALLIS

Date : 09.06.2021

1

Revision : 1.0

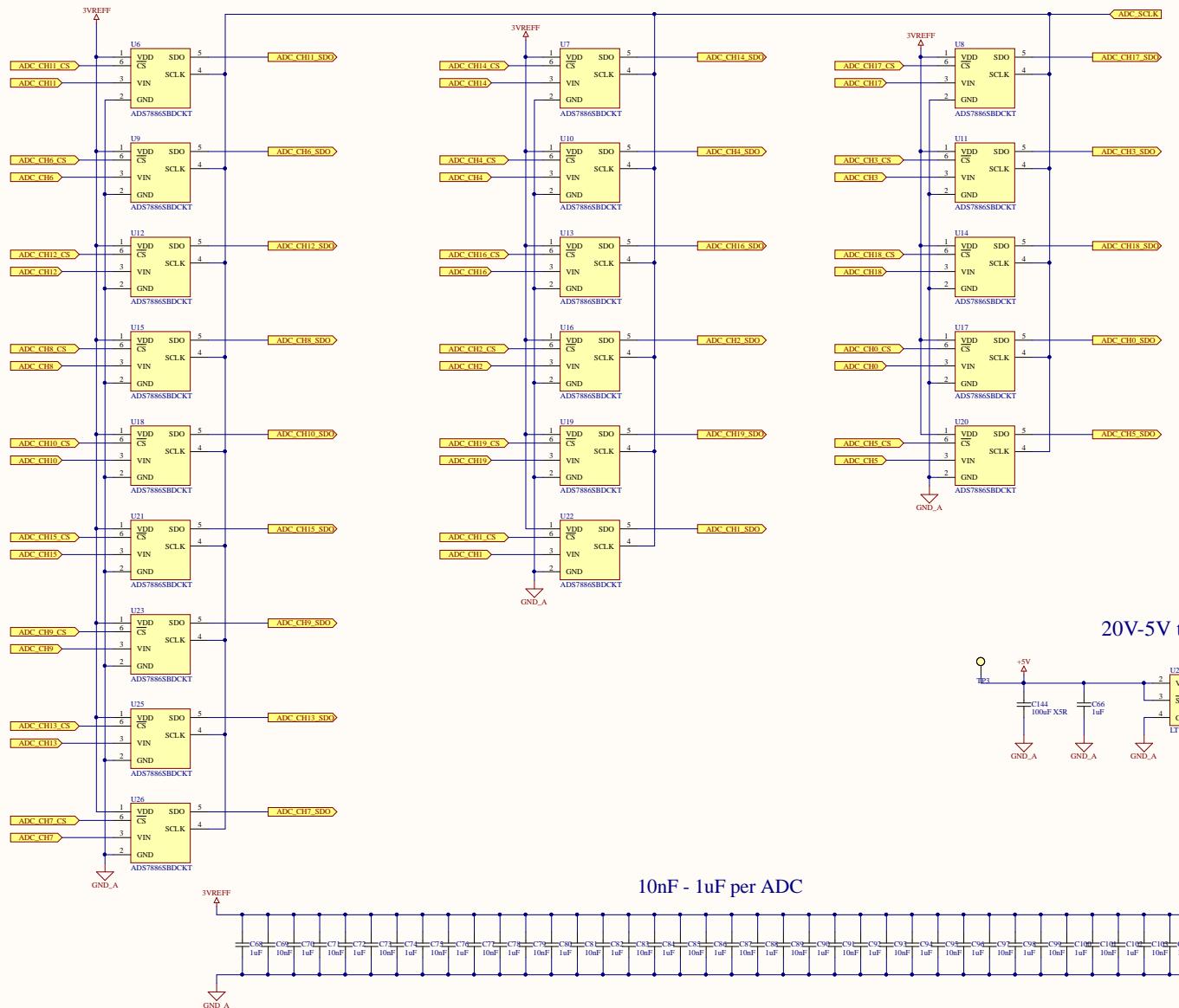
Sheet 2 of 6

Design by : Jean Nanchen

R:\Diploma\TD2021\SYND\jean.nanchen\Hardware\Mezzanine_Board\V1_01\AD_1_V1_01.

20 CHANNELS ADC @ 1MSPS

ITOT = 30mA



ANAI0G_MEZZANINE1.01.PrjPcb

AD_3_V1_01.SchDoc

Hes-SO VALAIS WALLIS

Date : 09.06.2021



Revision : 1.0

Sheet 3 of 6

Design by : Jean Nanchen

R:\Diploma\TD2021\SYND\jean.nanchen\Hardware\Mezzanine_Board\V1_01\AD_3_V1_01.

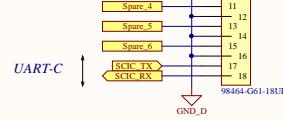
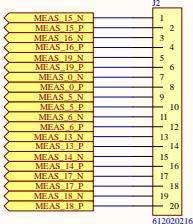
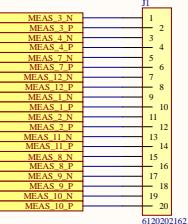
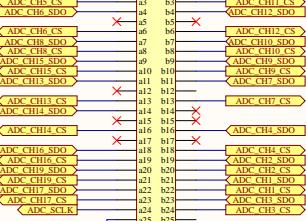
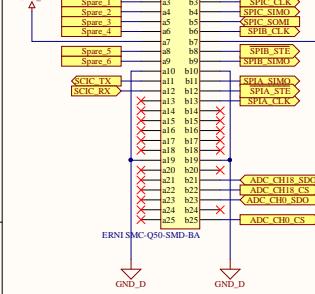


MEZZA B

314 IS NC !!!!!!!!

Connector for Analog Input Board

Spare connector



1

SPI_C_SCK_P
SPI_C_SCK_N
SPI_C_MOSI_P
SPI_C_MOSI_N
SPI_C_MISO_P
SPI_C_MISO_N

C

SPIC_SOMI_N	3
SPIC_SOMI_P	4
SPIC_CLK_N	5
SPIC_CLK_P	6

necto_0_V1_01.SchDoc

Date : 09.06.2021

ANALOG MEZZANINE1.01.PriPcb

Connector 0 V1 01.SchDoc

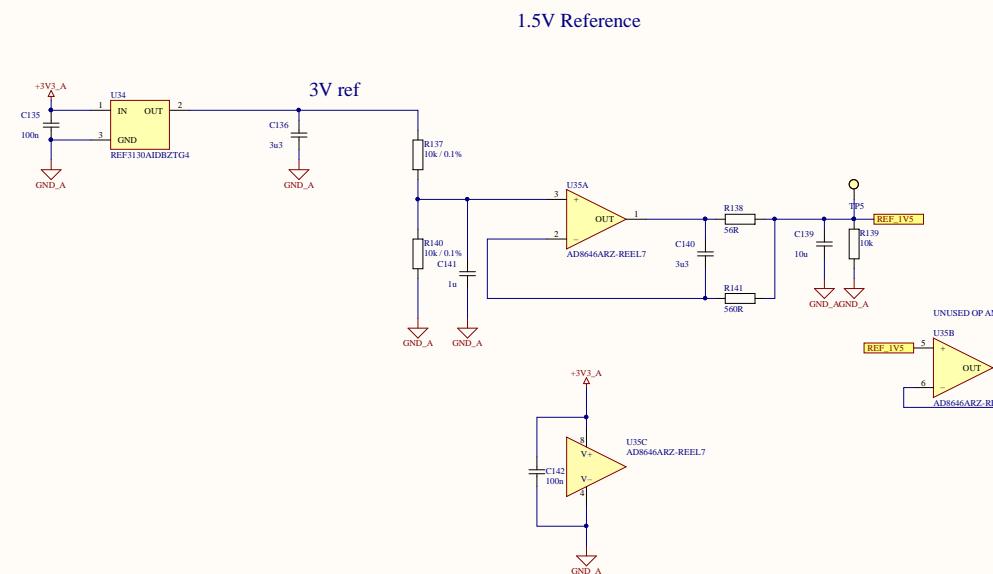
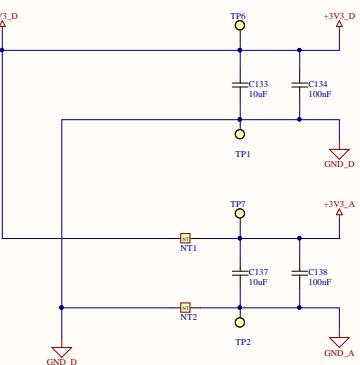
Hes-SO // VALAIS WALLIS

Date : 09-06-2021

Sheet 4 of 6

B:\Diploma\TR2021\SYND\icon-patch\Hardware\Mezzanine_Board\V1_01\Connect

3V3 BackPlane ----->3V3_D, 3V3_A



ANAIOG_MEZZANINE1.01.PrjPcb

Supply_0_V1_01.SchDoc

Hes-SO VALAIS WALLIS

Date : 09.06.2021



Revision : 1.0

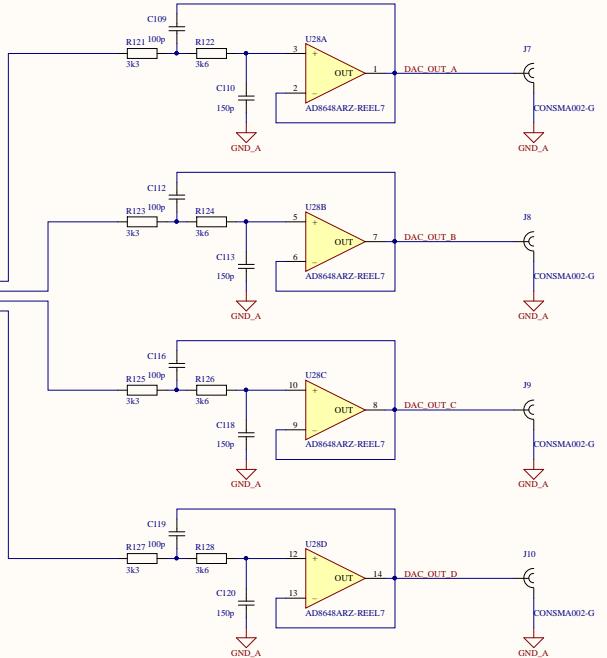
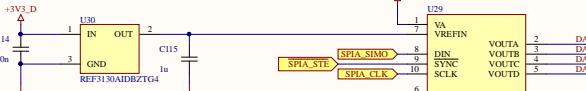
Sheet 5 of 6

Design by : Jean Nanchen

R:\Diploma\TD2021\SYND\jean.nanchen\Hardware\Mezzanine_Board\V1_01\Supply_0_V1

SPI-A (DAC)

Filter 2nd Order, Critical Attenuation
Fc: 200kHz
Attenuation @ 1MHz: -17dB
Phase Shift @ 10kHz: -3.5°

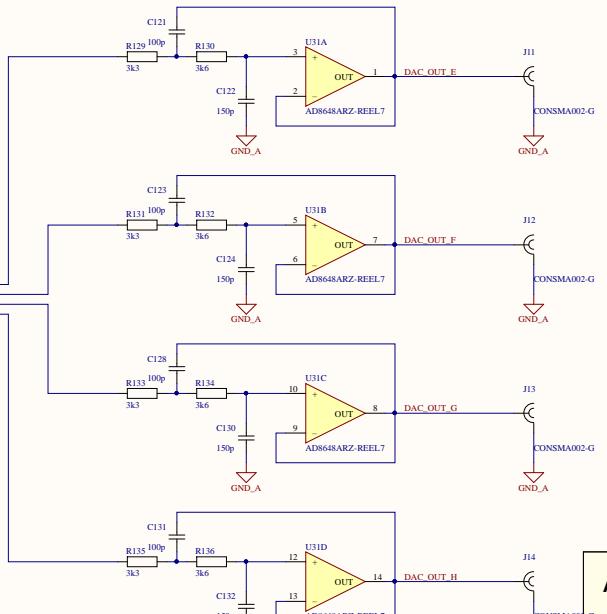


CONSM002-G

GND_A

SPI-B (DAC)

Filter 2nd Order, Critical Attenuation
Fc: 200kHz
Attenuation @ 1MHz: -17dB
Phase Shift @ 10kHz: -3.5°



