



Filière Systèmes Industriels

Orientation Infotronics

TRAVAIL DE BACHELOR

DIPLÔME 2023

Grobéty Christophe

High Speed Data Logger

Professor
Alexandra Andersson

Expert
Mario Clausen

Submission date of the report
18.08.2023



Filière / Studiengang SYND	Année académique / <i>Studienjahr</i> 2022-23	No TB / Nr. BA IT/2023/83
Mandant / <i>Auftraggeber</i>	<p><input type="checkbox"/> HES—SO Valais <input checked="" type="checkbox"/> Industrie <input type="checkbox"/> Etablissement partenaire <i>Partnerinstitution</i></p> <p>Etudiant / Student Christophe Grobety</p> <p>Professeur / Dozent Alexandra Andersson</p>	Lieu d'exécution / <i>Ausführungs</i> ort
		<p><input checked="" type="checkbox"/> HES—SO Valais <input type="checkbox"/> Industrie <input type="checkbox"/> Etablissement partenaire <i>Partnerinstitution</i></p>
Travail confidentiel / <i>vertrauliche Arbeit</i>	<p><input type="checkbox"/> oui / ja <input checked="" type="checkbox"/> non / nein</p>	Expert / <i>Experte</i> (<i>données complètes</i>) Mario Clausen , mario.clausen@armasuisse.ch Armasuisse, Feuerwerkerstrasse 39, CH-3602 Thun

Titre / *Titel***High Speed Data Logger**Description / *Beschreibung*

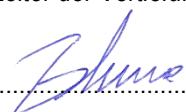
In order to record data from acceleration sensors and air pressure sensors during an explosion, a fast-sampling datalogger is needed. It needs to function with a battery and be able to interface with sensors that require a current source power supply to operate (IEPE Standard interface). Because of the fast sampling, an FPGA is responsible for interfacing the ADCs with fast external memory (FRAM). The FPGA is also responsible for generating the required timing signals for the acquisition and for the data transfer. A microprocessor is used for setup and control of various parts of the electronics, as well as for providing data readout via USB and user configuration options. If time permits, the analogue interface for the sensors including current sources need also to be designed, as well as the power system to generate required voltages from the battery.

Objectifs / *Ziele*

- Receive the newly built PCB and test all the subsystems: Analog front end, FPGA and microprocessor.
- Write the code for the FPGA to generate the various timing signals and signal muxing required
- Write the code for the microprocessor to handle acquisition setup, waveform readout and user interactions
- Test the implementation by sampling known signals and verify their correctness
- Optional goal: Design, test and integrate the analogue interface for the sensors
- Optional goal: Design, test and integrate the power module with its batteries and various DC/DC converters.

Signature ou visa / *Unterschrift oder Visum*

Responsable de l'orientation /
Leiter der Vertiefungsrichtung:



¹ Etudiant / Student:

Délais / *Termine*

Attribution du thème / *Ausgabe des Auftrags:*
15.05.2023

Présentation intermédiaire / *Zwischenpräsentation:*
19 - 20.06.2023

Remise du rapport final / *Abgabe des Schlussberichts:*
18.08.2023, 12:00

Expositions / *Ausstellungen der Diplomarbeiten:*
25.08.2023 – HEI
28.08.2023 – Monthey
31.08.2023 – Visp

Défense orale / *Mündliche Verfechtung:*
Semaine/Woche (04-07.09.2023)

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.



High Speed Data Logger



Graduate

Christophe Grobéty

Objectives

Création d'un système d'acquisition de données paramétrable, pouvant effectuer des mesures d'accélérations et de pression avec une fréquence de 32[MHz] avec un stockage sur carte SD.

Methods | Experiences | Results

Pour récolter le maximum d'informations provenant d'une explosion, un PCB a été développé avec 4 entrées de capteurs reliées à un contrôleur d'ADC (ANALOG DEVICES).

Afin de récolter ces datas, une FPGA dérive ses informations vers 4 FRAMs en fonction des canaux, qui vont ensuite pouvoir être lu par un stm32. Ce dernier va ensuite envoyer l'intégralité des mémoires FRAMs vers une carte SD dans des fichiers binaires pouvant être analyser via le programme MatLab.

Pour contrôler l'état du système durant son fonctionnement des leds de couleurs ont été mises en place.

Les résultats sont concluant et le système est fonctionnel, cependant la configuration de la FPGA s'efface durant la mise hors-tension et il y a encore quelques point d'améliorations possible.

Travail de diplôme | 2023 |



Filière

Système Industriel

Domaine d'application

Infotronics

Supervising professor

Andersson Alexandra

Alexendra.andersson@hevs.ch

Partner

Mario Clausen

Information about this report

Contact information

Author: Grob  t   Christophe
Bachelor Student
HES-SO//Valais Wallis
Switzerland
Email: chris_grobety@hotmail.com

Declaration of honor

I, undersigned, Grob  t   Christophe, hereby declare that the work submitted is the result of a personal work. I certify that I have not resorted to plagiarism or other forms of fraud. All sources of information used and the author quotes were clearly mentioned.

Place, date: August 17, 2023

Signature: _____

Table des Matières

Table des matières	vi
Tables des figures	vii
Tables des tableaux	viii
1 Introduction	1
1.1 Contexte	1
1.2 Objectifs	1
1.3 Cahier des charges	2
1.4 Lien GitHub	2
2 Analyse	3
2.1 Logiciels utilisés	4
2.2 Matériels utilisés	4
3 Développement	15
3.1 Programmation VHDL	15
3.2 Programmation STM32	20
3.3 Carte SD	31
4 Tests, résultats et corrections	33
4.1 SIMULATION VHDL	33
4.2 TESTS REELS	36
4.3 Corrections Pcb	41
4.4 Conclusion & améliorations	42
5 Conclusions	45
5.1 Résumé du projet	45
5.2 Comparaison avec les objectifs initiaux	45
5.3 Difficultés rencontrés	46
5.4 Remerciement	46
Bibliography	47
A Annexes	49
A.1 Schéma Altium	49
A.2 Code STM32	60
A.3 Bloc FPGA	93
A.4 Code FPGA	96

A.5	Bloc de test FPGA	108
A.6	Code du Test FPGA	110
A.7	Script Matlab	116
B	Durabilité	121
Acronyms		123

List of Figures

2.1	Schéma simplifié	3
2.2	WREN commande (Write setUp) [1]	5
2.3	Write commande [1]	5
2.4	READ commande [1]	6
2.5	Timing Diagrams [2]	7
2.6	Bits de configuration2 [2]	8
2.7	PROGRAMMATION CRAM	9
2.8	programmation SPI	10
2.9	PROGRAMMATION NVCM	11
2.10	shema fpga to flash [6]	12
2.11	schéma de configuration CRAM/NVCM	13
2.12	schéma de configuration SPI	13
2.13	schéma de connexion PINs SPI	14
3.1	FPGA BLOCKS SIMPLIFIE	16
3.2	ADC TO FRAM sous-bloc	17
3.3	ADC TO FRAM shema	17
3.4	service en fonction du Selector	18
3.5	configuration PINs	20
3.6	configuration NVIC	21
3.7	configuration SPI	22
3.8	CPOL CPHA	24
3.9	configuration Timer16	25
3.10	configuration SDMMC1	26
3.11	diagramme de classe	27
3.12	diagramme de séquence	29
3.13	Valeurs d'initialisation	31
3.14	Exemple de fichiers dans la carte SD	32
4.1	Simulation d'initialisation	33
4.2	Simulation d'écriture	34
4.3	Simulation de Trigger	34
4.4	Fin de simulation	35
4.5	Signals clock 1 & 32[MHz]	37

4.6	delta de temps	38
4.7	sinus à 10 [Hz] avec perturbation	39
4.8	perturbation du signal	40
4.9	sinus à 10 [Hz] sans perturbation	40
4.10	sinus à 10 [Hz]-précision	41
B.1	Les 17 objectifs de développement durable [14]	121

List of Tables

2.1	Liste des programmes	4
-----	--------------------------------	---

1 | Introduction

1.1 Contexte

Ce projet a été commandité par Armasuisse afin qu'ils puissent procéder à des tests plus poussés lors de l'explosion d'un véhicule par des mines ou autres types d'explosifs.

Le but principal de ce travail de Bachelor est de pouvoir enregistrer les valeurs d'accélération et de pression lors d'une explosion avec un enregistreur de données à échantillonneur rapide.

A cause de l'échantillonnage rapide, il est nécessaire de faire fonctionner le tout en tandem avec une FPGA et une mémoire externe rapide (FRAM).

La FPGA s'occupe aussi de générer les signaux de synchronisation pour l'acquisition et le transfert des données.

Le processeur est chargé de la configuration et du contrôle ainsi que de la lecture et l'écriture via la carte SD. Par manque de temps je n'ai pas pu implémenter la partie permettant de lire les données de la carte SD via le port USB-C présent sur le PCB.

1.2 Objectifs

Pour accomplir ce travail il a fallut le diviser en plusieurs tâches dont les principales sont :

- La réception du matériel et les tests de ces derniers afin de vérifier si le circuit est fonctionnel et les modifications à lui apporter le cas échéant.
- La configuration de la FPGA et de sa logique de fonctionnement pour la gestion de communication entre les différents éléments de la board.
- La configuration, la programmation et la routine d'interruption du micro Contrôleur.
- Les différentes phases de tests pour vérifier le bon fonctionnement de l'intégralité du circuit et de sa logique ainsi que de la réception des informations et leurs vérifications.
- Les différentes corrections à faire en fonction des résultats des tests jusqu'à obtenir une version fiable du système.

Ce travail est donc principalement orienté système embarqué et demande une bonne connaissance en programmation VHDL et en C. Ce rapport aura donc pour but d'expliquer le matériel et les logiciels utilisés ainsi que les phases de tests et de vérifications du bon fonctionnement de l'intégralité du système.

1.3 Cahier des charges

Les demandes techniques vis-à-vis du projet étaient donc :

- Une vitesse d'acquisition des données de 32 [Mhz]
- Écriture complète sur les FRAMs de 2^{19} Bytes de datas.
- 4 canaux de mesures pouvant être lus en simultanés.
- Envoi et stockage des mesures sur une carte SD.
- Affichage des données via MatLab.
- Boucle de programmation et affichage de l'état du programme via des LEDs.

1.4 Lien GitHub

Pour faciliter l'accès à mon travail pratique, j'ai créé un référentiel (repo) sur GitHub. Ce référentiel contient tous les fichiers, le code source et les ressources associés à mon projet. Vous pouvez consulter et télécharger ces informations en visitant le lien suivant : https://github.com/Drident/High_Speed_Data.git.

2 | Analyse

Le design général du fonctionnement de la board de la Figure 1 ci-dessous montre la communication d'une manière plus visible entre les différentes parties Hardware qui ont été imposés.

D'une manière simplifiée la board fonctionne comme suit: le STM32 initialise les FRAMs et l'ADC en passant par la FPGA en lui donnant les directives selon les valeurs comprises dans le fichier d'initialisation de la carte SD, puis l'ADC va communiquer avec les supports de mémoires en leur envoyant à chacune les valeurs d'un canal. Quand l'ADC a fini d'échantillonner, les FRAMs vont envoyer l'une après l'autre leurs valeurs au STM32 qui les transférera vers la carte SD. Quand l'utilisateur voudra lire ses valeurs, il pourra soit sortir la carte SD pour la transférer sur un pc, soit continuer la prise de mesures.

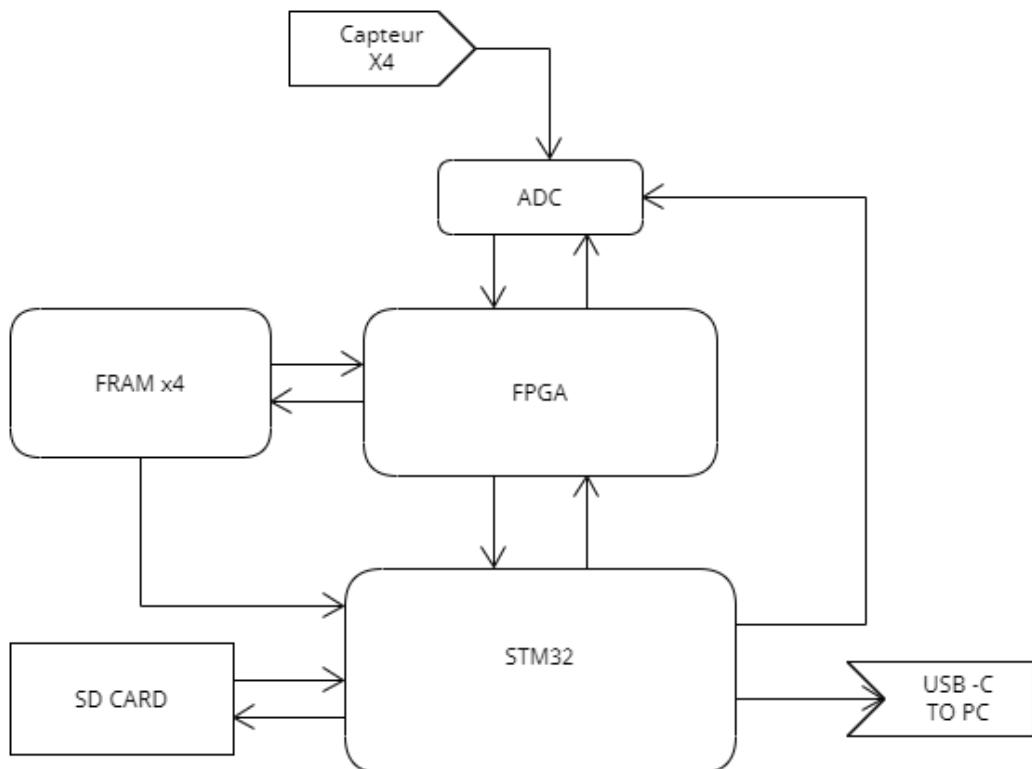


Figure 2.1 Schéma simplifié

On peut donc observer que le microControlleur communique de façon semi-directe avec la FRAM et l'ADC:

- **FRAM** : les valeurs d'initialisation données par le MOSI et sa clock passent par la FPGA tandis que les valeurs de sorties de la FRAM SDO vont directement sur le STM32.

Chapter 2. Analyse

- ADC : ce fonctionnement est presque l'inverse de celui avec la FRAM puisque les valeurs d'initialisation sont directement connectées au STM32 et celles de sortie passent par la FRAM. Cependant sa clock passe par la FPGA.

La partie importante à comprendre est que la FPGA possède sa propre clock et qu'elle reçoit en plus celle du STM32. Ce switch entre les clocks permet au STM32 de pouvoir passer en mode sleep quand l'ADC et les FRAMS sont entrain de communiquer, puisqu'à ce moment-là, ils utiliseront la clock de la FPGA.

2.1 Logiciels utilisés

Pour réussir à manipuler les composants HardWare, différents logiciels ont été utilisés afin de pouvoir implémenter les différentes logic SoftWare:

Programmes	Fonction
STM32CubelDE 1.9.0	Configuration et programmation du μ contrôleur
HDL Designer	Programmation VHDL de la FPGA
ICECube2	Adaptation du programme VHDL pour le transfert
Lattice Diamond	Transférer le design VHDL sur la FPGA
Altium Designer	Analyse et modification du circuit
MatLab	Analyse des résultats obtenu

Table 2.1 Liste des programmes

2.2 Matériels utilisés

Pour ce projet, le seul réel support utilisé était une board déjà préconçue avant le début du travail par madame Andersson. Dans cette partie, il s'agira de détailler les principaux composants qu'il a fallu tester, initialiser et programmer sur la board, à savoir :

- L'[ADC](#) DEVICES
- Les [FRAM](#)s
- Le [STM32](#)
- La [FPGA](#)

2.2.1 Memory FRAM (MB85RS4MT)

Pour enregistrer rapidement les valeurs envoyées par l'adc, on emploie une FRAM par capteur. Celles-ci ont une capacité de 2 puissance 19 bytes. Communicant par [SPI](#), elles ont besoin d'une alimentation entre 1.8 et 3.6[v] ainsi que d'une fréquence plafonnée à 40[Mhz]. Pour ce projet il a été décidé que son alimentation serait celle de la board (3.3[v]) et que sa fréquence serait de 32[Mhz].

Il existe de nombreuses fonctions pour ce chip, mais seulement 3 d'entre elles seront utilisées durant ce projet.

2.2 Matériaux utilisés

Afin de pouvoir écrire sur cette carte, il est nécessaire de l'initialiser en lui envoyant une clock et une commande opcode via le SPI (voir Figure ci-dessous) :

• WREN

The WREN command sets WEL (Write Enable Latch) . WEL has to be set with the WREN command before writing operation (WRSR command and WRITE command) .

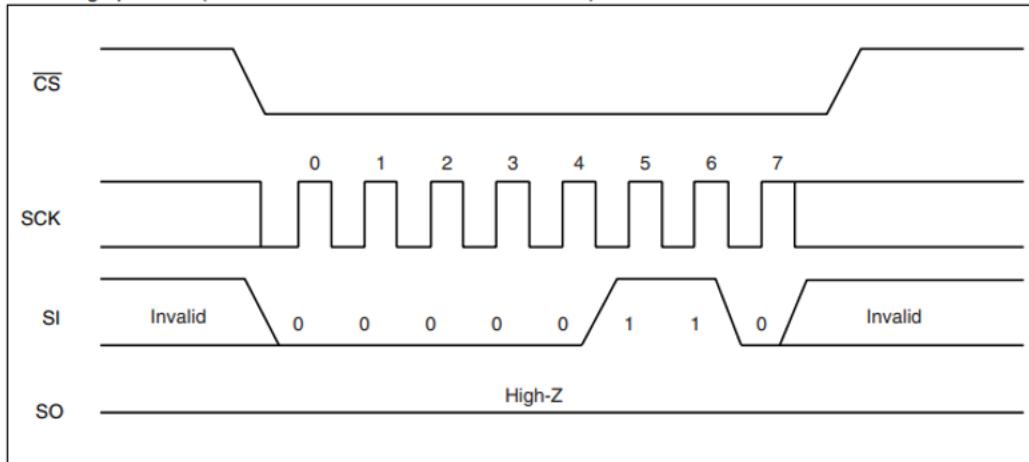


Figure 2.2 WREN commande (Write setUp) [1]

Il est important de noter que pour pouvoir écrire sur les registres de la FRAM, il faut impérativement que son signal d'entrée du NCS soit à '0' afin d'activer le chip et que celui-ci récupère les valeurs de l'opCode envoyées.

Une fois l'initialisation accomplie, il est possible de lui envoyer une commande WRITE afin de placer le chip en mode écriture en lui donnant l'adresse du début d'acquisition :

• WRITE

The WRITE command writes data to FRAM memory cell array. WRITE op-code, arbitrary 24 bits of address and 8 bits of writing data are input to SI. The 5-bit upper address bit is invalid. When 8 bits of writing data is input, data is written to FRAM memory cell array. Risen CS will terminate the WRITE command, but if you continue sending the writing data for 8 bits each before CS rising, it is possible to continue writing with automatic address increment. When it reaches the most significant address, it rolls over to the starting address, and writing cycle can be continued infinitely.

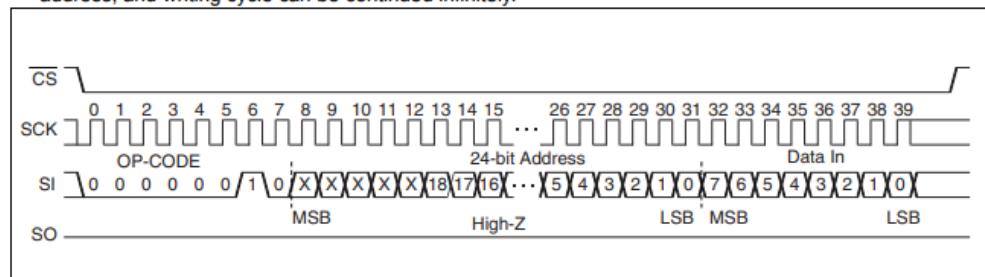


Figure 2.3 Write commande [1]

Chapter 2. Analyse

Cette opération est la plus complexe à réaliser, car ces premiers 32 MSB d'opcode et d'adressage sont gérés par la clock du STM32 tandis que les LSB restants des datas doivent utiliser la clock de la FPGA. Afin de pouvoir écrire continuellement sur la FRAM sans avoir à réécrire en permanence la commande complète, il suffit simplement de laisser le NCS à '0' durant la totalité de l'acquisition de l'ADC. Le chip va simplement incrémenter la valeur de l'adresse chaque 8 bits envoyés.

La dernière commande employée est celle du READ qui fonctionne comme pour celle présentée avant :

• READ

The READ command reads FRAM memory cell array data. Arbitrary 24 bits address and op-code of READ are input to SI. The 5-bit upper address bit is invalid. Then, 8-cycle clock is input to SCK. SO is output synchronously to the falling edge of SCK. While reading, the SI value is invalid. When CS is risen, the READ command is completed, but keeps on reading with automatic address increment which is enabled by continuously sending clocks to SCK in unit of 8 cycles before CS rising. When it reaches the most significant address, it rolls over to the starting address, and reading cycle keeps on infinitely.

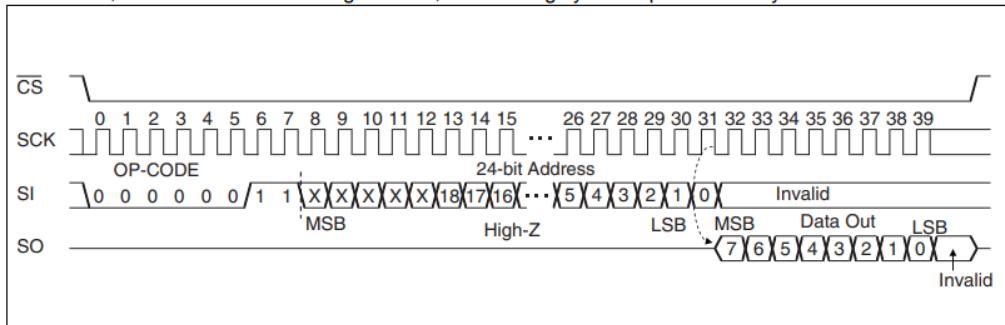


Figure 2.4 READ commande [1]

La différence notable avec la commande WRITE est que lorsque les bits de l'opCode et de l'adressage ont été envoyés, la FRAM va commencer à transmettre ces valeurs en sortie en fonction de la clock jusqu'à ce que le NCS remonte à '1'.

2.2.2 ADC DEVICES (AD7380-4)

Cette pièce permet de pouvoir récupérer les valeurs des différents signaux qui seront branchés sur le PCB. Elle transmet ces informations sur 4 sorties. Son fonctionnement Software n'est pas spécialement différent de celle de la FRAM présentée plus haut: elle s'initialise en communiquant avec un SPI et une clock ainsi qu'un NCS afin d'être active ou non. Cependant, contrairement à la FRAM elle communique par paquets de 16 bits et réclame une attention quant au temps de traitement des informations reçues ainsi qu'aux délais entre signaux :

2.2 Matériels utilisés

Timing Diagrams

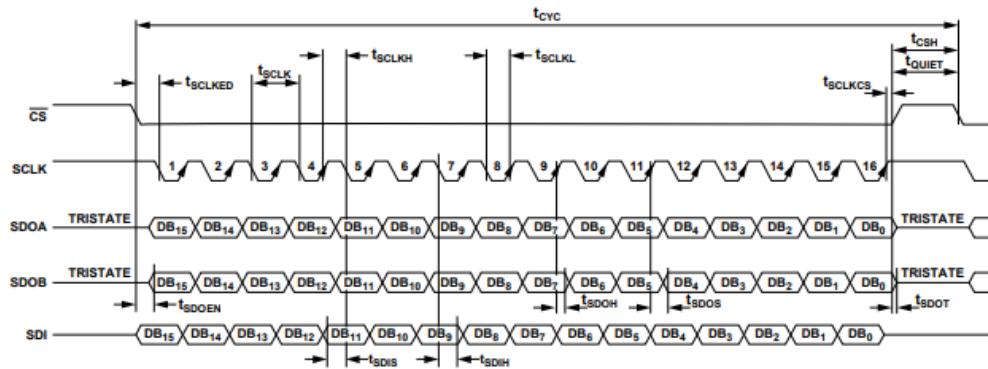


Figure 2.5 Timing Diagrams [2]

Sur cette image, on peut observer plusieurs laps de temps entre les signaux, la datasheet[2] précise que ce sont là des durées minimales à respecter pour le bon fonctionnement du chip. Les principaux temps qui nous intéressent sont :

- Tsclked -> Espace entre le NCS et le falling edge de la clock : Min. 5 [ns]
- Tsclk -> Durée d'un coup de clock (fréquence) : Min. 12.5 [ns] (Max. 80 [Mhz])
- Tsclkcs -> Espace entre le dernier coup de clock et le rising edge NCS : Min. 0 [ns]
- Tquiet -> temps minimal du NCS à l'état haut avant la prochaine aquisition : Min. 20 [ns]

Cette partie sera importante durant le développement, car elle indique la marche à suivre pour réclamer à l'ADC d'envoyer ses valeurs ainsi que l'utilisation de sa clock, qui consiste à rester à l'état haut quand le NCS l'est aussi.

Pour permettre à l'ADC d'envoyer ses datas, il faut le configurer afin d'initialiser ses registres à l'écriture. Parmi les nombreuses possibilités, la solution décrite dans la datasheet CONFIGURATION2 a été sélectionnée car c'est celle qui permet la conversion des résultats reçus par les 4 canaux afin d'envoyer ces valeurs sur les sorties. Les 16bits à configurer sont définis comme suit :

Chapter 2. Analyse

CONFIGURATION 2 REGISTER

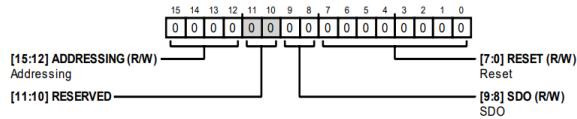


Table 19. Bit Descriptions for Configuration 2 Register

Bits	Bit Name	Description	Reset	Access
[15:12]	Addressing	Addressing Bits. These bits define the address of the relevant register. See the Addressing Registers section for further details.	0x0	R/W
[11:10]	Reserved	Reserved.	0x0	R
[9:8]	SDO	SDO. Conversion Results Serial Data Output 00: 2-wire output. Conversion data are output on both SDOA and SDOB. 01: 1-wire output. Conversion data are output on SDOA only. 10: 4-wire. Conversion data are output on SDOA, SDOB, SDOC, and SDOD/ALERT. 11: 1-wire. Conversion data are output on SDOA, only.	0x0	R/W
[7:0]	Reset	Reset.	0x0	R/W
		0x3C: performs a soft reset. Refreshes some blocks, register contents remain unchanged. Clears alert indication register and flushes any oversampling stored variables or active state machine.		
		0xFF: performs a hard reset. Resets all possible blocks in the device. Registers contents are set to defaults. All other values are ignored.		

Figure 2.6 Bits de configuration2 [2]

Pour remplir les 4 MSB, le premier doit être à '1' afin de permettre au registre de lire et d'écrire tandis que les 3 derniers sont à "010" ce qui correspond à l'adresse du registre en question. Les autres bits ne sont pas important quant à leurs valeurs à part 2 exceptions comme indiqué dans la partie "Reset" du tableau ci-dessus.

2.2.3 Micro Controller (STM32L496RGT6)

Le uContrôleur utilisé permet de communiquer les différentes fonctions d'utilisation à la FPGA et ses périphériques détaillés plus haut.

Ses principales fonctions utilisées pour ce projet seront donc de pouvoir lire et écrire sur une carte SD ainsi que de faire tourner le programme en boucle jusqu'à la lecture des données.

Il possède 64 pins pour un coeur Arm 32-bit-M4 CPU et demande une alimentation comprise entre 1.7 et 3.6[v] pouvant utiliser une fréquence allant jusqu'à 80 [Mhz].

Afin de programmer ce contrôleur, j'utilise un câble STLINK-V3SET via le programme STM32CUBEIDE. La datasheet de ce microContrôleur est disponible dans la bibliographie [3]

2.2.4 FPGA (ICE40 Ultra)

Ce module est une pièce fournie par l'entreprise Lattice de 48 pins. Elle permet donc de réceptionner et d'envoyer l'intégralité des signaux lui étant donné afin de permettre une communication entre les différentes pièces de la carte. Pour concevoir le fichier bin via le code VHDL, il faut utiliser le programme IceCube2 qui permettra aussi de pouvoir initialiser les pins de la FPGA ainsi que de leur placer ou non des "Pull up/down" en fonction des besoins dans le fichier "bachelor.pcf" sous l'onglet "Constraint Files" dans le programme (voir datasheet [4]). Sa configuration se fait via le programme Diamond Programmer et nécessite un câble LATTICE. Elle peut-être initialisée avec un fichier.bin qui correspond au fichier créé via le programme IceCube2 de 3 manières différentes :

CRAM

De cette façon la configuration se transfert sur la RAM de la FPGA et est réinitialisée dès que la board est éteinte.

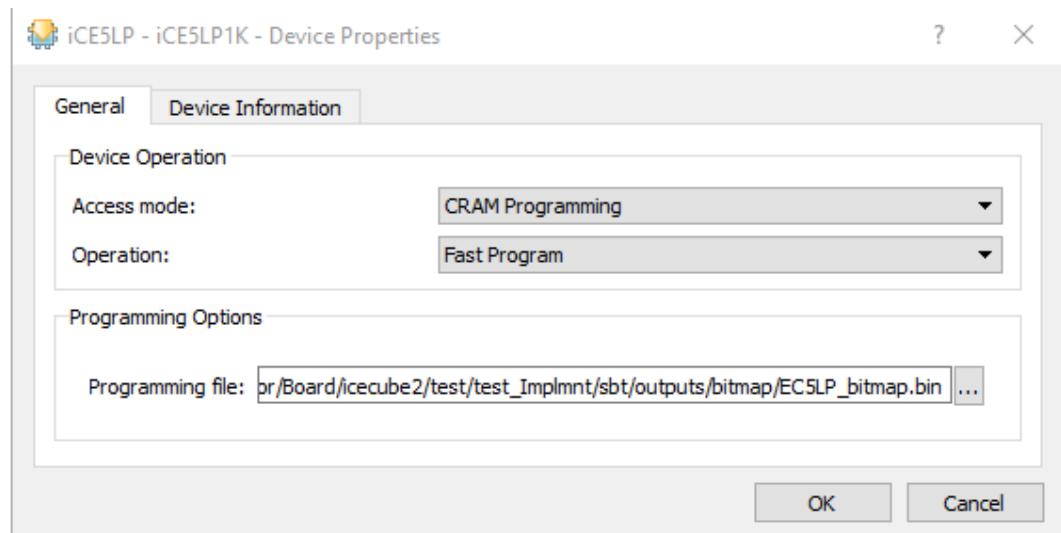


Figure 2.7 PROGRAMMATION CRAM

La figure représente l'onglet de configuration CRAM. Pour l'implémenter il faut sélectionner l'opération "Program, Verify".

Chapter 2. Analyse

SPI

La configuration se fait sur la mémoire **FLASH** qui lors de la mise en tension va communiquer ses informations à la FPGA.