CONSM002-G

GND_A

CONSM002-G

GND_A

CONSM002-G

GND_A

CONSM002-G

GND_A

ANAI0G_MEZZANINE1.01.PrjPcb**DAC_0_V1_01.SchDoc****Hes-SO VALAIS WALLIS**

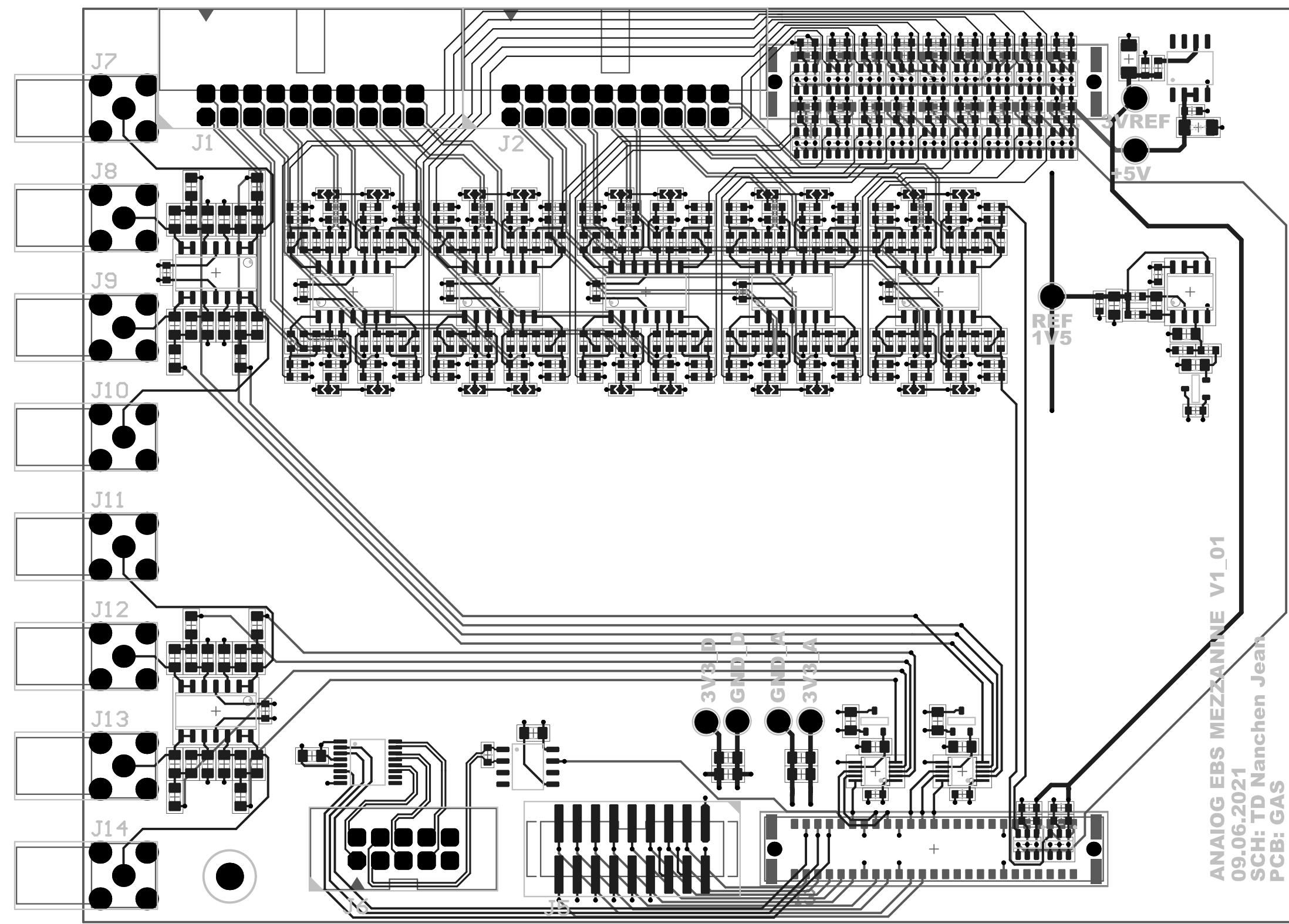
Date : 09.06.2021



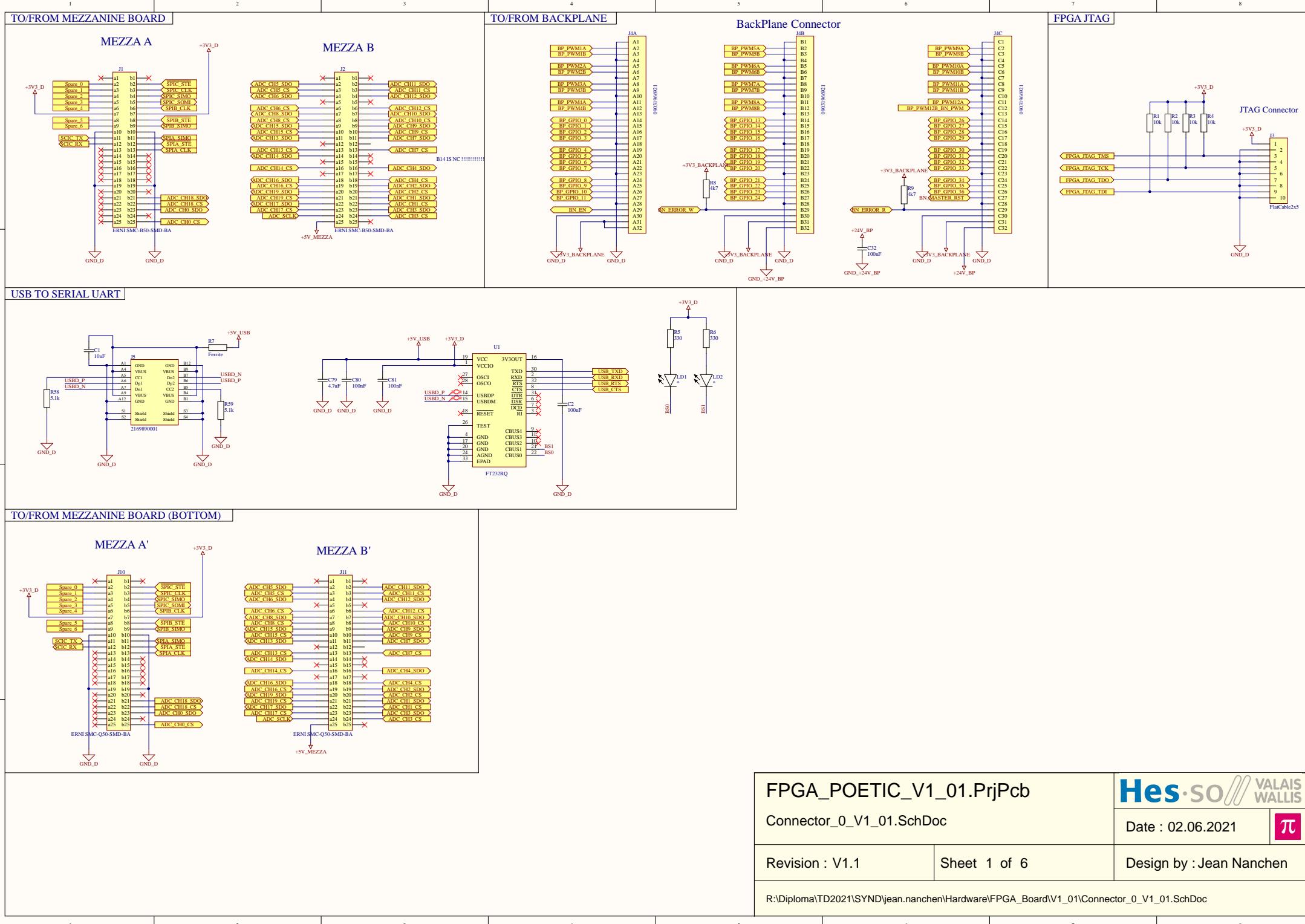
Revision : 1.0 | Sheet 6 of 6

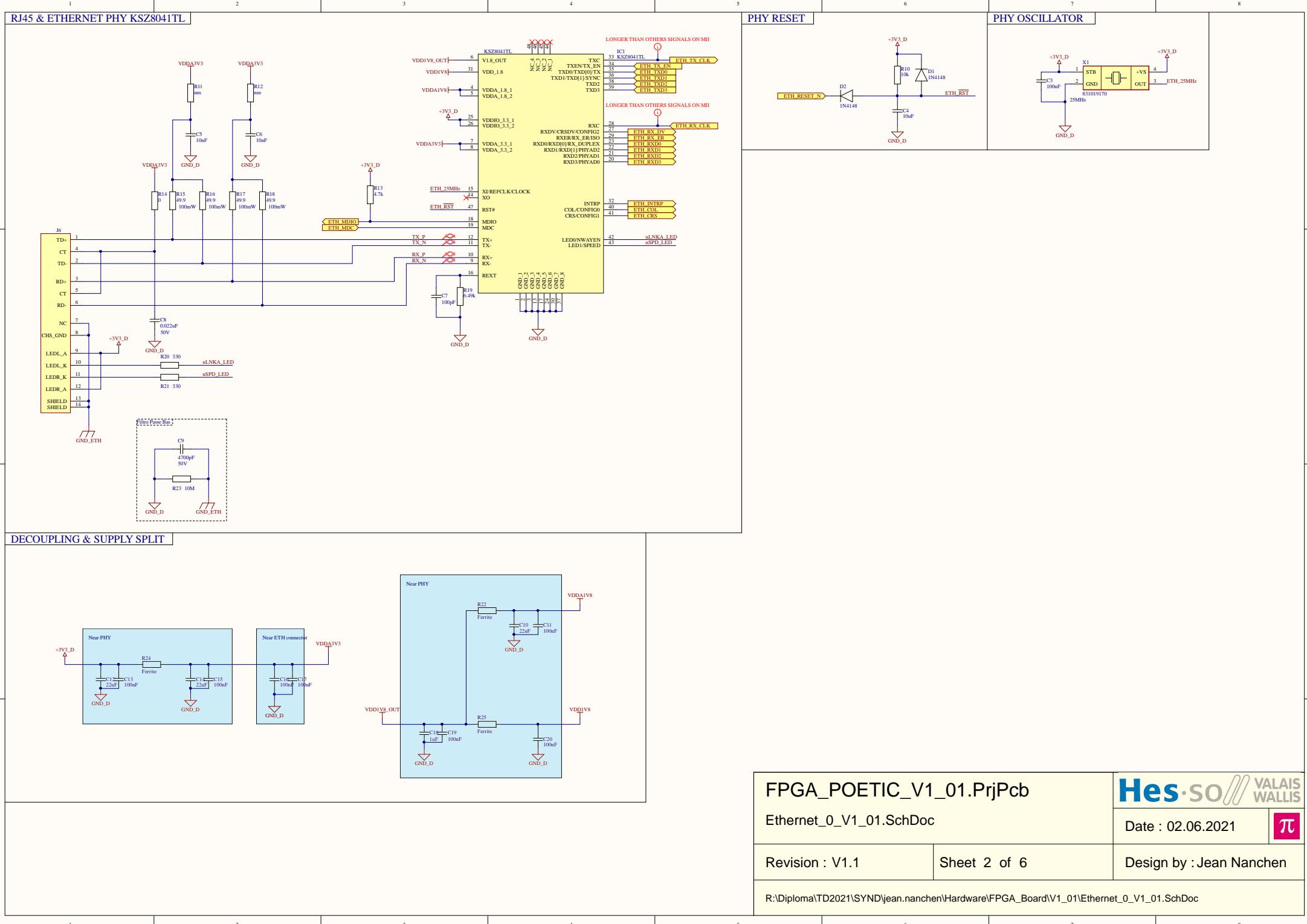
Design by : Jean Nanchen

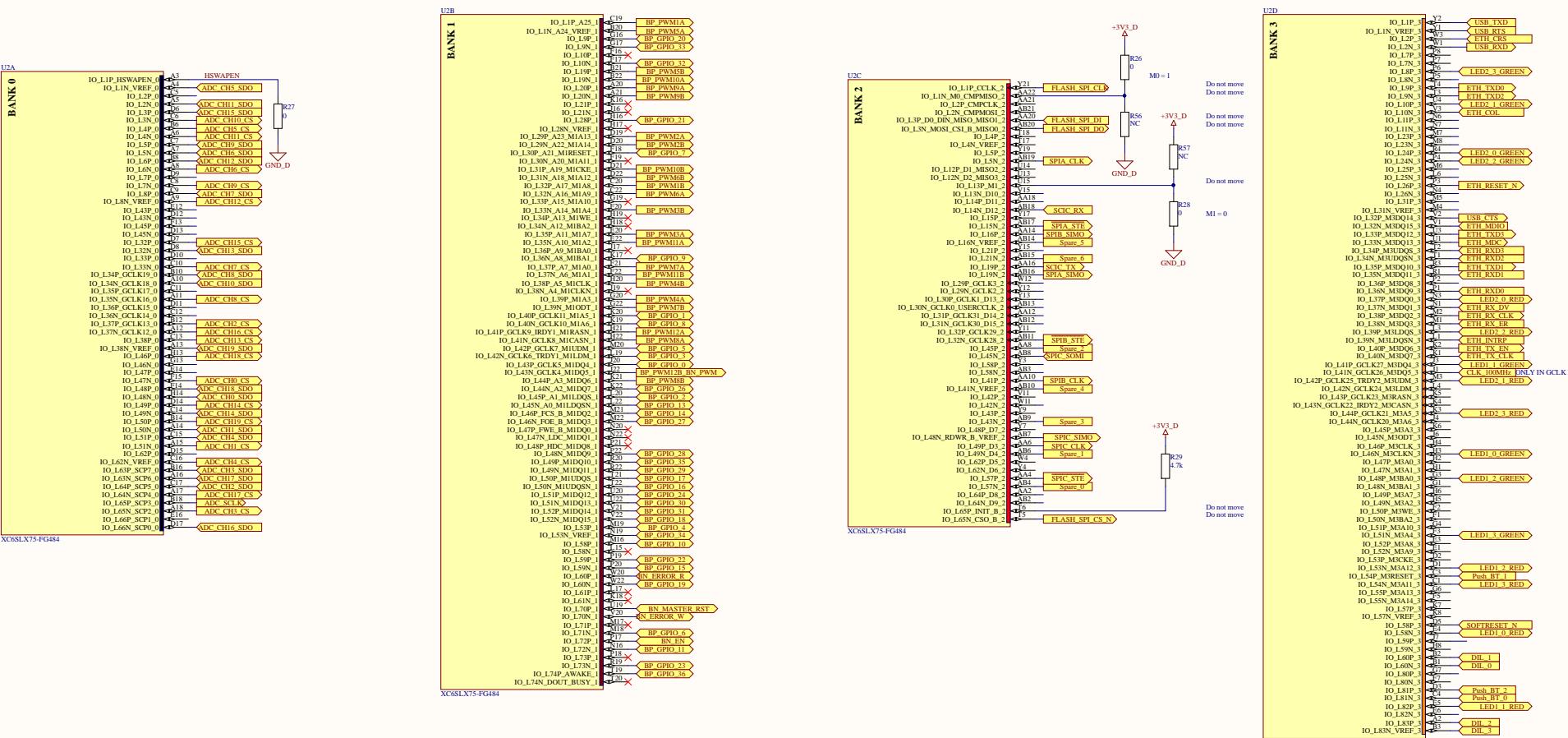
R:\Diploma\TD2021\SYND\jean.nanchen\Hardware\Mezzanine_Board\V1_01\DAC_0_V1_01



F | Schématique PCB FPGA







FPGA_POETIC_V1_01.PrjPcb

FPGA_0_V1_01.SchDoc

Hes-SO // VALAIS WALLIS

Date : 02.06.2021

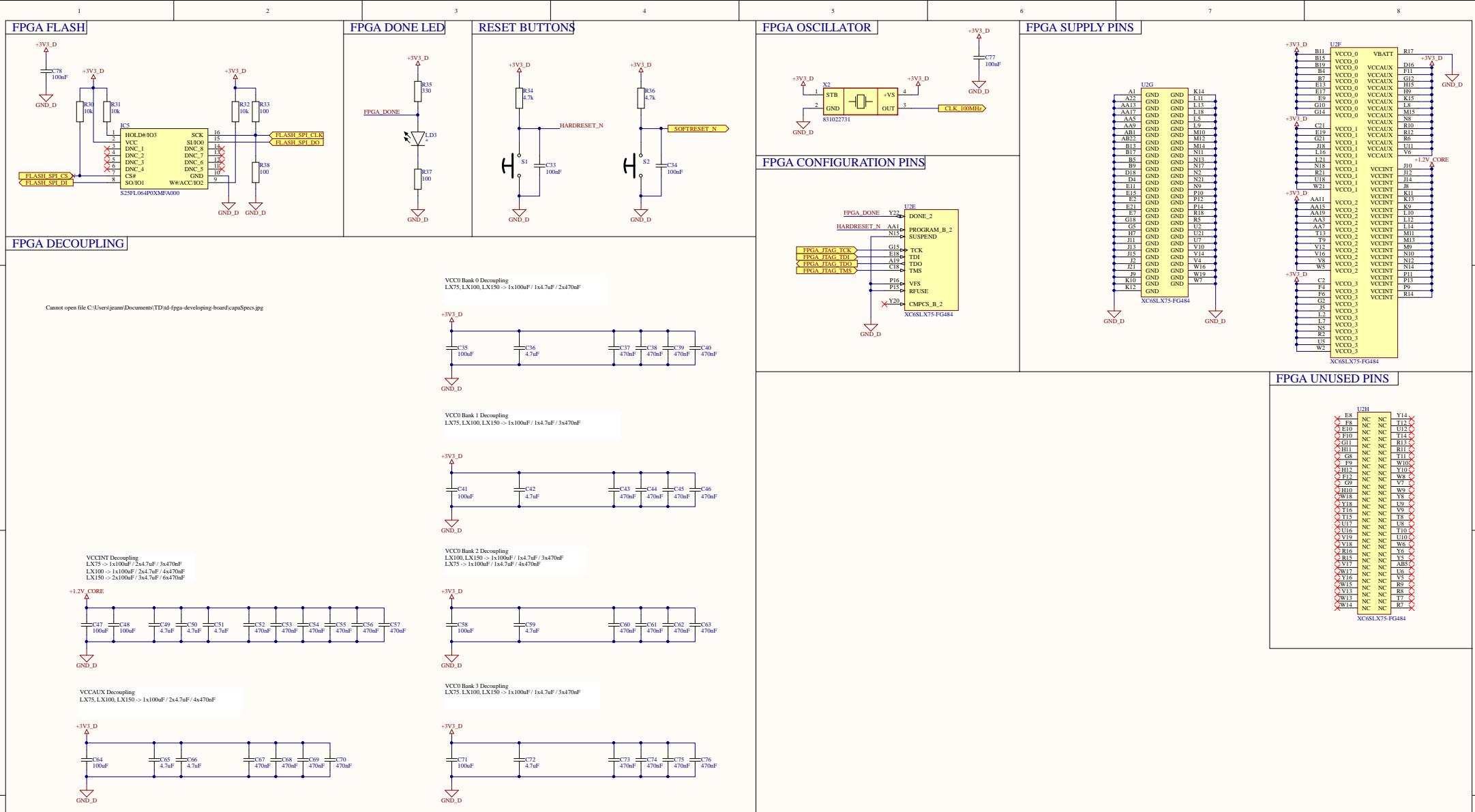
1

Revision : V1.1

Sheet 3 of 6

Design by : Jean Nancher

R:\Diploma\TD2021\SYND\jean.nanchen\Hardware\FPGA_Board\V1_01\FPGA_0_V1_01.SchDoc