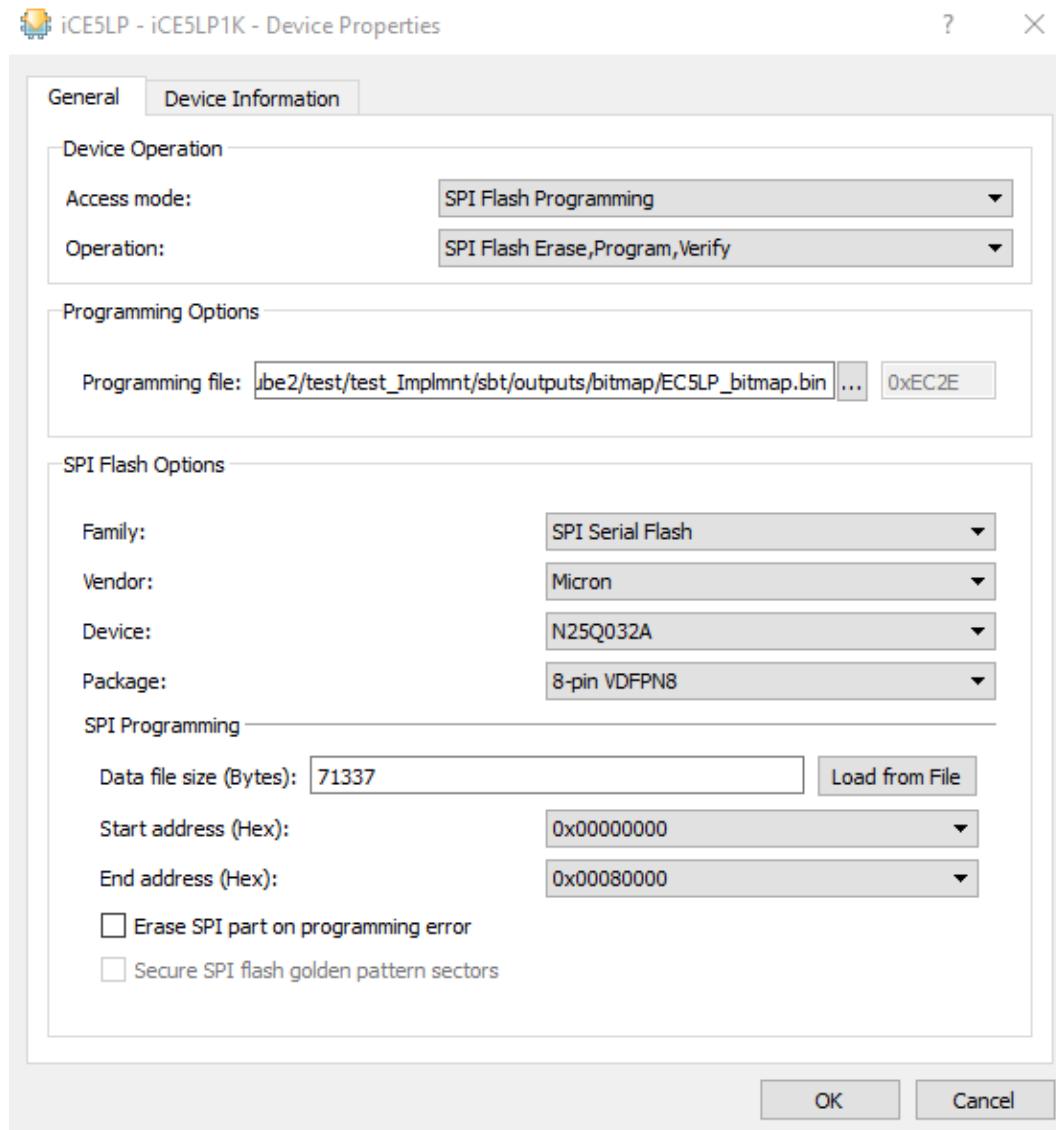


Figure 2.8 programmation SPI

Cette configuration demande le nom de constructeur, le nombre de pins de la mémoire flash et son nombre de bytes (qui se remplit automatiquement en fonction du fichier sélectionné). On peut aussi décider de l'adresse de début et de fin de configuration sur la mémoire flash.

L'opération à sélectionner sera le "SPI Flash Erase, Program, Verify" afin de supprimer la dernière configuration présente sur la flash.

NVCM

Ce mode est risqué, car il n'est possible de l'utiliser qu'une seule fois. Après quoi la FPGA bloque l'accès à cette partie de la mémoire qui ne peut plus être re-conditionnée.

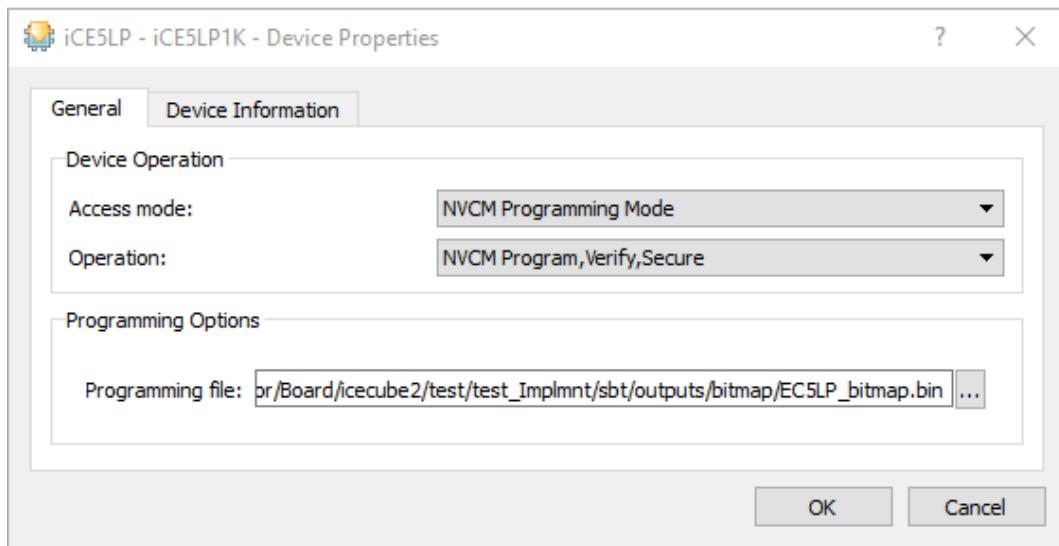


Figure 2.9 PROGRAMMATION NVCM

Il existe de nombreuses opérations disponibles pour chacune de ces configurations, mais elle concerne avant tout des possibilités d'informations sur l'état actuel de la flash ou de la FPGA.

2.2.5 Échec de programmation de la flash

Malheureusement je n'ai pas pu utiliser le mode SPI afin de conserver les valeurs d'initialisation de la FPGA, même après hors-tension. L'accès à la mémoire flash[5] fonctionne depuis le programme et la FPGA lui envoie une clock afin de réceptionner les valeurs. Mais elle ne lui envoie pas le signal nécessaire afin de la piloter pour qu'elle puisse envoyer ces valeurs :

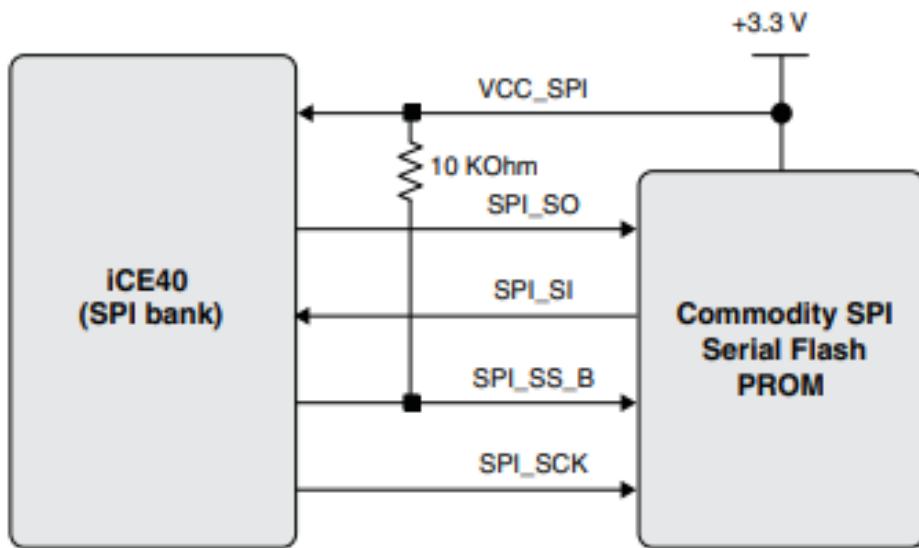


Figure 2.10 shema fpga to flash [6]

Comme vous pouvez le voir sur la figure tirée de la datasheet, la FPGA va envoyer une clock en direction de la flash une fois la mise sous tension effectuée. Elle doit lui envoyer un signal de commande (SPI_SS_B), puis l'échange est sensé commencer entre les deux appareils, mais il ne se passe rien. Les 2 autres modes de transfert présentés plus haut fonctionnent donc elle n'est pas bloquée dans un mode. Le circuit présent sur la board a été contrôlé plusieurs fois et ne présente aucune différence avec ce qui est réclamé dans les datasheets de la FPGA [4] [6] et de la mémoire flash.

2.2.6 Switch sur la board

La communication à transmettre entre les PINs de la broche permettant la configuration du programme sur le pc vers la FPGA et la FPGA elle-même ne sont pas les mêmes si l'on veut utiliser le mode SPI ou le mode CRAM et NVCM, comme démontré par le schéma suivant:

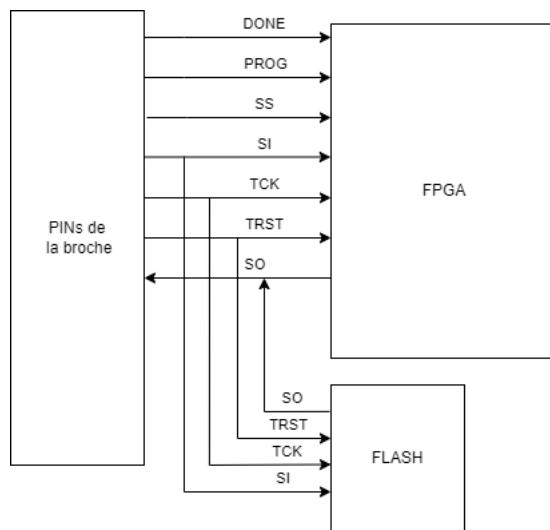


Figure 2.11 schéma de configuration CRAM/NVCM

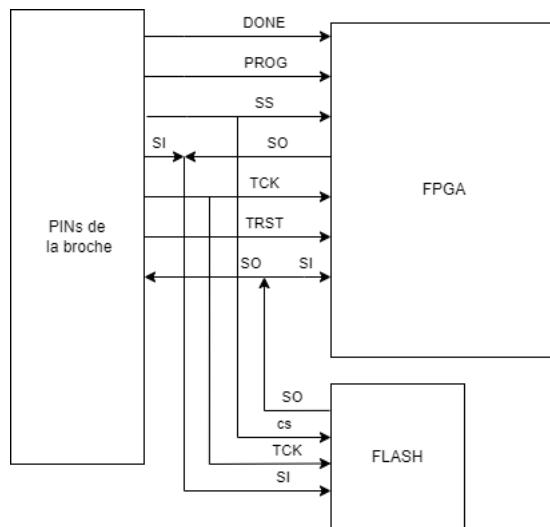


Figure 2.12 schéma de configuration SPI

Sur l'image 2.11, on peut constater que la FPGA partage ses signaux avec ceux de la flash mais qu'ils n'ont aucun moyen de communication entre eux.

Tandis que sur la 2ème image 2.12, on voit que la FPGA et la flash ont une communication SPI permettant à la FPGA de demander à la flash de lui envoyer sa configuration en pilotant sa pin cs.

Chapter 2. Analyse

Sur la board afin de switcher entre ces 2 méthodes, on a installé 6 pins afin de pouvoir changer de mode en fonction de la connexion effectuée entre elles:

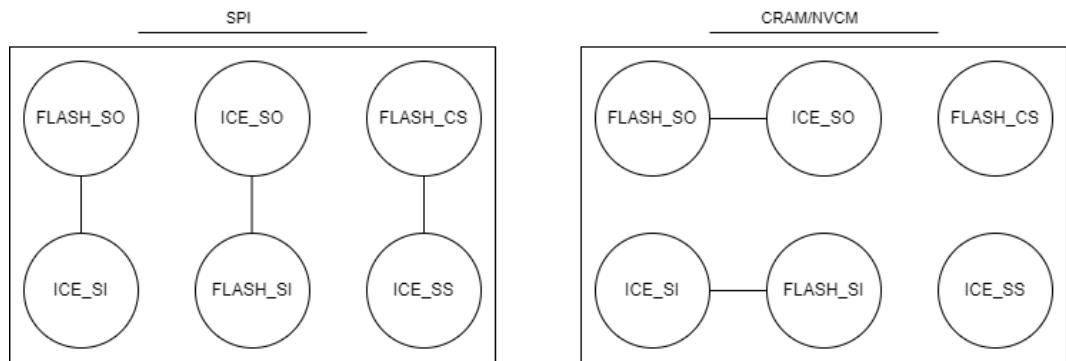


Figure 2.13 schéma de connexion PINs SPI

Ces pins se situent au-dessus du support de carte SD et sont disponibles visuellement dans l'annexe.

3 | Développement

Le développement de ce projet dans sa partie programmation, s'est principalement effectué sur le STM32 et la FPGA étant donné que ce sont les seules parties programmables en soit. Ce chapitre portera donc sur la logique de programmation de ces différentes parties

3.1 Programmation VHDL

La programmation de la partie FPGA s'est faite en la construction de différents blocs permettant de pouvoir traiter les multiples cas nécessaires au bon fonctionnement du projet. Une multitude de signaux entrent et sortent de cette FPGA, les principaux nous intéressant sont :

Entrant

- **Acquisition pretrig:** signal permettant au STM32 de préparer la FPGA à recevoir des informations en SPI pour le pretrigger et correspondant au pourcentage de mémoire censé être écrite une fois le Trigger capté par la FPGA. Ce signal est en trop car ce rôle peut être endossé par le bus "fpga m"
- **Acquisition Trigg:** signal indiquant la réception d'une valeur de tension de canal suffisante vis-vis de la configuration du STM32
- **Adc SDO:** bus de 4 signaux correspondants aux 4 canaux disponibles et envoyant leurs valeurs en SPI.
- **fpga m:** bus de 4 signaux dont les valeurs permettent de déterminer le service désiré par la FPGA. Cela sert de "[SELECTOR](#)". Pratiquement chaque bloc possède ce signal car il permet de pouvoir réguler la valeur de leurs signaux de sortie.
- **fpga MOSI:** (Master Out Slave In) signal permettant la communication SPI entre la FPGA et le STM32. Ce signal est accompagné par une clock.
- **fpga sck:** c'est la clock en provenance du STM32. Quand elle n'est pas active, son signal reste à '1'.

Sortant

- **adc sclk:** clock envoyée vers l'adc device
- **adc NCS:** signal pilotant l'adc device
- **fram sclk:** clock envoyée vers les FRAMS
- **fram NCS:** signal pilotant les FRAMs

- **fram SDI:** signal communiquant les valeurs à écrire ou des services à demander aux FRAMs
- **fpga MISO:** (Master In Slave Out) signal communiquant en SPI l'adresse mémoire des FRAMS où commencer à lire au STM32.

L'adc device ne possède pas de signal de communication de la part de la FPGA car il possède un lien direct avec le STM32 pour recevoir des datas.

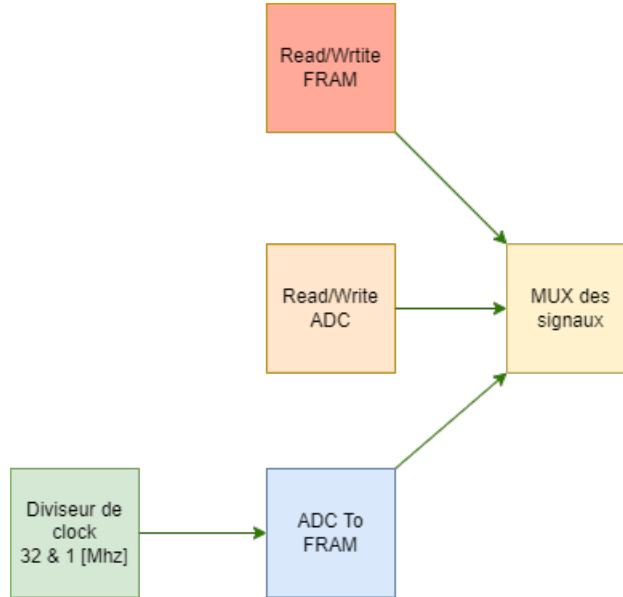


Figure 3.1 FPGA BLOCKS SIMPLIFIE

READ/WRITE FRAM

Ce bloc permet donc de recevoir une clock du STM32 et s'occupe de transférer les valeurs d'initialisations vers les FRAMs et de gérer son "chip-select".

READ/WRITE ADC

Identique à celui du dessus mais pour l'adc. Si j'ai créer 2 blocs différents pour cette opération c'était pour simplifier le traitement des erreurs et de simulations.

Diviseur de clock

Afin de permettre au bloc de l'adc vers les FRAMs d'obtenir une acquisition chaque 1 [Mhz], j'ai conçu 2 blocs divisant la fréquence de base vers 1 et 32[Mhz].

MUX des signaux

C'est le bloc central. La totalité des signaux sortant de la FPGA passent par lui afin d'éviter d'écrire avec 2 signaux en même temps sur une sortie. Il gère le mux des signaux en fonction de la valeur présente sur le selector. De ce fait, il y a un problème durant la modification du selector car pour le mux cela ne correspondra à aucune service connu. Il passera les sorties à leur état par défaut. En l'état, cela ne produit aucune erreur mais s'il doit y avoir une modification sur le code de base cela pourrait amener à de potentielles erreurs.

La partie "ADC TO FRAM" est constituée de plusieurs sous-blocs permettant diverses opérations:

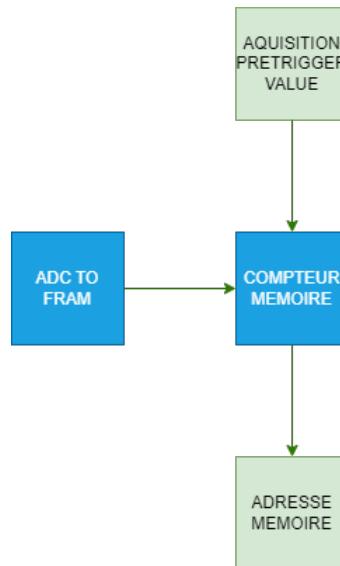


Figure 3.2 ADC TO FRAM sous-bloc

Ces blocs sont l'ensemble permettant la réception de la valeur du preTrigger envoyé par le STM32, la gestion des clocks et NCS des FRAMS avec l'ADC/FRAMs et la communication au STM32 du dernier emplacement mémoire utilisé.

ADC TO FRAM

C'est dans ce bloc que j'effectue la réception des mesures de l'adc vers les FRAMS.

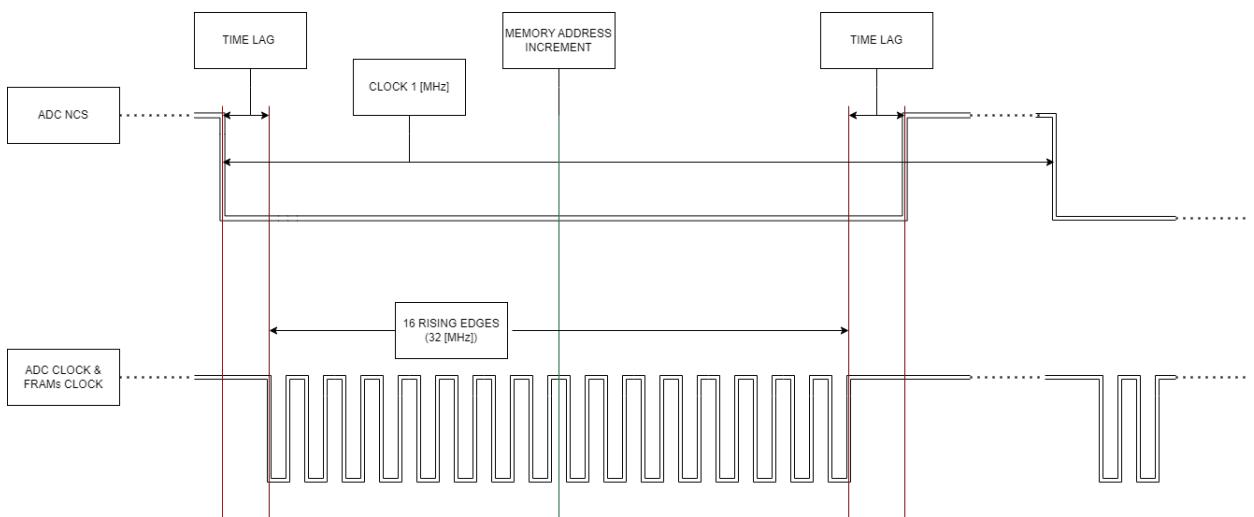


Figure 3.3 ADC TO FRAM shema

Pour ce faire, une clock A de 32 [Mhz] est envoyée vers l'adc ainsi que le NCS de l'adc afin de les commander. Le NCS des frams étant de base à l'état actif bas après la commande d'écriture dans le bloc FRAM WRITE. Il existe également un

Chapter 3. Développement

système afin d'incrémenter un compteur à chaque 8 bits de données envoyées vers les FRAMs, permettant de savoir où se situe la dernière adresse mémoire employée. Le tout fonctionne sur une période de 1[MHz] avec une clock B durant laquelle on va effectuer 16 transfert de données via la clock A, suite à quoi, elle restera à l'état actif haut.

3.1.1 AQUISITION PRETRIGGER VALUE

Ce bloc sert donc à recevoir la valeur de pretrigger avant d'être envoyée dans le bloc COMPTEUR MEMOIRE.

3.1.2 COMPTEUR MEMOIRE

Le but ici est de réceptionner la valeur du bloc présente ci-dessus afin de calculer la valeur mémoire où il faudra s'arrêter ainsi que le signal d'incrémentations du bloc de l'ADC TO FRAM. Ce bloc va donc compter à chaque signal émis en boucle, commençant à chaque fois que la valeur maximum est atteinte. Une fois le signal trigger reçu, il va démarrer un 2ème compteur jusqu'à la valeur mémoire calculée. Une fois cette valeur atteinte, le bloc de transfert de données va s'arrêter et envoyer un signal "done" vers le STM32 afin de le "réveiller" et transmettre la valeur d'adresse mémoire vers le bloc ADRESSE MEMOIRE.

3.1.3 ADRESSE MEMOIRE

Le but de ce bloc est simplement de pouvoir revoir l'adresse mémoire de la dernière valeur reçue et de l'envoyer au STM32 quand il recevra la commande via le "Selector"

3.1.4 Fonctionnement générale

Une fois la mise sous tension établie et la configuration implémentée dans la RAM, tout le système tournera en fournissant diverses "services" en fonction de la valeur du "Selector"(fpga_m) :

M	FUNCTION	CLK	TAILLE
0001	/	/	/
0010	WRITE ADC	STM32	16
0011	NCS ADC ON	STM32	1
0100	READ FRAM A	STM32	32
0101	READ FRAM B	STM32	32
0110	READ FRAM C	STM32	32
0111	READ FRAM D	STM32	32
1000	WRITE ALL FRAM	STM32	32 + MEMORY
1010	ADC TO FRAM	FPGA	32+ INF. + PRETRIG
1011	MEMORY ADD	STM32	19
1110	/	/	/

Figure 3.4 service en fonction du Selector

3.1 Programmation VHDL

Une grande variété de services sont donc proposés ici, en fonction des blocs décrits plus haut.

Un autre service est disponible sans être sur le tableau, il s'agit du service permettant de récupérer la valeur de pretrigg qui comme décrit plus haut, dépend du signal de pretrigg.

3.2 Programmation STM32

Cette partie traitant de l'implémentation du programme en 'C' sur le STM32 c'est faite en 3 parties : la configuration via stmcube des diverses fonctionnalités du microcontrôleur (clock,pin,FATFS...) et la programmation des classes nécessaire à son bon fonctionnement. Pour m'aider à réaliser cette partie, j'ai eu recours à l'utilisation de la documentation du STM32[3], le site STMicroelectronics[7], chatgpt[8] et stack-Overflow[9]

3.2.1 Configuration du projet - PINs & Interrupts

Afin de correspondre aux directives de ce travail, il fallait initialiser le STM32 afin de définir et remplir les besoins des différentes phases du programme.

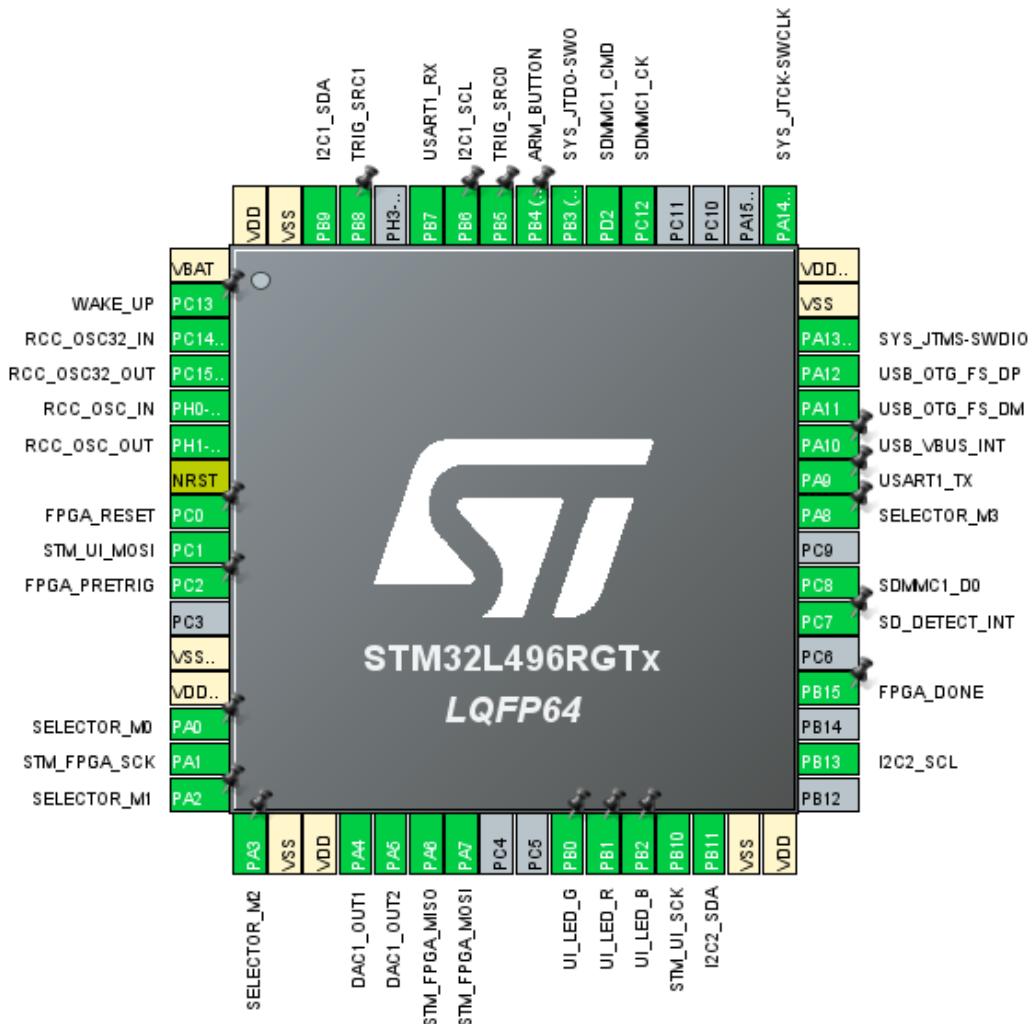


Figure 3.5 configuration PINs

Pour configurer le processeur, j'ai initialisé les pins en fonction des connexions déjà établies sur la board. Je ne vais pas détailler chacune d'entre elles mais les points importants étaient de configurer 3 pins spécifiques en interruption externe pour le projet :

FPGA DONE, c'est la pin PB15(36). Elle représente un signal en provenance de la FPGA et indique quand l'acquisition des données est terminée. Elle est configurée sans l'utilisation de Pull-up/Down étant donné qu'il y en a déjà une du coté de la FPGA

ARM BUTTON, c'est la pin PB4(56). Le signal provient d'un bouton. Il fallait lui rajouter une Pull-up depuis le STM32. Son implémention nécessite d'activer les interruptions externes 4 dans l'onglet NVIC sur le programme STM32CUBEIDE.

WAKE UP, c'est la pin PC13(2), le signal provient aussi d'un bouton. Il fallait lui rajouter une Pull-up depuis le STM32.

NVIC Mode and Configuration		
Configuration		
NVIC Interrupt Table	Enabled	Preemption
Non maskable interrupt	<input checked="" type="checkbox"/>	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0
Memory management fault	<input checked="" type="checkbox"/>	0
Prefetch fault, memory access fault	<input checked="" type="checkbox"/>	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0
Debug monitor	<input checked="" type="checkbox"/>	0
Pendable request for system service	<input checked="" type="checkbox"/>	0
Time base: System tick timer	<input checked="" type="checkbox"/>	15
PVD/PVM1/PVM2/PVM3/PVM4 interrupts through EXTI lines 16/35/36/37/...	<input type="checkbox"/>	0
Flash global interrupt	<input type="checkbox"/>	0
RCC global interrupt	<input type="checkbox"/>	0
EXTI line4 interrupt	<input checked="" type="checkbox"/>	5
DMA1 channel2 global interrupt	<input checked="" type="checkbox"/>	0
DMA1 channel3 global interrupt	<input checked="" type="checkbox"/>	0
TIM1 update interrupt and TIM16 global interrupt	<input checked="" type="checkbox"/>	0
I2C1 event interrupt	<input type="checkbox"/>	0
I2C1 error interrupt	<input type="checkbox"/>	0
I2C2 event interrupt	<input type="checkbox"/>	0
I2C2 error interrupt	<input type="checkbox"/>	0
SPI1 global interrupt	<input type="checkbox"/>	0
SPI2 global interrupt	<input type="checkbox"/>	0
USART1 global interrupt	<input type="checkbox"/>	0
EXTI line[15:10] interrupts	<input checked="" type="checkbox"/>	5
SDMMC1 global interrupt	<input type="checkbox"/>	0
TIM6 global interrupt, DAC channel1 and channel2 underrun error interrupts	<input type="checkbox"/>	0
USB OTG FS global interrupt	<input type="checkbox"/>	0
FPU global interrupt	<input type="checkbox"/>	0

Figure 3.6 configuration NVIC

Chapter 3. Développement

La priorité de ces interupts a été définie à 5 afin qu'elle n'empête pas sur d'autres fonctions prioritaires du STM32. L'interruption a aussi été sélectionnée pour les Timers que je vais détailler plus tard.

3.2.2 Configuration du projet - clocks, SPI, SD

La clock du système a été définie à 64 [Mhz] pour tourner à la même vitesse que la FPGA et empêcher au STM32 d'avoir des opérations de communications allant plus vite que ce que les autres appareils ne peuvent supporter.

Pour pouvoir communiquer avec la FPGA et les FRAMs, j'ai configurer un SPI :

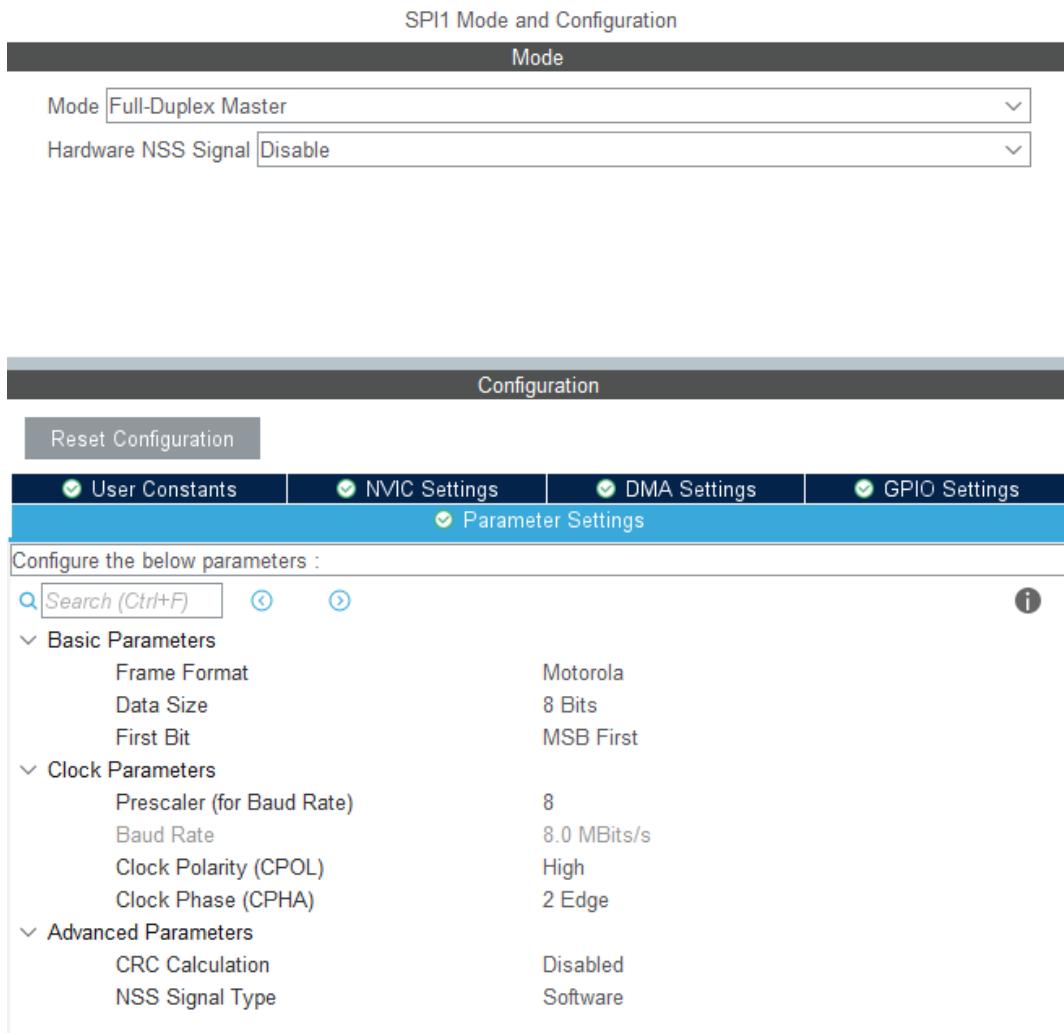


Figure 3.7 configuration SPI

SPI

La communication s'effectue en envoyant des blocs de 8 bits en commençant par le bit de point fort. Le prescaler est de 8, ce qui correspond à une clock de 6.4 [Mhz] selon le calcul :

$$\text{Prescaler} = \frac{f_{\text{clock}}}{f_{\text{SPI}} \times 2} \quad (3.1)$$

$$f_{\text{clock}} = 64[\text{Mhz}] \quad (3.2)$$

$$f_{\text{SPI}} = 6.4[\text{Mhz}] \quad (3.3)$$

$$\text{Prescaler} = 8 \quad (3.4)$$

J'ai choisi un prescaler à 8 afin d'améliorer la vitesse de transmission au maximum. Au-delà de cette valeur, il n'y a plus de différence de vitesse à cause de la façon dont je communique avec les FRAMS.

Chapter 3. Développement

Le tout fonctionne avec un CPOL et une CPHA à '1' :

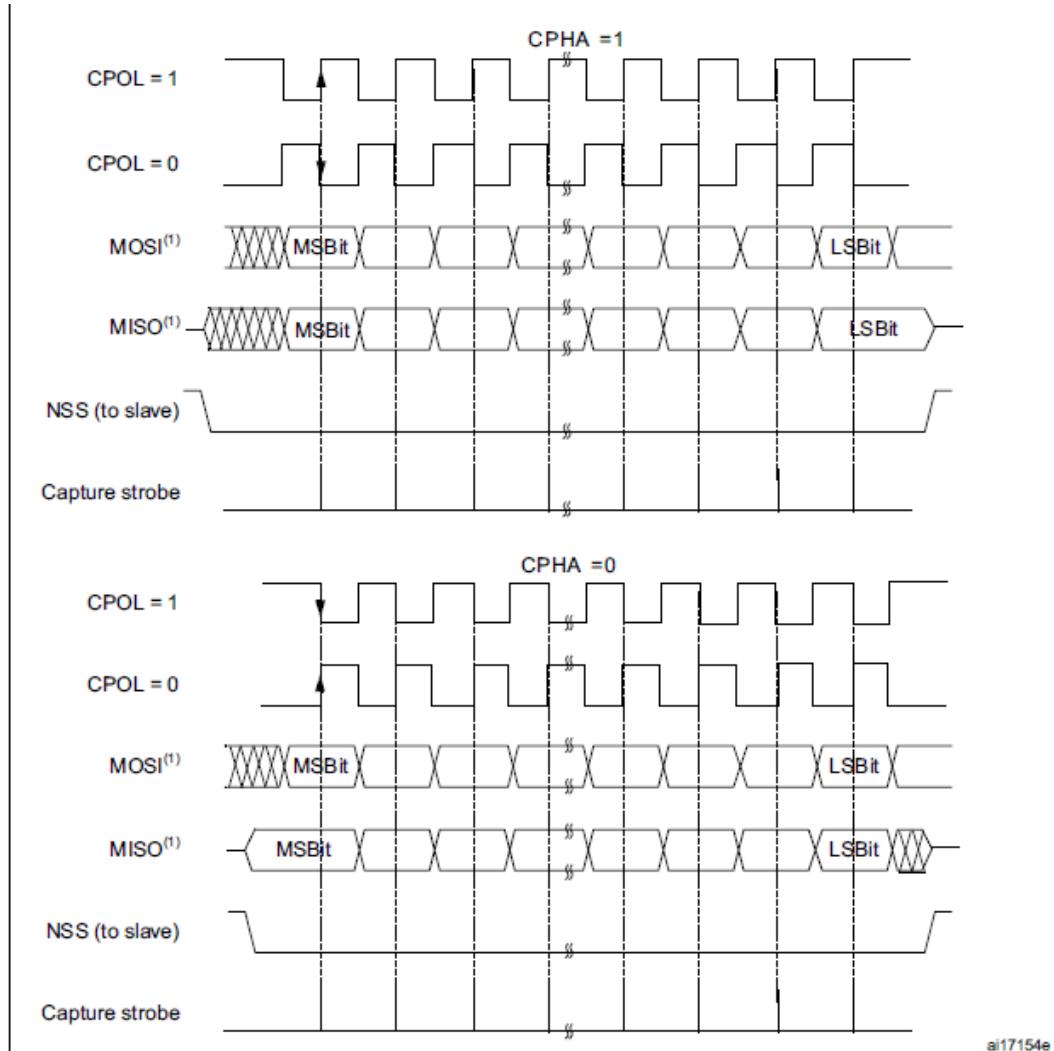


Figure 3.8 CPOL CPHA

Comprenez par là que la clock est de base à l'état actif haut et que le transfert de données ne se fait que durant le changement de clock vers l'état actif haut.

TIMER

Il a fallu aussi implémenter un timer afin de permettre aux leds de clignoter pendant que le programme est en train de tourner. Le timer va compter pendant l'exécution du programme. Une fois arrivé au bout, il va créer une interruption afin d'activer/éteindre la led.

3.2 Programmation STM32

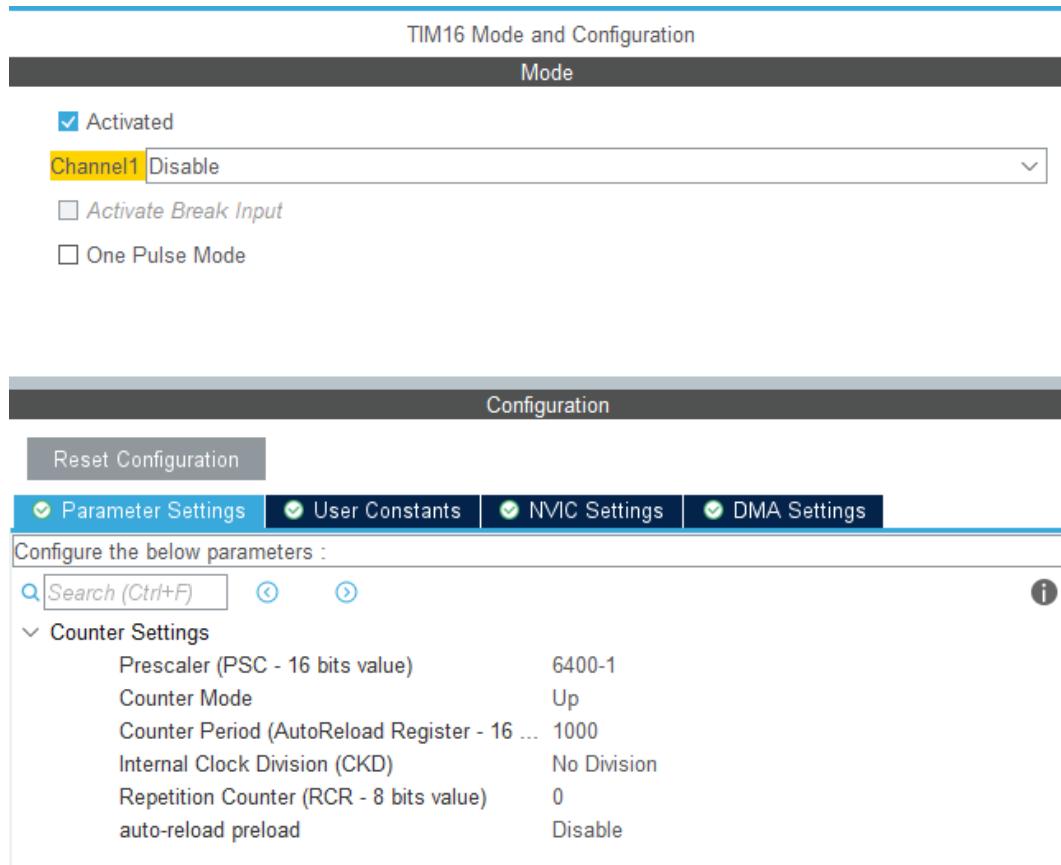


Figure 3.9 configuration Timer16

Sur cette image on peut donc voir la configuration de ce Timer, son prescaler est de 6'400 et correspond à la division de la clock qui va être effectuée pour en obtenir une moins rapide(ici une clock de 10[KHz]) et un compteur de 1'000 coups d'horloge. Sa vitesse n'a pas vraiment besoin d'être précise puisqu'ici il s'agit simplement de faire clignoter une led.

SD CARD

Pour échanger avec la carte SD, j'utilise un autre système que le SPI puisque les pins permettant l'utilisation du 2ème SPI du STM32 ont été réservés pour un autre appareil (initialement un écran) mais qui n'a malheureusement pas pu être implémenté ici. Pour cette communication il a fallu employer un autre moyen disponible le SDMMC1, qui concrètement agit comme le spi mais est spécialisé pour les cartes SD. Ce service s'occupe aussi de l'initialisation et de la gestion d'erreurs entre autre. La clock de ce système en interne est configurée à 48 [MHz] avant d'être divisée en fonction des paramètres sélectionnés.

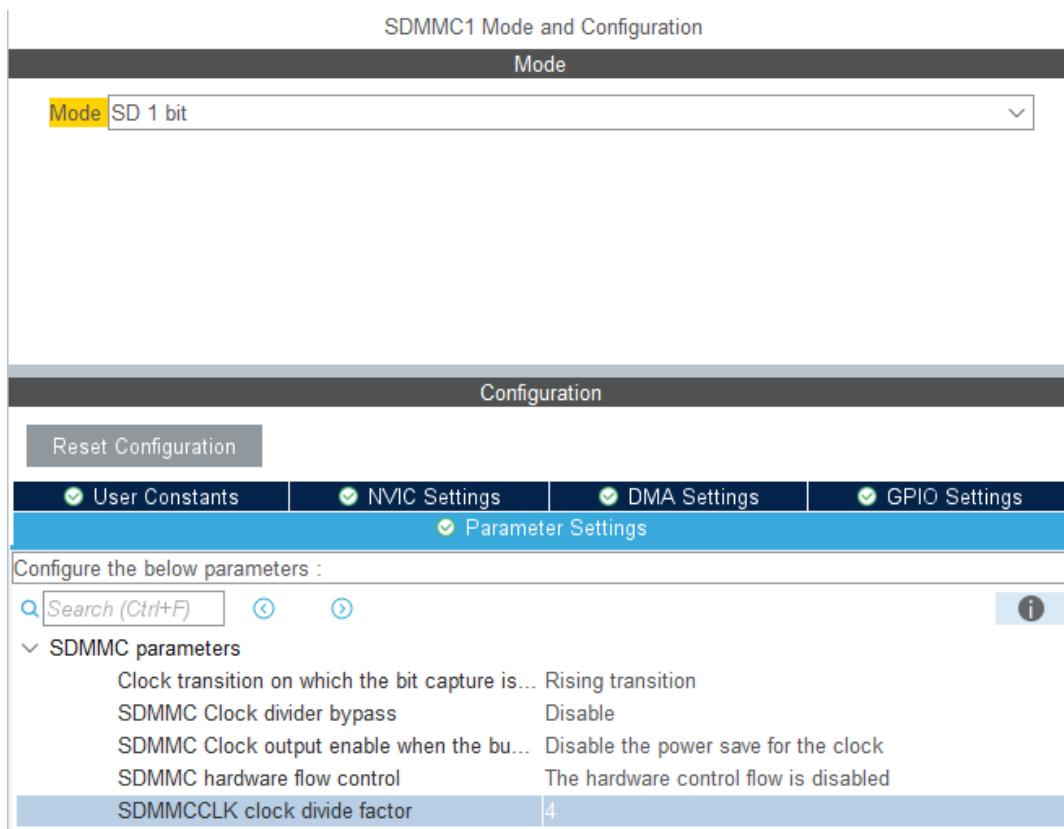


Figure 3.10 configuration SDMMC1

Le calcul effectué pour sa clock finale qui sera employé est donc :

$$\text{SDMMC_CK} = \frac{\text{SDMMCLCK}}{\text{CLKDIV} + 2} \quad (3.5)$$

$$\text{SDMMC_CK} = \text{clockrel} \quad (3.6)$$

$$\text{SDMMCLCK} = \text{clock48}[MHz] \quad (3.7)$$

$$\text{CLKDIV} = \text{diviseur de clock} \quad (3.8)$$

Ce qui donne une clock à 8[MHz], elle a été prise pour avoir un temps correct vis-à-vis du transfert de données. Ici cela ne représente que 1 à 2 secondes pour finir le transfert total.

3.2.3 Classes et fonctions

Dans cette partie, je vais expliquer de manière globale l'utilité des classes que j'ai créé pour le projet, pour ce qui concerne les explications du code, vous pouvez les retrouver en annexe.

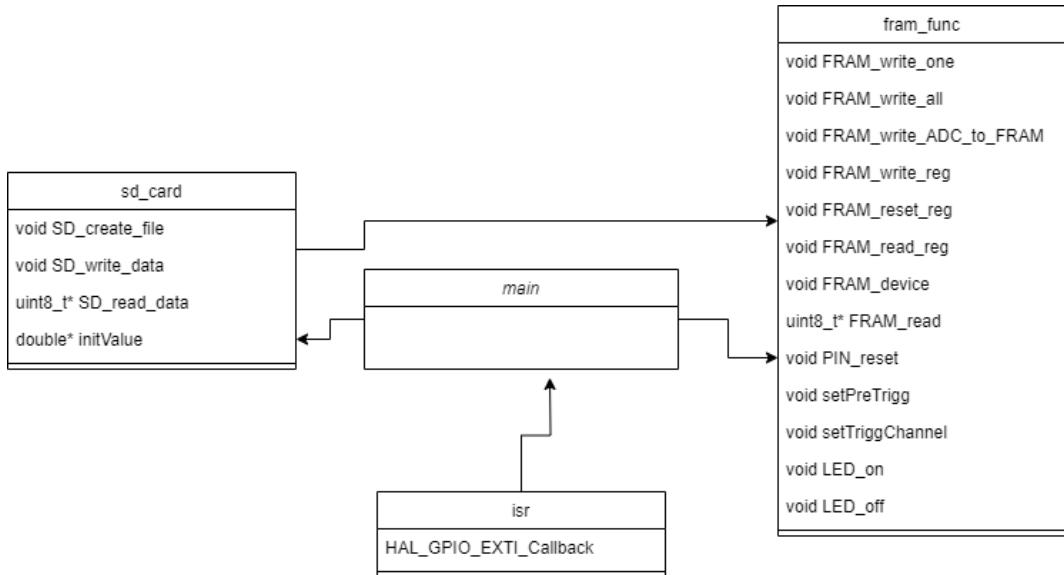


Figure 3.11 diagramme de classe

main

Classe principale où l'intégralité du code s'effectue, les initialisations, la boucle de programmation, les paramètres externes etc...

fram_func

Cette classe rassemble l'ensemble des fonctions permettant de pouvoir communiquer avec les FRAMs et demander ainsi l'ensemble de leurs services (lecture, écriture, accès aux registres, etc...). Par conséquent, c'est dans cette classe que je gère la valeur du "Selector" décrit dans la partie FPGA.

Il y aussi les quelques fonctions nécessaires au bon fonctionnement du programme comme la gestion des LEDs, l'envoi du preTrigger, la sélection du canal de trigger et le reset des pins du Selector.

Pour l'instant, afin de lire les valeurs des FRAMS, la fonction envoie l'instruction de lecture puis envoie une lecture de 2 bytes sur la FRAMS afin de réceptionner une valeur complète de l'adc.

sd_card

Ici, comme son nom l'indique, c'est la partie qui s'occupe de l'écriture/lecture de la carte sd, la création des fichiers ainsi que de la transformation des valeurs d'initialisation du fichier texte en valeurs binaires. En cas d'erreur en lien avec la carte, que ce soit

Chapter 3. Développement

l'absence de carte ou d'une mauvaise lecture de fichier, la led va s'allumer en rouge et le restera jusqu'à un reset du système. En mode FATFS, si durant le programme la carte SD est enlevée, le système va considérer qu'elle est corrompue même si on l'a réintégrée dans le système en dehors des lectures/écritures sur la carte, il faut alors relancer le programme via le bouton reset après avoir re-branché la carte.

isr

Elle ne s'occupe de réceptionner les changements survenus sur les pins configurées en "interrupts" et de réveiller le contrôleur quand il y a changement sur ces pins ou de lancer un reset du system.

3.2.4 Fonctionnement global

La boucle de programmation va donc se lancer selon le schéma suivant :

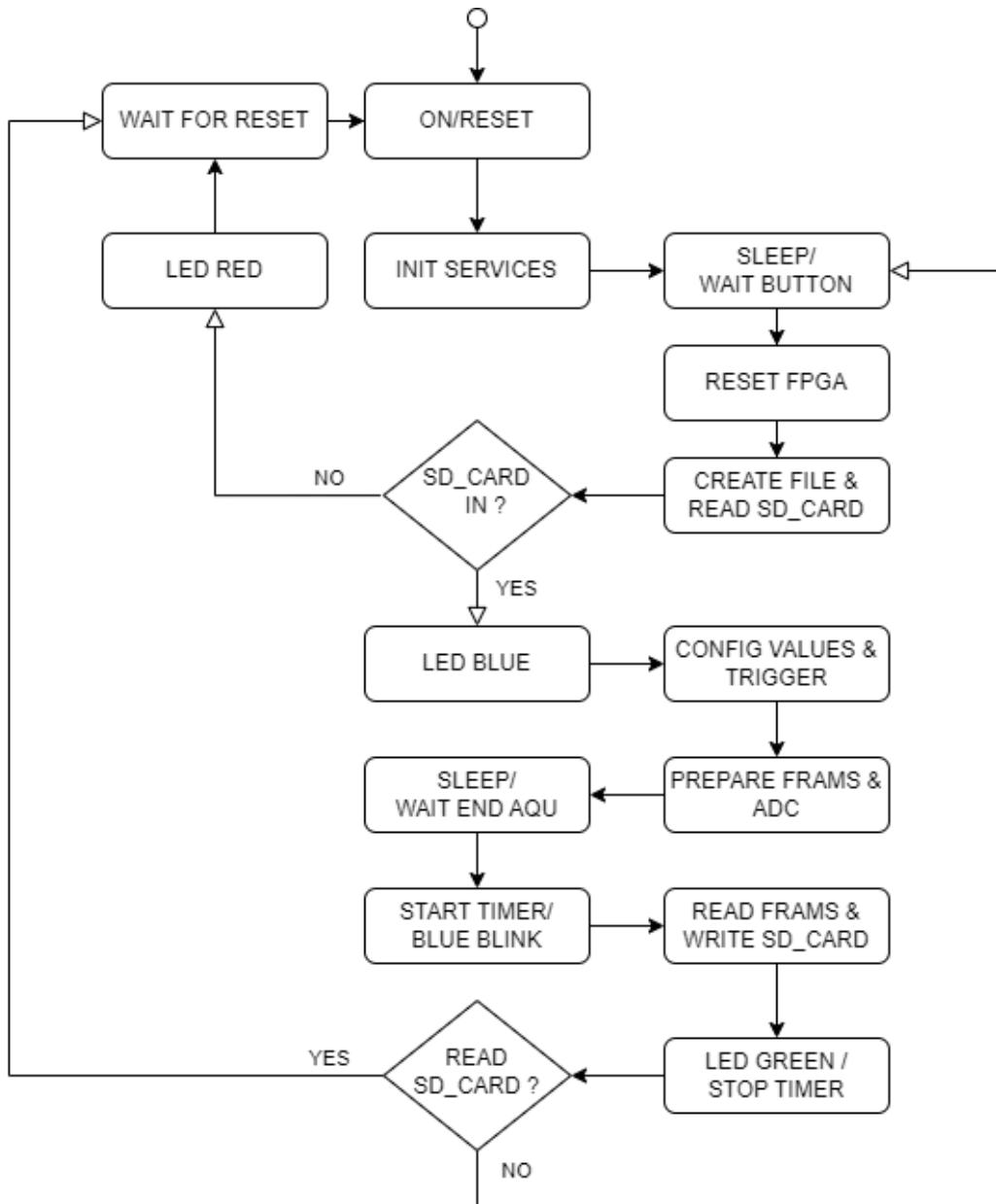


Figure 3.12 diagramme de séquence

Une fois la board mise sous tension le programme va initialiser ses différents services configurés (FATFS, HAL, DAC, SPI1, etc...), puis va passer en état sleep attendant l'action sur le bouton "wake up".

Une fois le bouton appuyé le programme va créer 4 fichiers, un par canal, et va lire les valeurs comprises dans le fichier "INIT" de la carte SD afin de préparer la valeur du preTrigg, le canal de trigg et la tension de trigg via le DAC.

Le dac permettant au STM32 d'envoyer une tension comprise entre 1 et 3.3[v], cette tension sera ensuite comparée en dehors du STM32 dans un appareil avec celle du

Chapter 3. Développement

canal sélectionné, une fois sa tension supérieure ou égale à celle du DAC un signal à '1' sera émis en direction de la FPGA pour avertir d'un TRIGGER.

Si le système passe la Led en rouge, c'est qu'il y a un problème avec la carte sd et qu'il faut reset le programme.

Si la led est en bleu le système continue sa routine en configurant les différentes valeurs et va initialiser les FRAMs et l'adc afin de les préparer à envoyer/recevoir des datas, avant de transmettre la commande à la FPGA pour préparer l'acquisition des valeurs puis va passer en mode sleep à nouveau.

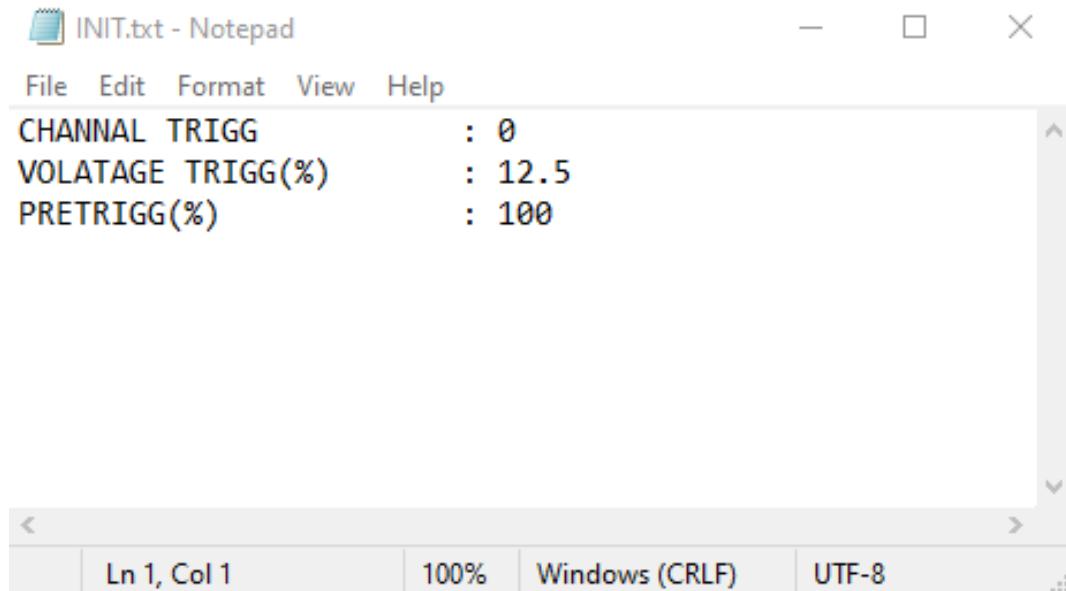
Quand l'acquisition est terminée, le STM32 se réveille et va demander la valeur de la dernière adresse mémoire comptée par la FPGA où il pourra commencer à lire les datas comprises dans les FRAMs une par une, remplissant des tableaux de 32'768 valeurs puis les concatènent 2 part 2 pour obtenir les valeurs de 16 bits de l'adc, avant de les envoyer dans leur fichier de la carte SD 16 fois en tout pour parcourir toute la mémoire des FRAMs.

La taille des tableaux n'a pas été prise au hasard. La valeur maximale des adresses mémoires d'une FRAM correspond à 2^{19} (524'288) et la taille maximale d'un tableau de 8bits sur le STM32 utilisé est de 320'000 unités, vu que c'est sa mémoire SRAM qui est utilisée durant l'exécution du programme et non pas sa mémoire flash qui possède une plus grande disposition. Ainsi pour faciliter la lecture des données, j'ai donc divisé la mémoire par 16.

Quand l'intégralité des mémoires a été transférée, le système se remet en début de programme dans l'attente soit d'une lecture de la carte SD sur l'ordinateur via Matlab (dans ce cas il aura besoin d'un reset), soit d'une pression sur l'autre bouton afin de relancer le programme pour créer de nouveaux fichiers et ainsi refaire une prise de valeurs.

3.3 Carte SD

La carte SD possède 2 fichiers de base: un fichier d'initialisation "INIT.txt" où l'on peut retrouver les 3 valeurs de base de Trigger et un fichier "README.txt" où l'on peut retrouver les informations de bases sur l'utilisation de la board.



```

INIT.txt - Notepad
File Edit Format View Help
CHANNEL TRIGG      : 0
VOLATAGE TRIGG(%)  : 12.5
PRETRIGG(%)        : 100
Ln 1, Col 1 100% Windows (CRLF) UTF-8

```

Figure 3.13 Valeurs d'initialisation

les 3 valeurs pouvant être réglé sont donc :

- **CHANNEL TRIGG**, on peut lui écrire une donnée allant de 0 à 3 et correspondant au 4 canaux visible sur la board.
- **VOLTAGE TRIGG(%)**, valeur en pourcentage allant de 0 à 100% et correspondant à une tension de 1 à 3[v].
- **PRETRIGG(%)**, valeur en pourcentage, indiquant le total de mémoire à remplir une fois le Trigger détecté.

Chapter 3. Développement

Une fois le système lancé, il va créer des fichiers correspondants aux canaux. A savoir : FRAM0_0.BIN, FRAM1_0.BIN, FRAM2_0.BIN, FRAM3_0.BIN. A chaque nouvelle prise de mesure, le STM32 va créer de nouveaux 4 fichiers en remplaçant le " _0" par " _1" et ainsi de suite :

Name	Date modified	Type	Size
INIT.txt	11.08.2023 11:33	Text Document	1 KB
FRAM3_0.BIN		BIN File	0 KB
FRAM2_0.BIN		BIN File	0 KB
FRAM1_0.BIN		BIN File	0 KB
FRAM0_0.BIN		BIN File	0 KB
FRAM_01.TXT		Text Document	1 KB
DEVICE.TXT		Text Document	0 KB
.		File	1 KB
README.txt	17.08.2023 09:08	Text Document	0 KB

Figure 3.14 Exemple de fichiers dans la carte SD

4 | Tests, résultats et corrections

Dans la continuité de ce rapport, je vais maintenant parler des différentes parties de tests importants qui ont été accomplis tout au long de ce projet afin de confirmer et de corriger les programmations et configurations de la FPGA, du STM32 ainsi que des modifications effectuées sur la board.

4.1 SIMULATION VHDL

Durant les premiers mois du projet, je m'étais exclusivement concentré sur la programmation VHDL car grâce aux simulations disponibles sur le programme, il était possible de mieux visualiser les résultats espérés après discussions.

4.1.1 SIMULATION-INITIALISION

Une grosse partie des modifications établies ont dépendu de ces simulations. Grâce à un bloc de tests, disponibles en annexe, j'ai pu simuler le comportement probable qu'aurait le STM32 et j'ai pu ainsi obtenir, au bout de plusieurs corrections, la logique de ma configuration VHDL pour ces simulations :

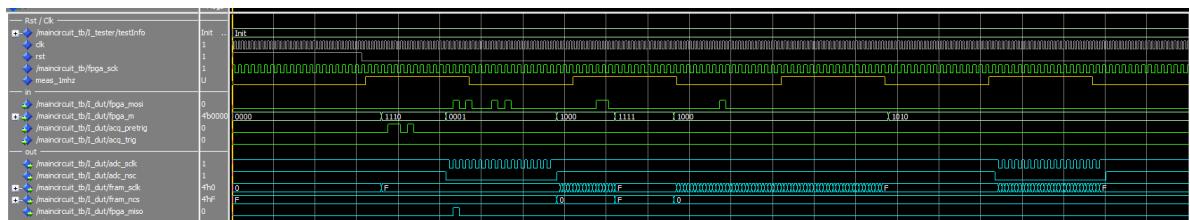


Figure 4.1 Simulation d'initialisation

Sans décrire tous les signaux présents sur l'image, on peut observer les différents services effectués en fonction des valeurs du selector^{3.4}. Mais étant donné que ce sont là de vieilles simulations le service "0001" correspond à l'écriture sur l'ADC et le "1110" à l'écriture du pretrig. On peut aussi noter qu'avant le signal pretrig servait de spi pour sa propre initialisation.

Ces détails mis de côté, on peut observer que les divers services fonctionnent comme prévu avec pour cette simulation une valeur de pretrig à 80% de la valeur de mémoire. La partie qui fut complexe à réaliser concerne le passage entre l'écriture du registre des FRAMS avec l'adresse mémoire où commencer l'écriture des valeurs, puisqu'il fallait que le NCS des FRAMS reste à l'état actif bas. Comme expliqué dans le chapitre 2 à la figure 2.3.

On peut donc observer que l'ensemble des services présentés ici fonctionne comme ce qui a été décrit dans la partie 3 concernant la FPGA.

Chapter 4. Tests, résultats et corrections

4.1.2 SIMULATION-ADC TO FRAMS

Durant la suite de cette simulation on passe à la partie écriture de l'adc vers les FRAMs, ce qui donne :

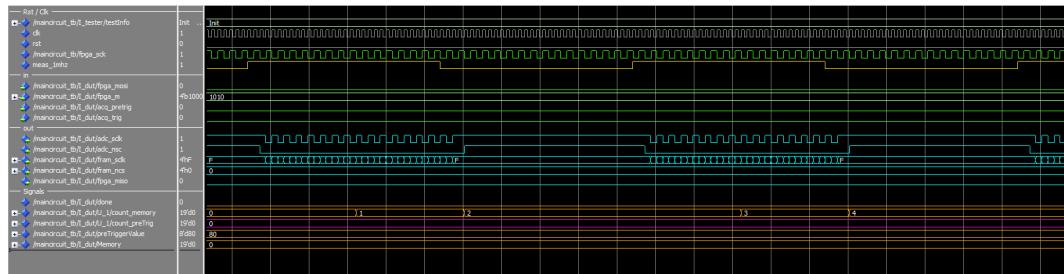


Figure 4.2 Simulation d'écriture

On peut alors constater ici, que la clock de l'adc et des FRAMs sont synchronisées, que le NCS des FRAMs restent à '0' et que celui de l'adc copie le signal de la clock de 1[MHz], restant ainsi à '0' quand la clock de 32 [MHz] est active et qu'elle effectue bien 16 coup de clock avant de rester à '1'. On remarque aussi que le compteur de mémoire s'incrémente chaque 8bits de données envoyées.

4.1.3 SIMULATION-TRIGGER

Après une certaine période d'activité dans la simulation, je lance un Trigger :

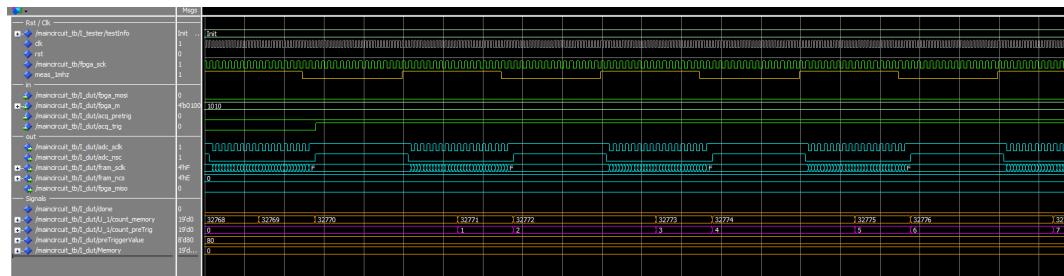


Figure 4.3 Simulation de Trigger

Sur cette image on peut donc observer que lors de la détection du signal de Trigger, un deuxième compteur se met en route jusqu'à atteindre la valeur mémoire calculée en fonction du pourcentage de preTrigger reçus.