FPGA_POETIC_V1_01.PrjPcb

FPGA_1_V1_01.SchDoc

Hes-SO // VALAIS WALLIS

Date : 02.06.2021

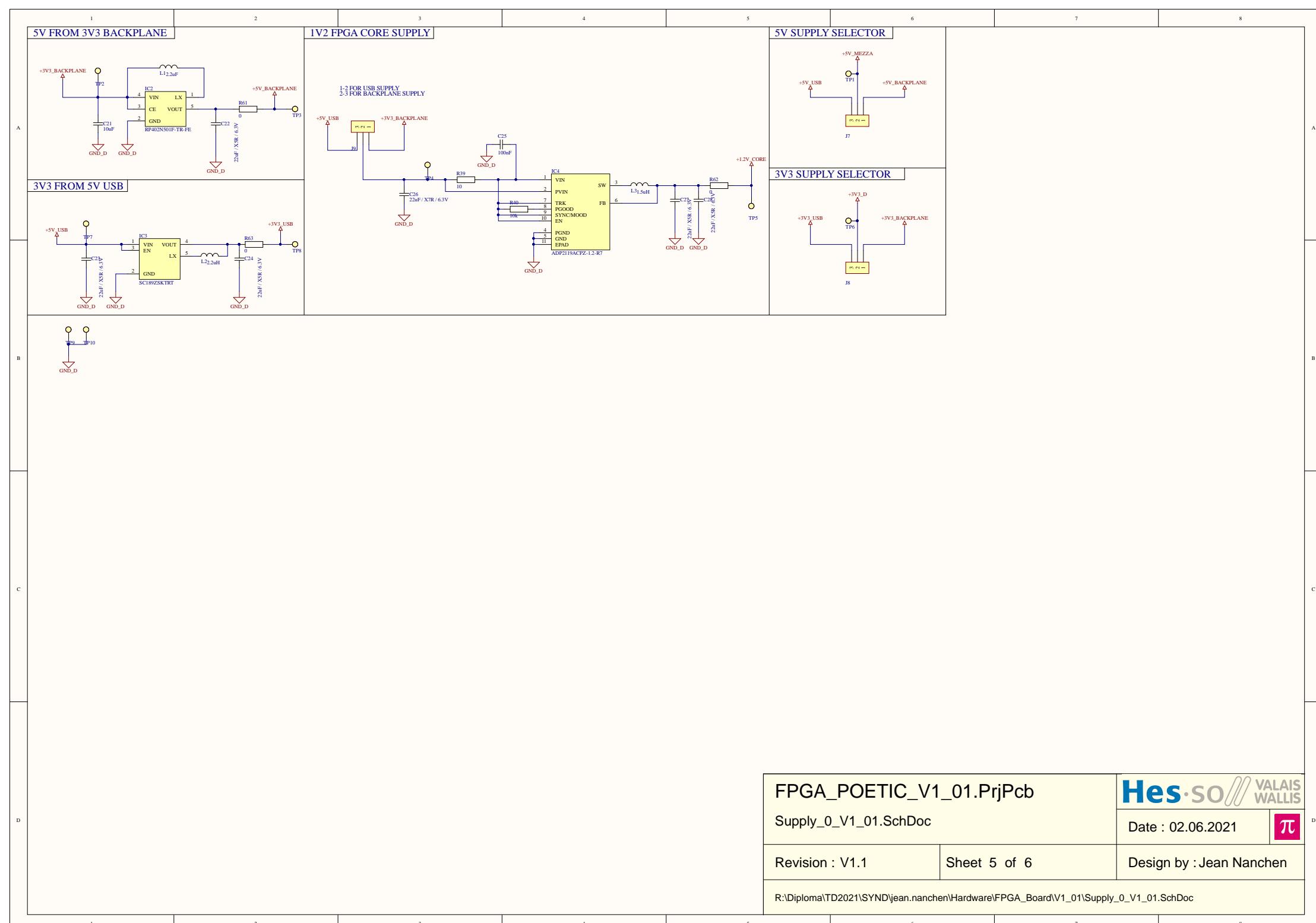


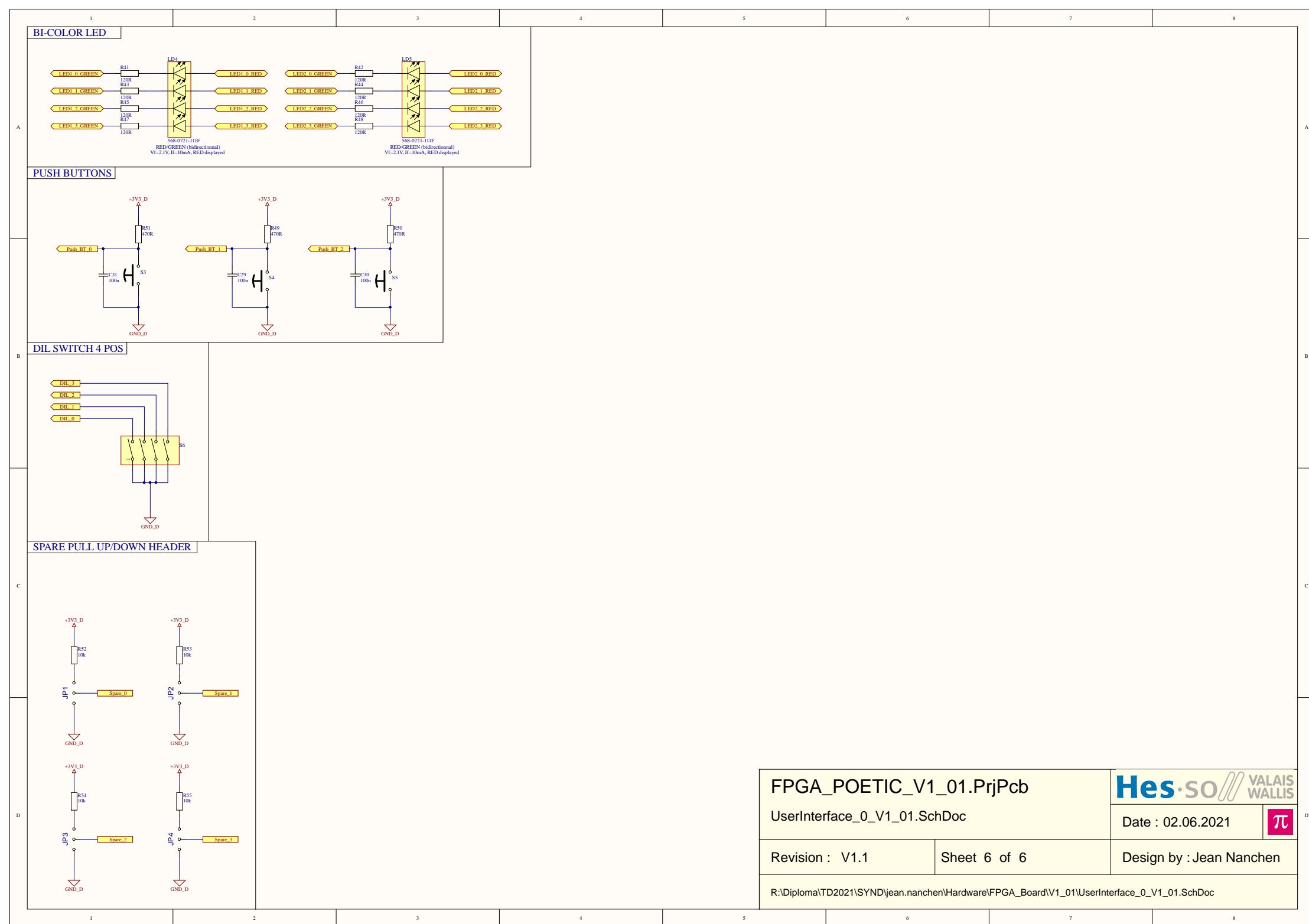
Revision : V1.1

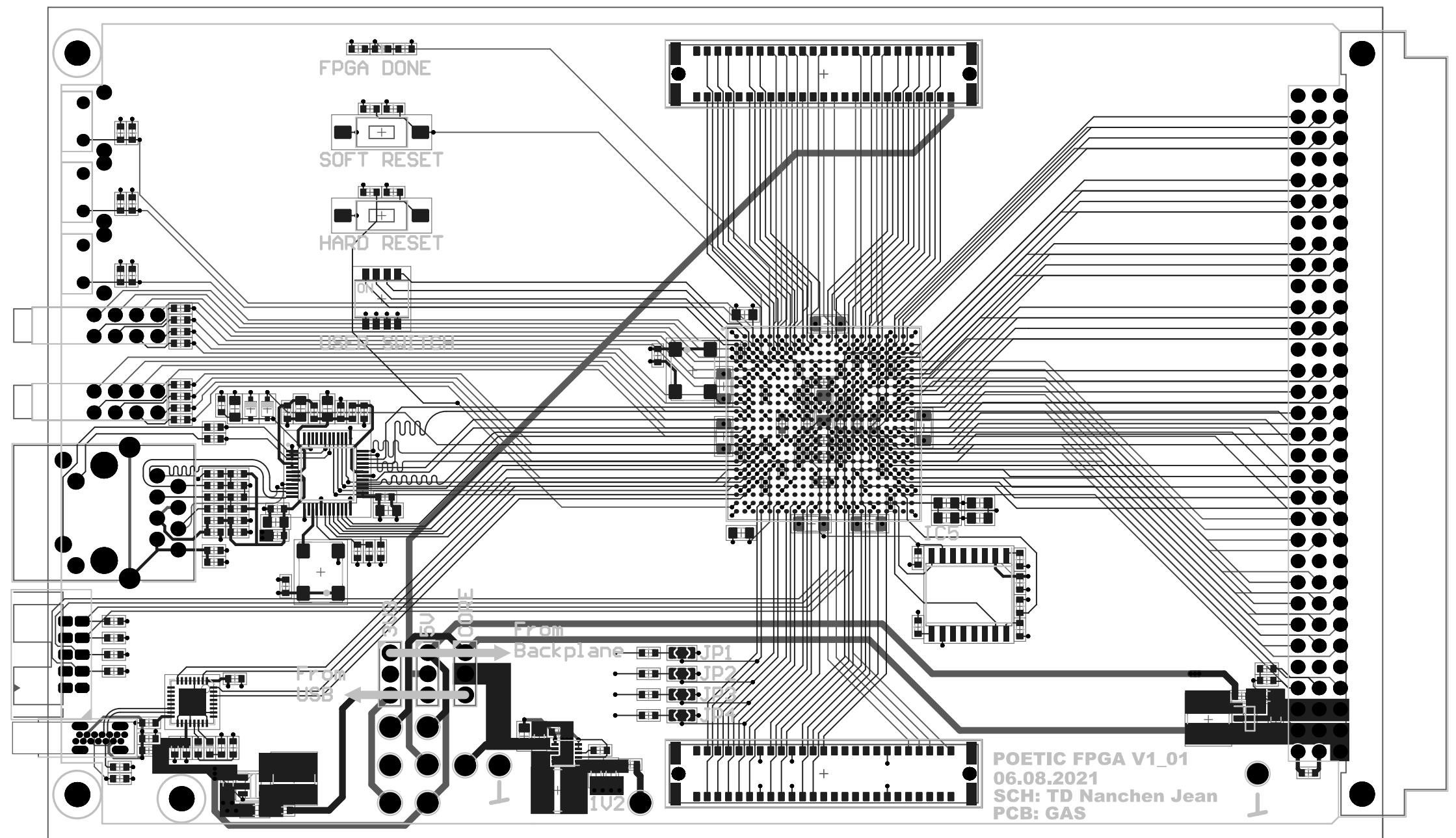
Sheet 4 of 6

Design by : Jean Nanchen

R:\Diploma\TD2021\SYND\jean.nanchen\Hardware\FPGA_Board\V1_01\FPGA_1_V1_01.SchDoc







G | Interface de l'ADC ADS7886

```
--  
-- VHDL Architecture Poetic.ADC.ads7886_decoder  
--  
-- Created:  
--     by - jeann.UNKNOWN (DESKTOP-V46KISN)  
-- at - 14:09:37 17.06.2021  
--  
-- using Mentor Graphics HDL Designer(TM) 2019.2 (Build 5)  
--  
ARCHITECTURE ads7886_decoder OF ADC IS  
    type decodeState is (  
        sendLowCS, waitForData, readData, sendHighCS, ready, print  
    );  
    signal mainState : decodeState;  
    signal memSCLK, risingSCLK : std_ulogic;  
    signal counterWaitData : unsigned(1 DOWNTO 0);  
    signal counterReadData : unsigned(10 DOWNTO 0);  
    signal dataReg : unsignedadcBitNb-1 DOWNTO 0);  
BEGIN  
    decode : process(reset, clock)  
        variable counterTq : integer;  
    begin  
        if reset = '1' then  
            counterTq := 0;  
            mainState <= ready;  
            Data <= (others => '0');  
            CS_n <= '1';  
            counterReadData <= (others => '0');  
            counterWaitData <= (others => '0');  
            dataReg <= (others => '0');  
        elsif rising_edge(clock) then  
            case mainState is  
                when ready =>  
                    if enable = '1' AND risingSCLK = '1' then  
                        counterTq := 0;  
                        mainState <= sendLowCS;  
                    end if;  
                when sendLowCS =>  
                    CS_n <= '0';  
                    mainState <= waitForData;  
                when waitForData =>  
                    if risingSCLK = '1' then  
                        counterWaitData <= counterWaitData +1;  
                        if counterWaitData = 3 + highSpeedEn then  
                            mainState <= readData;  
                            counterWaitData <= (others => '0');  
                        end if;  
                    end if;  
                when readData =>  
                    if risingSCLK = '1' then  
                        counterReadData <= counterReadData + 1;  
                        dataReg <= shift_left(dataReg, 1);  
                        dataReg(dataReg'low) <= SDO;  
                        if (counterReadData = adcBitNb-1) then  
                            counterReadData <= (others => '0');  
                            mainState <= print;  
                        end if;  
                    end if;  
                when print =>  
                    Data <= std_ulogic_vector(dataReg);
```

Annexe G. Interface de l'ADC ADS7886

```
    mainState <= sendHighCS;
when sendHighCS =>
    if risingSCLK = '1' then
        CS_n <= '1';
        counterTq := counterTq + 1;
        if counterTq = 2 then
            mainState <= ready;
        end if;
    end if;
end case;
end if;
end process decode;

detectRisingSCLK : process (reset, clock)
begin
    if reset = '1' then
        memSCLK <= '0';
        risingSCLK <= '0';
    elsif rising_edge(clock) then
        risingSCLK <= '0';
        if memSCLK = '0' AND SCLK = '1' then
            risingSCLK <= '1';
            memSCLK <= SCLK;
        else
            memSCLK <= SCLK;
        end if;
    end if;
end process detectRisingSCLK;
END ARCHITECTURE ads7886_decoder;
```

H | Modèle de l'ADC ADS7886

```
--  
-- VHDL Architecture Poetic_test.AD.ADS7886  
--  
-- Created:  
--     by - jeann.UNKNOWN (DESKTOP-V46KISN)  
-- at - 15:32:09 30.06.2021  
--  
-- using Mentor Graphics HDL Designer(TM) 2019.2 (Build 5)  
--  
ARCHITECTURE ADS7886 OF AD IS  
    type decodeState is (  
        waitHighCS, sendData, sendZeros, waitLowCS, incrementCounterSendData  
    );  
    signal mainState : decodeState;  
    signal memCS, fallingCS : std_ulogic;  
    signal memSCLK, fallingSCLK : std_ulogic;  
    signal dataReg : unsigned(adcBitNb-1 DOWNTO 0);  
  
BEGIN  
    decode : process(reset, clock)  
        variable counterWaitData : integer;  
        variable counterSendData : integer;  
    begin  
        if reset = '1' then  
            mainState <= waitLowCS;  
            counterWaitData := 0;  
            counterSendData := 0;  
        elsif rising_edge(clock) then  
            case mainState is  
                when waitLowCS =>  
                    if fallingCS = '1' then  
                        mainState <= sendZeros;  
                    end if;  
                when sendZeros =>  
                    if fallingSCLK = '1' then  
                        counterWaitData := counterWaitData + 1;  
                        SDO <= '0';  
                    if counterWaitData = 3 then  
                        mainState <= sendData;  
                        dataReg <= unsigned(DataToSend);  
                        counterWaitData := 0;  
                    end if;  
                end if;  
                when sendData =>  
                    if fallingSCLK = '1' then  
                        if counterSendData >= adcBitNb then  
                            mainState <= waitHighCS;  
                            SDO <= '0';  
                            counterSendData := 0;  
                        else  
                            SDO <= dataReg(adcBitNb-1 - counterSendData);  
                            mainState <= incrementCounterSendData;  
                        end if;  
                    end if;  
                when incrementCounterSendData =>  
                    counterSendData := counterSendData + 1;  
                    mainState <= sendData;  
                when waitHighCS =>  
                    if CS_n = '1' then  
                        mainState <= waitLowCS;
```

Annexe H. Modèle de l'ADC ADS7886

```
        end if;
    end case;
end if;
end process decode;

detectFallingCS : process (reset, clock)
begin
    if reset = '1' then
        memCS <= '1';
        fallingCS <= '0';
    elsif rising_edge(clock) then
        fallingCS <= '0';
        if memCS = '1' AND CS_n = '0' then
            fallingCS <= '1';
            memCS <= CS_n;
        else
            memCS <= CS_n;
        end if;
    end if;
end process detectFallingCS;
detectFallingSCLK : process (reset, clock)
begin
    if reset = '1' then
        memSCLK <= '0';
        fallingSCLK <= '0';
    elsif rising_edge(clock) then
        fallingSCLK <= '0';
        if memSCLK = '1' AND SCLK = '0' then
            fallingSCLK <= '1';
            memSCLK <= SCLK;
        else
            memSCLK <= SCLK;
        end if;
    end if;
end process detectFallingSCLK;
END ARCHITECTURE ADS7886;
```

Interface du DAC DACXX4S085

```
--  
-- VHDL Architecture Poetic.DAC.DAC124S085  
--  
-- Created:  
--     by - jeann.UNKNOWN (DESKTOP-V46KISN)  
-- at - 09:21:02 18.06.2021  
--  
-- using Mentor Graphics HDL Designer(TM) 2019.2 (Build 5)  
--  
ARCHITECTURE DAC124S085 OF DAC IS  
    constant numberOfDataToSend : integer := 12;  
    signal memSCLK, risingSCLK : std_ulogic;  
    signal masterWr : std_ulogic;  
    signal masterData : std_logic_vector(dacBitNb+dacChBitNb+dacOpBitNb-1 DOWNTO  
    ↳ 0);  
    type transmitState is(  
        waitForATransmission, sendData, sendLowSync, sendHighSync, sendZero  
    );  
    signal mainState : transmitState;  
BEGIN  
    concatenate : process (reset, clock)  
    begin  
        if reset = '1' then  
            masterData <= (others => '0');  
            masterWr <= '0';  
        elsif rising_edge(clock) then  
            masterWr <= '0';  
            if send = '1' then  
                masterData <= dacSel & mode & data;  
                masterWr <= '1';  
            end if;  
        end if;  
    end process concatenate;  
  
    transmit : process (reset, clock)  
    variable decounter : integer;  
    variable counter : integer;  
    begin  
        if reset = '1' then  
            mainState <= waitForATransmission;  
            decounter := masterData'length-1;  
            Sync_n <= '1';  
            Dout <= '0';  
            counter := 0;  
        elsif rising_edge(clock) then  
            case mainState is  
                when waitForATransmission =>  
                    if masterWr = '1' then  
                        mainState <= sendLowSync;  
                    end if;  
                when sendData =>  
                    if risingSCLK = '1' then  
                        decounter := decounter - 1;  
                        Dout <= masterData(decounter);  
                        if decounter = 0 then  
                            mainState <= sendZero;  
                        end if;  
                    end if;  
                when sendLowSync =>  
                    if risingSCLK = '1' then
```

Annexe I. Interface du DAC DACXX4S085

```
Sync_n <= '0';
Dout <= masterData(decounter);
mainState <= sendData;
end if;
when sendHighSync =>
decounter := masterData'length-1;
counter := 0;
if risingSCLK = '1' then
Sync_n <= '1';
mainState <= waitForATransmission;
end if;
when sendZero =>
if risingSCLK = '1' then
Dout <= '0';
counter := counter + 1;
end if;
if counter >= numberOfWorkToSend - dacBitNb then
mainState <= sendHighSync;
end if;
end case;
end if;
end process transmit;

detectRisingSCLK : process (reset, clock)
begin
if reset = '1' then
memSCLK <= '0';
risingSCLK <= '0';
elsif rising_edge(clock) then
risingSCLK <= '0';
if memSCLK = '0' AND SCLK = '1' then
risingSCLK <= '1';
memSCLK <= SCLK;
else
memSCLK <= SCLK;
end if;
end if;
end process detectRisingSCLK;
END ARCHITECTURE DAC124S085;
```

J | Régulateur PI

```
--  
-- VHDL Architecture Poetic.regulator.PDI3  
--  
-- Created:  
--     by - jean.nanchen.UNKNOWN (WEA30407)  
-- at - 20:04:32 22.07.2021  
--  
-- using Mentor Graphics HDL Designer(TM) 2019.2 (Build 5)  
--  
ARCHITECTURE PDI3 OF regulator IS  
    type State is (  
        ready, calculateNewError, calculatePID, checkOverFlow, print  
    );  
    constant Kp : integer := 4;  
    constant Ki : integer := 3;  
    signal mainState : State;  
    signal updatePID : std_ulogic;  
    signal PIDHasBeenUpdate : std_ulogic;  
    signal error : signed(pidBitNb DOWNTO 0);  
    signal error_sum : signed(pidBitNb+20 DOWNTO 0); --BIG NUMBER  
    signal pidValue : signed(pidBitNb+5+1 DOWNTO 0);  
    signal p : signed(pidBitNb+5 DOWNTO 0);  
    signal i : signed(pidBitNb+5 DOWNTO 0);  
    signal d : signed(pidBitNb+5 DOWNTO 0);  
    signal sOutput : unsigned(pidBitNb-1 DOWNTO 0);  
BEGIN  
    process (clock, reset)  
    begin  
        if reset = '1' then  
            output <= (others => '0');  
            mainState <= ready;  
            error <= (others => '0');  
            p <= (others => '0');  
            i <= (others => '0');  
            d <= (others => '0');  
            pidValue <= (others => '0');  
            sOutput <= (others => '0');  
            PIDHasBeenUpdate <= '0';  
            error_sum <= (others => '0');  
        elsif rising_edge(clock) then  
            case mainState is  
                when ready =>  
                    mainState <= calculateNewError;  
                when calculateNewError =>  
                    error <= signed(resize(unsigned(Setval), error'length)) -  
                        signed(resize(unsigned(adc_data), error'length));  
                    if updatePID = '1' AND ki_sw = '1' then  
                        error_sum <= error_sum + error;  
                        PIDHasBeenUpdate <= '1';  
                    else  
                        PIDHasBeenUpdate <= '0';  
                    end if;  
                    mainState <= calculatePID;  
                when calculatePID =>  
                    if kp_sw = '1' then  
                        p <= resize(Kp * error, p'length);  
                    end if;  
                    if ki_sw = '1' then  
                        i <= resize(Ki*shift_right(error_sum,0), i'length);  
                    end if;
```

Annexe J. Régulateur PI

```
pidValue <= resize(p, pidValue'length) + resize(i, pidValue'length) +
    ↪ resize(d, pidValue'length);
mainState <= checkOverFlow;
when checkOverFlow =>
    if pidValue > 4095 then
        sOutput <= (others => '1');
    elsif pidValue < 0 then
        sOutput <= (others => '0');
    else
        sOutput <= resize(unsigned(pidValue), sOutput'length);
    end if;
    mainState <= print;
when print =>
    mainState <= ready;
    output <= sOutput;
end case;
end if;
end process;

process (clock, reset)
    variable counter : integer := 0;
begin
    if reset = '1' then
        counter := 0;
        updatePID <= '0';
    elsif rising_edge(clock) then
        counter := counter + 1;
        if PIDHasBeenUpdate = '1' then
            updatePID <= '0';
        end if;
        if counter > 100000 then
            updatePID <= '1';
            counter := 0;
        end if;
    end if;
end process;
END ARCHITECTURE PDI3;
```

K | Modèle moteur DC

```
--  
-- VHDL Architecture Poetic_test.motor.dc  
--  
-- Created:  
--     by - jeann.UNKNOWN (DESKTOP-V46KISN)  
-- at - 14:32:12 01.07.2021  
--  
-- using Mentor Graphics HDL Designer(TM) 2019.2 (Build 5)  
--  
ARCHITECTURE dc OF motor IS  
    constant clockFrequency: real := 100.0E03;  
    constant clockPeriod: real := (1.0/clockFrequency);  
    constant L : real := 0.1;  
    constant Kphi : real := 0.2;  
    constant J : real := 0.1;  
    constant B : real := 0.1;  
    constant R : real := 1.6;  
BEGIN  
    decode : process(reset, clock)  
        variable di_dt : real;  
        variable dw_dt : real;  
        variable w : real;  
        variable i : real;  
    begin  
        if reset = '1' then  
            di_dt := 0.0;  
            dw_dt := 0.0;  
            i := 0.0;  
            w := 0.0;  
            speed <= (others => '0');  
        elsif rising_edge(clock) then  
            di_dt := (real(to_integer(Vapp))/L) - ((R/L) * i) - ((Kphi/L)*w);  
            dw_dt := ((Kphi/J) * i) - ((B/J) * w);  
            w := w + dw_dt * clockPeriod;  
            i := i + di_dt * clockPeriod;  
            speed <= std_logic_vector(to_unsigned(integer(w), speed'length));  
        end if;  
    end process decode;  
END ARCHITECTURE dc;
```


L | Générateur de clock

```
--  
-- VHDL Architecture Poetic.clockGenerator.clockDivider  
--  
-- Created:  
--     by - jeann.UNKNOWN (DESKTOP-V46KISN)  
-- at - 11:15:13 17.06.2021  
--  
-- using Mentor Graphics HDL Designer(TM) 2019.2 (Build 5)  
--  
ARCHITECTURE clockDivider OF clockGenerator IS  
    constant countValue : integer := integer(real(integer(real(frequencyIn) /  
        → real((frequencyOut)))) / 2.0);  
    signal counter : unsigned(counterBitNb-1 DOWNTO 0);  
    signal clockOut_int : std_ulogic;  
BEGIN  
    clockOut <= clockOut_int;  
    increment : process(reset, clock)  
    begin  
        if reset = '1' then  
            clockOut_int <= '0';  
            counter <= (others => '0');  
            counter(counter'low) <= '1';  
        elsif rising_edge(clock) then  
            if enable = '1' then  
                counter <= counter + 1;  
                if (counter >= countValue) then  
                    clockOut_int <= not clockOut_int;  
                    counter <= (others => '0');  
                    counter(counter'low) <= '1';  
                end if;  
            else  
                clockOut_int <= '0';  
            end if;  
        end if;  
    end process increment;  
END ARCHITECTURE clockDivider;
```


M | Contrôleur UART

```
--  
-- VHDL Architecture Poetic uartController.fsm  
--  
-- Created:  
--     by - jeann.UNKNOWN (DESKTOP-V46KISN)  
-- at - 17:17:44 11.08.2021  
--  
-- Generated by Mentor Graphics' HDL Designer(TM) 2019.2 (Build 5)  
--  
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE ieee.numeric_std.all;  
  
ARCHITECTURE fsm OF uartController IS  
  
    TYPE STATE_TYPE IS (  
        waitForCommand,  
        dispatcher,  
        S_select,  
        S_receive,  
        S_waitForOtherCharacter  
    );  
  
    -- Declare current and next state signals  
    SIGNAL current_state : STATE_TYPE;  
    SIGNAL next_state : STATE_TYPE;  
  
    -- Declare any pre-registered internal signals  
    SIGNAL consigne_cld : std_ulogic_vector (dataBitNb-1 DOWNTO 0);  
    SIGNAL endOfMsg_cld : std_ulogic ;  
    SIGNAL newCharacter_cld : std_ulogic ;  
    SIGNAL rxRd_cld : std_ulogic ;  
    SIGNAL txData_cld : std_ulogic_vector (dataBitNb-1 DOWNTO 0);  
    SIGNAL txWr_cld : std_ulogic ;  
  
BEGIN  
  
    -----
    clocked_proc : PROCESS (  
        clock,  
        reset  
    )  
    -----  
BEGIN  
        IF (reset = '1') THEN  
            current_state <= waitForCommand;  
            -- Default Reset Values  
            consigne_cld <= (others => '0');  
            endOfMsg_cld <= '0';  
            newCharacter_cld <= '0';  
            rxRd_cld <= '0';  
            txData_cld <= (others => '0');  
            txWr_cld <= '0';  
        ELSIF (clock'EVENT AND clock = '1') THEN  
            current_state <= next_state;  
  
            -- Combined Actions  
            CASE current_state IS  
                WHEN waitForCommand =>  
                    newCharacter_cld <= '0';  
-----
```

Annexe M. Contrôleur UART

```

        rxRd_cld <= '0';
        consigne_cld <= (others => '0');
        endOfMsg_cld <= '0';
        txData_cld <= (others => '0');
        txWr_cld <= '0';
    WHEN dispatcher =>
        IF (rxData = X"53") THEN
            rxRd_cld <= '1';
        ELSE
            rxRd_cld <= '1';
        END IF;
    WHEN S_select =>
        rxRd_cld <= '0';
        newCharacter_cld <= '0';
        IF (rxEmpty = '0' AND rxData/=X"53") THEN
            rxRd_cld <= '1';
        END IF;
    WHEN S_receive =>
        consigne_cld <= rxData;
        if rxData /= X"65" then
            newCharacter_cld <= '1';
        end if;
        IF (rxData /= X"65") THEN
        ELSIF (rxData = X"65") THEN
            endOfMsg_cld <= '1';
        END IF;
    WHEN S_waitForOtherCharacter =>
        rxRd_cld <= '0';
        newCharacter_cld <= '0';
        IF (rxEmpty = '0') THEN
            rxRd_cld <= '1';
        END IF;
    WHEN OTHERS =>
        NULL;
    END CASE;
END IF;
END PROCESS clocked_proc;

-----
nextstate_proc : PROCESS (
    current_state,
    rxData,
    rxEmpty
)
-----
BEGIN
    CASE current_state IS
        WHEN waitForCommand =>
            IF (rxEmpty = '0') THEN
                next_state <= dispatcher;
            ELSE
                next_state <= waitForCommand;
            END IF;
        WHEN dispatcher =>
            IF (rxData = X"53") THEN
                next_state <= S_select;
            ELSE
                next_state <= waitForCommand;
            END IF;
        WHEN S_select =>
            IF (rxEmpty = '0' AND rxData/=X"53") THEN
                next_state <= S_receive;
            ELSE
                next_state <= S_select;
            END IF;
    END CASE;
END;

```

```

WHEN S_receive =>
    IF (rxData /= X"65") THEN
        next_state <= S_waitForOtherCharacter;
    ELSIF (rxData = X"65") THEN
        next_state <= waitForCommand;
    ELSE
        next_state <= S_receive;
    END IF;
WHEN S_waitForOtherCharacter =>
    IF (rxEmpty = '0') THEN
        next_state <= S_receive;
    ELSE
        next_state <= S_waitForOtherCharacter;
    END IF;
WHEN OTHERS =>
    next_state <= waitForCommand;
END CASE;
END PROCESS nextstate_proc;

-- Concurrent Statements
-- Clocked output assignments
consigne <= consigne_cld;
endOfMsg <= endOfMsg_cld;
newCharacter <= newCharacter_cld;
rxRd <= rxRd_cld;
txData <= txData_cld;
txWr <= txWr_cld;
END fsm;