4.1.4 SIMULATION-ACQUISITION DONE

A la fin de cette simulation nous obtenons :

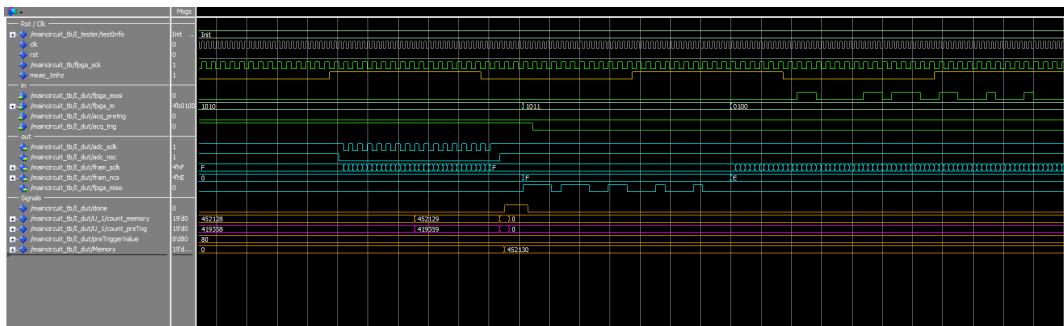


Figure 4.4 Fin de simulation

Quand la valeur d'adresse mémoire finale est atteinte par le 2ème compteur, ici 419'359 ce qui correspond une fois le calcul effectué vis-à-vis des 80% du pretrig, à 524'198.75 pour 100%, on peut observer une différence de 89.25 valeurs perdues par rapport à la valeur mémoire total, à savoir une erreur de 0.016% (négligeable après discussion). Une fois cette valeur atteinte les compteurs sont remis à 0 pour attendre la prochaine acquisition et la valeur mémoire atteinte par le premier compteur est envoyée en SPI vers le STM32.

A noter que sur cette simulation l'envoie de l'adresse mémoire commence avec le "MSB" à '1', car au départ je voulais l'utiliser comme repère quand une communication entre le STM32 et la FPGA commence. Ce système a été abandonné car inutile en l'occurrence.

4.1.5 SIMULATION-conclusion

En conclusion, les simulations respectaient les directives tel qu'on a pu le constater dans les chapitres vu plus haut.

Cependant, comme j'ai pu le préciser durant ces explications, beaucoup d'éléments ont du être modifiés, voir supprimés du fait de la différence entre le monde théorique et pratique. J'ai noté également que le programme de simulation est plus permissif quant aux possibilités de configuration en VHDL. Par exemple : la simulation permet dans un process "reset,clock" l'utilisation en simultané de la détection d'un "rising_edge" et d'un "falling_edge" d'une clock, alors que la FPGA sélectionnée empêche cette utilisation.

L'utilisation de certaines pin étant des RGBs nécessitaient aussi certaines réadaptations afin de permettre leur utilisation. Notamment, selon la datasheet, elles sont équipées en interne de pull-up activables depuis le programme IceCube. Mais, après plusieurs tests, il s'est avéré qu'elles n'y étaient pas. J'ai donc du souder directement sur la board de nouvelles résistances. Je détaillerais plus en détail cette partie de modifications dans le chapitre 4 Correction décrit plus bas.

4.2 TESTS REELS

4.2.1 Test carte SD

Une fois les simulations VHDL terminées, il a fallu commencer le code du STM32, les premiers tests effectués concernaient l'écriture et la lecture de la carte SD, après de nombreux tests j'ai pu établir une connexion entre les 2 appareils. Cependant, je n'utilise qu'un seul signal sur 4 de communications. Effectivement, sur la configuration du SDMMC1 décrite selon la figure [3.10](#), le mode de communication a été fixé à 1 bit. Une méthode permet de communiquer avec le mode bus 4 bits afin d'augmenter la vitesse de communication. Faute de temps je n'ai pas eu l'opportunité d'explorer plus en détail cette option.

4.2.2 Test des FRAMs

Actuellement, le temps nécessaire pour lire et écrire l'intégralité des valeurs des FRAMs vers des fichiers sur la carte SD prend 1 minute. Cette durée s'explique de part la façon dont je vais lire ces valeurs comme expliqué dans le chapitre [3](#). Ceci correspond à un point à améliorer puisque l'instruction de lecture demande du temps. Cependant lorsque je demande plus de 2 valeurs aux FRAMs ces dernières deviennent corrompues rendant les graphes illisibles. Malgré mes recherches je n'ai pas réussi à trouver la raison de cette erreur.

Une analyse plus poussée à une fréquence plus basse serait envisageable afin de déterminer d'où pourrait provenir le dysfonctionnement. Mais les FRAMs sont très sensibles. Quand on pose un appareil de mesure sur elles, leur impédance va changer, ce qui va émettre une erreur interne et empêcher la réception correcte des datas.

4.2.3 Test FPGA

Afin de pouvoir tester le bon fonctionnement des diviseurs de clock, 2 pins de la FPGA ont été connectés à 2 canaux pouvant être reliés à un oscilloscope pour inspecter leurs signaux :



Figure 4.5 Signals clock 1 & 32[MHz]

De ce fait on peut observer que selon l'oscilloscope les clocks de 1 et 32[MHz], tel décrit dans les chapitres précédants sont bien effectuées par la FPGA.

Chapter 4. Tests, résultats et corrections

En diminuant la plage de temps de l'oscilloscope pour agrandir les 2 signaux, on peut mesurer la différence de temps entre le NCS de l'adc et la clock employée pour faire la réception des 16 bits afin d'observer s'il respecte le temps décrit dans le chapitre 2:

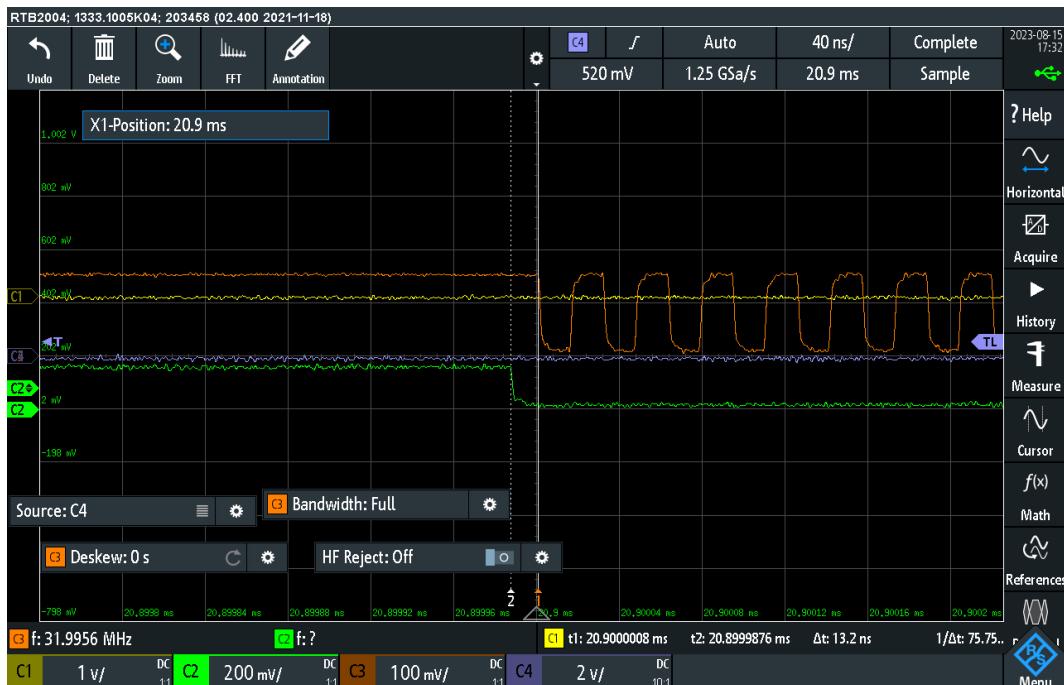


Figure 4.6 delta de temps

La valeur mesurée est donc de 13.2[ns] et celle prescrite ne doit pas être plus basse que 5[ns], l'objectif est donc accompli.

4.2.4 Analyse Matlab

Analyse de fréquences

Afin de pouvoir déterminer si ces 2 clocks sont efficaces durant une période de temps indéterminée, j'ai laissé tourner la FPGA bloqué en mode acquisition durant toute une soirée en connectant les mesures récoltées par l'adc vers le pc afin d'analyser l'ensemble de ses valeurs avec un script matlab.

Les mesures observées permettaient de contrôler si la clock à 32 faisait effectivement à chaque fois 16 coups de clock et si la fréquence de la clock à 1 était constante durant toute la période de test.

Après avoir inspecté les valeurs reçues, le constat était que la clock à 32 effectuait tout au long de la période à '0' de la clock à '1' 16 coups de clocks.

La clock à 1 était constante tout du long avec une légère variation de +/- 400[Hz] , correspondant à une erreur de 0.04% avec à 4 reprises une erreur de 800[Hz](0.08%). Après discussion et observation il a été décidé que cette légère marge d'erreur était négligeable et qu'il n'y avait pas d'impact sur le fonctionnement de l'appareil. Les scripts employés afin d'obtenir ces résultats sont disponibles en annexe

Analyse des FRAMs

Les simulations présentées ici correspondent à un signal de 10[Hz] d'une amplitude 1 [v] pour un offset de 1.25[v]. Le pretrigger était configuré à 100%, le canal de trigg était celui du signal à savoir le n°0 et le Trigg à été sélectionné à 1.25 [v] aussi. Après avoir récolté l'ensemble des valeurs en mémoire des FRAMs et les avoir envoyé sur la carte SD, j'ai employé un script MatLab afin de pouvoir les afficher sur un graphe :

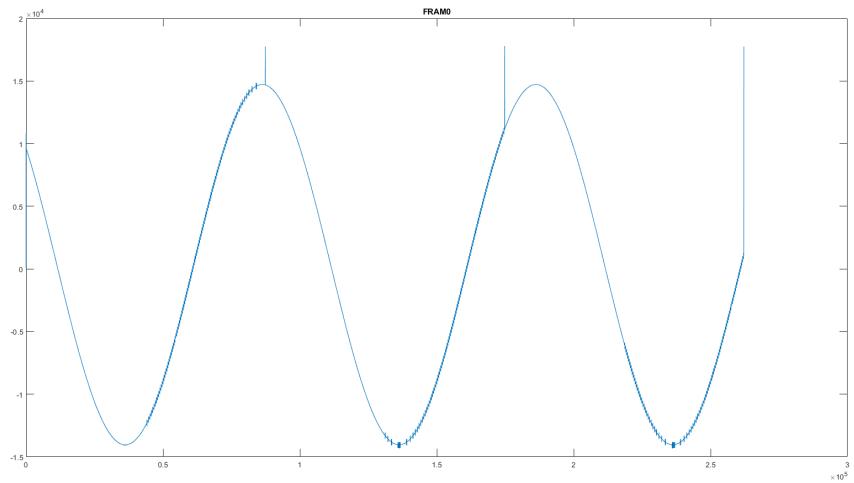


Figure 4.7 sinus à 10 [Hz] avec perturbation

Sur cette figure, on peut constater que le signal correspond bien à celui désiré, mais qu'il possède des anomalies qui se répètent de manière périodique à 3 répétitions.

D'abord le signal est lisse, puis il y a des perturbations qui apparaissent et se terminent par une valeur "piquée" :

Chapter 4. Tests, résultats et corrections

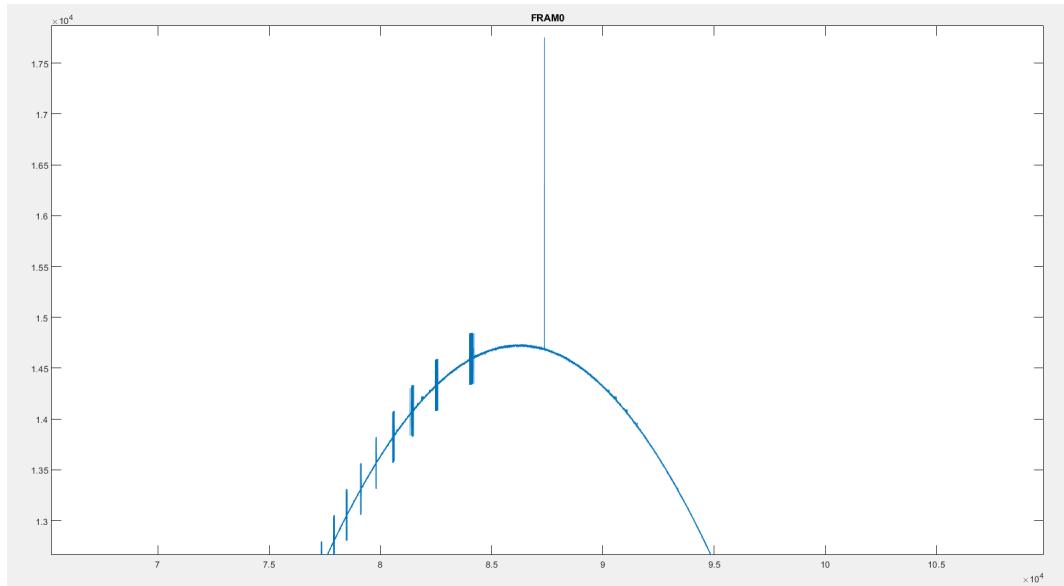


Figure 4.8 perturbation du signal

Je n'ai pas pu clairement établir la raison de l'apparition de ces perturbations et de cette valeur pic. Pour corriger ce problème, j'ai justement diminué la taille des tableaux pour arriver à la valeur décrite dans le chapitre 3.

Grâce à cela j'ai pu obtenir un signal plus lisse :

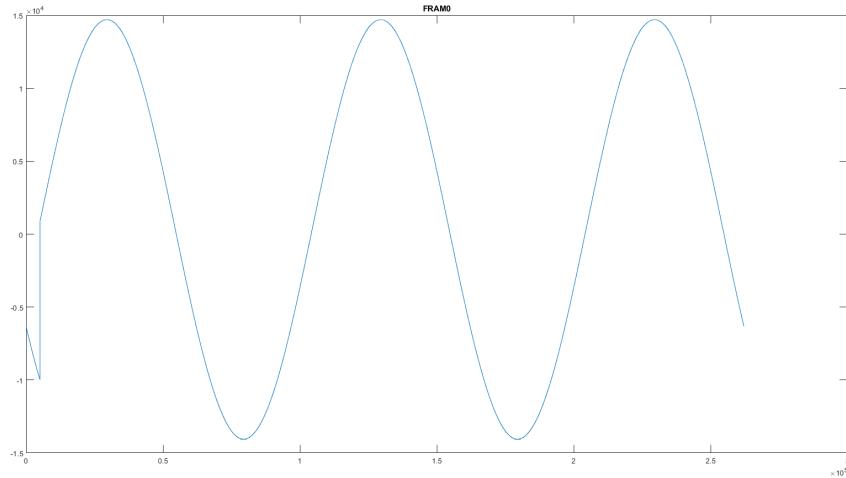


Figure 4.9 sinus à 10 [Hz] sans perturbation

Le signal est à ce moment-là plus propre avec un bruit nettement plus diminué. Cependant, il y a toujours une erreur quant à l'adresse où il faut commencer à lire les valeurs dans les FRAMs. On peut observer un léger décalage sur l'image. Dans cette exemple, j'ai démarré la lecture directement sur l'adresse '0' des FRAMs car l'adresse envoyée par la FPGA créait toujours un décalage égal à sa valeur. Pour résumer, les FRAMs ne se remplissent que lors de la réception d'un signal trigger, avec un décalage de

plus ou moins 5'000 valeurs (1.9%). Pourtant, même en prolongeant la durée d'un test, le résultat est le même. J'ai déjà contrôlé le NCS et la clock des FRAMs et de l'ADC durant la prise de mesures, mais les valeurs sont correctes et correspondent aux directives. Cela fait donc aussi partie des points à améliorer.

4.2.5 Précision des mesures

En augmentant la taille de l'image on peut observer la précision des mesures effectuées sur la dernière image présentée :

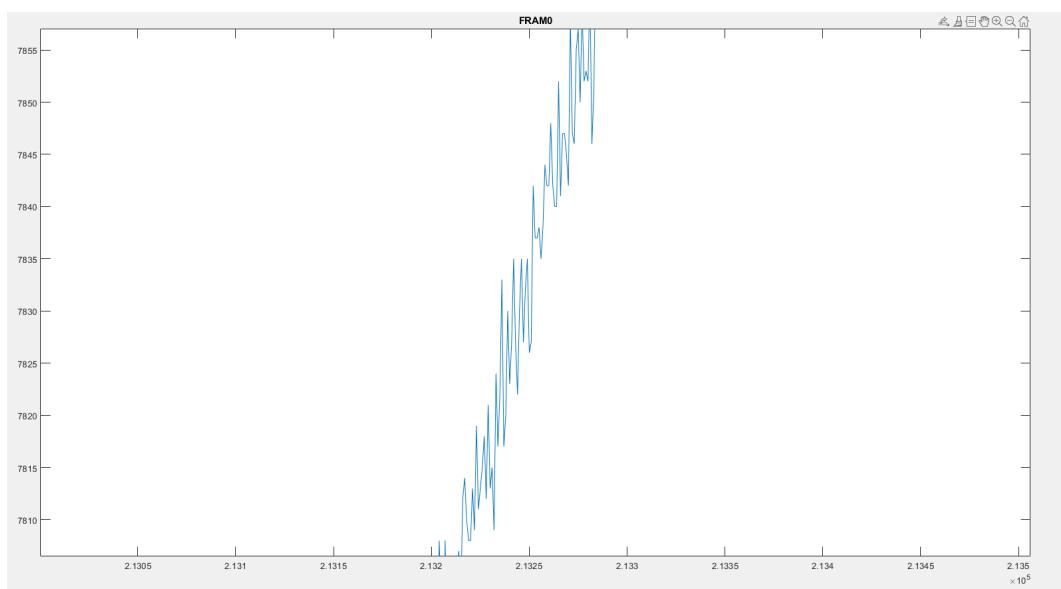


Figure 4.10 sinus à 10 [Hz]-précision

Le signal a une variation aléatoire entre 2 mesures, avec un pic entre 2 valeurs pouvant aller jusqu'à 15 (valant ici 15[uV]).

4.3 Corrections Pcb

Le PCB comportait plusieurs erreurs qu'il a fallu détecter et corriger tout au long de ce projet. Les corrections ont été apportées sur les schémas du pcb disponible en annexe.

- J'ai dû retirer un régulateur de courant(U7[10]) qui s'occupait de l'alimentation de toute la partie réception des signaux sur les canaux, car la tension générée par ce dernier n'était pas stable. Il a fallu brancher cette partie directement sur les points de tests 3.3[v] et GND.
- Il a aussi fallu brancher des Pull-up directement sur les pistes de la FPGA correspondant à ses pins RGB (n°39,40 et 41), car elles n'en possédaient pas en interne.
- Brancher un condensateur entre la sortie(pin 6) et la masse de l'U74[11] qui s'occupe d'atténuer les bruits pour une sortie à 2.5[v], car après mesure elle n'était que de 1,5 [v].

Chapter 4. Tests, résultats et corrections

- Brancher la pin 4 de l'U75[12] à la masse pour respecter ces directives de datasheet.
- Retirer les Pull-up R66 et R69 branchées entre la pin 16 de la FPGA et la pin 1 de la flash(U17)[5] car elles étaient en trop étant donné que la FPGA en possède déjà une en interne.
- Couper la piste 3 de l'U12[13], car elle en empêchait le bon fonctionnement.

4.4 Conclusion & améliorations

En conclusion de ces phases de tests et de mesures, j'ai pu déterminer que le système est fonctionnel et qu'il respecte correctement les objectifs demandés. Cependant il y a encore des fonctionnalités pouvant être améliorées, comme j'ai pu le préciser durant les chapitres précédents. Voici un récapitulatif:

La communication flash-FPGA

Comme précisé dans le chapitre 2 analyse dans la partie FPGA, je n'ai pas réussi à utiliser la mémoire flash à cause de la communication entre elle et la FPGA ce qui empêche de pouvoir conserver le fichier de programmation après une hors-tension.

La vitesse de transfert des données FRAMs-STM32

Le deuxième point d'amélioration possible est de réduire le temps d'acquisition des datas qui est actuellement d'une minute. Le meilleur moyen serait d'augmenter le nombre de datas transmises qui est actuellement de 2 datas à chaque demande. Cependant en augmentant ce nombre cela corrompt les valeurs présentes dans la FRAMs. Il faudrait pouvoir analyser avec un oscilloscope ou un Analog devices les trames reçues par le stm32 en fonction de la clock envoyée. Dans cette optique, il faudrait pour cela étudier un moyen de pouvoir brancher ces appareils sans pour autant perturber les FRAMs.

L'adresse mémoire de début de signal

Un autre point plus problématique concerne la façon dont les datas se transmettent entre l'adc device et les FRAMs, étant donné qu'il y a toujours un décalage entre le début du signal mesuré et l'adresse où l'on va lire la première valeur ainsi que la valeur du compteur mémoire envoyé par la FPGA qui ne correspond pas à la dernière valeur recue par les FRAMs. Le problème doit sans doute se situer au niveau de la FPGA, mais après des semaines à essayer de repérer le problème je n'ai pas pu exactement situer la zone à corriger.

Une meilleure interface Matlab

Pour lire les valeurs comprises dans les fichiers de la carte SD, j'ai conçu un script Matlab. Je pense que l'on peut encore améliorer ce script afin d'avoir un réel programme qui permet d'afficher directement les graphiques de l'ensemble des fichiers de valeurs.

Connexion USB-C vers PC

Afin de pouvoir lire directement les valeurs de la carte SD sur le PC. Avec plus de temps à disposition, ce serait une partie intéressante à rajouter, Cette partie serait

4.4 Conclusion & améliorations

intéressante à rajouter, car elle permettrait de pouvoir lire directement les datas de la carte SD sans avoir à sortir la carte SD de la board évitant ainsi d'avoir à reset le système à chaque analyse des datas.

Refaire un PCB

Comme expliqué plus haut diverses pièces ont du être modifiées sur la board et certaines ont dû être retirées. Cela implique que la board à plus un état de prototype actuellement et ne possède pas de réel boîtier. Refaire un PCB une fois les améliorations établies permettrait d'assurer au système une meilleure protection dans le temps et d'éviter des problèmes externes.

5 | Conclusions

5.1 Résumé du projet

Ce projet a donc pour but de pouvoir acquérir à haute fréquence des données provenant de différents canaux présents sur la board. Pour ce faire, on prépare un fichier d'initialisation dans la carte SD afin de sélectionner les valeurs du preTrigger, le canal sur lequel on va effectuer le trigger et la valeur de Trigg. Une fois ce fichier édité, on va connecter la carte SD à la board puis enclencher le bouton "Wake up" afin de commencer l'initialisation des valeurs dans le programme pour préparer l'adc device et les FRAMs. Puis le programme va passer en mode "sleep" et la FPGA prendra le relais pour transférer les valeurs jusqu'à l'apparition d'un Trigger sur le bon canal, suite à quoi elle va remplir les mémoires en fonction de la valeur envoyée par le STM32.

Quand toutes les adresses désirées seront remplies, la FPGA va envoyer un signal au STM32 afin de le réveiller. Ainsi il pourra lire les adresses mémoires des FRAMs et remplira les fichiers de la carte SDF prévue à cet effet. Quand toutes les adresses mémoires auront été lues le STM32 repassera en mode veille et l'utilisateur aura le choix entre sortir la carte SD du système afin de lire les valeurs grâce au script matlb ou de relancer une nouvelle prise de mesures en appuyant sur le bouton "Wake up". En cas de problème sur la board ou si la carte SD a été sortie, il y a un 2ème bouton le "Arm" qui permet de pouvoir faire un reset du STM32.

Des leds(rouge,bleu et verte) sont présentes sur la board et permettent de visualiser l'état des opérations en cours:

- OFF: le système est éteint
- BLEU: le système a fini d'initialiser les valeurs et passe en mode veille en attendant le signal "done" de la FPGA.
- BLEU CLIGNOTANT : le système s'est réveillé et est entrain de récupérer les valeurs des FRAMS
- ROUGE: problème avec la carte SD, veuillez vérifier qu'elle est présente dans le système et appuyez sur le bouton reset.
- VERT: Acquisition terminée. Pour relancer le système veuillez soit appuyer sur le bouton "Wake up" pour relancer une prise de mesures, soit lire la carte SD et appuyer ensuite sur le bouton reset.

5.2 Comparaison avec les objectifs initiaux

Le projet a donc pu être mené à bien et les objectifs ont été atteints selon ce qui a été défini dans le chapitre 1 :

- Réception, analyse et correction du circuit : Le système est fonctionnel après les retouches décrites plus haut, à l'exception de la programmation SPI de la flash vers la FPGA.

Chapter 5. Conclusions

- Configuration de la FPGA : Elle fonctionne comme désiré, transférant les valeurs et réceptionnant d'autres. A l'exception de l'adresse mémoire par laquelle il faudrait commencer la lecture. Le reste de ses fonctionnalités sont conformes à ce qui a été demandé
- Les phases de tests ont été menées à bien et ont pu permettre de corriger beaucoup d'erreurs aussi bien dans la programmation que sur le circuit hardware, mais malheureusement le temps n'a pas été suffisant pour corriger l'intégralité des défauts.

Le projet est donc fonctionnel mais nécessite encore quelques modifications afin d'avoir un système plus rapide, autonome et plus précis.

5.3 Difficultés rencontrés

De nombreux problèmes sont arrivés tout au long de ce projet comme des bugs, des incompréhensions vis-à-vis de l'utilisation de certaines fonctions en 'c', datasheets, utilisation de programmes, différence entre les simulations et les tests réels, etc...

Il y a 2 principales difficultés auxquelles je me suis heurté tout au long de ce projet : Premièrement le temps, j'ai du pafois passer plusieurs semaines sur un problème tout en rédigeant un rapport ce qui m'a empêché de me concentrer suffisamment dans le traitement de certaines mal-fonctionnalité qui se répéteraient plus tard. Deuxièmement la méthode pour pouvoir repérer et isoler une altération du système lors du développement. Quand il y avait des erreurs de mesures il fallait pouvoir réussir à découvrir si le problème venait du code 'c', VHDL, des FRAMS, de la FPGA, des pistes sur le PCB ou d'un autre composant Hardware. Il pouvait aussi venir de la génération du signal ou de la façon de l'analyser sur Matlab. Si certaines de ses possibilités pouvaient facilement être écartées ce n'est pas le cas pour les autres.

Tous ces points ont rendu ce travail de bachelor challengeant, complexe et passionnant et m'ont permis de comprendre certaines problématiques en premier lieu les deux citées le temps et la méthode. J'ai été amené à utiliser et à développer encore plus en profondeur beaucoup de connaissances acquises au cours de ma formation.

5.4 Remerciement

Je tiens à remercier:

- M. Amand Axel pour son aide concernant l'utilisation des programmes et de la configuration de la FPGA.
- Mme. Alexandra Andersson de m'avoir suivi pendant ce travail de bachelor.
- M. Pascal Sartoretti pour l'aide apporté sur les FRAMS
- M. Thomas Sterren & M. Medard Rieder pour leur aide sur le uContrôleur.
- M. Francois Corthay pour l'aide apporté sur la configuration de la flash
- M. Alban Theytaz & M. David Tagan pour le montage et les modifications sur le PCB.

Bibliography

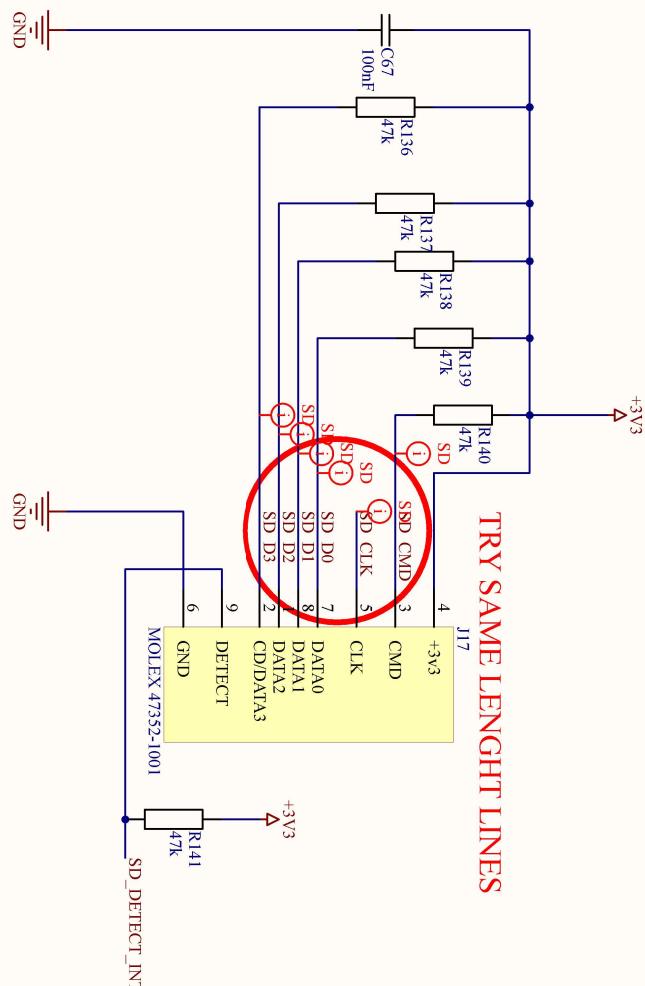
- [1] *MB85RS4MT Datasheet* Fujitsu, Download PDF. en. URL: <https://www.radiolocman.com/datasheet/data.html?di=533923&/MB85RS4MT> (visited on 08/17/2023).
- [2] *AD7380-4 Datasheet and Product Info* / Analog Devices. URL: <https://www.analog.com/en/products/ad7380-4.html#product-overview> (visited on 08/17/2023).
- [3] *STM32L496AE - Ultra-low-power with FPU Arm Cortex-M4 MCU 80 MHz with 512 kbytes of Flash memory, USB OTG, LCD, DFSDM* - STMicroelectronics. en. URL: <https://www.st.com/en/microcontrollers-microprocessors/stm32l496ae.html> (visited on 08/17/2023).
- [4] *ICE40LP1K-CM49 iCE40 LP/HX Family* from Lattice Semiconductor - VEKEMO FPGA. URL: https://www.vemeko.com/product/lattice-semiconductor-ice40lp1k-cm49-10369.html?gclid=CjwKCAjw5_GmBhBIEiwA5QSMxJoo3BrYXVrW37yBwE (visited on 08/17/2023).
- [5] *N25Q032A13ESECOF Electronic Distributor* | Micron Technology | Ariat-Tech.com. URL: <https://www.ariat-tech.com/part/micron-technology/N25Q032A13ESECOF?msclkid=879d4cbd7caa1d2c14362860a0058413> (visited on 08/17/2023).
- [6] *FPGA-TN-02001-3-3-i CE40-Programming-Configuration* - Technical Note FPGA-TN-02001-3. October 2021 - Studocu. en-us. URL: <https://www.studocu.com/en-us/document/gibson-southern-high-school/earth-science/fpga-tn-02001-3-3-i-ce40-programming-configuration/55660603> (visited on 08/17/2023).
- [7] *STMicroelectronics: Our technology starts with you.* en. URL: https://www.st.com/content/st_com/en.html (visited on 08/17/2023).
- [8] *ChatGPT*. URL: <https://chat.openai.com> (visited on 08/17/2023).
- [9] *Stack Overflow - Where Developers Learn, Share, & Build Careers.* en. URL: <https://stackoverflow.com/> (visited on 08/17/2023).
- [10] *LD29080PT33R*. Digi-Key Electronics. URL: <https://www.digikey.ch/en/products/detail/stmicroelectronics/LD29080PT33R/725232> (visited on 08/17/2023).
- [11] *ADR4525BRZ-R7 Electronic Distributor* | Ariat-Tech.com. URL: [https://www.ariat-tech.com/part/ADI\(Analog-Devices, Inc.\)/ADR4525BRZ-R7?msclkid=29df2cf81fa018789f9c4825c3254117](https://www.ariat-tech.com/part/ADI(Analog-Devices, Inc.)/ADR4525BRZ-R7?msclkid=29df2cf81fa018789f9c4825c3254117) (visited on 08/17/2023).
- [12] *ADA4500-2ARMZ-R7*. Digi-Key Electronics. URL: <https://www.digikey.ch/en/products/detail/analog-devices-inc/ADA4500-2ARMZ-R7/3820045> (visited on 08/17/2023).
- [13] *LM3880MFE-1AF/NOPB*. Digi-Key Electronics. URL: <https://www.digikey.ch/en/products/detail/texas-instruments/LM3880MFE-1AF-NOPB/2682782> (visited on 08/17/2023).

Bibliography

- [14] Office fédéral du développement territorial ARE. *Agenda 2030 pour le développement durable.* fr. URL: <https://www.are.admin.ch/are/fr/home/nachhaltige-entwicklung/internationale-zusammenarbeit/agenda-2030-fuer-nachhaltige-entwicklung.html> (visited on 08/17/2023).

A | Annexes

A.1 Shéma Altium





A

A

B

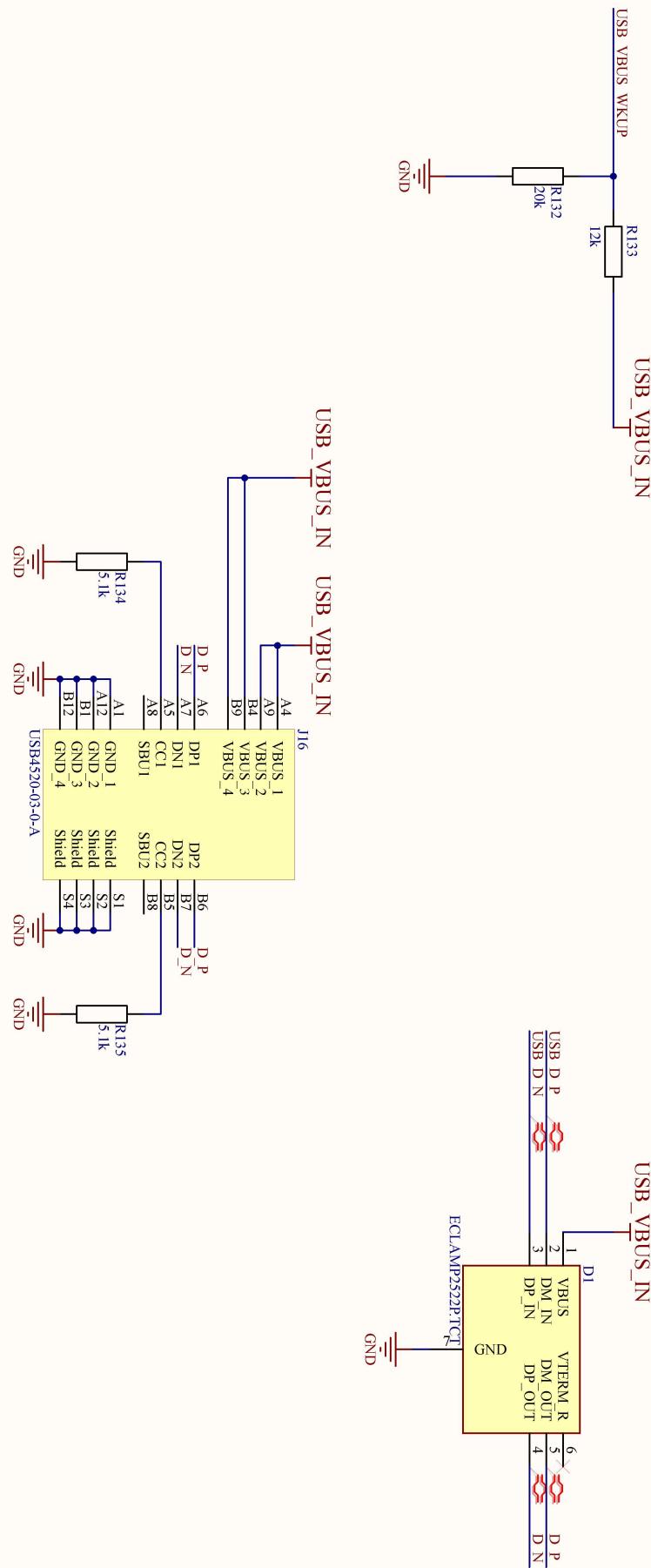
B

C

C

D

D



1

2

3

4

BalEv_PriPcb

BalEv_USB_v1_0.SchDoc

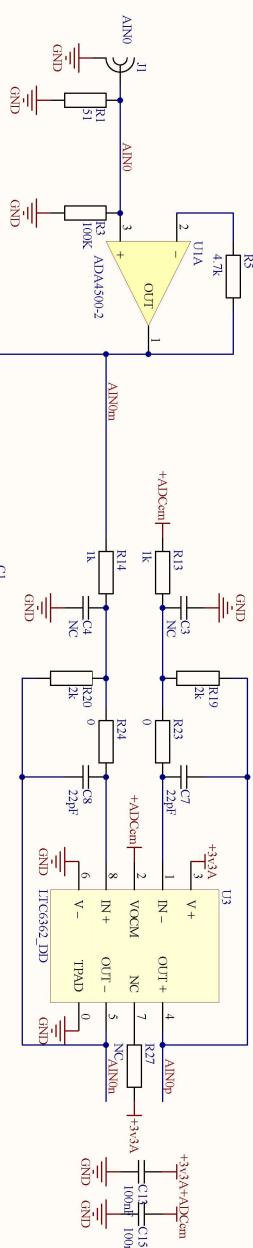
Revision: 1.0a

Sheet 3 of 9

Design by: anx/sap

C:\Users\christoph.grobety\bachelorChrisBachelor\TB_2023_LoggerPCB\BalEv_v1\BalEv_USB_v1_0.SchDoc

ANALOG INPUT 0

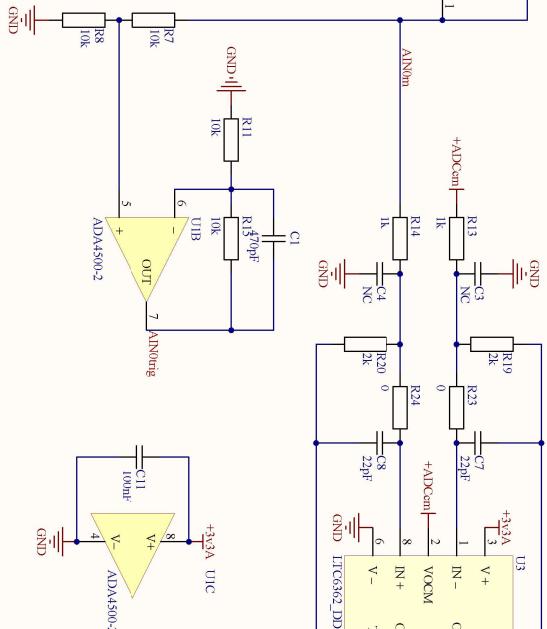


5

7

8

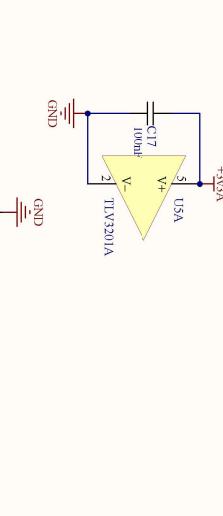
ANALOG INPUT 1



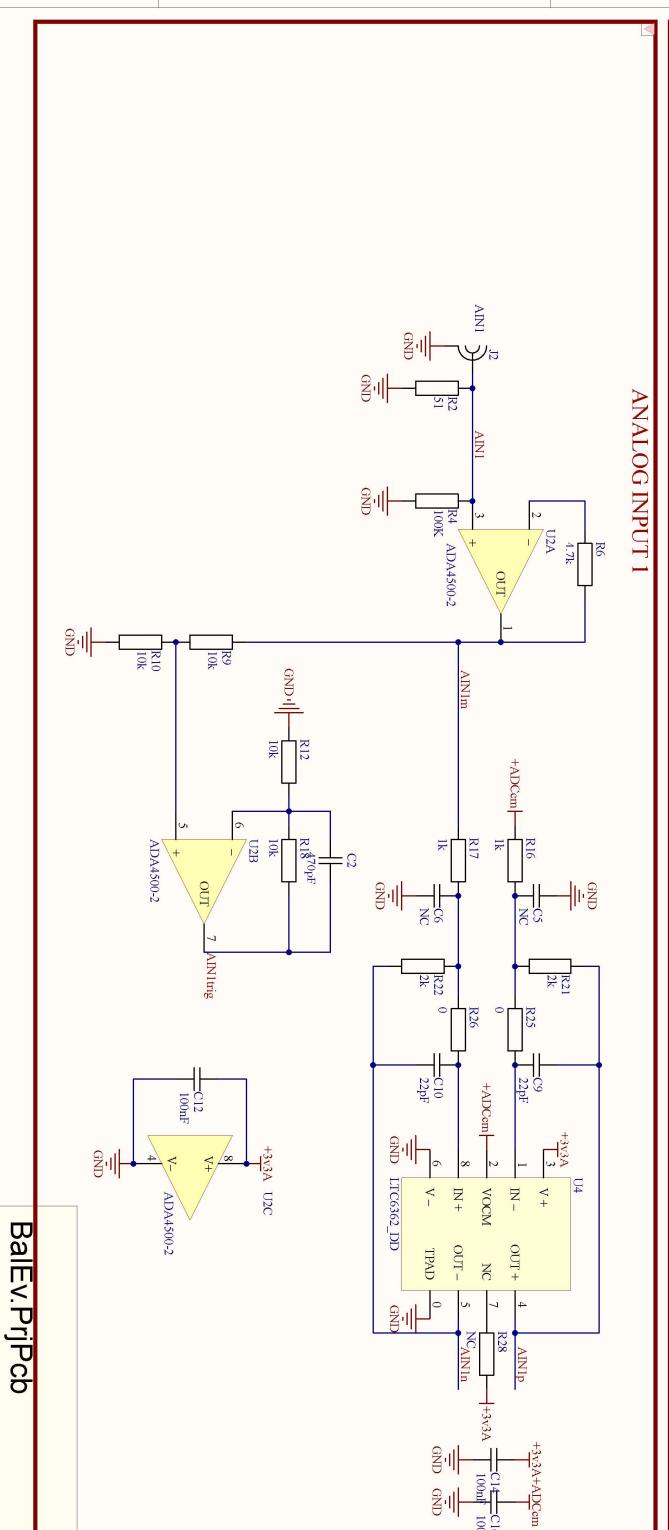
4

6

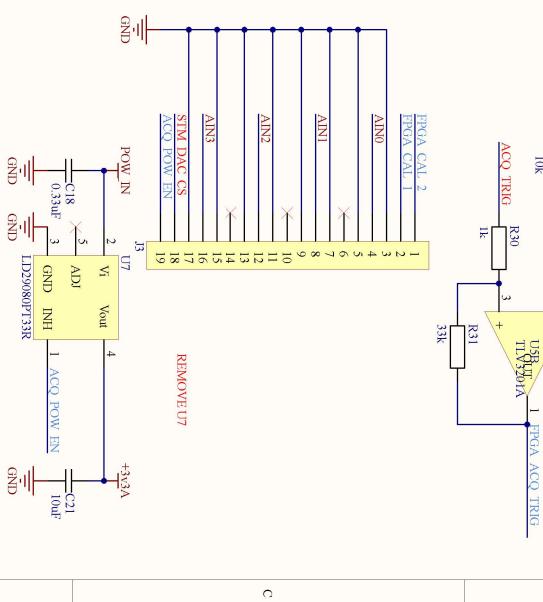
8



B



C



C

BalEv.PjPcb

BalEv_AINa_v1_0.SchDoc

Revision : 1.0a

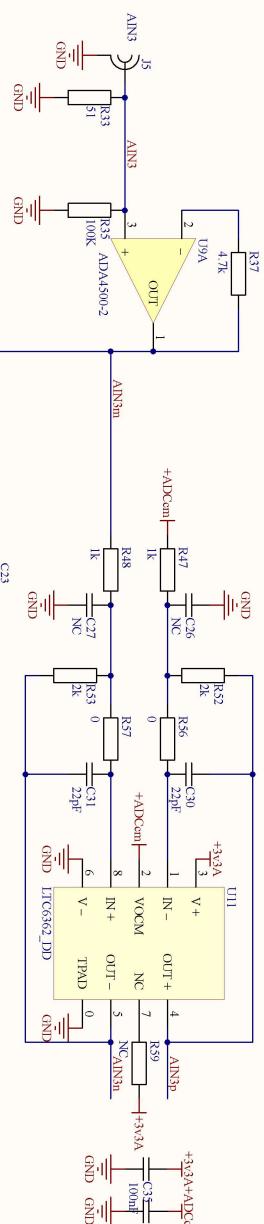
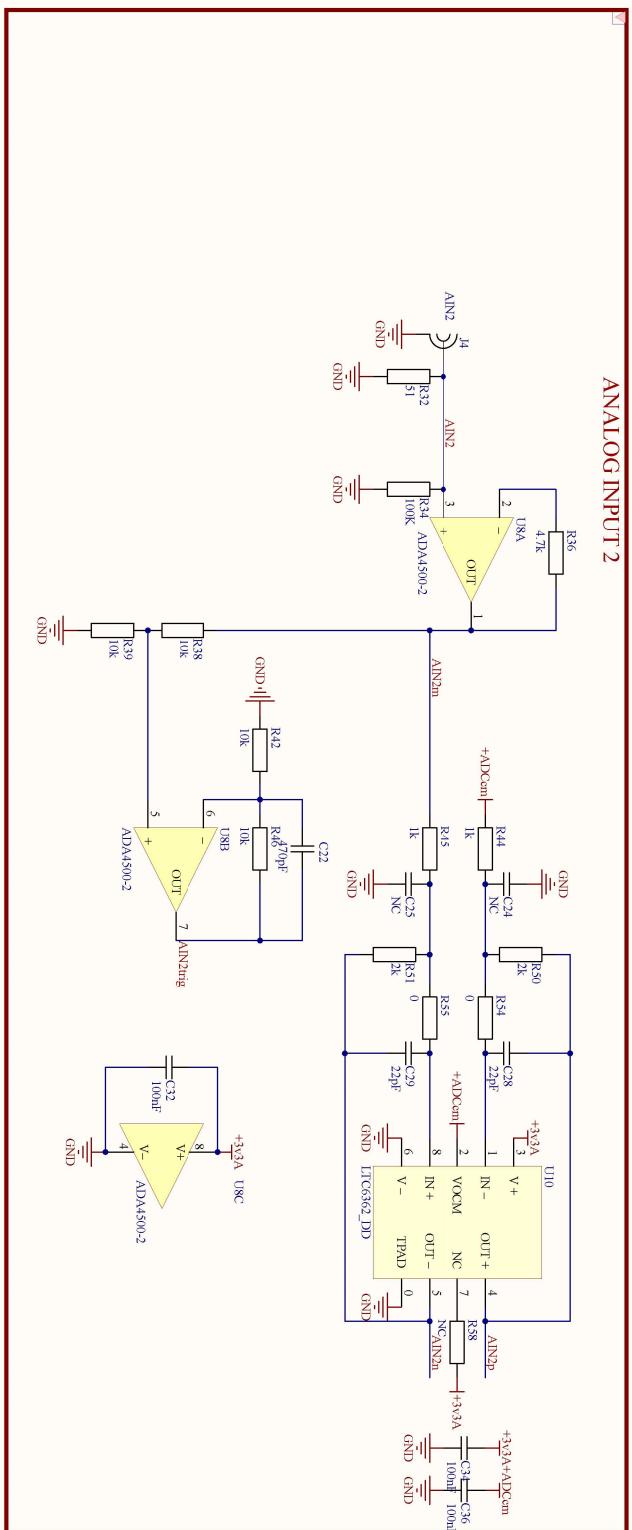
Sheet 6 of 9

**Hess-SO//VALAIS
WALLIS**

Date : 17.01.2022



D



ANALOG INPUT

BalEv.PrjPcb

BalEv AlNb v1 0.SchDoc

111

110

Hes-SO VALAIS WALLIS

Date : 17.01.2022

卷之三

1

D

1

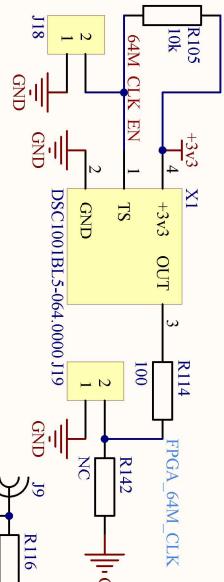
5

三

Signaux rose ->
connexions court.
Peut être échangé
sur le même BANK.

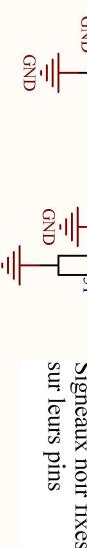
Rose foncé peut être
plus longues, garde
avec les autres rose.

FPGA_64M_CLK



U13B	BANK 0	
FPGA_FRAM_0_SCLK	R119	25
FPGA_FRAM_1_SCLK	R120	23
IOT_36b DIFF	IOT_37a	
FPGA_FRAM_2_SCLK	NQ61	27
FPGA_FRAM_3_SCLK	R122	26
IOT_38b DIFF	IOT_39a	
FPGA_ADC_SCLK	RQ63	28
FPGA_ADC_nSC	RQ64	31
FPGA_ADC_nSC	R125	32
IOT_42b DIFF	IOT_43a	
FPGA_TMHz_Meas	NQ66	34
FPGA_SCCLK_MEAS	R116	37
IOT_44b DIFF	IOT_45a	31
STM_FPGA_Reset	35	IOT_46b_G0
FPGA_ACQ_PRETRIG	36	IOT_48b DIFF
FPGA_ACQ_TRIGGER	43	IOT_49a
STM_FPGA_M2	38	IOT_50a DIFF
STM_FPGA_M3	42	IOT_51a
FPGA_ACQ_DONI39	RQ62	RGB2 DIFF
FPGA_CAL_2	40	RGB1 High Current
FPGA_CAL_1	41	RGB0

Resistors on this page could be m0603

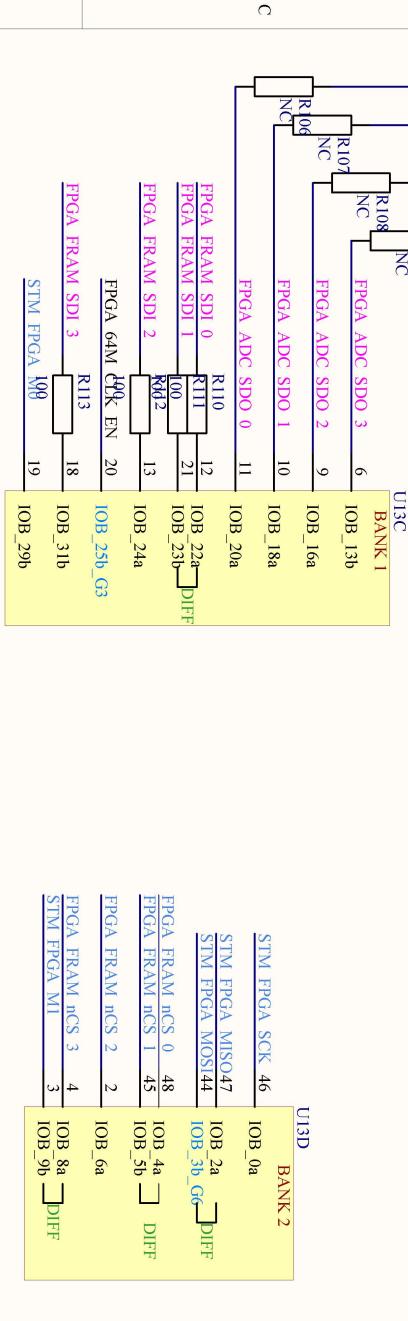


Signaux noir fixes
sur leurs pins

STM_FPGA_RESET	35	IOT_46b_G0
FPGA_ACQ_TRIGGER	43	IOT_48b DIFF
STM_FPGA_M2	38	IOT_49a
STM_FPGA_M3	42	IOT_50a DIFF
FPGA_ACQ_DONI39	RQ62	RGB2 DIFF
FPGA_CAL_2	40	RGB1 High Current
FPGA_CAL_1	41	RGB0

iCE40 Ultra / iCE5LP

Signaux bleu peut
être échangé



iCE40 Ultra / iCE5LP

iCE40 Ultra / iCE5LP

BalEv.PriPcb

BalEv_FPTG_IO_v1_0.SchDoc

Date : 17.01.2022

Revision : 1.0a

Sheet 8 of 9

Design by : anx/sap

C:\Users\christoph.grobety\bachelorChrisBachelor\TB_2023\LoggerPCB\BalEv_v1\BalEv_FPTG_IO_v1_0.SchDoc

A

1

A

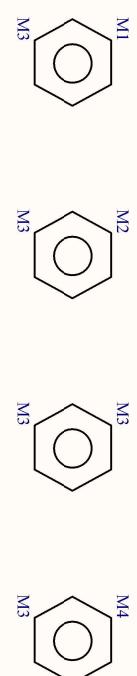
2

B

3

B

4



B

C

C

D

BalEv.PnjPcb

Hes-SO//VALAIS WALLIS

BalEv_Mechanic_v1_0.SchDoc

Date : 17.01.2022

π

Revision : 1.0a

Sheet 9 of 9

Design by: anx/sap

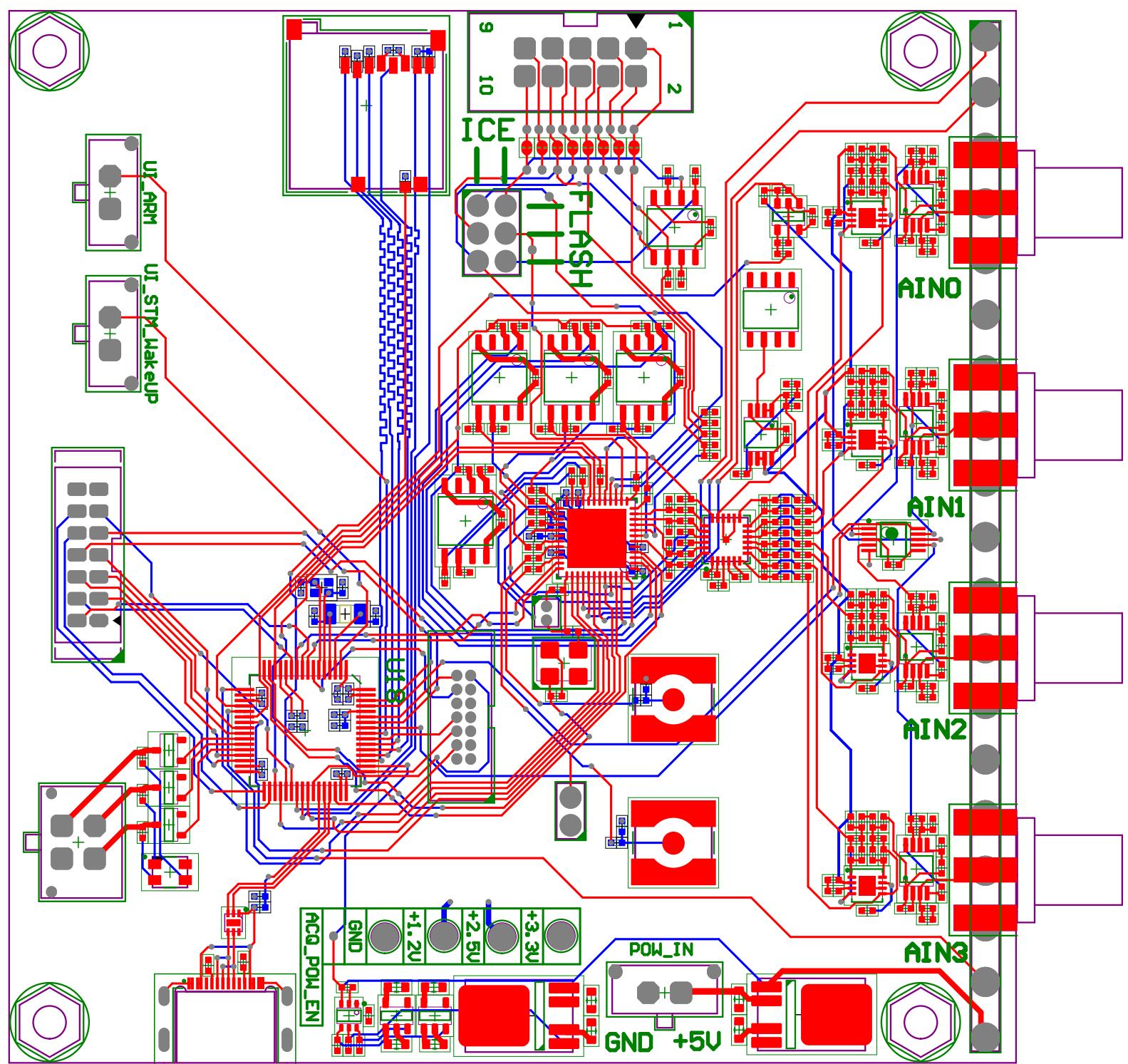
C:\Users\christoph.grobety\bachelorChrisBachelor\TB_2023_LoggerPCB\BalEv_v1\BalEv_Mechanic_v1_0.SchDoc

1

2

3

4



A.2 Code STM32

A.2.1 main.c

```

1  /* USER CODE BEGIN Header */
2  /**
3   * @file          : main.h
4   * @brief         : Header for main.c file.
5   *                   This file contains the common defines of the application.
6   * @attention
7   *
8   * Copyright (c) 2023 STMicroelectronics.
9   * All rights reserved.
10  *
11  * This software is licensed under terms that can be found in the LICENSE file
12  * in the root directory of this software component.
13  * If no LICENSE file comes with this software, it is provided AS-IS.
14  *
15  ****
16  */
17  /*
18  */
19  /* USER CODE END Header */
20
21  /* Define to prevent recursive inclusion -----*/
22  #ifndef __MAIN_H
23  #define __MAIN_H
24
25  #ifdef __cplusplus
26  extern "C" {
27  #endif
28
29  /* Includes -----*/
30  #include "stm32l4xx_hal.h"
31
32  /* Private includes -----*/
33  /* USER CODE BEGIN Includes */
34  #include <stdio.h>
35  #include <string.h>
36  #include <stdbool.h>
37  /* USER CODE END Includes */
38
39  /* Exported types -----*/
40  /* USER CODE BEGIN ET */
41  enum State {
42      FRAM_0,
43      FRAM_1,
44      FRAM_2,
45      FRAM_3,
46  };
47
48  enum color_Led {
49      RED,
50      BLUE,
51      GREEN,
52      OFF,
53  };
54
55  enum trig_channel {
56      CHANNEL_0,
57      CHANNEL_1,
58      CHANNEL_2,
59      CHANNEL_3,
60  };
61
62  extern const uint32_t TAB_SIZE;
63
64  extern enum color_Led LED_STATE;
65
66  extern bool reset;
67  extern bool start;
68  extern bool acqu;
69
70  /* USER CODE END ET */
71
72  /* Exported constants -----*/
73  /* USER CODE BEGIN EC */

```

```

74
75 /* USER CODE END EC */
76
77 /* Exported macro -----*/
78 /* USER CODE BEGIN EM */
79
80 /* USER CODE END EM */
81
82 /* Exported functions prototypes -----*/
83 void Error_Handler(void);
84
85 /* USER CODE BEGIN EFP */
86
87 /* USER CODE END EFP */
88
89 /* Private defines -----*/
90 #define WAKE_UP_Pin GPIO_PIN_13
91 #define WAKE_UP_GPIO_Port GPIOC
92 #define WAKE_UP_EXTI_IRQn EXTI15_10_IRQn
93 #define FPGA_RESET_Pin GPIO_PIN_0
94 #define FPGA_RESET_GPIO_Port GPIOC
95 #define STM_UI_MOSI_Pin GPIO_PIN_1
96 #define STM_UI_MOSI_GPIO_Port GPIOC
97 #define FPGA_PPRETRIG_Pin GPIO_PIN_2
98 #define FPGA_PPRETRIG_GPIO_Port GPIOC
99 #define SELECTOR_M0_Pin GPIO_PIN_0
100 #define SELECTOR_M0_GPIO_Port GPIOA
101 #define STM_FPGA_SCK_Pin GPIO_PIN_1
102 #define STM_FPGA_SCK_GPIO_Port GPIOA
103 #define SELECTOR_M1_Pin GPIO_PIN_2
104 #define SELECTOR_M1_GPIO_Port GPIOA
105 #define SELECTOR_M2_Pin GPIO_PIN_3
106 #define SELECTOR_M2_GPIO_Port GPIOA
107 #define STM_FPGA_MISO_Pin GPIO_PIN_6
108 #define STM_FPGA_MISO_GPIO_Port GPIOA
109 #define STM_FPGA_MOSI_Pin GPIO_PIN_7
110 #define STM_FPGA_MOSI_GPIO_Port GPIOA
111 #define UI_LED_G_Pin GPIO_PIN_0
112 #define UI_LED_G_GPIO_Port GPIOB
113 #define UI_LED_R_Pin GPIO_PIN_1
114 #define UI_LED_R_GPIO_Port GPIOB
115 #define UI_LED_B_Pin GPIO_PIN_2
116 #define UI_LED_B_GPIO_Port GPIOB
117 #define STM_UI_SCK_Pin GPIO_PIN_10
118 #define STM_UI_SCK_GPIO_Port GPIOB
119 #define FPGA_DONE_Pin GPIO_PIN_15
120 #define FPGA_DONE_GPIO_Port GPIOB
121 #define FPGA_DONE_EXTI_IRQn EXTI15_10_IRQn
122 #define SD_DETECT_INT_Pin GPIO_PIN_7
123 #define SD_DETECT_INT_GPIO_Port GPIOC
124 #define SELECTOR_M3_Pin GPIO_PIN_8
125 #define SELECTOR_M3_GPIO_Port GPIOA
126 #define USB_VBUS_INT_Pin GPIO_PIN_10
127 #define USB_VBUS_INT_GPIO_Port GPIOA
128 #define USB_VBUS_INT_EXTI_IRQn EXTI15_10_IRQn
129 #define ARM_BUTTON_Pin GPIO_PIN_4
130 #define ARM_BUTTON_GPIO_Port GPIOB
131 #define ARM_BUTTON_EXTI_IRQn EXTI4_IRQn
132 #define TRIG_SRC0_Pin GPIO_PIN_5
133 #define TRIG_SRC0_GPIO_Port GPIOB
134 #define TRIG_SRC1_Pin GPIO_PIN_8
135 #define TRIG_SRC1_GPIO_Port GPIOB
136 /* USER CODE BEGIN Private defines */
137 #define MEMORY_SIZE 524288
138 // #define TAB_SIZE 87381
139 #define DAC_MAX_VALUE 4096
140 #define VREF_VOLTAGE 3.3
141 /* USER CODE END Private defines */
142
143 #ifdef __cplusplus
144 }
145#endif
146
```

147 #endif /* __MAIN_H */
148

A.2.2 main.h

```

1  /* USER CODE BEGIN Header */
2  /**
3   * @file          : main.c
4   * @brief         : Main program body
5   * @attention
6   *
7   * Copyright (c) 2023 STMicroelectronics.
8   * All rights reserved.
9   *
10  * This software is licensed under terms that can be found in the LICENSE file
11  * in the root directory of this software component.
12  * If no LICENSE file comes with this software, it is provided AS-IS.
13  *
14  */
15  /**
16  */
17  /**
18  /* USER CODE END Header */
19  /* Includes -----*/
20 #include "main.h"
21 #include "fatfs.h"
22
23 /* Private includes -----*/
24 /* USER CODE BEGIN Includes */
25
26 #include "fram_func.h"
27 #include "sd_card_func.h"
28 /* USER CODE END Includes */
29
30 /* Private typedef -----*/
31 /* USER CODE BEGIN PTD */
32
33 /* USER CODE END PTD */
34
35 /* Private define -----*/
36 /* USER CODE BEGIN PD */
37 // #define MEMORY_SIZE    524288
38 /* USER CODE END PD */
39
40 /* Private macro -----*/
41 /* USER CODE BEGIN PM */
42
43 /* USER CODE END PM */
44
45 /* Private variables -----*/
46 DAC_HandleTypeDef hdac1;
47
48 I2C_HandleTypeDef hi2c1;
49 I2C_HandleTypeDef hi2c2;
50
51 SD_HandleTypeDef hsd1;
52
53 SPI_HandleTypeDef hspi1;
54 SPI_HandleTypeDef hspi2;
55 DMA_HandleTypeDef hdma_spi1_rx;
56 DMA_HandleTypeDef hdma_spi1_tx;
57
58 TIM_HandleTypeDef htim16;
59
60 UART_HandleTypeDef huart1;
61
62 HCD_HandleTypeDef hhcd_USB_OTG_FS;
63
64 /* USER CODE BEGIN PV */
65
66 /* USER CODE END PV */
67
68 /* Private function prototypes -----*/
69 void SystemClock_Config(void);
70 static void MX_GPIO_Init(void);
71 static void MX_DAC1_Init(void);
72 static void MX_SDMMC1_SD_Init(void);
73 static void MX_SPI1_Init(void);