```


N | Décodeur ASCII (UART)

```
--  
-- VHDL Architecture Poetic.serialAsciiDecoder.serialAsciiDecoder  
--  
-- Created:  
--     by - jeann.UNKNOWN (DESKTOP-V46KISN)  
-- at - 10:55:13 29.06.2021  
--  
-- using Mentor Graphics HDL Designer(TM) 2019.2 (Build 5)  
--  
ARCHITECTURE serialAsciiDecoder OF serialAsciiDecoder IS  
    type decodeState is (  
        isReceiving, endOfReceive, ready, print, increment, incrementAdder  
    );  
    signal mainState : decodeState;  
    type t_Memory is array (0 to 11) of std_ulogic_vector(7 downto 0);  
    signal r_Mem : t_Memory;  
BEGIN  
    decode : process(reset, clock)  
        variable counterCharacter : integer;  
        variable counterAdder : integer;  
        variable uOutput : integer;  
    begin  
        if reset = '1' then  
            uOutput := 0;  
            output <= (others => '0');  
            mainState <= ready;  
            counterCharacter := 0;  
            counterAdder := 0;  
            for i in 0 to 11 loop  
                r_Mem(i) <= (others => '0');  
            end loop;  
        elsif rising_edge(clock) then  
            case mainState is  
                when ready =>  
                    uOutput := 0;  
                    if newCharacter = '1' then  
                        r_Mem(counterCharacter) <= consigne;  
                        mainState <= increment;  
                    end if;  
                when isReceiving =>  
                    if endOfMsg = '1' then  
                        mainState <= endOfReceive;  
                    end if;  
                    if newCharacter = '1' then  
                        mainState <= increment;  
                        r_Mem(counterCharacter) <= consigne;  
                    end if;  
                when increment =>  
                    counterCharacter := counterCharacter + 1;  
                    mainState <= isReceiving;  
                when endOfReceive =>  
                    if counterAdder = counterCharacter then  
                        mainState <= print;  
                    else  
                        r_Mem(counterAdder) <= (others => '0');  
                        mainState <= incrementAdder;  
                        if counterAdder /= counterCharacter - 1 then  
                            case r_Mem(counterAdder) is  
                                when X"30" => uOutput := uOutput * 10;  
                                when X"31" => uOutput := (uOutput + 1) * 10;  
                        end if;  
                    end if;  
                end case;  
            end if;  
        end if;  
    end begin
```

Annexe N. Décodeur ASCII (UART)

```
when X"32" => uOutput := (uOutput + 2) * 10;
when X"33" => uOutput := (uOutput + 3) * 10;
when X"34" => uOutput := (uOutput + 4) * 10;
when X"35" => uOutput := (uOutput + 5) * 10;
when X"36" => uOutput := (uOutput + 6) * 10;
when X"37" => uOutput := (uOutput + 7) * 10;
when X"38" => uOutput := (uOutput + 8) * 10;
when X"39" => uOutput := (uOutput + 9) * 10;
when others =>
end case;
else
case r_Mem(counterAdder) is
when X"30" => uOutput := uOutput;
when X"31" => uOutput := (uOutput + 1);
when X"32" => uOutput := (uOutput + 2);
when X"33" => uOutput := (uOutput + 3);
when X"34" => uOutput := (uOutput + 4);
when X"35" => uOutput := (uOutput + 5);
when X"36" => uOutput := (uOutput + 6);
when X"37" => uOutput := (uOutput + 7);
when X"38" => uOutput := (uOutput + 8);
when X"39" => uOutput := (uOutput + 9);
when others =>
end case;
end if;
when incrementAdder =>
counterAdder := counterAdder + 1;
mainState <= endOfReceive;
when print =>
output <= std_logic_vector(to_unsigned(uOutput, output'length));
counterCharacter := 0;
counterAdder := 0;
mainState <= ready;
end case;
end if;
end process decode;
END ARCHITECTURE serialAsciiDecoder;
```

O | Contrôleur BLDC

```
--  
-- VHDL Architecture Poetic.BLDCController.BLDC  
--  
-- Created:  
--     by - jean.nanchen.UNKNOWN (WEA30407)  
-- at - 11:42:58 22.07.2021  
--  
-- using Mentor Graphics HDL Designer(TM) 2019.2 (Build 5)  
--  
ARCHITECTURE BLDC OF BLDCController IS  
BEGIN  
    logic : process (Hall_A, Hall_B, Hall_C, PWM_in)  
        variable hallReg : std_ulogic_vector(2 DOWNTO 0);  
    begin  
        hallReg := Hall_A & Hall_B & Hall_C;  
        case hallReg is  
            when "001" => --Sa_top 1 & Sb_bot 1  
                Sa_top <= PWM_in;  
                Sa_bot <= '0';  
                Sb_top <= '0';  
                Sb_bot <= PWM_in;  
                Sc_top <= '0';  
                Sc_bot <= '0';  
            when "101" => --Sa_top 1 & Sc_bot 1  
                Sa_top <= PWM_in;  
                Sa_bot <= '0';  
                Sb_top <= '0';  
                Sb_bot <= '0';  
                Sc_top <= '0';  
                Sc_bot <= PWM_in;  
            when "100" =>  
                Sa_top <= '0';  
                Sa_bot <= '0';  
                Sb_top <= PWM_in;  
                Sb_bot <= '0';  
                Sc_top <= '0';  
                Sc_bot <= PWM_in;  
            when "110" =>  
                Sa_top <= '0';  
                Sa_bot <= PWM_in;  
                Sb_top <= PWM_in;  
                Sb_bot <= '0';  
                Sc_top <= '0';  
                Sc_bot <= '0';  
            when "010" =>  
                Sa_top <= '0';  
                Sa_bot <= PWM_in;  
                Sb_top <= '0';  
                Sb_bot <= '0';  
                Sc_top <= PWM_in;  
                Sc_bot <= '0';  
            when "011" =>  
                Sa_top <= '0';  
                Sa_bot <= '0';  
                Sb_top <= '0';  
                Sb_bot <= PWM_in;  
                Sc_top <= '0';  
                Sc_bot <= '0';  
            when others =>  
                Sa_top <= '0';
```

Annexe O. Contrôleur BLDC

```
Sa_bot <= '0';
Sb_top <= '0';
Sb_bot <= '0';
Sc_top <= '0';
Sc_bot <= '0';
end case;
end process logic;
END ARCHITECTURE BLDC;
```

P | Fichier UCF

```
CONFIG VCCAUX = "3.3";
#-----
# Clock and reset
#
NET "clock"          LOC = J1 | IOSTANDARD = LVCMOS33;
NET "reset_N"        LOC = D5 | PULLUP | IOSTANDARD = LVCMOS33;

NET "USB_TX"          LOC = Y2 | IOSTANDARD = LVCMOS33;
NET "USB_RX"          LOC = W1 | IOSTANDARD = LVCMOS33;

NET "DAC_SCLK"        LOC = AB19 | IOSTANDARD = LVCMOS33;
NET "DAC_SDO"          LOC = AB16 | IOSTANDARD = LVCMOS33;
NET "DAC_SYNC"         LOC = AB17 | IOSTANDARD = LVCMOS33;

NET "LED0_0"           LOC = E4 | IOSTANDARD = LVCMOS33;
NET "LED0_1"           LOC = H3 | IOSTANDARD = LVCMOS33;

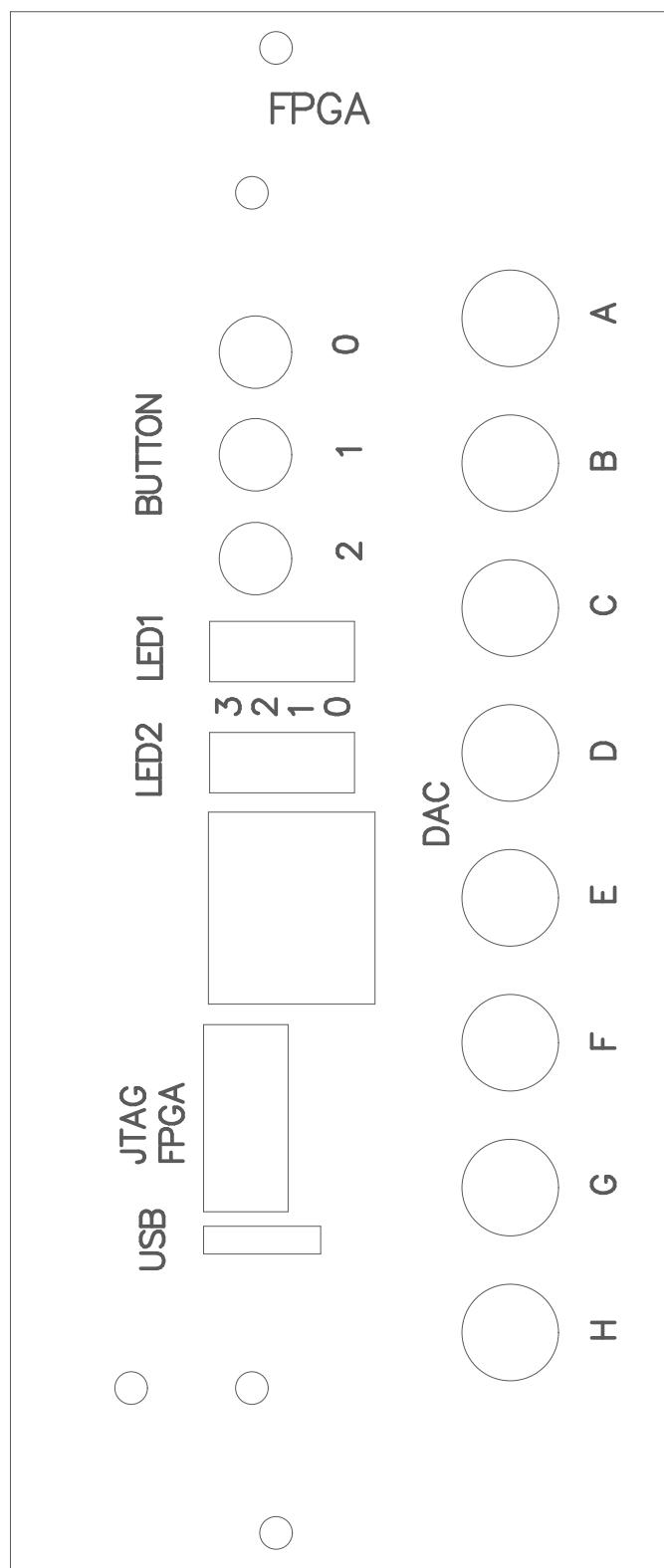
NET "Hall_A"           LOC = A20 | IOSTANDARD = LVCMOS33;
NET "Hall_B"           LOC = A21 | IOSTANDARD = LVCMOS33;
NET "Hall_C"           LOC = B22 | IOSTANDARD = LVCMOS33;

NET "Sa_top"           LOC = C19 | IOSTANDARD = LVCMOS33;
NET "Sa_bot"           LOC = C20 | IOSTANDARD = LVCMOS33;
NET "Sb_top"           LOC = D19 | IOSTANDARD = LVCMOS33;
NET "Sb_bot"           LOC = D20 | IOSTANDARD = LVCMOS33;
NET "Sc_top"           LOC = E20 | IOSTANDARD = LVCMOS33;
NET "Sc_bot"           LOC = F20 | IOSTANDARD = LVCMOS33;

NET "ADC_SCLK"         LOC = B18 | IOSTANDARD = LVCMOS33;
NET "ADC0_SDO"          LOC = H14 | IOSTANDARD = LVCMOS33;
NET "ADC0_CS"           LOC = F15 | IOSTANDARD = LVCMOS33;

NET "BP_PWM_12A"        LOC = H21 | IOSTANDARD = LVCMOS33;
NET "BP_PWM_12B"        LOC = J22 | IOSTANDARD = LVCMOS33;
NET "BP_GPIO_26"         LOC = K22 | IOSTANDARD = LVCMOS33;
NET "BP_GPIO_27"         LOC = M22 | IOSTANDARD = LVCMOS33;
NET "BP_GPIO_28"         LOC = P22 | IOSTANDARD = LVCMOS33;
```


Q | Face avant



R | Mise en place du démonstrateur

R.1 Matériel requis

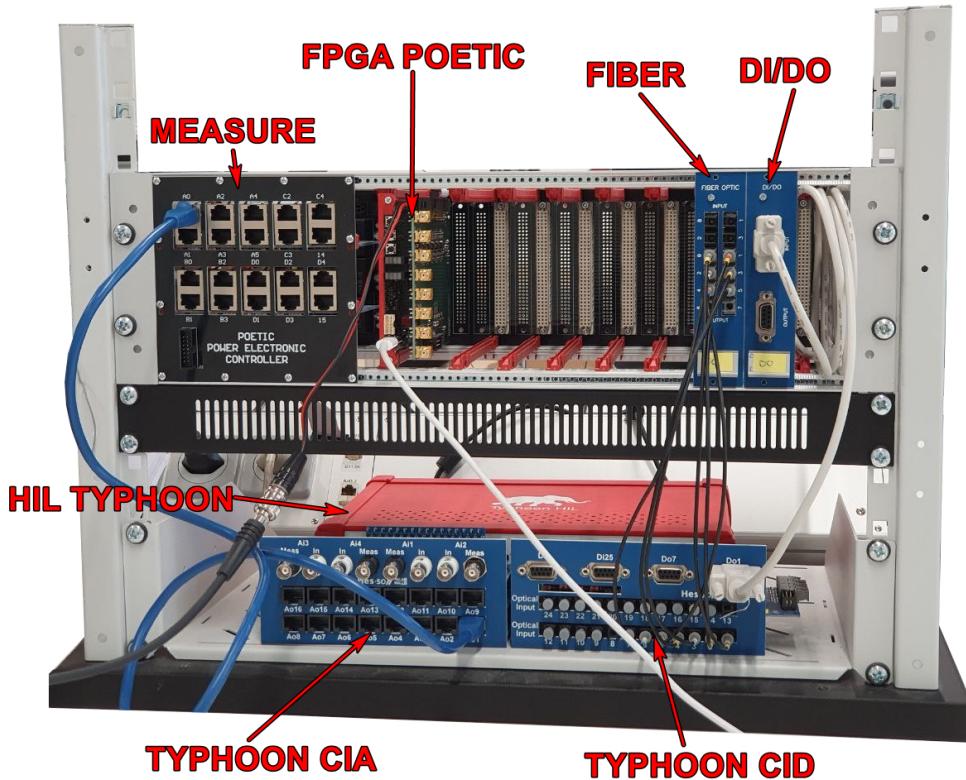


Figure R.1 Disposition du rack Poetic pour le démonstrateur

Voici la liste du matériel requis pour ce démonstateur :

- Module FPGA POETIC
- Module MEASURE
- Module FIBER (en configuration 00¹)
- Module DI/DO (en configuration 00¹)
- Typhoon HIL 402
- Module TYPHOON CIA
- Module TYPHOON CID
- 1x câble d'interface 9 pins
- 1x câble RJ45
- 6x fibres optiques
- 1x câble USB-C
- 1x Xilinx Programmer

1. La configuration se trouve sur la poignée du module (en jaune)

R.2 Câblage

Dans cette section, est expliqué, le câblage entre le rack Poetic et le simulateur Typhoon HIL.

La première étape est de placer tous les modules dans le rack Poetic comme sur la figure R.1.

Ensuite, il faut connecter le module mesure et le module Poetic FPGA à l'aide d'un câble plat 20 pins (voir l'illustration R.2).

Un schéma bloc à la figure R.3, illustre la connexion entre les modules du rack Poetic et les modules du simulateur HIL Typhoon.

Un câble RJ45 relie la sortie «Ao1» du module TYPHOON CIA à l'entrée A0 du module de mesure.

Un câble d'interface 9 pins relie la sortie «Do1» du module TYPHOON CID et l'entrée du module DI/DO.

Six fibres optiques relient les sorties (0..5) du module fibres optique aux entrées optiques du module TYPHOON CID (1..6).