```

```

74 static void MX_USART1_UART_Init(void);
75 static void MX_I2C1_Init(void);
76 static void MX_I2C2_Init(void);
77 static void MX_SPI2_Init(void);
78 static void MX_DMA_Init(void);
79 static void MX_USB_OTG_FS_HCD_Init(void);
80 static void MX_TIM16_Init(void);
81 /* USER CODE BEGIN PFP */
82
83 /* USER CODE END PFP */
84
85 /* Private user code -----*/
86 /* USER CODE BEGIN 0 */
87 const uint32_t TAB_SIZE = (MEMORY_SIZE/16);
88 enum color_Led LED_STATE = OFF;
89 bool reset = true;
90 bool start = true;
91 bool acqu = true;
92 /* USER CODE END 0 */
93
94 /**
95 * @brief The application entry point.
96 * @retval int
97 */
98 int main(void)
99 {
100     /* USER CODE BEGIN 1 */
101     //FRESULT res; /* FatFs function common result code */
102     TCHAR* fileName0 ;
103     TCHAR* fileName1 ;
104     TCHAR* fileName2 ;
105     TCHAR* fileName3 ;
106     TCHAR* file_Init = "INIT.txt";
107     uint8_t TX_Data_ADC[2];
108     uint8_t TX_FRAM_TEST_R[10] ;
109     uint16_t DATA_SD;
110     uint16_t DATA_ind [TAB_SIZE/2];
111     uint8_t pretrigValue = 100;
112
113
114     //uint8_t RX_DATA [524288];    -> TOO BIG
115
116     /* USER CODE END 1 */
117
118     /* MCU Configuration-----*/
119
120     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
121     HAL_Init();
122
123     /* USER CODE BEGIN Init */
124
125     /* USER CODE END Init */
126
127     /* Configure the system clock */
128     SystemClock_Config();
129
130     /* USER CODE BEGIN SysInit */
131
132     /* USER CODE END SysInit */
133
134     /* Initialize all configured peripherals */
135     MX_GPIO_Init();
136     MX_DAC1_Init();
137     MX_SDMMC1_SD_Init();
138     MX_SPI1_Init();
139     MX_USART1_UART_Init();
140     MX_I2C1_Init();
141     MX_I2C2_Init();
142     MX_SPI2_Init();
143     MX_DMA_Init();
144     MX_USB_OTG_FS_HCD_Init();
145     MX_FATFS_Init();
146     MX_TIM16_Init();

```

```

147 /* USER CODE BEGIN 2 */
148
149 /* USER CODE END 2 */
150
151 /* Infinite loop */
152 /* USER CODE BEGIN WHILE */
153     uint8_t start_Add[4];
154     uint32_t start_Add_tot = 0;
155     uint32_t Add_ = 0;
156     uint8_t TX_32 = 0;
157     uint16_t boucle = 0;
158     uint8_t wtext_0 [100];
159     uint8_t wtext_1 [100];
160     uint8_t wtext_2 [100];
161     uint8_t wtext_3 [100];
162     uint8_t* receive;
163     uint8_t chann;
164     double* dataInit_;
165     double triggDacValue;
166     PIN_reset();
167 //FRAM_device(hspi1);
168 while (1)
169 {
170     acqu = true;
171     TX_32 = 0;
172     start = true;
173     HAL_SuspendTick();
174     __WFI();
175
176 //-----RESET
177 //FPGA-----
178 HAL_GPIO_WritePin(FPGA_RESET_GPIO_Port, FPGA_RESET_Pin, GPIO_PIN_RESET);
179 HAL_Delay(100);
180 HAL_GPIO_WritePin(FPGA_RESET_GPIO_Port, FPGA_RESET_Pin, GPIO_PIN_SET);
181 //-----RESET
182 M-----
183 PIN_reset();
184 //-----INIT-----
185 //-----init file sd -----
186 sprintf((char*)wtext_0, "FRAM0_%d.bin", boucle);
187 fileName0 = (TCHAR*)wtext_0;
188 SD_create_file(SDFile, fileName0);
189 sprintf((char*)wtext_1, "FRAM1_%d.bin", boucle);
190 fileName1 = (TCHAR*)wtext_1;
191 SD_create_file(SDFile, fileName1);
192 sprintf((char*)wtext_2, "FRAM2_%d.bin", boucle);
193 fileName2 = (TCHAR*)wtext_2;
194 SD_create_file(SDFile, fileName2);
195 sprintf((char*)wtext_3, "FRAM3_%d.bin", boucle);
196 fileName3 = (TCHAR*)wtext_3;
197 SD_create_file(SDFile, fileName3);
198 //-----init
199 value-----
200 dataInit_ = initValue(SDFile, file_Init);
201
202 chann = (uint8_t) dataInit_[0];
203 if(chann >4) {
204     chann = 0;
205 }
206 triggDacValue = dataInit_[1]* (0.02)+1;
207 if(triggDacValue <1 || triggDacValue>3.3) {
208     triggDacValue = 1;
209 }
210 pretrigValue = (uint8_t) dataInit_[2];
211 if(pretrigValue >100) {
212     pretrigValue = 100;
213 }
214 //-----LED
215 CONTROL-----
216 if(LED_STATE != RED) {

```

```

214     LED_on(BLUE);
215 }
216 else{
217     while(1){
218         //reset the systeme
219     }
220 }
221
222 //-----init trigg dc value-----
223 uint32_t triggDac = (uint32_t)((triggDacValue/VREF_VOLTAGE)*DAC_MAX_VALUE);
224 HAL_DAC_SetValue(&hdac1, DAC1_CHANNEL_2, DAC_ALIGN_12B_R, triggDac);
225 HAL_DAC_Start(&hdac1, DAC1_CHANNEL_2);
226 HAL_DAC_SetValue(&hdac1, DAC1_CHANNEL_1, DAC_ALIGN_12B_R, triggDac);
227 HAL_DAC_Start(&hdac1, DAC1_CHANNEL_1);
228
229 //-----init pretrigg value-----
230 HAL_SPI_Transmit(&hspil, (uint8_t*)&pretrigValue,1, 100);
231 HAL_GPIO_WritePin(FPGA_PRETRIG_GPIO_Port, FPGA_PRETRIG_Pin, GPIO_PIN_SET);
232 HAL_SPI_Transmit(&hspil, (uint8_t*)&pretrigValue,1, 100);
233 HAL_GPIO_WritePin(FPGA_PRETRIG_GPIO_Port, FPGA_PRETRIG_Pin, GPIO_PIN_RESET);
234
235 //-----init trigg channel -----
236 setTriggChannel(chann);
237 //-----RESET FRAM
238 REG-----  

239 FRAM_reset_reg(hspil);
240 //-----WRITE FRAM
241 REG-----  

242 FRAM_write_reg(hspil);
243 //-----ADC WRITE
244 REG-----  

245 TX_Data_ADC[0] = 0xA2;
246 TX_Data_ADC[1] = 0x80;
247 HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_SET);
248 HAL_SPI_Transmit(&hspil, (uint8_t*)TX_Data_ADC,2, 100);
249 HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_RESET);
250 HAL_Delay(100);
251
252 //-----WRITE
253 FRAM-----  

254 PIN_reset();
255 TX_FRAM_TEST_R[0] = 2;
256 for(int i =1; i<5;i++){
257     TX_FRAM_TEST_R[i] = 0;
258 }
259 TX_FRAM_TEST_R[3] = 0;
260 TX_FRAM_TEST_R[4] = 0;
261 TX_FRAM_TEST_R[5] = 56;
262
263 HAL_GPIO_WritePin(SELECTOR_M3_GPIO_Port, SELECTOR_M3_Pin, GPIO_PIN_SET);
264 HAL_SPI_Transmit(&hspil, (uint8_t*)TX_FRAM_TEST_R,6, 100);
265 HAL_Delay(100);
266
267 //-----ADC TO
268 FRAM-----  

269 HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_SET);
270 //-----SLEEP
271 MODE-----  

272 //HAL_PWR_EnterSLEEPMode(PWR_LOWPOWERREGULATOR_ON, PWR_SLEEPENTRY_WFI);
273 while(acqu){
274     HAL_SuspendTick();
275     __WFI();
276 }
277 //HAL_PWR_ENTER
278 PIN_reset();
279 while(TX_32 <16) {
280     if(TX_32 == 0){
281         PIN_reset();
282         HAL_GPIO_WritePin(SELECTOR_M0_GPIO_Port, SELECTOR_M0_Pin, GPIO_PIN_SET);
283         HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_SET);
284         HAL_GPIO_WritePin(SELECTOR_M3_GPIO_Port, SELECTOR_M3_Pin, GPIO_PIN_SET);
285         HAL_SPI_Receive(&hspil, (uint8_t*)start_Add,4, 100);
286         HAL_Delay(100);

```

```

281     start_Add_tot = (((uint32_t)start_Add[0])<<24)+(((uint32_t)start_Add[1]
282                                         ])<<16)
283                                         +(((uint32_t)start_Add[2])<<8)+((uint32_t)
284                                         start_Add[3]);
285                                         PIN_reset();
286                                         start_Add_tot = start_Add_tot>>12;
287                                         //start_Add_tot = 0;
288                                         HAL_TIM_Base_Start_IT(&htim16);
289 }
290 //-----READ FRAM
291 0-----
292 for(uint32_t i = 0; i<(TAB_SIZE/2) ; i++){
293     Add_ = i*2+TAB_SIZE*TX_32+(MEMORY_SIZE);
294     if(Add_ > MEMORY_SIZE){
295         Add_ = Add_ - MEMORY_SIZE;
296     }
297     //receive = FRAM_read(FRAM_0,i*2+TAB_SIZE*TX_32, hspil, 2);
298     receive = FRAM_read(FRAM_0, Add_, hspil, 2);
299     DATA_SD = (((uint16_t)receive[0])<<8)+(uint16_t)(receive[1]);
300     // DATA_SD = (((uint16_t)RX_FAST[i*2])<<8)+(uint16_t)(RX_FAST[i*2+1]);
301     DATA_ind[i] = DATA_SD;
302 }
303 SD_write_data(SDFile, fileName0, DATA_ind);
304 //-----READ FRAM
305 1-----
306 for(uint32_t i = 0; i<(TAB_SIZE/2) ; i++){
307     Add_ = i*2+TAB_SIZE*TX_32+(MEMORY_SIZE);
308     if(Add_ > MEMORY_SIZE){
309         Add_ = Add_ - MEMORY_SIZE;
310     }
311     receive = FRAM_read(FRAM_1, Add_, hspil, 2);
312     DATA_SD = (((uint16_t)receive[0])<<8)+(uint16_t)(receive[1]);
313     //DATA_SD = (short)DATA_SD;
314     DATA_ind[i] = DATA_SD;
315 }
316 SD_write_data(SDFile, fileName1, DATA_ind);
317 //-----READ FRAM
318 2-----
319 for(uint32_t i = 0; i<TAB_SIZE/2 ; i++){
320     Add_ = i*2+TAB_SIZE*TX_32+(MEMORY_SIZE-start_Add_tot);
321     if(Add_ > MEMORY_SIZE){
322         Add_ = Add_ - MEMORY_SIZE;
323     }
324     receive = FRAM_read(FRAM_2,Add_, hspil, 2);
325     DATA_SD = (((uint16_t)receive[0])<<8)+(uint16_t)(receive[1]);
326     //DATA_SD = (short)DATA_SD;
327     DATA_ind[i] = DATA_SD;
328 }
329 SD_write_data(SDFile, fileName2, DATA_ind);
330 //-----READ FRAM
331 3-----
332 for(uint32_t i = 0; i<TAB_SIZE/2 ; i++){
333     Add_ = i*2+TAB_SIZE*TX_32+(MEMORY_SIZE-start_Add_tot);
334     if(Add_ > MEMORY_SIZE){
335         Add_ = Add_ - MEMORY_SIZE;
336     }
337     receive = FRAM_read(FRAM_3,Add_, hspil, 2);
338     DATA_SD = (((uint16_t)receive[0])<<8)+(uint16_t)(receive[1]);
339     //DATA_SD = (short)DATA_SD;
340     DATA_ind[i] = DATA_SD;
341 }
342 SD_write_data(SDFile, fileName3, DATA_ind);
343 TX_32++;
344 //receive = FRAM_read(FRAM_3, TX_32, hspil, 2);
345 //DATA_SD = (((uint16_t)receive[0])<<8)+(uint16_t)(receive[1]);
346 //DATA_SD = (short)DATA_SD;
347 //SD_write_data(SDFile, fileName3, DATA_SD);
348 }
349 if(TX_32==16){
350     HAL_TIM_Base_Stop(&htim16);
351     if(LED_STATE != RED){
352         LED_off();
353         LED_on(GREEN); //6

```

```

348     }
349     else{
350         LED_off();
351         LED_on(RED);
352     }
353     boucle++;
354 /* USER CODE END WHILE */
355
356 /* USER CODE BEGIN 3 */
357 }
358 /* USER CODE END 3 */
359 }
360
361 /**
362 * @brief System Clock Configuration
363 * @retval None
364 */
365 void SystemClock_Config(void)
366 {
367     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
368     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
369
370     /** Configure the main internal regulator output voltage
371     */
372     if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
373     {
374         Error_Handler();
375     }
376
377     /** Initializes the RCC Oscillators according to the specified parameters
378     * in the RCC_OscInitTypeDef structure.
379     */
380     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI48|RCC_OSCILLATORTYPE_MSI;
381     RCC_OscInitStruct.HSI48State = RCC_HSI48_ON;
382     RCC_OscInitStruct.MSISState = RCC_MSIS_ON;
383     RCC_OscInitStruct.MSICalibrationValue = 0;
384     RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
385     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
386     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_MSI;
387     RCC_OscInitStruct.PLL.PLLM = 1;
388     RCC_OscInitStruct.PLL.PLLN = 32;
389     RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
390     RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
391     RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
392     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
393     {
394         Error_Handler();
395     }
396
397     /** Initializes the CPU, AHB and APB buses clocks
398     */
399     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
400             |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
401     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
402     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
403     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
404     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
405
406     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
407     {
408         Error_Handler();
409     }
410 }
411
412 /**
413 * @brief DAC1 Initialization Function
414 * @param None
415 * @retval None
416 */
417
418 static void MX_DAC1_Init(void)
419 {
420

```

```

421 /* USER CODE BEGIN DAC1_Init_0 */
422
423 /* USER CODE END DAC1_Init_0 */
424
425 DAC_ChannelConfTypeDef sConfig = {0};
426
427 /* USER CODE BEGIN DAC1_Init_1 */
428
429 /* USER CODE END DAC1_Init_1 */
430
431 /** DAC Initialization
432 */
433 hdac1.Instance = DAC1;
434 if (HAL_DAC_Init(&hdac1) != HAL_OK)
435 {
436     Error_Handler();
437 }
438
439 /** DAC channel OUT1 config
440 */
441 sConfig.DAC_SampleAndHold = DAC_SAMPLEANDHOLD_DISABLE;
442 sConfig.DAC_Trigger = DAC_TRIGGER_NONE;
443 sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;
444 sConfig.DAC_ConnectOnChipPeripheral = DAC_CHIPCONNECT_DISABLE;
445 sConfig.DAC_UserTrimming = DAC_TRIMMING_FACTORY;
446 if (HAL_DAC_ConfigChannel(&hdac1, &sConfig, DAC_CHANNEL_1) != HAL_OK)
447 {
448     Error_Handler();
449 }
450
451 /** DAC channel OUT2 config
452 */
453 if (HAL_DAC_ConfigChannel(&hdac1, &sConfig, DAC_CHANNEL_2) != HAL_OK)
454 {
455     Error_Handler();
456 }
457 /* USER CODE BEGIN DAC1_Init_2 */
458
459 /* USER CODE END DAC1_Init_2 */
460
461 }
462
463 /**
464 * @brief I2C1 Initialization Function
465 * @param None
466 * @retval None
467 */
468 static void MX_I2C1_Init(void)
469 {
470
471     /* USER CODE BEGIN I2C1_Init_0 */
472
473     /* USER CODE END I2C1_Init_0 */
474
475     /* USER CODE BEGIN I2C1_Init_1 */
476
477     /* USER CODE END I2C1_Init_1 */
478     hi2c1.Instance = I2C1;
479     hi2c1.Init.Timing = 0x10707DBC;
480     hi2c1.Init.OwnAddress1 = 0;
481     hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
482     hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
483     hi2c1.Init.OwnAddress2 = 0;
484     hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
485     hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
486     hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
487     if (HAL_I2C_Init(&hi2c1) != HAL_OK)
488     {
489         Error_Handler();
490     }
491
492     /** Configure Analogue filter
493 */

```

```

494     if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
495     {
496         Error_Handler();
497     }
498
499     /** Configure Digital filter
500     */
501     if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
502     {
503         Error_Handler();
504     }
505     /* USER CODE BEGIN I2C1_Init 2 */
506
507     /* USER CODE END I2C1_Init 2 */
508
509 }
510
511 /**
512 * @brief I2C2 Initialization Function
513 * @param None
514 * @retval None
515 */
516 static void MX_I2C2_Init(void)
517 {
518
519     /* USER CODE BEGIN I2C2_Init 0 */
520
521     /* USER CODE END I2C2_Init 0 */
522
523     /* USER CODE BEGIN I2C2_Init 1 */
524
525     /* USER CODE END I2C2_Init 1 */
526     hi2c2.Instance = I2C2;
527     hi2c2.Init.Timing = 0x10707DBC;
528     hi2c2.Init.OwnAddress1 = 0;
529     hi2c2.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
530     hi2c2.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
531     hi2c2.Init.OwnAddress2 = 0;
532     hi2c2.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
533     hi2c2.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
534     hi2c2.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
535     if (HAL_I2C_Init(&hi2c2) != HAL_OK)
536     {
537         Error_Handler();
538     }
539
540     /** Configure Analogue filter
541     */
542     if (HAL_I2CEx_ConfigAnalogFilter(&hi2c2, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
543     {
544         Error_Handler();
545     }
546
547     /** Configure Digital filter
548     */
549     if (HAL_I2CEx_ConfigDigitalFilter(&hi2c2, 0) != HAL_OK)
550     {
551         Error_Handler();
552     }
553     /* USER CODE BEGIN I2C2_Init 2 */
554
555     /* USER CODE END I2C2_Init 2 */
556
557 }
558
559 /**
560 * @brief SDMMC1 Initialization Function
561 * @param None
562 * @retval None
563 */
564 static void MX_SDMMC1_SD_Init(void)
565 {
566

```

```

567 /* USER CODE BEGIN SDMMC1_Init 0 */
568
569 /* USER CODE END SDMMC1_Init 0 */
570
571 /* USER CODE BEGIN SDMMC1_Init 1 */
572
573 /* USER CODE END SDMMC1_Init 1 */
574 hsd1.Instance = SDMMC1;
575 hsd1.Init.ClockEdge = SDMMC_CLOCK_EDGE_RISING;
576 hsd1.Init.ClockBypass = SDMMC_CLOCK_BYPASS_DISABLE;
577 hsd1.Init.ClockPowerSave = SDMMC_CLOCK_POWER_SAVE_DISABLE;
578 hsd1.Init.BusWide = SDMMC_BUS_WIDE_1B;
579 hsd1.Init.HardwareFlowControl = SDMMC_HARDWARE_FLOW_CONTROL_DISABLE;
580 hsd1.Init.ClockDiv = 4;
581 /* USER CODE BEGIN SDMMC1_Init 2 */
582
583 /* USER CODE END SDMMC1_Init 2 */
584
585 }
586
587 /**
588 * @brief SPI1 Initialization Function
589 * @param None
590 * @retval None
591 */
592 static void MX_SPI1_Init(void)
593 {
594
595 /* USER CODE BEGIN SPI1_Init 0 */
596
597 /* USER CODE END SPI1_Init 0 */
598
599 /* USER CODE BEGIN SPI1_Init 1 */
600
601 /* USER CODE END SPI1_Init 1 */
602 /* SPI1 parameter configuration*/
603 hspi1.Instance = SPI1;
604 hspi1.Init.Mode = SPI_MODE_MASTER;
605 hspi1.Init.Direction = SPI_DIRECTION_2LINES;
606 hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
607 hspi1.Init.CLKPolarity = SPI_POLARITY_HIGH;
608 hspi1.Init.CLKPhase = SPI_PHASE_2EDGE;
609 hspi1.Init.NSS = SPI_NSS_SOFT;
610 hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
611 hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
612 hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
613 hspi1.Init.CRCCalculation = SPI_CRCALCULATION_DISABLE;
614 hspi1.Init.CRCPolynomial = 7;
615 hspi1.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
616 hspi1.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
617 if (HAL_SPI_Init(&hspi1) != HAL_OK)
618 {
619     Error_Handler();
620 }
621 /* USER CODE BEGIN SPI1_Init 2 */
622
623 /* USER CODE END SPI1_Init 2 */
624
625 }
626
627 /**
628 * @brief SPI2 Initialization Function
629 * @param None
630 * @retval None
631 */
632 static void MX_SPI2_Init(void)
633 {
634
635 /* USER CODE BEGIN SPI2_Init 0 */
636
637 /* USER CODE END SPI2_Init 0 */
638
639 /* USER CODE BEGIN SPI2_Init 1 */

```

```

640
641     /* USER CODE END SPI2_Init 1 */
642     /* SPI2 parameter configuration*/
643     hspi2.Instance = SPI2;
644     hspi2.Init.Mode = SPI_MODE_MASTER;
645     hspi2.Init.Direction = SPI_DIRECTION_1LINE;
646     hspi2.Init.DataSize = SPI_DATASIZE_4BIT;
647     hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
648     hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
649     hspi2.Init.NSS = SPI_NSS_SOFT;
650     hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
651     hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
652     hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
653     hspi2.Init.CRCCalculation = SPI_CRCALCULATION_DISABLE;
654     hspi2.Init.CRCPolynomial = 7;
655     hspi2.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
656     hspi2.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
657     if (HAL_SPI_Init(&hspi2) != HAL_OK)
658     {
659         Error_Handler();
660     }
661     /* USER CODE BEGIN SPI2_Init 2 */
662
663     /* USER CODE END SPI2_Init 2 */
664
665 }
666
667 /**
668 * @brief TIM16 Initialization Function
669 * @param None
670 * @retval None
671 */
672 static void MX_TIM16_Init(void)
673 {
674
675     /* USER CODE BEGIN TIM16_Init 0 */
676
677     /* USER CODE END TIM16_Init 0 */
678
679     /* USER CODE BEGIN TIM16_Init 1 */
680
681     /* USER CODE END TIM16_Init 1 */
682     htim16.Instance = TIM16;
683     htim16.Init.Prescaler = 6400-1;
684     htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
685     htim16.Init.Period = 1000;
686     htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
687     htim16.Init.RepetitionCounter = 0;
688     htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
689     if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
690     {
691         Error_Handler();
692     }
693     /* USER CODE BEGIN TIM16_Init 2 */
694
695     /* USER CODE END TIM16_Init 2 */
696
697 }
698
699 /**
700 * @brief USART1 Initialization Function
701 * @param None
702 * @retval None
703 */
704 static void MX_USART1_UART_Init(void)
705 {
706
707     /* USER CODE BEGIN USART1_Init 0 */
708
709     /* USER CODE END USART1_Init 0 */
710
711     /* USER CODE BEGIN USART1_Init 1 */
712

```

```

713 /* USER CODE END USART1_Init 1 */
714 huart1.Instance = USART1;
715 huart1.Init.BaudRate = 115200;
716 huart1.Init.WordLength = UART_WORDLENGTH_8B;
717 huart1.Init.StopBits = UART_STOPBITS_1;
718 huart1.Init.Parity = UART_PARITY_NONE;
719 huart1.Init.Mode = UART_MODE_TX_RX;
720 huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
721 huart1.Init.OverSampling = UART_OVERSAMPLING_16;
722 huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
723 huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
724 if (HAL_UART_Init(&huart1) != HAL_OK)
725 {
726     Error_Handler();
727 }
728 /* USER CODE BEGIN USART1_Init 2 */
729
730 /* USER CODE END USART1_Init 2 */
731
732 }
733
734 /**
735 * @brief USB_OTG_FS Initialization Function
736 * @param None
737 * @retval None
738 */
739 static void MX_USB_OTG_FS_HCD_Init(void)
740 {
741
742     /* USER CODE BEGIN USB_OTG_FS_Init 0 */
743
744     /* USER CODE END USB_OTG_FS_Init 0 */
745
746     /* USER CODE BEGIN USB_OTG_FS_Init 1 */
747
748     /* USER CODE END USB_OTG_FS_Init 1 */
749     hhcd_USB_OTG_FS.Instance = USB_OTG_FS;
750     hhcd_USB_OTG_FS.Init.Host_channels = 12;
751     hhcd_USB_OTG_FS.Init.speed = HCD_SPEED_FULL;
752     hhcd_USB_OTG_FS.Init.dma_enable = DISABLE;
753     hhcd_USB_OTG_FS.Init.phy_itface = HCD_PHY_EMBEDDED;
754     hhcd_USB_OTG_FS.Init.Sof_enable = DISABLE;
755     if (HAL_HCD_Init(&hhcd_USB_OTG_FS) != HAL_OK)
756     {
757         Error_Handler();
758     }
759     /* USER CODE BEGIN USB_OTG_FS_Init 2 */
760
761     /* USER CODE END USB_OTG_FS_Init 2 */
762
763 }
764
765 /**
766 * Enable DMA controller clock
767 */
768 static void MX_DMA_Init(void)
769 {
770
771     /* DMA controller clock enable */
772     __HAL_RCC_DMA1_CLK_ENABLE();
773
774     /* DMA interrupt init */
775     /* DMA1_Channel2_IRQHandler interrupt configuration */
776     HAL_NVIC_SetPriority(DMA1_Channel2_IRQn, 0, 0);
777     HAL_NVIC_EnableIRQ(DMA1_Channel2_IRQn);
778     /* DMA1_Channel3_IRQHandler interrupt configuration */
779     HAL_NVIC_SetPriority(DMA1_Channel3_IRQn, 0, 0);
780     HAL_NVIC_EnableIRQ(DMA1_Channel3_IRQn);
781
782 }
783
784 /**
785 * @brief GPIO Initialization Function

```

```

786     * @param None
787     * @retval None
788     */
789 static void MX_GPIO_Init(void)
790 {
791     GPIO_InitTypeDef GPIO_InitStruct = {0};
792
793     /* GPIO Ports Clock Enable */
794     __HAL_RCC_GPIOC_CLK_ENABLE();
795     __HAL_RCC_GPIOH_CLK_ENABLE();
796     __HAL_RCC_GPIOA_CLK_ENABLE();
797     __HAL_RCC_GPIOB_CLK_ENABLE();
798     __HAL_RCC_GPIOD_CLK_ENABLE();
799
800     /*Configure GPIO pin Output Level */
801     HAL_GPIO_WritePin(GPIOC, FPGA_RESET_Pin|FPGA_PRETRIG_Pin, GPIO_PIN_RESET);
802
803     /*Configure GPIO pin Output Level */
804     HAL_GPIO_WritePin(GPIOA, SELECTOR_M0_Pin|SELECTOR_M1_Pin|SELECTOR_M2_Pin|
805     SELECTOR_M3_Pin, GPIO_PIN_RESET);
806
807     /*Configure GPIO pin Output Level */
808     HAL_GPIO_WritePin(GPIOB, UI_LED_G_Pin|UI_LED_R_Pin|UI_LED_B_Pin|TRIG_SRC0_Pin
809                         |TRIG_SRC1_Pin, GPIO_PIN_RESET);
810
811     /*Configure GPIO pin : WAKE_UP_Pin */
812     GPIO_InitStruct.Pin = WAKE_UP_Pin;
813     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
814     GPIO_InitStruct.Pull = GPIO_PULLUP;
815     HAL_GPIO_Init(WAKE_UP_GPIO_Port, &GPIO_InitStruct);
816
817     /*Configure GPIO pins : FPGA_RESET_Pin FPGA_PRETRIG_Pin */
818     GPIO_InitStruct.Pin = FPGA_RESET_Pin|FPGA_PRETRIG_Pin;
819     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
820     GPIO_InitStruct.Pull = GPIO_NOPULL;
821     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
822     HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
823
824     /*Configure GPIO pins : SELECTOR_M0_Pin SELECTOR_M1_Pin SELECTOR_M2_Pin
825     SELECTOR_M3_Pin */
826     GPIO_InitStruct.Pin = SELECTOR_M0_Pin|SELECTOR_M1_Pin|SELECTOR_M2_Pin|
827     SELECTOR_M3_Pin;
828     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
829     GPIO_InitStruct.Pull = GPIO_NOPULL;
830     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
831     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
832
833     /*Configure GPIO pins : UI_LED_G_Pin UI_LED_R_Pin UI_LED_B_Pin TRIG_SRC0_Pin
834                         |TRIG_SRC1_Pin */
835     GPIO_InitStruct.Pin = UI_LED_G_Pin|UI_LED_R_Pin|UI_LED_B_Pin|TRIG_SRC0_Pin
836                         |TRIG_SRC1_Pin;
837     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
838     GPIO_InitStruct.Pull = GPIO_NOPULL;
839     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
840     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
841
842     /*Configure GPIO pin : FPGA_DONE_Pin */
843     GPIO_InitStruct.Pin = FPGA_DONE_Pin;
844     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING_FALLING;
845     GPIO_InitStruct.Pull = GPIO_NOPULL;
846     HAL_GPIO_Init(FPGA_DONE_GPIO_Port, &GPIO_InitStruct);
847
848     /*Configure GPIO pin : SD_DETECT_INT_Pin */
849     GPIO_InitStruct.Pin = SD_DETECT_INT_Pin;
850     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
851     GPIO_InitStruct.Pull = GPIO_NOPULL;
852     HAL_GPIO_Init(SD_DETECT_INT_GPIO_Port, &GPIO_InitStruct);
853
854     /*Configure GPIO pin : USB_VBUS_INT_Pin */
855     GPIO_InitStruct.Pin = USB_VBUS_INT_Pin;
856     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
857     GPIO_InitStruct.Pull = GPIO_NOPULL;
858     HAL_GPIO_Init(USB_VBUS_INT_GPIO_Port, &GPIO_InitStruct);

```

```

856
857     /*Configure GPIO pin : ARM_BUTTON_Pin */
858     GPIO_InitStruct.Pin = ARM_BUTTON_Pin;
859     GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
860     GPIO_InitStruct.Pull = GPIO_PULLUP;
861     HAL_GPIO_Init(ARM_BUTTON_GPIO_Port, &GPIO_InitStruct);
862
863     /* EXTI interrupt init*/
864     HAL_NVIC_SetPriority(EXTI4_IRQn, 5, 0);
865     HAL_NVIC_EnableIRQ(EXTI4_IRQn);
866
867     HAL_NVIC_SetPriority(EXTI15_10_IRQn, 5, 0);
868     HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
869
870 }
871
872 /* USER CODE BEGIN 4 */
873 #ifdef __GNUC__
874     /* With GCC, small printf (option LD Linker->Libraries->Small printf
875      set to 'Yes') calls __io_putchar() */
876     int __io_putchar(int ch)
877 #else
878     int fputc(int ch, FILE *f)
879 #endif /* __GNUC__ */
880 {
881     /* Place your implementation of fputc here */
882     /* e.g. write a character to the UART3 and Loop until the end of transmission */
883     HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, HAL_MAX_DELAY);
884     return ch;
885 }
886 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
887     if(htim==&htim16){
888         HAL_GPIO_TogglePin(UI_LED_B_GPIO_Port, UI_LED_B_Pin);
889     }
890 }
891 /* USER CODE END 4 */
892
893 /**
894  * @brief This function is executed in case of error occurrence.
895  * @retval None
896  */
897 void Error_Handler(void)
898 {
899     /* USER CODE BEGIN Error_Handler_Debug */
900     /* User can add his own implementation to report the HAL error return state */
901     disable_irq();
902     while (1)
903     {
904     }
905     /* USER CODE END Error_Handler_Debug */
906 }
907
908 #ifdef USE_FULL_ASSERT
909 /**
910  * @brief Reports the name of the source file and the source line number
911  * where the assert_param error has occurred.
912  * @param file: pointer to the source file name
913  * @param line: assert_param error line source number
914  * @retval None
915  */
916 void assert_failed(uint8_t *file, uint32_t line)
917 {
918     /* USER CODE BEGIN 6 */
919     /* User can add his own implementation to report the file name and line number,
920      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
921     /* USER CODE END 6 */
922 }
923 #endif /* USE_FULL_ASSERT */
924