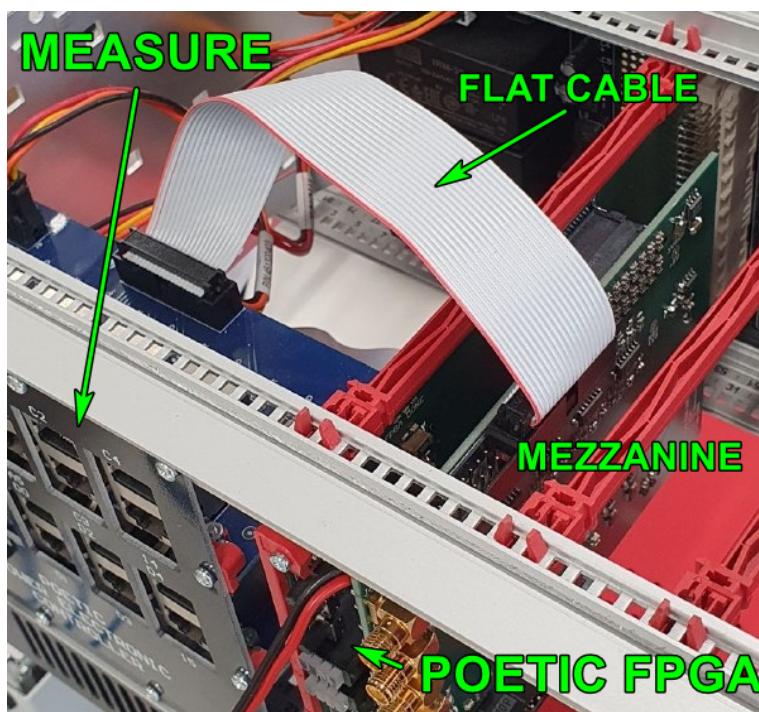


Figure R.2 Câble plat reliant le module mesure et le module Poetic FPGA

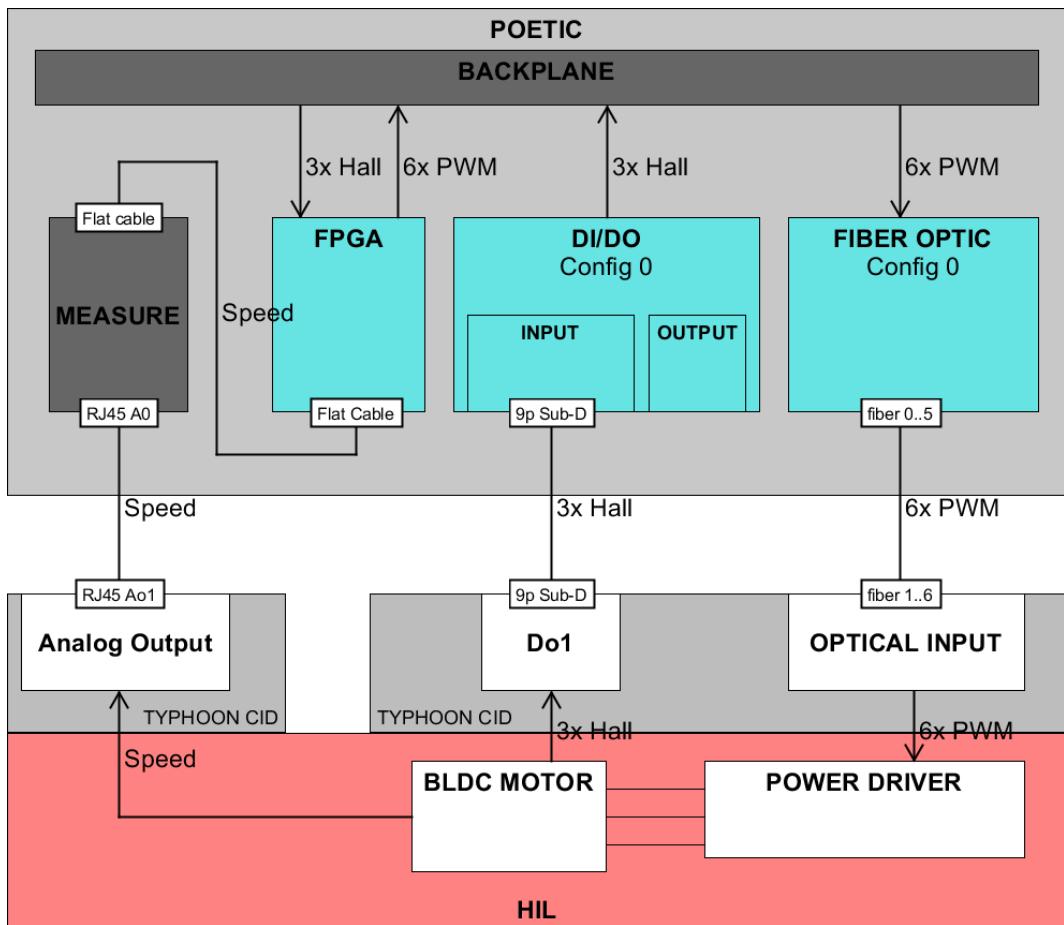


Figure R.3 Schéma bloc du câblage entre la FPGA et le simulateur Typhoon HIL

R.3 Installations des fichiers

Les fichiers nécessaire à la mise en place du démonstrateur ainsi que les fichiers nécessaires au développement se trouve sur le repo : https://github.com/73jn/td_fpga-developing-board-demonstrator

```
sysadmin@localhost:~$ git clone
→ https://github.com/73jn/td_fpga-developing-board-demonstrator.git
```

Ce repo contient :

- la schématique Altium des deux PCB (FPGA et Mezzanine)
- l'architecture software implémentée sur HDL Designer
- un fichier excel contenant le pinning de l'environnement poetic
- le projet contenant la simulation sur le simulateur HIL de Typhoon

R.4 Flash BLDC POETIC sur la FPGA

Le tutoriel de cette section permet, l'aide du logiciel iMPACT, de flasher le fichier de configuration POETIC BLDC sur le module FPGA. Une fois réalisé, une LED verte sur la face avant doit se mettre à clignoter.

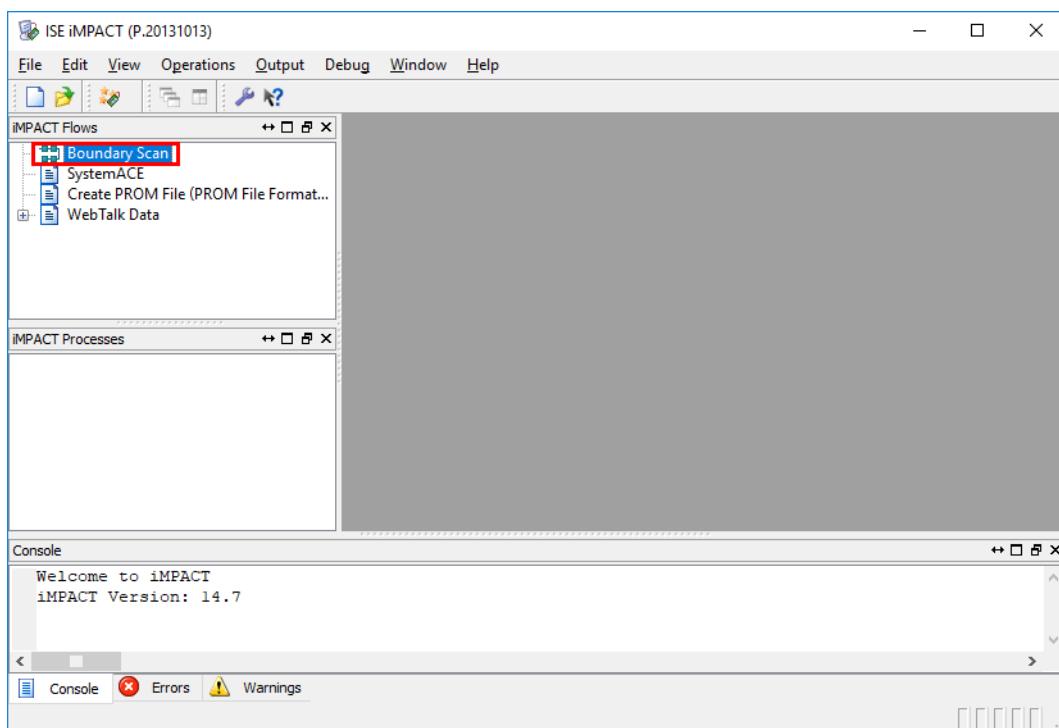


Figure R.4 Double clicker sur «Boundary Scan»

R.4. Flash BLDC POETIC sur la FPGA

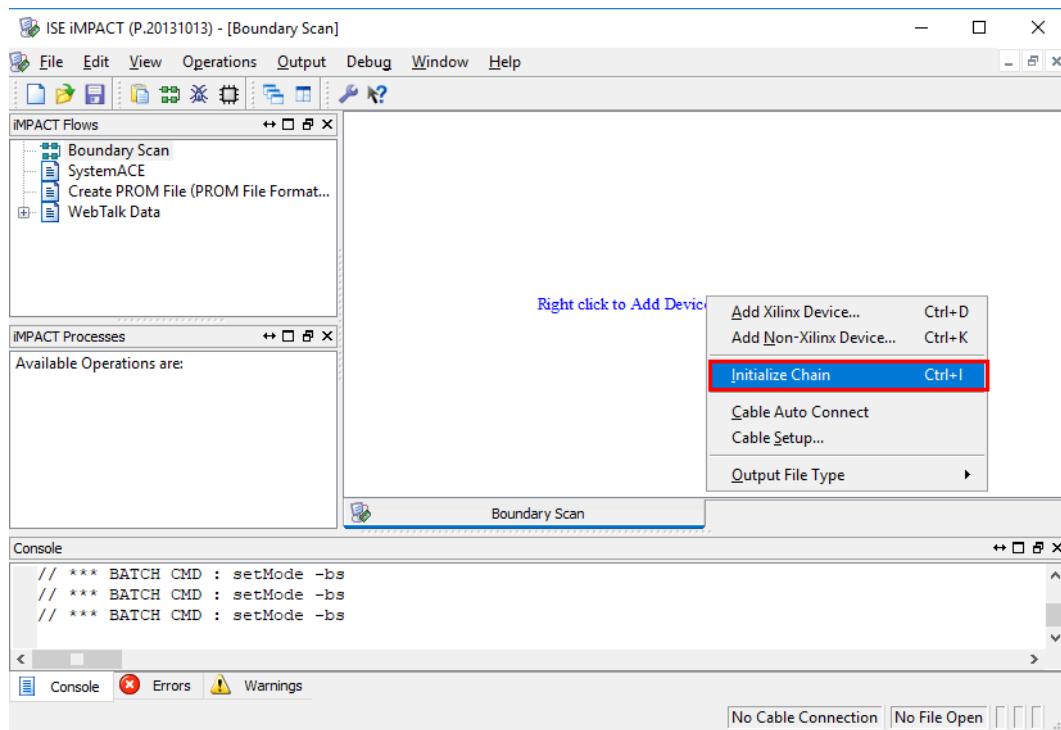


Figure R.5 Click droit -> «Initialize Chain» ou CTRL+I

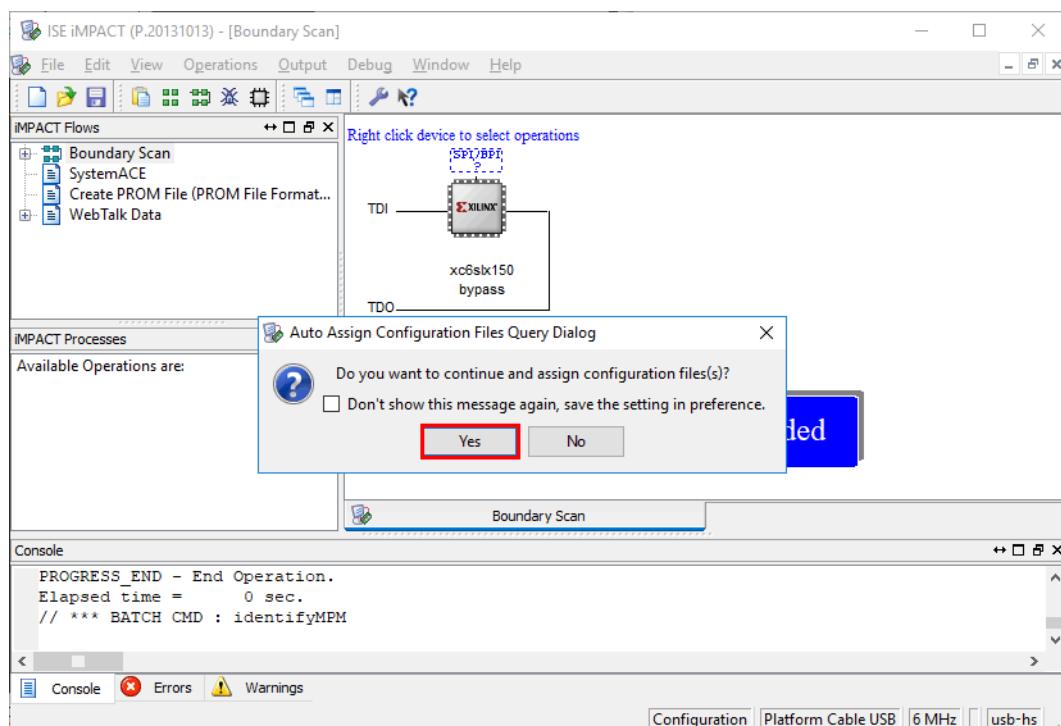


Figure R.6 Cliquer sur «Yes» pour assigner un fichier .bit de configuration

Annexe R. Mise en place du démonstrateur

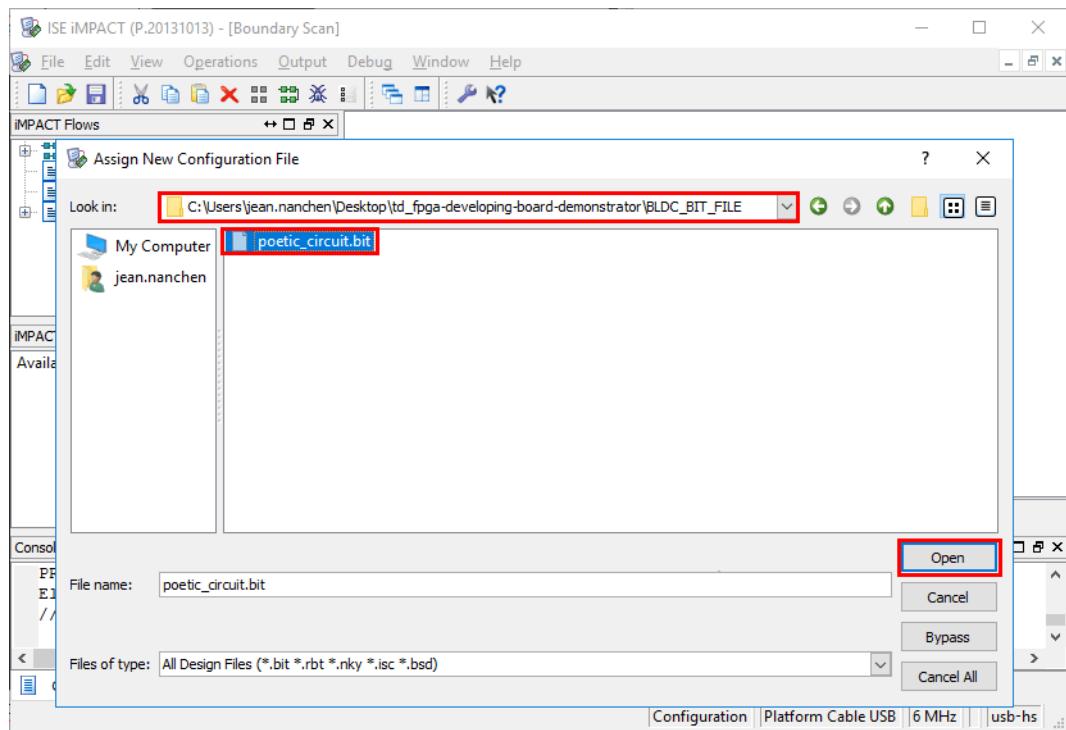


Figure R.7 Aller dans le fichier /BLDC_BIT_FILE du repo et ouvrir le fichier «poetic_circuit.bit»

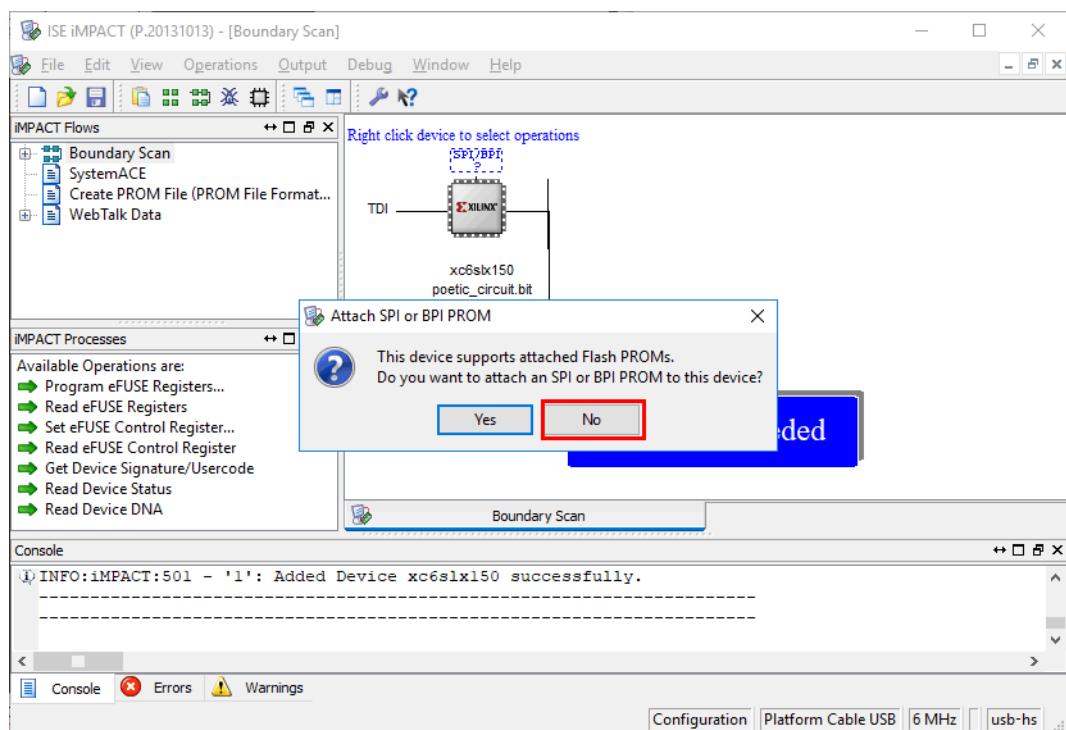


Figure R.8 Cliquer sur «No» pour flasher uniquement la FPGA

R.4. Flash BLDC POETIC sur la FPGA

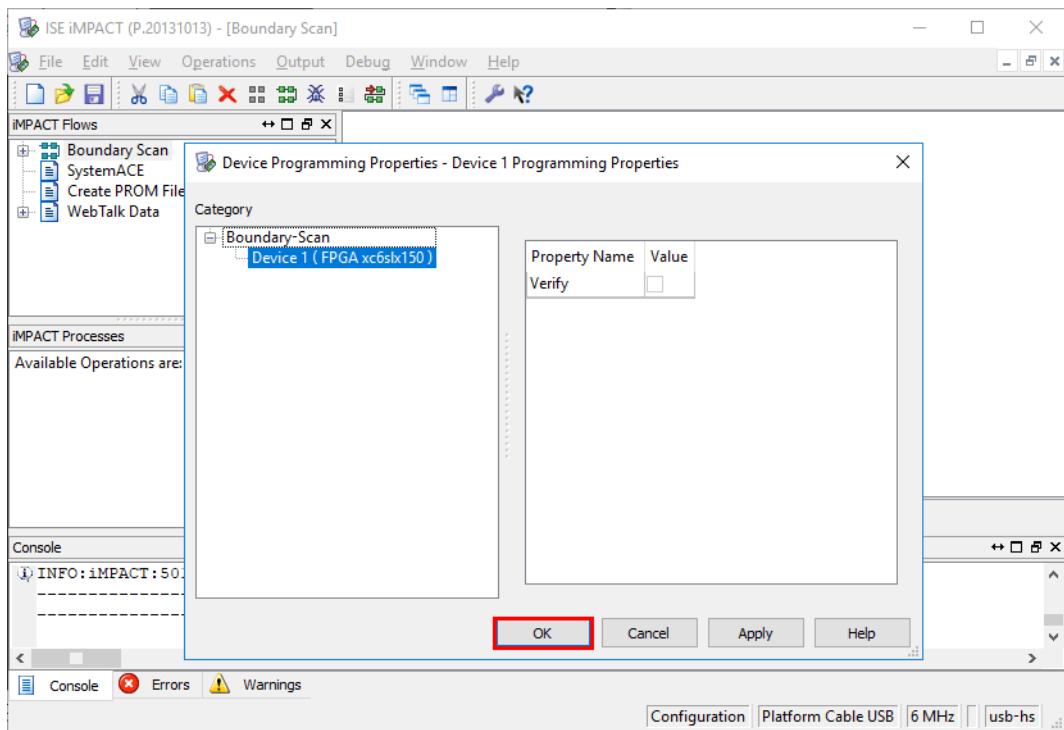


Figure R.9 La FPGA a été reconnue. Clicker sur «Ok»

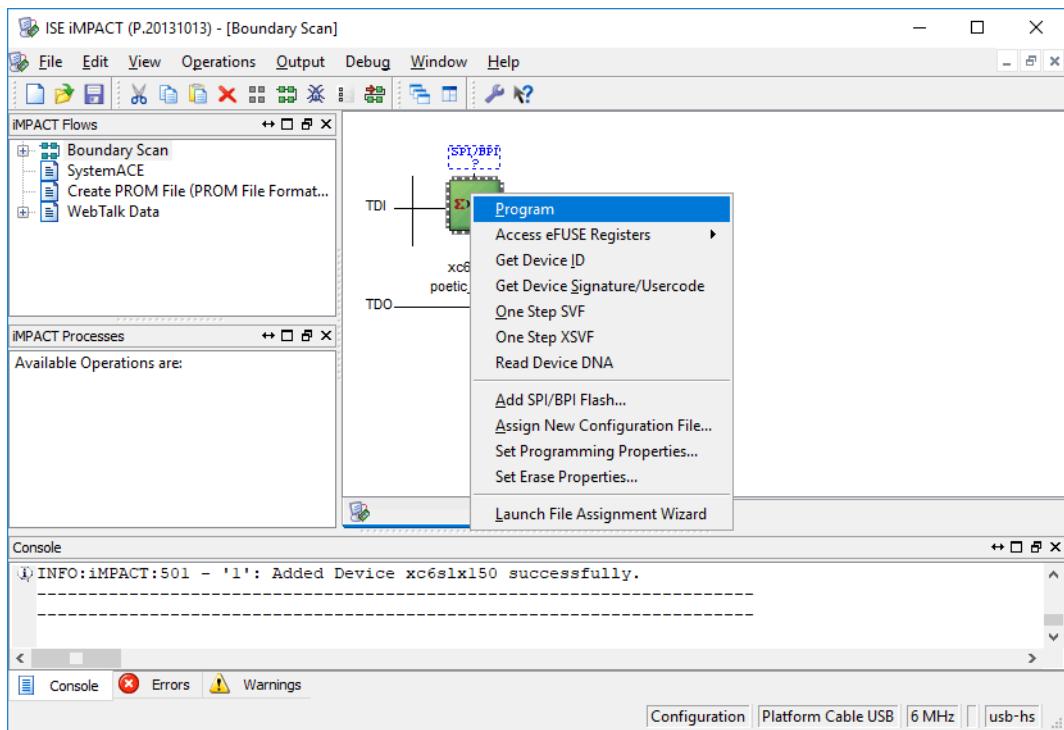


Figure R.10 Pour programmer la FPGA, il effectuer un Click droit sur celle-ci et clicker sur «Program»

Annexe R. Mise en place du démonstrateur

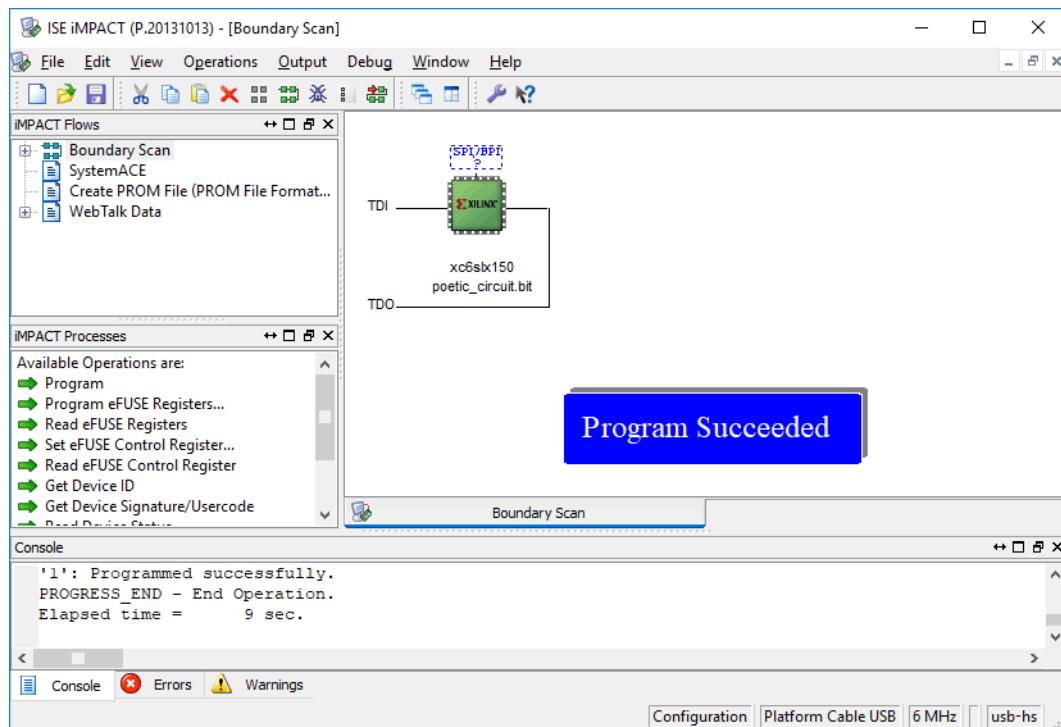


Figure R.11 Programmation réussie

R.5 Simulateur HIL Typhoon

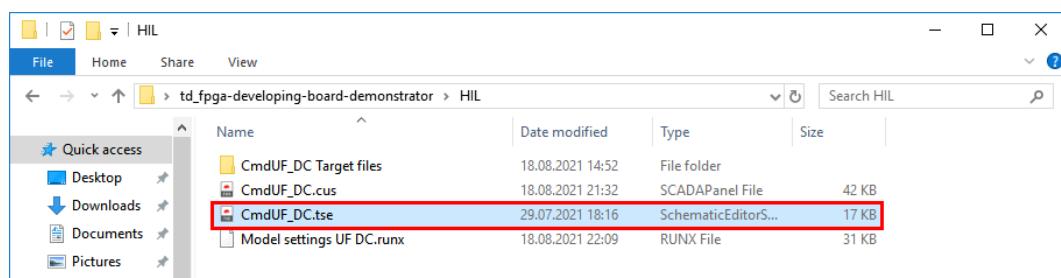


Figure R.12 Ouvrir la schématique du simulateur avec le fichier «CmdUF_DC.tse»