```

A.2.3 fram_func.h

```

1  /*
2   * fram_func.h
3   *
4   * Created on: 6 juil. 2023
5   * Author: christop.grobety
6   */
7
8 #ifndef INC_FRAM_FUNC_H_
9 #define INC_FRAM_FUNC_H_
10
11#define MEMORY_SIZE 524288
12#include "main.h"
13#include <stdbool.h>
14#include <stdio.h>
15
16/**
17 * @brief Write one or more value in a FRAM
18 * @param State state_ : FRAM where to send the value(s)
19 * @param TX_FRAM: Pointer of the table to send
20 * @param data_size: Number of values to send on the FRAM
21 * @param hspi_1 : SPI of the STM32 to use to send the value
22 */
23void FRAM_write_one(enum State state_, uint8_t*TX_FRAM, uint8_t data_size,
SPI_HandleTypeDef hspi_1);
24
25/**
26 * @brief Write one or more value in all the FRAMs
27 * @param TX_FRAM: Pointer of the table to send
28 * @param data_size: Number of values to send on the FRAMs
29 * @param hspi_1 : SPI of the STM32 to use to send the value
30 */
31void FRAM_write_all(SPI_HandleTypeDef hspi_1, uint8_t*TX_FRAM, uint8_t data_size);
32
33/**
34 * @brief Write in all the FRAMs the instruction of receive datas from ADC DEVICE
35 * @param hspi_1 : SPI of the STM32 to use to send the value
36 */
37void FRAM_write_ADC_to_FRAM(SPI_HandleTypeDef hspi_1);
38
39/**
40 * @brief Write in all the FRAMs the instruction to set the register for write
41 * instruction
42 * @param hspi_1 : SPI of the STM32 to use to send the value
43 */
44void FRAM_write_reg(SPI_HandleTypeDef hspi_1);
45
46/**
47 * @brief Write in all the FRAMs the instruction to reset the register
48 * @param hspi_1 : SPI of the STM32 to use to send the value
49 */
50void FRAM_reset_reg(SPI_HandleTypeDef hspi_1);
51
52/**
53 * @brief Write in all the FRAMs the instruction to reset the register
54 * @param State state_ : FRAM where to read the Register's values
55 * @param hspi_1 : SPI of the STM32 to use to send the value
56 */
57void FRAM_read_reg(enum State state_, SPI_HandleTypeDef hspi_1);
58
59/**
60 * @brief Read in a FRAM his Device's value
61 * @param hspi_1 : SPI of the STM32 to use to send the value
62 */
63void FRAM_device(SPI_HandleTypeDef hspi_1);
64
65/**
66 * @brief Read in a FRAM his Device's value
67 * @param State state_ : FRAM where to read the Register's values
68 * @param add: Location of the first address to start read the value of a FRAM
69 * @param hspi_1 : SPI of the STM32 to use to send the value
70 * @param data_size : size of the address where to read value
71 * @return a pointer of a table with the FRAM's values
72 */

```

```
72 uint8_t* FRAM_read(enum State state_, uint32_t add,SPI_HandleTypeDef hspi_1, uint8_t
73 data_size);
74 /**
75 * @brief Reset the values of the Selector's pins to send in the FPGA
76 */
77 void PIN_reset();
78 /**
79 * @brief Send the value of pretrigger to the FPGA
80 * @param hspi_1 : SPI of the STM32 to use to send the value
81 * @param pretrig : pretrig's value to send to the FPGA
82 */
83 void setPreTrigg(SPI_HandleTypeDef hspi_1, uint8_t pretrig);
84 /**
85 * @brief Selection of the channel to trigg
86 * @param chan: value of the channel to use
87 */
88 void setTriggChannel(enum trig_channel chan);
89 /**
90 * @brief To turn on a Led
91 * @param color: color of the Led to turn on
92 */
93 void LED_on(enum color_Led color);
94 /**
95 * @brief To turn off a Led
96 */
97 void LED_off();
```

A.2.4 fram_func

.C

```

1  /*
2   * fram_func.c
3   *
4   * Created on: 6 juil. 2023
5   * Author: christop.grobety
6   */
7 #include "fram_func.h"
8 #include "fatfs.h"
9 SPI_HandleTypeDef hspi_2;
10
11 FRESULT res; /* FatFs function common result code */
12 uint32_t byteswritten, bytesread; /* File write/read counts */
13 uint8_t wtext[20] = "DATA FRAM IS : \n"; /* File write buffer */
14 uint8_t rtext[_MAX_SS];/* File read buffer */
15 uint8_t TX_Data_ADC[2];
16 uint8_t TX_Data_PRETRIG = 80;
17 uint8_t TX_Data_FRAM_REG = 6;
18 uint8_t TX_Read_FRAM_REG = 5;
19 uint8_t TX_ReadId_FRAM_REG = 0x9F;
20 uint8_t TX_Data_FRAM_RESET = 4;
21 uint32_t TX_Data_FRAM_ADD = 0x40;
22 uint8_t TX_FRAM_TAB[50];
23 uint8_t TX_FRAM_TEST_R[4];
24 uint8_t RX_SAMPLE[2];
25 uint8_t TX_SAMPLE;
26 uint8_t RX_ADD [3];
27 uint8_t RX_FRAM_READ[6];
28 uint8_t TX_FRAM_TEST_R[4];
29 uint16_t device_0,device_1;
30 TCHAR* fileName_0 = "REG_0.txt";
31 TCHAR* fileName_1 = "REG_1.txt";
32 TCHAR* fileName_2 = "REG_2.txt";
33 TCHAR* fileName_3 = "REG_3.txt";
34 TCHAR* reg_name;
35
36 void FRAM_write_one(enum State state_, uint8_t*TX_FRAM, uint8_t data_size,
37 SPI_HandleTypeDef hspi_1){
38     PIN_reset();
39     switch(state_){
40         case FRAM_0:
41             HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_SET);
42             HAL_SPI_Transmit(&hspi_1,TX_FRAM,data_size, 100);
43             HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_RESET);
44             HAL_Delay(100);
45             break;
46         case FRAM_1:
47             HAL_GPIO_WritePin(SELECTOR_M0_GPIO_Port, SELECTOR_M0_Pin, GPIO_PIN_SET);
48             HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_SET);
49             HAL_SPI_Transmit(&hspi_1,TX_FRAM,data_size, 100);
50             HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_RESET);
51             HAL_GPIO_WritePin(SELECTOR_M0_GPIO_Port, SELECTOR_M0_Pin, GPIO_PIN_RESET);
52             HAL_Delay(100);
53             break;
54         case FRAM_2:
55             HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_SET);
56             HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_SET);
57             HAL_SPI_Transmit(&hspi_1,TX_FRAM,data_size, 100);
58             HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_RESET);
59             HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_RESET);
60             HAL_Delay(100);
61             break;
62         case FRAM_3:
63             HAL_GPIO_WritePin(SELECTOR_M0_GPIO_Port, SELECTOR_M0_Pin, GPIO_PIN_SET);
64             HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_SET);
65             HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_SET);
66             HAL_SPI_Transmit(&hspi_1,TX_FRAM,data_size, 100);
67             HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_RESET);
68             HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_RESET);
69             HAL_GPIO_WritePin(SELECTOR_M0_GPIO_Port, SELECTOR_M0_Pin, GPIO_PIN_RESET);
70             HAL_Delay(100);
71             break;
72     }
}

```

```

73 void FRAM_write_all(SPI_HandleTypeDef hspi_1, uint8_t*TX_FRAM, uint8_t data_size){
74     PIN_reset();
75
76     HAL_GPIO_WritePin(SELECTOR_M3_GPIO_Port, SELECTOR_M3_Pin, GPIO_PIN_SET);
77     HAL_SPI_Transmit(&hspi_1, TX_FRAM, data_size, 100);
78     HAL_GPIO_WritePin(SELECTOR_M3_GPIO_Port, SELECTOR_M3_Pin, GPIO_PIN_RESET);
79     HAL_Delay(100);
80 }
81
82 void FRAM_write_ADC_to_FRAM(SPI_HandleTypeDef hspi_1){
83     PIN_reset();
84
85     TX_FRAM_TAB[0] = 2;
86     for(int i = 1; i<5;i++) {
87         TX_FRAM_TAB[i] = 0;
88     }
89     TX_FRAM_TAB[4] = 0; // "A"
90
91     HAL_GPIO_WritePin(SELECTOR_M3_GPIO_Port, SELECTOR_M3_Pin, GPIO_PIN_SET);
92     HAL_SPI_Transmit(&hspi_1, (uint8_t*)TX_FRAM_TAB, 4, 100);
93     //-----ADC TO
94     //-----FRAM-----
95     HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_SET);
96     //HAL_GPIO_WritePin(SELECTOR_M3_GPIO_Port, SELECTOR_M3_Pin, GPIO_PIN_RESET);
97     //HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_RESET);
98 }
99
100 void FRAM_write_reg(SPI_HandleTypeDef hspi_1){
101     HAL_GPIO_WritePin(SELECTOR_M3_GPIO_Port, SELECTOR_M3_Pin, GPIO_PIN_SET);
102     HAL_SPI_Transmit(&hspi_1, (uint8_t*)&TX_Data_FRAM_REG, 1, 100);
103     HAL_GPIO_WritePin(SELECTOR_M3_GPIO_Port, SELECTOR_M3_Pin, GPIO_PIN_RESET);
104     HAL_Delay(100);
105 }
106
107 void FRAM_reset_reg(SPI_HandleTypeDef hspi_1){
108
109     HAL_GPIO_WritePin(SELECTOR_M3_GPIO_Port, SELECTOR_M3_Pin, GPIO_PIN_SET);
110     HAL_SPI_Transmit(&hspi_1, (uint8_t*)&TX_Data_FRAM_RESET, 1, 100);
111     HAL_GPIO_WritePin(SELECTOR_M3_GPIO_Port, SELECTOR_M3_Pin, GPIO_PIN_RESET);
112     HAL_Delay(100);
113 }
114
115 void FRAM_read_reg(enum State state_, SPI_HandleTypeDef hspi_1){
116     PIN_reset();
117     switch(state_){
118         case FRAM_0:
119             HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_SET);
120             HAL_SPI_Transmit(&hspi_1, (uint8_t*)&TX_Read_FRAM_REG, 1, 1);
121             HAL_SPI_Receive(&hspi_1, (uint8_t*)&TX_SAMPLE, 1, 100);
122             HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_RESET);
123             //HAL_Delay(100);
124             reg_name = fileName_0;
125             break;
126         case FRAM_1:
127             HAL_GPIO_WritePin(SELECTOR_M0_GPIO_Port, SELECTOR_M0_Pin, GPIO_PIN_SET);
128             HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_SET);
129             HAL_SPI_Transmit(&hspi_1, (uint8_t*)&TX_Read_FRAM_REG, 1, 1);
130             HAL_SPI_Receive(&hspi_1, (uint8_t*)&TX_SAMPLE, 1, 100);
131             HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_RESET);
132             HAL_GPIO_WritePin(SELECTOR_M0_GPIO_Port, SELECTOR_M0_Pin, GPIO_PIN_RESET);
133             //HAL_Delay(100);
134             reg_name = fileName_1;
135             break;
136         case FRAM_2:
137             HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_SET);
138             HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_SET);
139             HAL_SPI_Transmit(&hspi_1, (uint8_t*)&TX_Read_FRAM_REG, 1, 1);
140             HAL_SPI_Receive(&hspi_1, (uint8_t*)&TX_SAMPLE, 1, 100);
141             HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_RESET);
142             HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_RESET);
143             //HAL_Delay(100);
144             reg_name = fileName_2;
145             break;
146         case FRAM_3:
147             HAL_GPIO_WritePin(SELECTOR_M0_GPIO_Port, SELECTOR_M0_Pin, GPIO_PIN_SET);

```

```

145 HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_SET);
146 HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_SET);
147 HAL_SPI_Transmit(&hspi_1, (uint8_t*)&TX_Read_FRAM_REG_1, 1);
148 HAL_SPI_Receive(&hspi_1, (uint8_t*)&TX_SAMPLE, 1, 100);
149 HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_RESET);
150 HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_RESET);
151 HAL_GPIO_WritePin(SELECTOR_M0_GPIO_Port, SELECTOR_M0_Pin, GPIO_PIN_RESET);
152 //HAL_Delay(100);
153 reg_name = fileName_3;
154 break;
155 }
156 }
157 void FRAM_device(SPI_HandleTypeDef hspi_1){
158 PIN_reset();
159 HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_SET);
160 HAL_SPI_Transmit(&hspi_1, (uint8_t*)&TX_ReadId_FRAM_REG_1, 100);
161 HAL_SPI_Receive(&hspi_1, (uint8_t*)&TX_FRAME_TEST_R, 4, 100);
162 HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_RESET);
163 HAL_Delay(100);
164 reg_name = "DEVICE.txt";
165
166 SD_create_file(SDFile, reg_name);
167 SD_write_data(SDFile, reg_name, TX_FRAME_TEST_R[0]);
168 SD_write_data(SDFile, reg_name, TX_FRAME_TEST_R[1]);
169 SD_write_data(SDFile, reg_name, TX_FRAME_TEST_R[2]);
170 SD_write_data(SDFile, reg_name, TX_FRAME_TEST_R[3]);
171 }
172 uint8_t* FRAM_read(enum State state_, uint32_t add, SPI_HandleTypeDef hspi_1, uint8_t
data_size){
173 PIN_reset();
174 RX_FRAME_READ[3] = (uint8_t)(add&255);
175 RX_FRAME_READ[2] = (uint8_t)((add>>8)&255);
176 RX_FRAME_READ[1] = (uint8_t)((add>>16)&7);
177 RX_FRAME_READ[0] = 3;
178 switch(state_){
179 case FRAM_0:
180 HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_SET);
181 HAL_SPI_Transmit(&hspi_1, (uint8_t*)&RX_FRAME_READ, 4, 100);
182 HAL_SPI_Receive(&hspi_1, (uint8_t*)&RX_SAMPLE, data_size, 100);
183 HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_RESET);
184 //HAL_Delay(100);
185 break;
186 case FRAM_1:
187 HAL_GPIO_WritePin(SELECTOR_M0_GPIO_Port, SELECTOR_M0_Pin, GPIO_PIN_SET);
188 HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_SET);
189 HAL_SPI_Transmit(&hspi_1, (uint8_t*)&RX_FRAME_READ, 4, 100);
190 HAL_SPI_Receive(&hspi_1, (uint8_t*)&RX_SAMPLE, data_size, 100);
191 HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_RESET);
192 HAL_GPIO_WritePin(SELECTOR_M0_GPIO_Port, SELECTOR_M0_Pin, GPIO_PIN_RESET);
193 //HAL_Delay(100);
194 break;
195 case FRAM_2:
196 HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_SET);
197 HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_SET);
198 HAL_SPI_Transmit(&hspi_1, (uint8_t*)&RX_FRAME_READ, 4, 100);
199 HAL_SPI_Receive(&hspi_1, (uint8_t*)&RX_SAMPLE, data_size, 100);
200 HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_RESET);
201 HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_RESET);
202 //HAL_Delay(100);
203 break;
204 case FRAM_3:
205 HAL_GPIO_WritePin(SELECTOR_M0_GPIO_Port, SELECTOR_M0_Pin, GPIO_PIN_SET);
206 HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_SET);
207 HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_SET);
208 HAL_SPI_Transmit(&hspi_1, (uint8_t*)&RX_FRAME_READ, 4, 100);
209 HAL_SPI_Receive(&hspi_1, (uint8_t*)&RX_SAMPLE, data_size, 100);
210 HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_RESET);
211 HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_RESET);
212 HAL_GPIO_WritePin(SELECTOR_M0_GPIO_Port, SELECTOR_M0_Pin, GPIO_PIN_RESET);
213 //HAL_Delay(100);
214 break;
215 }
216 return (uint8_t*)&RX_SAMPLE;

```

```

217 }
218 void setPreTrigg(SPI_HandleTypeDef hspi_1, uint8_t pretrig) {
219     PIN_reset();
220
221     HAL_GPIO_WritePin(FPGA_PRETRIG_GPIO_Port, FPGA_PRETRIG_Pin, GPIO_PIN_SET);
222     HAL_SPI_Transmit(&hspi_1, &pretrig, 1, 100);
223     HAL_GPIO_WritePin(FPGA_PRETRIG_GPIO_Port, FPGA_PRETRIG_Pin, GPIO_PIN_RESET);
224 }
225
226
227 void setTriggChannel(enum trig_channel chan) {
228     switch(chan) {
229         case CHANNEL_0:
230             HAL_GPIO_WritePin(TRIG_SRC0_GPIO_Port, TRIG_SRC0_Pin, GPIO_PIN_RESET);
231             HAL_GPIO_WritePin(TRIG_SRC1_GPIO_Port, TRIG_SRC1_Pin, GPIO_PIN_RESET);
232             break;
233         case CHANNEL_1:
234             HAL_GPIO_WritePin(TRIG_SRC0_GPIO_Port, TRIG_SRC0_Pin, GPIO_PIN_RESET);
235             HAL_GPIO_WritePin(TRIG_SRC1_GPIO_Port, TRIG_SRC1_Pin, GPIO_PIN_SET);
236             break;
237         case CHANNEL_2:
238             HAL_GPIO_WritePin(TRIG_SRC0_GPIO_Port, TRIG_SRC0_Pin, GPIO_PIN_RESET);
239             HAL_GPIO_WritePin(TRIG_SRC1_GPIO_Port, TRIG_SRC1_Pin, GPIO_PIN_SET);
240             break;
241         case CHANNEL_3:
242             HAL_GPIO_WritePin(TRIG_SRC0_GPIO_Port, TRIG_SRC0_Pin, GPIO_PIN_SET);
243             HAL_GPIO_WritePin(TRIG_SRC1_GPIO_Port, TRIG_SRC1_Pin, GPIO_PIN_SET);
244             break;
245     }
246 }
247
248
249 void PIN_reset() {
250     HAL_GPIO_WritePin(SELECTOR_M3_GPIO_Port, SELECTOR_M3_Pin, GPIO_PIN_RESET);
251     HAL_GPIO_WritePin(SELECTOR_M2_GPIO_Port, SELECTOR_M2_Pin, GPIO_PIN_RESET);
252     HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_RESET);
253     HAL_GPIO_WritePin(SELECTOR_M0_GPIO_Port, SELECTOR_M0_Pin, GPIO_PIN_RESET);
254 }
255
256 void LED_on(enum color_Led color) {
257     switch(color) {
258         case RED:
259             HAL_GPIO_WritePin(UI_LED_R_GPIO_Port, UI_LED_R_Pin, SET);
260             HAL_GPIO_WritePin(UI_LED_B_GPIO_Port, UI_LED_B_Pin, RESET);
261             HAL_GPIO_WritePin(UI_LED_G_GPIO_Port, UI_LED_G_Pin, RESET);
262             break;
263         case BLUE:
264             HAL_GPIO_WritePin(UI_LED_R_GPIO_Port, UI_LED_R_Pin, RESET);
265             HAL_GPIO_WritePin(UI_LED_B_GPIO_Port, UI_LED_B_Pin, SET);
266             HAL_GPIO_WritePin(UI_LED_G_GPIO_Port, UI_LED_G_Pin, RESET);
267             break;
268         case GREEN:
269             HAL_GPIO_WritePin(UI_LED_R_GPIO_Port, UI_LED_R_Pin, RESET);
270             HAL_GPIO_WritePin(UI_LED_B_GPIO_Port, UI_LED_B_Pin, RESET);
271             HAL_GPIO_WritePin(UI_LED_G_GPIO_Port, UI_LED_G_Pin, SET);
272             break;
273         case OFF:
274             LED_off();
275             break;
276     }
277     LED_STATE = color;
278 }
279
280 void LED_off() {
281     HAL_GPIO_WritePin(UI_LED_R_GPIO_Port, UI_LED_R_Pin, RESET);
282     HAL_GPIO_WritePin(UI_LED_B_GPIO_Port, UI_LED_B_Pin, RESET);
283     HAL_GPIO_WritePin(UI_LED_G_GPIO_Port, UI_LED_G_Pin, RESET);
284     LED_STATE = OFF;
285 }
286
287
288

```

A.2.5 sd_card.h

```

1  /*
2   *  sd_card_func.h
3   *
4   *  Created on: Jul 7, 2023
5   *      Author: christop.grobety
6   */
7
8 #ifndef INC_SD_CARD_FUNC_H_
9 #define INC_SD_CARD_FUNC_H_
10 #include "main.h"
11 #include "fatfs.h"
12 #include "fram_func.h"
13 #include <stdbool.h>
14
15 /**
16  * @brief Create a file in the SD card with a name and a type (.bin,.txt,...)
17  * @param sd_file: SD_card emplacement
18  * @param fileName: name and type of the file
19  */
20 void SD_create_file(FIL sd_file, TCHAR* fileName);
21
22 /**
23  * @brief Write a table of 16 bits of values in a SD's file
24  * @param sd_file: SD_card emplacement
25  * @param fileName: name and type of the file
26  * @param data: pointer on the first adress of the table to write
27  */
28 void SD_write_data(FIL sd_file, TCHAR* fileName, uint16_t* data);
29
30 /**
31  * @brief Read all the value in a SD's file
32  * @param sd_file: SD_card emplacement
33  * @param fileName: name and type of the file
34  * @return the pointer on the first adress of the data's file table
35  */
36 uint8_t* SD_read_data(FIL sd_file, TCHAR* fileName);
37
38 /**
39  * @brief Convert all the values(char) in the SD's file receive in 3 double values
40  * @param sd_file: SD_card emplacement
41  * @param fileName: name and type of the file
42  * @return the pointer on the first adress of the InitTrigger's values
43  */
44 double* initValue(FIL sd_file, TCHAR* fileName);
45
46
47 #endif /* INC_SD_CARD_FUNC_H_ */
48

```

A.2.6 sd_card.c

```

1  /*
2   *  sd_card_func.c
3   *
4   *  Created on: Jul 7, 2023
5   *      Author: christop.grobety
6   */
7
8 #include "sd_card_func.h"
9 #include <math.h>
10
11 FRESULT res_sd;
12 uint32_t byteswritten_sd, bytesread_sd;
13 uint8_t rtext_ [_MAX_SS] ;
14 uint8_t wtext_2 [100] ;
15 uint8_t test [2] ;
16 double init_Value [4] ;
17
18 void SD_create_file(FIL sd_file, TCHAR* fileName){
19     res_sd = f_mount(&SDFatFS, (TCHAR const*)SDPath, 0) ;
20     if(res_sd != FR_OK){
21         LED_on(RED);
22     }
23     else{
24         //Open file for writing (Create)
25         res_sd = f_open(&sd_file, fileName, (FA_CREATE_ALWAYS | FA_WRITE));
26         if(res_sd != FR_OK){
27             LED_on(RED);
28         }
29         else{
30             //LED_on(BLUE);
31             f_close(&sd_file);
32         }
33     }
34     f_mount(&SDFatFS, (TCHAR const*)NULL, 0);
35 }
36
37 void SD_write_data(FIL sd_file, TCHAR* fileName, uint16_t* data){
38     //sprintf((char*)wtext_, "%d \n", data);
39     res_sd = f_mount(&SDFatFS, (TCHAR const*)SDPath, 0) ;
40     if(res_sd != FR_OK){
41         LED_on(RED);
42     }
43     else{
44         //Open file for writing (Create)
45         res_sd = f_open(&sd_file, fileName, (FA_OPEN_ALWAYS | FA_WRITE));
46         if(res_sd != FR_OK){
47             LED_on(RED);
48         }
49         else{
50             //Write to the text file
51             if(f_lseek(&sd_file, f_size(&sd_file)) == FR_OK){
52                 //res_sd = f_write(&sd_file, wtext_, strlen((char *)wtext_), (void *)
53                 *byteswritten_sd);
54                 res_sd = f_write(&sd_file, data, (TAB_SIZE), (void *)&byteswritten_sd);
55                 if((byteswritten_sd == 0) || (res_sd != FR_OK)){
56                     LED_on(RED);
57                 }
58                 else{
59                     //LED_on(BLUE);
60                     f_close(&sd_file);
61                 }
62             }
63         }
64     f_mount(&SDFatFS, (TCHAR const*)NULL, 0);
65 }
66
67 uint8_t* SD_read_data(FIL sd_file, TCHAR* fileName){
68     res_sd = f_mount(&SDFatFS, (TCHAR const*)SDPath, 0) ;
69     if(res_sd != FR_OK){
70         LED_on(RED);
71     }
72     else{

```

```

73 //Open file for writing (Create)
74 res_sd = f_open(&sd_file, "INIT.txt", FA_OPEN_ALWAYS | FA_READ);
75 if(res_sd != FR_OK){
76     LED_on(RED);
77 }
78 else{
79     //Write to the text file
80     if(f_lseek(&sd_file, 0) == FR_OK){
81         while(bytesread_sd == 0){
82             res_sd = f_read(&sd_file, rtext_, sizeof(rtext_), (UINT *)&
83             bytesread_sd);
84             //f_gets(wtext_, sd_file.fptr, &sd_file);
85             if((bytesread_sd == 0) || (res_sd != FR_OK)){
86                 LED_on(RED);
87             }
88             else{
89                 //LED_on(BLUE);
90                 f_close(&sd_file);
91             }
92         }
93     }
94 }
95 f_mount(&SDFatFS, (TCHAR const*)NULL, 0);
96 return (uint8_t*)&rtext_;
97 }
98 double* initValue(FIL sd_file, TCHAR* fileName){
99     uint8_t* data = SD_read_data(sd_file, fileName);
100    uint8_t value[_MAX_SS/2];
101    uint8_t number = 0;
102    uint8_t number_ = 0;
103    uint8_t pow_ = 0;
104    uint8_t div_ = 0;
105    bool fraction = false;
106    for(uint32_t i = 0 ; i<_MAX_SS; i++){
107        if(data[i] >= '0' && data[i] <= '9'){
108            value[number] = data[i]-'0';
109            number++;
110            pow_++;
111        }
112        else if(data[i] == '.' || data[i] == ','){
113            fraction = true;
114            div_ = pow_;
115        }
116        else if(data[i] == '\n' || data[i] == '\0'){
117            for(uint8_t j = 0; j<number ; j++){
118                init_Value[number_] = init_Value[number_] + value[j]*pow(10,number-j-1);
119            }
120            if(fraction){
121                fraction = false;
122                init_Value[number_] = init_Value[number_]/(pow(10,number-div_));
123            }
124            pow_ = 0;
125            div_ = 0;
126            number = 0;
127            number++;
128            if(data[i] == '\0'){
129                break;
130            }
131        }
132    }
133    return (double*)&init_Value;
134 }
135
136

```

A.2.7 isrs.c

```

1  /*
2   *  isrs.c
3   *
4   *  Created on: Jun 30, 2023
5   *      Author: christop.grobety
6   */
7
8 #include "stm32l4xx_hal.h"
9 #include "main.h"
10 #include "fram_func.h"
11
12 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
13 {
14     if(GPIO_Pin == FPGA_DONE_Pin){
15         //-----RECEIVE
16         ADD-----
17         LED_on(BLUE);
18         start = true;
19         HAL_ResumeTick();
20         acqu = false;
21         //sleep_ = 1;
22     }
23     else if(GPIO_Pin == WAKE_UP_Pin ){
24         //-----START
25         ACQU-----
26         if(start){
27             HAL_ResumeTick();
28             start = false;
29         }
30     }
31     else if(GPIO_Pin == ARM_BUTTON_Pin){
32         //-----RESET
33         SYSTEM-----
34         if(reset){
35             reset = false;
36             HAL_NVIC_SystemReset();
37         }
38     }
39 }

```

A.3 Bloc FPGA

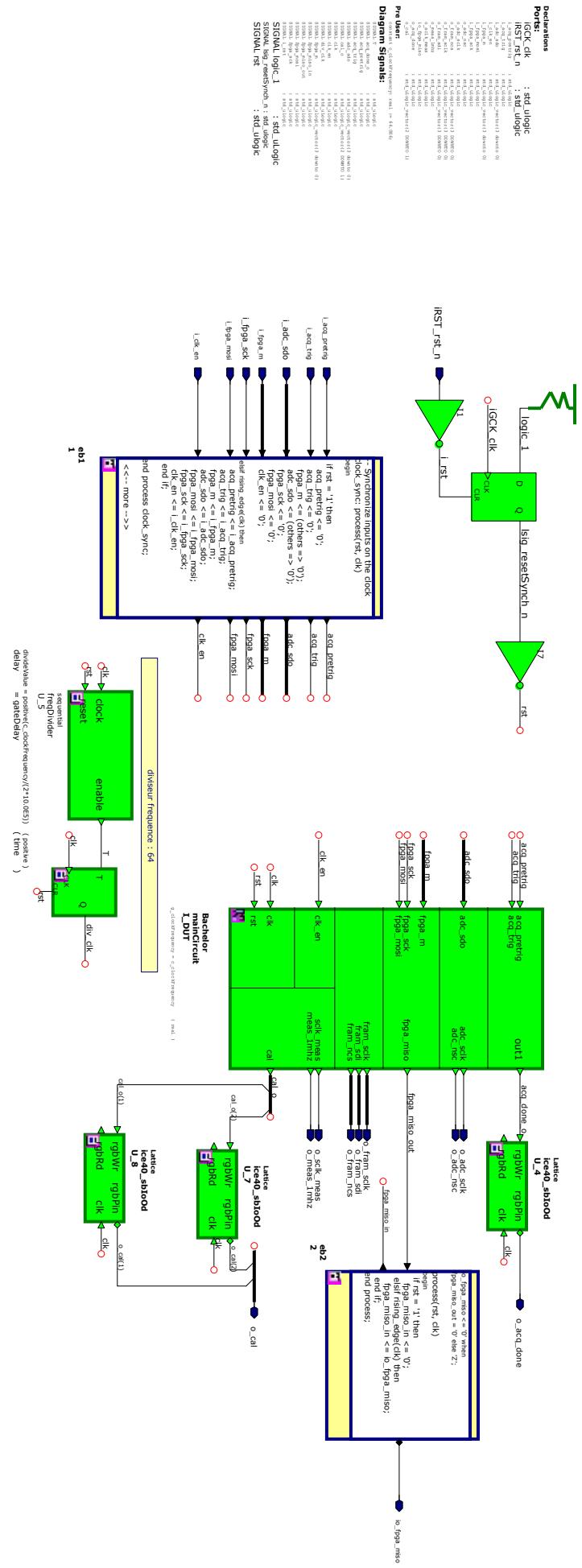
Board/EC5LP/struct

Package List

- LIBRARY ieee;
- USE ieee.std_logic_1164.all;
- USE ieee.numeric_std.all;

LIBRARY gates;

USE gates.gates.all;



Bachelor/mainCircuit/struct

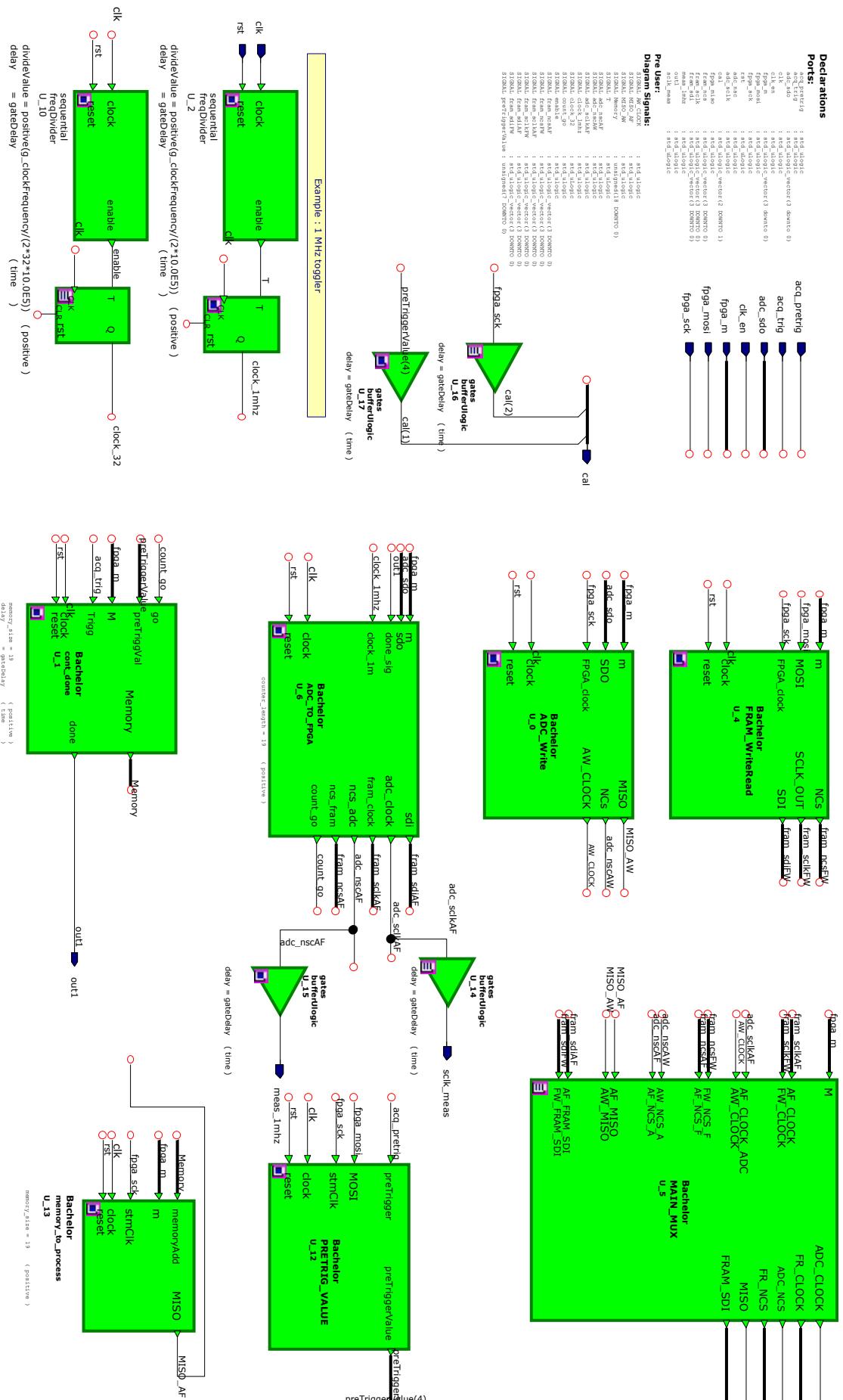
Package List	
LIBRARY ieee;	
USE ieee.std_logic_1164.all;	
USE ieee.std_logic_unsigned.all;	
USE ieee.numeric_std.all;	
	Project: both
	Comment Help
Title:	<center> Download file here </center>
Pan:	BachelorThesisCircuitStruct
Edited:	by chintan.goyal on 11/01/2023

```

Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

LIBRARY gates;
USE gates.gates.all;