R.5. Simulateur HIL Typhoon

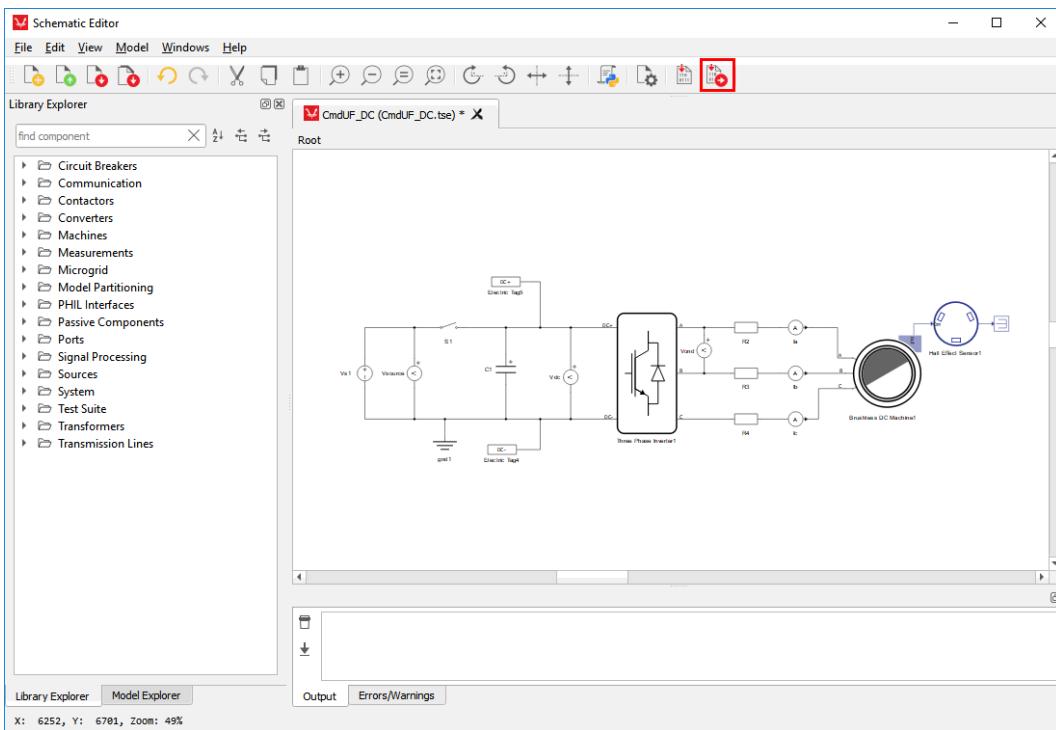


Figure R.13 Compiler la schématique et lancer le simulateur

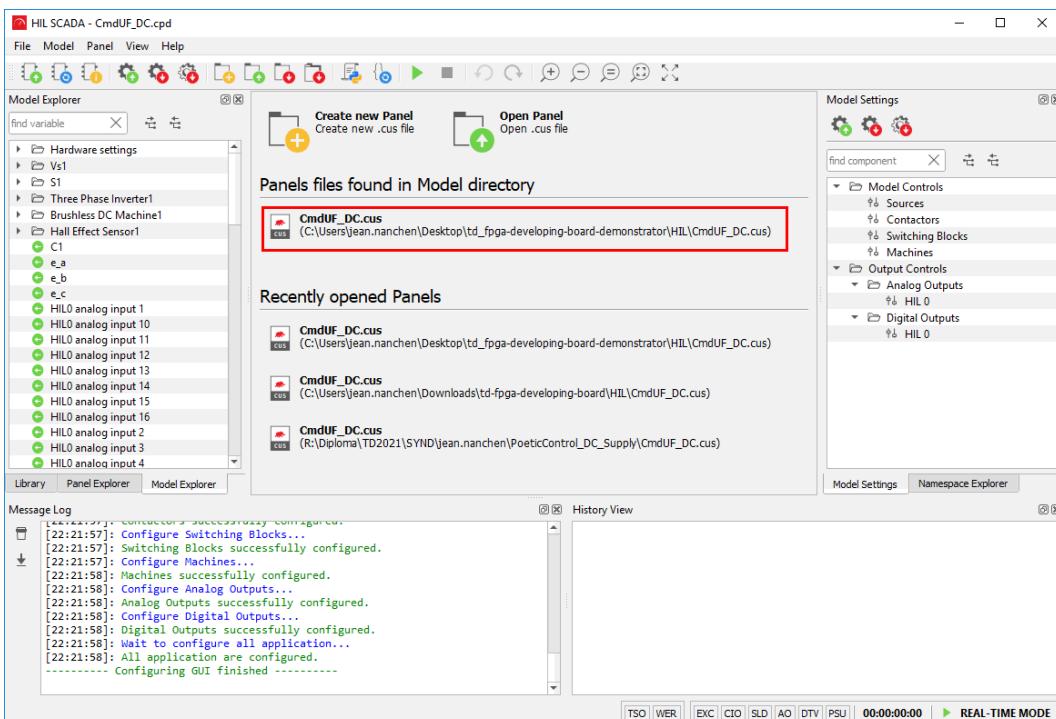


Figure R.14 Ouvrir le panel «CmdUF_DC.cus» qui se trouve dans le dossier /HIL

Annexe R. Mise en place du démonstrateur

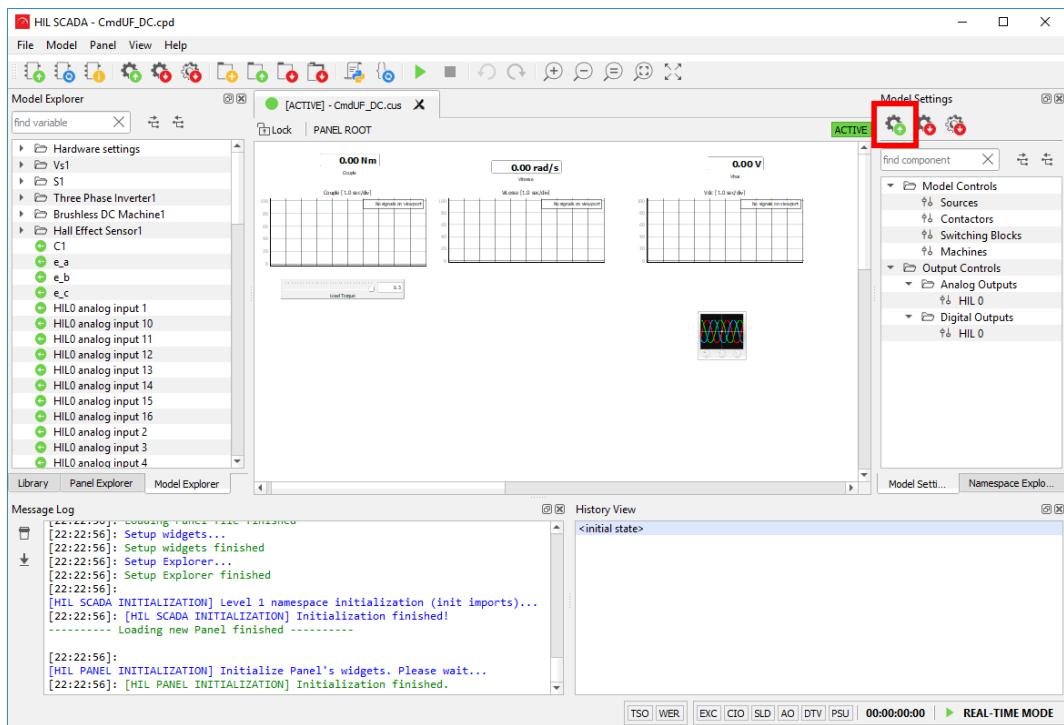


Figure R.15 Charger les paramètres du modèle

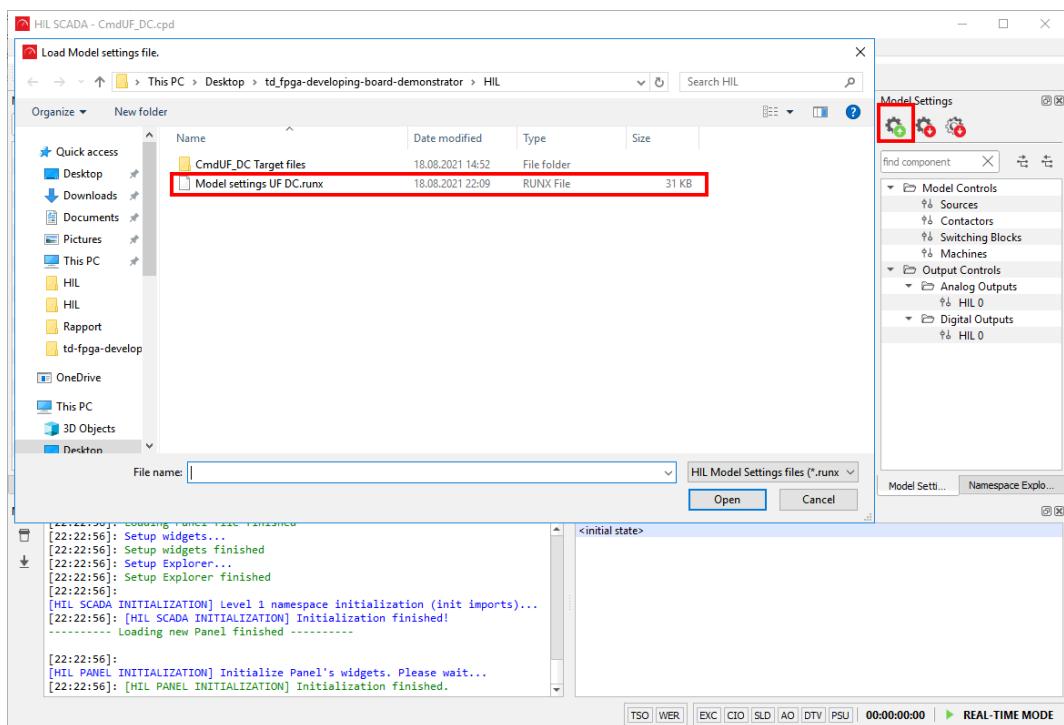


Figure R.16 Sélectionner le fichier «Model settings UF DC.runx» contenant les paramètres du modèle

R.6. Envoie de la consigne à travers l'UART

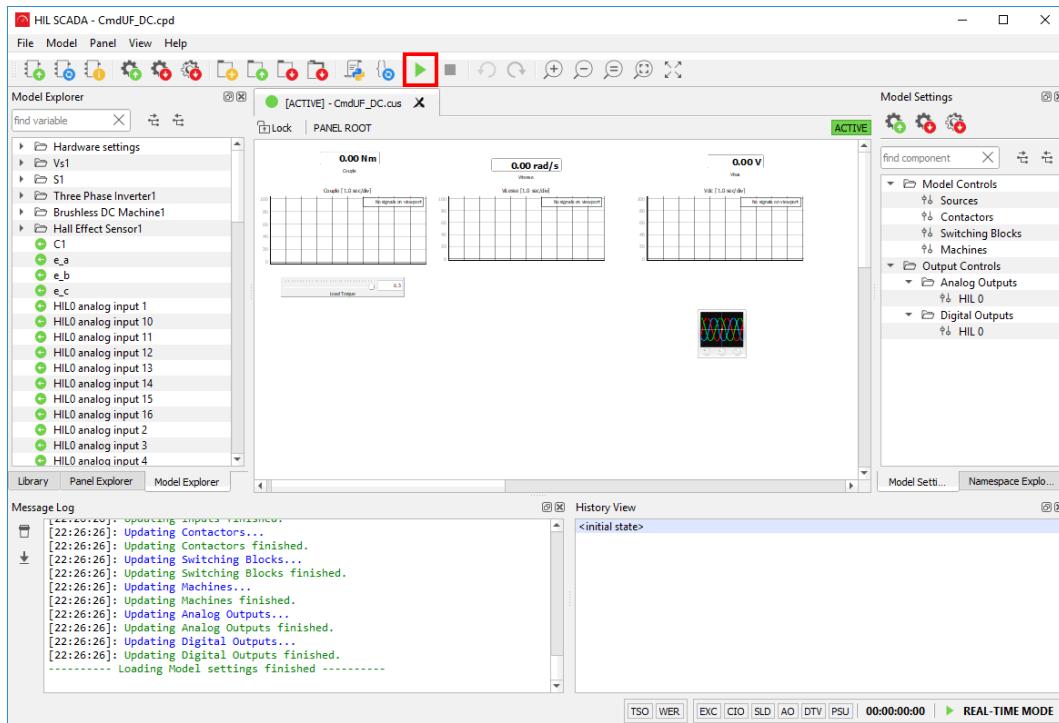


Figure R.17 Lancer le simulateur

R.6 Envoie de la consigne à travers l'UART

La consigne du régulateur PI est envoyée via l'UART du module FPGA Poetic. Il est donc nécessaire de relier un ordinateur avec le module à l'aide d'un câble USB-C. L'UART communique à 9600 baud/s. Le tableau R.1 indique quel sont les caractères à envoyer pour changer la consigne. Ceci est un rappel du chapitre 4.7.

Commande	Données				Fin de transmission
S	2	7	3	0	e

Table R.1 Paquet pour définir une consigne de 2730

Bibliographie

- [1] Wikipedia. *Convertisseur analogique-numérique*. url : https://fr.wikipedia.org/wiki/Convertisseur_analogique-num%C3%A9rique.
- [2] Wikipedia. *Fréquence d'échantillonnage*. url : https://fr.wikipedia.org/wiki/Fr%C3%A9quence_d%27%C3%A9chantillonnage.
- [3] Wikipedia. *Quantification (signal)*. url : [https://fr.wikipedia.org/wiki/Quantification_\(signal\)](https://fr.wikipedia.org/wiki/Quantification_(signal)).
- [4] Texas Instrument. *TMS320F2837xS Microcontrollers*. url : https://www.ti.com/lit/ds/symlink/tms320f28377s.pdf?ts=1629364755927&ref_url=https%253A%252F%252Fwww.ti.com%252Fstore%252Fti%252Fen%252Fp%252Fproduct%252F%253Fp%253DTMS320F28377SPTPT.
- [5] Wikipédia. *Théorème d'échantillonnage*. url : https://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_d%C3%A9chantillonnage.
- [6] Texas Instrument. *Choose the right A/D converter for your application*. url : <https://www.ti.com/europe/downloads/Choose%20the%20right%20data%20converter%20for%20your%20application.pdf>.
- [7] Texas Instrument. *ADS7886 12-Bit, 1-MSPS, Micro-Power, Miniature SAR Analog-to-Digital Converters*. url : <https://www.ti.com/lit/ds/symlink/ads7886.pdf?ts=1629335775516>.
- [8] Xilinx. *Spartan-6 FPGA Data Sheet : DC and Switching Characteristics*. url : https://www.xilinx.com/support/documentation/data_sheets/ds162.pdf.
- [9] Xilinx. *Product table and selection guide*. url : <https://www.xilinx.com/support/documentation/selection-guides/cost-optimized-product-selection-guide.pdf#S6>.
- [10] Xilinx. *Spartan-6 FPGA Packaging and Pinouts*. url : https://www.xilinx.com/support/documentation/user_guides/ug385.pdf.
- [11] Xilinx. *Xilinx UG393 Spartan-6 FPGA PCB Design Guide*. url : https://www.xilinx.com/support/documentation/user_guides/ug393.pdf.
- [12] Xilinx. *Spartan-6 FPGA Configuration User Guide*. url : https://www.xilinx.com/support/documentation/user_guides/ug380.pdf.
- [13] Texas Instrument. *Introduction to USB-C*. url : https://cdn.sparkfun.com/assets/e/b/4/f/7/USB-C_Datasheet.pdf.
- [14] Ricoh. *High Efficiency Small Packaged Step-up DC/DC Converter*. url : <https://www.mouser.com/datasheet/2/792/rp402-ea-911532.pdf>.
- [15] ANALOG DEVICES. *LT1461 : Micropower Precision Low Dropout Series Voltage Reference Family*. url : <https://www.mouser.ch/datasheet/2/609/LT1461-1267485.pdf>.

Bibliographie

- [16] Texas Instrument. *DAC124S085 12-Bit Micro Power Quad Digital-to-Analog Converter With Rail-to-Rail Output*. url : https://www.ti.com/lit/ds/symlink/dac124s085.pdf?ts=1629365652732&ref_url=https%253A%252F%252Fwww.google.com%252F.
- [17] Texas Instrument. *DAC084S085 8-Bit Micropower QUAD Digital-to-Analog Converter With Rail-to-Rail Output*. url : https://www.ti.com/lit/ds/symlink/dac084s085.pdf?ts=1629365923399&ref_url=https%253A%252F%252Fwww.google.com%252F.
- [18] Digikey. *How to Power and Control Brushless DC Motors*. url : <https://www.digikey.com/en/articles/how-to-power-and-control-brushless-dc-motors>.
- [19] Roboteq. *Field Oriented Control*. url : <https://www.roboteq.com/technology/field-oriented-control>.
- [20] ENSICAEN. *TP Sigma Delta*. url : https://foad.ensicaen.fr/pluginfile.php/5619/mod_resource/content/4/TP-3A-Micro-Simulation-Sigma-Delta.pdf.
- [21] JOSEPH MOERSCHELL. *Convertisseurs D/A et A/D, Module ELN2*. url : <https://drive.google.com/file/d/1E6oqCpNDniV8NJS8710HX1ba8YrKf8Le/view?usp=sharing>.
- [22] Digikey. *Principes de base analogiques - 3e partie : présentation et utilisation des CAN pipelines*. url : <https://www.digikey.ch/fr/articles/analog-basics-part-3-pipeline-adcs-and-how-to-use-them>.
- [23] Wikipédia. *Fond de panier*. url : https://fr.wikipedia.org/wiki/Fond_de_panier.
- [24] Wikipédia. *Low Voltage Differential Signaling*. url : https://fr.wikipedia.org/wiki/Low_Voltage_Differential_Signaling.
- [25] Wikipédia. *Modulation de largeur d'impulsion*. url : https://fr.wikipedia.org/wiki/Modulation_de_largeur_d%27impulsion.
- [26] Wikipédia. *Serial Peripheral Interface*. url : https://fr.wikipedia.org/wiki/Serial_Peripheral_Interface.
- [27] Wikipédia. *UART*. url : <https://fr.wikipedia.org/wiki/UART>.

Acronymes

ADC Convertisseur analogique vers digital (Analog to Digital Converter). [4](#), [7–9](#), [12](#), [25](#), [28](#), [33](#), [45](#)

BLDC Brushless DC. [xi](#), [44](#), [58](#)

DAC Convertisseur digital vers analogique (Digital to Analog Converter). [3](#), [7](#), [14](#), [16](#), [41](#), [43](#), [45](#), [58](#)

FIFO First In First Out. [46](#)

FPGA Réseau de portes programmables (Field Programmable Gate Arrays). [2](#), [3](#), [7–9](#), [12](#), [14–18](#), [20–25](#), [30](#), [32–34](#), [37](#), [38](#), [44](#), [45](#), [57](#)

GPIO General Purpose Input/Output. [5](#)

USB Universal Serial Bus. [3](#), [4](#), [7](#), [16](#), [23–26](#), [34](#)

VHDL VHSIC Hardware Description Language. [37–39](#), [41](#), [45](#), [47](#), [49](#)

Glossaire

fond de panier Selon Wikipédia : «*Un fond de panier (en anglais backplane) est une carte ou un bâti sur laquelle plusieurs connecteurs sont disponibles pour la connexion d'autres cartes à un appareil.*»[23]. [1](#), [4](#)

LVDS Selon Wikipédia : «*Low Voltage Differential Signaling (qu'on pourrait traduire mot à mot par « transmission différentielle basse-tension »), abrégé en LVDS, est une norme de transmission de signaux électriques à une fréquence élevée (typiquement plusieurs centaines de mégahertz) sur une ligne symétrique, de type transmission différentielle.*»[24]. [3](#), [16](#), [24](#), [33](#)

PWM Selon Wikipédia : «*La modulation de largeur d'impulsions (MLI ; en anglais : Pulse Width Modulation, soit PWM), est une technique couramment utilisée pour synthétiser des signaux pseudo analogiques à l'aide de circuits à fonctionnement tout ou rien, ou plus généralement à états discrets.*»[25]. [4](#), [5](#), [7](#), [16](#), [37](#), [44](#), [48–50](#), [53](#), [55](#), [56](#)

SPI Selon Wikipédia : «*Une liaison SPI (pour Serial Peripheral Interface) est un bus de données série synchrone baptisé ainsi par Motorola, au milieu des années 1980.*»... «*Les circuits communiquent selon un schéma maître-esclave, où le maître contrôle la communication. Plusieurs esclaves peuvent coexister sur un même bus, dans ce cas, la sélection du destinataire se fait par une ligne dédiée entre le maître et l'esclave appelée « Slave Select (SS) ».*»[26]. [3](#), [16](#), [18](#), [20](#), [24](#), [33](#)

UART Selon Wikipédia : «*Un UART, pour Universal Asynchronous Receiver Transmitter, est un émetteur-récepteur asynchrone universel.*»[27]. [7](#), [23](#), [24](#), [46](#), [48](#), [59](#)

Glossaire