```



A.4 Code FPGA

```

-- VHDL Architecture Bachelor.ADC_TO_FPGA.STUDENT
--
-- Created:
--     by - christop.grobety.UNKNOWN (WE2332207)
--     at - 11:23:31 31.05.2023
--
-- using Mentor Graphics HDL Designer(TM) 2019.2 (Build 5)
--
LIBRARY Common;
USE Common.CommonLib.all;

ARCHITECTURE STUDENT OF ADC_TO_FPGA IS
    signal done16 : std_uLogic ;
    signal down_ncs : std_uLogic ;
    signal down_ncs_old : std_uLogic ;
    signal is_up : std_uLogic ;
    signal finish_d : std_uLogic ;
    signal count: unsigned(requiredBitNb(counter_length)-1 downto 0);
BEGIN
    adc_to_fpga: process(clock,reset)
    begin
        if reset = '1' then
            fram_clock <= (others => '1');
            adc_clock <= '1';
            finish_d <= '0';
            --sdi <= (others => '0');
            ncs_fram <= (others => '0');
            ncs_adc <= '1';
            done16 <= '0';
            count <= (others => '0');
            is_up <= '1';
            count_go <= '0';
            down_ncs_old <= '1';
        elsif rising_edge(clock) then
            --if m = "0010" and down_ncs = '0' then
            count_go <= '0';
            if down_ncs /= down_ncs_old and m = "1010" and finish_d = '0' then
                down_ncs_old <= down_ncs;
                is_up <= '0';
            elsif is_up = '0' and finish_d = '0' and m = "1010" then
                ncs_fram <= (others => '0');
                ncs_adc <= '0';
                --sdi <= sdo;
                if count >= 1 and count <= counter_length-3 then
                    if count = 8 and count_go = '0' then
                        count_go <= '1';
                    end if;

                    if adc_clock = '1' then
                        adc_clock <= '0';
                        fram_clock <= (others => '0');
                    else
                        adc_clock <= '1';
                    end if;
                end if;
            end if;
        end if;
    end process;
end STUDENT;

```

```

        fram_clock <= (others => '1');
        --sdi <= sdo;
        count <= count + 1;
    end if;
elsif count = counter_length-2 then
    count_go <= '1';
    adc_clock <= '1';
    fram_clock <= (others => '1');
    count <= count+1 ;
elsif count = counter_length-1 then
    is_up <= '1';
    ncs_adc <= '1';
    --sdi <= (others => '0');
    ncs_fram <= (others => '0');
elsif count = 0 then
    count <= count + 1;
end if ;
else
    count <= (others => '0');
end if;

if done_sig = '1' then
    finish_d <= '1';
    ncs_fram <= (others => '1');
    ncs_adc <= '1';
elsif m /= "1010" then
    finish_d <= '0';
end if ;

end if;
end process adc_to_fpga;

NCS_clock: process(clock_1m,reset)
begin
    if reset = '1' then
        down_ncs <= '1';
    elsif rising_edge(clock_1m) then
        --if m = "0010" then
        if down_ncs = '1' and m = "1010" then
            down_ncs <= '0';
        else
            down_ncs <= '1';
        end if;
    end if;
end process NCS_clock;

data_to_trans: process(sdo, m)
begin
    if m = "1010" then
        sdi <= sdo;
    else
        sdi <= (others => '0');
    end if;
end process data_to_trans;

```

END ARCHITECTURE STUDENT;

```

-- VHDL Architecture Bachelor.ADC_Write.STUDENT_V3
--
-- Created:
--     by - christop.grobety.UNKNOWN (WE2332207)
--     at - 15:30:17 05.07.2023
--
-- using Mentor Graphics HDL Designer(TM) 2019.2 (Build 5)
--
ARCHITECTURE STUDENT_V3 OF ADC_Write IS
    signal count: unsigned(5 downto 0);
    signal go : std_uLogic;
    signal MISO_IN : std_uLogic;
    signal clockOut : std_uLogic;
    signal clockGO : std_uLogic;
BEGIN
    spiMS: process(m, SDO)
        begin
            if m = "0010" then
                NCS <= '0';
                --MISO_IN <= SDO(SDO'high);
            else
                NCS <= '1';
                --MISO_IN <= '0';
            end if;
        end process spiMS;

    colckSwitch: process(FPGA_clock)
        begin
            if unsigned(m) <= "0010" then
                clockOut <= FPGA_clock;
            else
                clockOut <= '1';
            end if ;
        end process colckSwitch;
    AW_CLOCK <= clockOut;
    --MISO <= MISO_IN;
END ARCHITECTURE STUDENT_V3;

```

```

LIBRARY Common;
USE Common.CommonLib.all;

ARCHITECTURE RTL OF freqDivider IS

    signal count: unsigned(requiredBitNb(divideValue)-1 downto 0);

BEGIN

    countEndlessly: process(reset, clock)
    begin
        if reset = '1' then
            count <= (others => '0');
        elsif rising_edge(clock) then
            if count = 0 then
                count <= to_unsigned(divideValue-1, count'length);
            else
                count <= count-1 ;
            end if;
        end if;
    end process countEndlessly;

    enable <= '1' after delay when count = 0
        else '0' after delay;

END ARCHITECTURE RTL;

```

```

-- VHDL Architecture Bachelor.FRAM_WriteRead.STUDENT_V3
--
-- Created:
--     by - christop.grobety.UNKNOWN (WE2332207)
--     at - 15:27:53 05.07.2023
--
-- using Mentor Graphics HDL Designer(TM) 2019.2 (Build 5)
--

ARCHITECTURE STUDENT_V3 OF FRAM_WriteRead IS
    signal go          : std_uLogic;
    signal count       : unsigned(7 downto 0);
    signal countValue : unsigned(7 downto 0);
    signal sclk_in    : std_uLogic_vector(3 downto 0);
    constant value_8  : natural := 8;
    constant value_32 : natural := 32;
BEGIN
    spiMS: process(MOSI,m)
    begin
        if m = "0100" then
            NCs <= "1110";
            SDI(SDI'high-3) <= MOSI;
        elsif m = "0101" then
            NCs <= "1101";
            SDI(SDI'high-2) <= MOSI;
        elsif m = "0110" then
            NCs <= "1011";
            SDI(SDI'high-1) <= MOSI;
        elsif m = "0111" then
            NCs <= "0111";
            SDI(SDI'high) <= MOSI;
        elsif m = "1000" then
            NCs <= (others => '0');
            SDI(SDI'high) <= MOSI;
            SDI(SDI'high-1) <= MOSI;
            SDI(SDI'high-2) <= MOSI;
            SDI(SDI'high-3) <= MOSI;
        else
            --NCs <= (others => '1');
            SDI <= (others => '0');
        end if;
    end process spiMS;

    colckSwitch: process(FPGA_clock)
    begin
        if unsigned(m) <= "1000" then
            sclk_in(sclk_in'high) <= FPGA_clock;
            sclk_in(sclk_in'high-1) <= FPGA_clock;
            sclk_in(sclk_in'high-2) <= FPGA_clock;
            sclk_in(sclk_in'high-3) <= FPGA_clock;
        else
            sclk_in <= (others =>'1');
        end if ;
    end process colckSwitch;

```

```
SCLK_OUT<= clk_in;  
END ARCHITECTURE STUDENT_V3;
```

```

-- VHDL Architecture Bachelor.memory_to_process.student
-- Created:
--     by - christop.grobety.UNKNOWN (WE2332207)
--     at - 10:24:22 07.06.2023
-- using Mentor Graphics HDL Designer(TM) 2019.2 (Build 5)

LIBRARY Common;
USE Common.CommonLib.all;
ARCHITECTURE student OF memory_to_process IS
    signal count: unsigned((memory_size-1) downto 0);
    signal count_memory: unsigned((memory_size-1) downto 0);
    signal start_bit: std_uLogic;
    signal stmClk_old: std_uLogic;
    signal MISO_OUT: std_uLogic;
    signal done_int: std_uLogic := '0';
BEGIN
    sendAdd : process(reset, clock)
    begin
        if reset = '1' then
            start_bit <= '0';
            MISO_OUT <= '0';
            stmClk_old <= stmClk;
            count_memory <= "100000000000000000000001";
            count <= (others => '0');
        elsif rising_edge(clock) then

            stmClk_old <= stmClk;
            if m = "1011" then
                if stmClk='1' and stmClk_old = '0' then
                    MISO_OUT <= count_memory(count_memory'HIGH);
                    count_memory <= SHIFT_LEFT(count_memory,1);
                end if;
            else
                count_memory <= memoryAdd;

                -- React on rising_edge(stmClk)
                --if stmClk='1' and stmClk_old = '0' then
                --    --MISO_OUT <= not MISO_OUT;
                --end if;
            end if;
        end if ;
    end process sendAdd;

    MISO <= MISO_OUT;
END ARCHITECTURE student;

```

```

-- 
-- VHDL Architecture Bachelor.MAIN_MUX.STUDENT
-- 
-- Created:
--     by - christop.grobety.UNKNOWN (WE2332207)
--     at - 13:08:20 31.05.2023
-- 
-- using Mentor Graphics HDL Designer(TM) 2019.2 (Build 5)
-- 
ARCHITECTURE STUDENT OF MAIN_MUX IS
BEGIN
    mux: process(FW_NCS_F,AF_CLOCK,AF_CLOCK_ADC,FW_CLOCK,AW_CLOCK,
                 AW_NCS_A,AF_NCS_A,M,AF_MISO)
        -- WARNING IF ERROR IN PHYSIC -> SWITCH CASE METHODE ALEX
    begin
        case M is
            -- init
            when "0000" =>
                ADC_CLOCK <= '1';
                FR_CLOCK <= (others => '1');
                ADC_NCS <= '1';
                FR_NCS <= (others => '1');
                FRAM_SDI <= (others => '0');
                MISO <= '1';
            -- Read/Write ADC
            when "0010" =>
                ADC_CLOCK <= AW_CLOCK;
                FR_CLOCK <= (others => '1');
                ADC_NCS <= AW_NCS_A;
                --FR_NCS <= (others => '1');
                --FRAM_SDI <= (others => '0');
                MISO <= '1';

            -- ADC to FRAM
            when "1010" =>
                ADC_CLOCK <= AF_CLOCK_ADC;
                FR_CLOCK <= AF_CLOCK;
                ADC_NCS <= AF_NCS_A;
                FR_NCS <= AF_NCS_F;
                FRAM_SDI <= AF_FRAM_SDI;
                MISO <= '1';

            -- ADC to FRAM
            when "1011" =>
                ADC_CLOCK <= '1';
                FR_CLOCK <= (others => '1');
                ADC_NCS <= '1';
                FR_NCS <= (others => '1');
                FRAM_SDI <= (others => '0');
                MISO <= AF_MISO;

            -- read/write FRAM
            when others =>
                MISO <= '1';
    end process;
end;

```

```
if M <= "1000" and M >="0100" then
    --ADC_CLOCK <= '0';
    FR_CLOCK <= FW_CLOCK;
    ADC_NCS <= '1';
    FR_NCS <= FW_NCS_F;
    FRAM_SDI <= FW_FRAM_SDI;
else
    --FR_NCS <= (others => '1');
    ADC_NCS <= '1';
    ADC_CLOCK <= '1';
    --FR_CLOCK <= (others => '1');
    FR_CLOCK <= (others => '1');
    FRAM_SDI <= (others => '0');
end if;
end case;
end process mux;
--MISO <= '1';
END ARCHITECTURE STUDENT;
```

```

-- VHDL Architecture Bachelor.PRETRIG_VALUE.STUDENT
--
-- Created:
--     by - christop.grobety.UNKNOWN (WE2332207)
--     at - 13:12:09 05.06.2023
--
-- using Mentor Graphics HDL Designer(TM) 2019.2 (Build 5)
--
ARCHITECTURE STUDENT OF PRETRIG_VALUE IS
    signal count_pre: unsigned(3 downto 0);
    signal valuePreTrig_int: unsigned(7 downto 0);
    signal valuePreTrig: unsigned(7 downto 0);
    signal stmClk_old: std_uLogic;
BEGIN
    value_of_preTrig: process(reset, clock)
    begin
        if reset = '1' then
            count_pre <= (others => '0');
            valuePreTrig_int <= "11110000";
            stmClk_old <= stmClk;

        elsif rising_edge(clock) then
            stmClk_old <= stmClk;
            if preTrigger = '1' then
                if stmClk='1' and stmClk_old = '0' then
                    valuePreTrig_int <= SHIFT_LEFT(valuePreTrig_int,1);
                    valuePreTrig_int(0) <= MOSI;
                end if;
            end if;
        end if;
    end process value_of_preTrig;

    preTriggerValue <= valuePreTrig_int;
END ARCHITECTURE STUDENT;

```

A.5 Bloc de test FPGA

Package List

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
LIBRARY gates;
USE gates.gates.all;

```

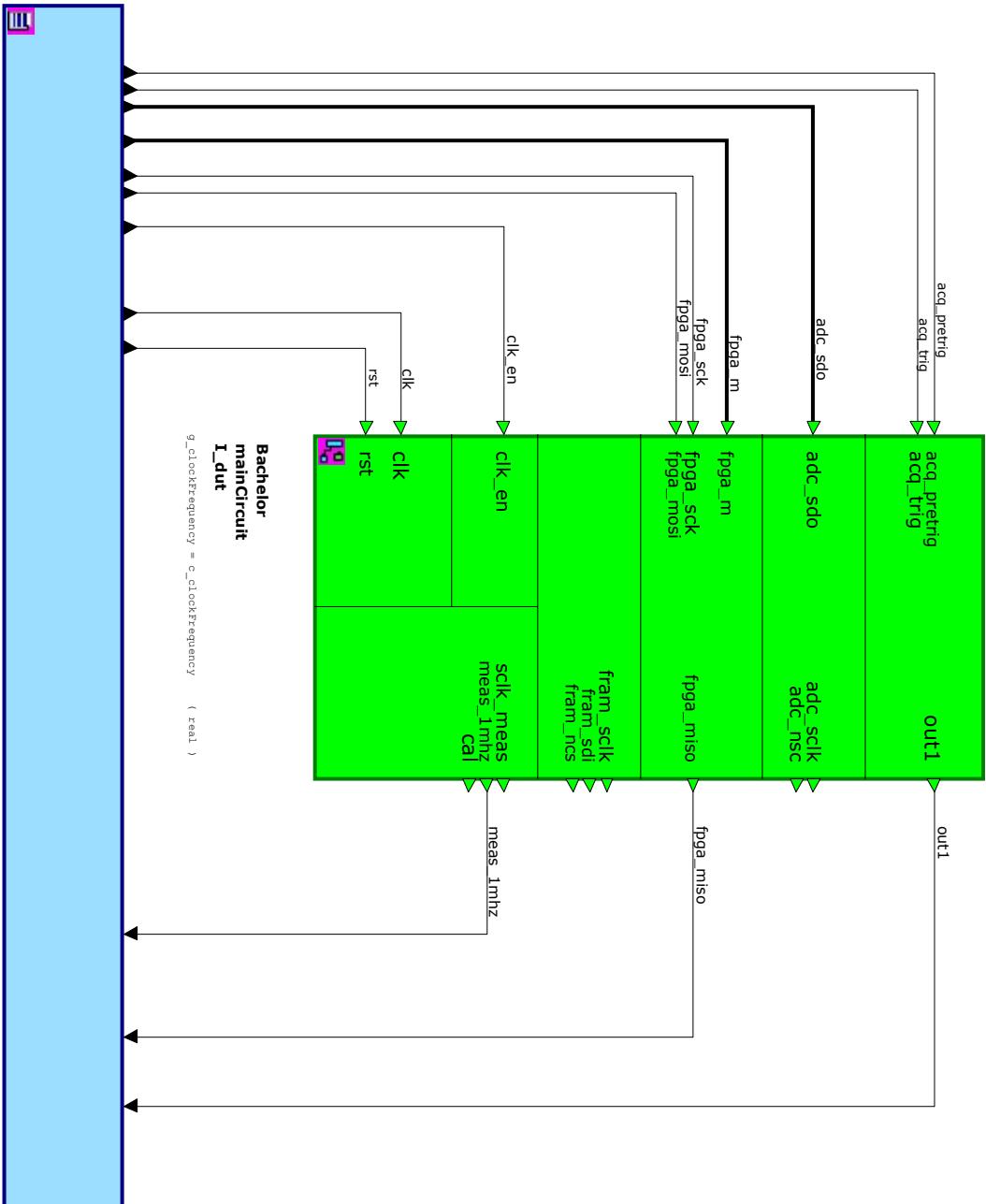
Declarations

Ports:
Pre User:
 constant c_clockFrequency : real := 64.0E6;

Diagram Signals:
 SIGNAL acq_preqtrig : std_ulogic;
 SIGNAL acq_trig : std_ulogic_vector(3 downto 0);
 SIGNAL adc_sdo : std_ulogic;
 SIGNAL clk_en : std_ulogic;
 SIGNAL fram_miso : std_ulogic_vector(3 downto 0);
 SIGNAL fpga_mosi : std_ulogic;
 SIGNAL fpga_sck : std_ulogic;
 SIGNAL fpga_miso : std_ulogic;
 SIGNAL meas_mhz : std_ulogic;
 SIGNAL otbl : std_ulogic;
 SIGNAL rst : std_ulogic;

Post User:

Bachelor_test/mainCircuit_tb/struct	
<company name>	Project: hds
<enter diagram title here>	<enter comments here>
Title: Bachelor_test/mainCircuit_tb/instuct	Path: Bachelor_test/mainCircuit_tb/instuct
Edid: by christop.grobelny on 13 Jun 2023	Edited: by christop.grobelny on 13 Jun 2023



```

Bachelor_test
mainCircuit _tester
I_tester
  g_clockfrequency = c_clockfrequency ( real )

```

A.6 Code du Test FPGA

```

LIBRARY std;
USE std.textio.ALL;

LIBRARY ieee;
USE ieee.std_logic_textio.ALL;

LIBRARY Common_test;
USE Common_test.testutils.all;

ARCHITECTURE test OF mainCircuit_tester IS

constant clockPeriodInn : time := 1.0/g_clockFrequency * 1 sec;
constant clockPeriod   : time := 2.0/g_clockFrequency * 1 sec;
signal sClock          : std_uLogic := '1';
signal sReset          : std_uLogic ;
signal testInfo        : string(1 to 40) := (others => ' ');

signal counter_test    : unsigned(5 downto 0);
signal trigger_counter : unsigned(18 downto 0);

signal sClock_fpga     : std_uLogic := '1';
constant value_8        : natural := 8;
constant value_16       : natural := 16;
constant value_19       : natural := 19;
constant value_32       : natural := 32;

signal ADC_WRITE_INIT   : unsigned(15 downto 0);
signal count_miso       : unsigned(5 downto 0);
signal FRAM_WREN_INIT   : unsigned(7 downto 0);
signal FRAM_WRDI_INIT   : unsigned(7 downto 0);
signal pretrigVal      : unsigned(7 downto 0);
signal count_memoryMax : unsigned(18 downto 0);
signal count_memory     : unsigned(18 downto 0);
signal FRAM_READ_INIT   : unsigned(31 downto 0);
signal FRAM_READ_ADD    : unsigned(18 downto 0);
signal FRAM_WRITE_INIT  : unsigned(31 downto 0);
signal init_done        : std_uLogic ;
signal miso_start       : std_uLogic ;
signal clk_start        : std_uLogic ;
signal selector         : unsigned(3 downto 0);

-- An example of procedure (function which returns nothing)
-- Here checks a value and log given error message if sim is not the same
procedure checkMeas(
  msg :          string;
  measArg :      std_ulogic) is
begin

  std.textio.write(std.textio.output, LF & "======" & LF);
  std.textio.write(std.textio.output, "Testing " & msg & LF);

  assert (meas_1mhz = measArg)
    report ("meas_1mhz error - expected " & to_string(measArg)) severity error;
  if (meas_1mhz = measArg) then
    report " ** Ok" severity note;

```

```

end if;

std.textio.write(std.textio.output, "======" & LF);

-- Force clock synch.
wait until clk'event and clk = '1';
end procedure checkMeas;

BEGIN

----- -- reset and clock

rst <= sReset;

sClock <= not sClock after clockPeriodInn/2;
clk <= sClock;

sClock_fpga <= not sClock_fpga after clockPeriod/2;
fpga_sck <= sClock_fpga ;

----- -- tester

process
  variable state : std_ulogic := '0';
begin
  -- Outputs default values
  sReset <= '1';
  acq_pretrig <= '0';
  acq_trig <= '0';
  adc_sdo <= (others => '1');
  fpga_m <= (others => '0');
  fpga_mosi <= '0';
  clk_en <= '0';
  clk_start <= '0';

----- -- TESTER MCU

  counter_test <= (others => '0');
  count_memory <= (others => '0');
  count_memoryMax <= (others => '1');
  ADC_WRITE_INIT <= "1010001010000000"; --1 010 00 10 10000000
  FRAM_READ_INIT <= SHIFT_LEFT(RESIZE("0000001100000",FRAM_READ_INIT),19);
  FRAM_WRDI_INIT <= "00000100";
  FRAM_WREN_INIT <= "00000110";
  pretrigVal <= "01010000";
  FRAM_WRITE_INIT <= SHIFT_LEFT(RESIZE("0000001000000",FRAM_READ_INIT),19);
  init_done <= '0';
  miso_start <= '0';
  selector <= (others => '0');
  count_miso <= (others => '0');
  FRAM_READ_ADD <= (others => '0');

  testInfo <= pad("Init", testInfo'length);
  wait for 20*clockPeriod;

```

```

sReset <= '0';
wait until rising_edge(sClock_fpga);

while true loop
    -- Wait until toggler should toggle
    wait for clockPeriod;

    if selector = "0000" then
        selector <= "1110";

    elsif selector = "1110" then
        fpga_m <= "1110";
        if clk_start = '0' then
            wait for clockPeriod;
            clk_start <= '1';
        end if;
        acq_pretrig <= '1';
        fpga_mosi <= pretrigVal(pretrigVal'high);
        pretrigVal <= SHIFT_LEFT(pretrigVal,1);
        counter_test <= counter_test+1 ;
        if counter_test = value_8 then
            counter_test <= (others => '0');
            clk_start <= '0';
            selector <= "0001";
            acq_pretrig <= '0';
        end if ;

    elsif selector = "0001" then
        fpga_m <= "0001";
        if clk_start = '0' then
            wait for clockPeriod;
            clk_start <= '1';
        end if;
        fpga_mosi <= ADC_WRITE_INIT(ADC_WRITE_INIT'HIGH);
        ADC_WRITE_INIT <= SHIFT_LEFT(ADC_WRITE_INIT,1);
        counter_test <= counter_test +1;
        if counter_test = value_16-1 then
            counter_test <= (others => '0');
            clk_start <= '0';
            selector <= "1000";
        end if ;

    elsif selector = "1000" then
        fpga_m <= "1000";
        if clk_start = '0' then
            wait for clockPeriod;
            clk_start <= '1';
        end if;
        if init_done <= '0' then
            fpga_mosi <= FRAM_WREN_INIT(FRAM_WREN_INIT'high);
            FRAM_WREN_INIT <= SHIFT_LEFT(FRAM_WREN_INIT,1);
            counter_test <= counter_test +1;
            if counter_test = value_8-1 then
                counter_test <= (others => '0');

```

```

    clk_start <= '0';
    selector <= "1111";
end if ;
else
    fpga_mosi <= FRAM_WRITE_INIT(FRAM_WRITE_INIT'high);
    FRAM_WRITE_INIT <= SHIFT_LEFT(FRAM_WRITE_INIT,1);
    counter_test <= counter_test +1;
    if counter_test = value_32-1 then
        counter_test <= (others => '0');
        clk_start <= '0';
        selector <= "1010";
    end if ;
end if ;

elsif selector = "1111" then
    fpga_m <= "1111";
    if clk_start = '0' then
        wait for clockPeriod;
        clk_start <= '1';
    end if;
    counter_test <= counter_test +1;
    if counter_test = value_8-1 then
        counter_test <= (others => '0');
        init_done <= '1';
        clk_start <= '0';
        selector <= "1000";
    end if ;

elsif selector = "1010" then
    fpga_m <= "1010";
    if clk_start = '0' then
        wait for clockPeriod;
        clk_start <= '1';
    end if;
    count_memory <= count_memory+1;
    if count_memory = count_memoryMax then
        acq_trig <= '1';
    end if ;
    if out1 = '1' then
        clk_start <= '0';
        selector <= "1011";
    end if ;

elsif selector = "1011" then
    fpga_m <= "1011";
    if clk_start = '0' then
        wait for clockPeriod;
        clk_start <= '1';
    end if;
    if fpga_miso = '1' and miso_start = '0' then
        --count_miso <= count_miso+1;
        miso_start <= '1';
        acq_trig <= '0';
    elsif miso_start = '1' and count_miso <19 then
        count_miso <= count_miso+1;
    end if;
end if;

```

```

    FRAM_READ_ADD <= SHIFT_LEFT(FRAM_READ_ADD,1);
    FRAM_READ_ADD(0) <= fpga_miso;
  elsif count_miso = 19 and miso_start = '1' then
    miso_start <= '0';
    count_miso <= (others => '0');
    FRAM_READ_INIT <= RESIZE(FRAM_READ_INIT+
                                RESIZE(FRAM_READ_ADD,
                                       FRAM_READ_INIT),
                                FRAM_READ_INIT);
    clk_start <= '0';
    selector <= "0100";
  end if;

  elsif selector = "0100" then
    fpga_m <= "0100";
    if clk_start = '0' then
      wait for clockPeriod;
      clk_start <= '1';
    end if;
    fpga_mosi <= FRAM_READ_INIT(FRAM_READ_INIT'high);
    FRAM_READ_INIT <= SHIFT_LEFT(FRAM_READ_INIT,1);
  end if ;
  -- Invert state and loop
  state := not state;

end loop;

end process;

```

END ARCHITECTURE test;

A.7 Script Matlab

```
%x = 0:1:29;
%y = [0 51 255 17 2 255 255 255 255 68 51 34 17 2 255 255 255 255 68 51 34 17 2 255 255 255 68 51 34 17 2 255 ];
%plot(x,y)
fileID0 = fopen('FRAM0_0.BIN','r');
FRAM0 = fread(fileID0, Inf, 'int16');
figure;
plot(FRAM0);
title('FRAM0');

fileID1 = fopen('FRAM1_0.BIN','r');
FRAM1 = fread(fileID1, Inf, 'int16');
figure;
plot(FRAM1);
title('FRAM1');

fileID2 = fopen('FRAM2_0.BIN','r');
FRAM2 = fread(fileID2, Inf, 'int16');
figure;
plot(FRAM2);
title('FRAM2');

fileID3 = fopen('FRAM3_0.BIN','r');
FRAM3 = fread(fileID3, Inf, 'int16');
figure;
plot(FRAM3);
title('FRAM3');
```

```

visaresource = 'USB0::2733::470::203458::0::INSTR'; %define visa
%resource name UNCOMMENT when name has been found

if(~exist('vsdev'))
    if(~exist('visaresource') || ~any(visaresource))
        visadevlist
        return
    else
        vsdev = visadev(visaresource);
    end
end

Nedges = [];
f2s = [];

veroff = 10;

%for(k=1:10)
while(1)

    ch1 = 0;
    ch2 = 0;
    ch3 = 0;
    ch4 = 0;

    Navg = 1;

    [ch1tmp, time] = RSscopeReadData(vsdev,1);
    ch1=ch1+ch1tmp;

    ch3 = ch3+RSscopeReadData(vsdev,3);

    pause(0.1)

    ch1 = ch1(1:3e3)';
    ch3 = ch3(1:3e3)';
    time = time(1:3e3)';
    edgeCh1 = diff(ch1);
    [x,ind1] = min(edgeCh1);
    edgeCh1(ind1-veroff:ind1+veroff) = 0;
    [x,ind2] = min(edgeCh1);
    f2 = 1/abs(time(ind2)-time(ind1));
    f2s(end+1) = f2;

    %count SCLK edges

    ch1 = ch1(1:2e3)';
    ch3 = ch3(1:2e3)';
    posedge = (((0.14-ch1(1:end-1)).*diff(ch3))>0).*((0.14-ch1(1:end-1)).*diff(ch3));

```

```
lim = 4e-3;
horOff = 10;
inds = find(posedge>lim);
Nedge = 0;
while(inds)
    Nedge = Nedge+1;
    posedge = posedge(inds(1)+horOff:end);
    inds = find(posedge>lim);
end

Nedges(end+1) = Nedge;
end

Nedges
f2s
```


B | Durabilité

Le 25 septembre 2015, les dirigeants nationaux ont approuvé à l'unanimité la résolution intitulée "Transformer notre monde : le Programme de développement durable à l'horizon 2030". Ce programme établit un cadre mondial de référence pour promouvoir le développement durable.

La Suisse est résolument engagée à mettre en œuvre ce programme, que ce soit à l'échelle internationale ou nationale.

Ce programme établi, cible les défis mondiaux majeurs et pose les priorités pour un développement durable. Il traite entre autre de l'éradication de la pauvreté extrême et la faim, la préservation de notre planète, la promotion de la paix et la prospérité, et renforcer la coopération internationale. Les résultats de la Conférence internationale sur le financement du développement de juillet 2015 sont également intégrés dans l'Agenda 2030 pour le développement durable.



Figure B.1 Les 17 objectifs de développement durable [14]

Au cœur de cet agenda se trouvent les 17 objectifs de développement durable (ODD) et leurs 169 cibles correspondantes. Ces objectifs ont pour échéance 2030 et s'appliquent à tous les États membres, qui peuvent les adapter à leurs besoins. Les ODD sont interconnectés et englobent les dimensions sociales, économiques et environnementales de manière équilibrée.

Dans le contexte du travail de bachelor de cette année, il nous a été demandé de revoir notre approche vis-à-vis de ces objectifs et de nos projets.

Cependant, en ce qui concerne mon travail, dont le but final et de pouvoir récolter des valeurs en provenance de test d'explosion, il devient complexe de déterminer comment

Appendix B. Durabilité

rendre mon projet plus "durable".

Peut-être serait-il envisageable de réévaluer la consommation de cet appareil ou encore les matériaux choisis, afin de le rendre potentiellement plus respectueux de l'environnement.

Cet appareil n'étant pas destiné à une production de masse et sa consommation électrique demeurant relativement faible il respecte déjà plusieurs points des 17 objectifs de développement durable tel que les objectifs n°7,9,12 et 13.

Acronyms

ADC Analog-to-Digital converter. [4](#)

FLASH Mémoire de masse. [10](#)

FPGA Field-programmable gate array. [4](#)

FRAM Ferroelectric Random Access Memory, mémoire non-volatile. [4](#)

PCB Printed Circuit Board (circuit imprimé). [6](#)

SD Carte SD (Secure Digital). [3](#)

SELECTOR Bus de 4 bits permettant l'utilisation des services de la FPGA. [15](#)

SPI Service Provider Interface, service de communication. [4](#)

STM32 micro contrôleur employé sur la board. [4](#)

