Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

Degree Course Systems Engineering

Major: Infotronics

Diploma 2013

Stéphane Lovejoy

CubETH Command and Data Management System

Professor
Christophe Bianchi

Expert
Luzius Kronig

Submission date of the report
12.07.2013

HES-SO Valais-Wallis • rte du Rawyl 47 • C.P. • 1950 Sion 2
+41 27 606 85 11 • info@hevs.ch • www.hevs.ch

HES-SO Valais

SI	TV
X	X

Données du travail de diplôme
Daten der Diplomarbeit

FO 1.2.02.07.AB
pof/31/01/2009

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2012/13**	No TD / *Nr. DA* **it/2013/19**
Mandant / *Auftraggeber* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Stéphane Lovejoy** Professeur / *Dozent* **Christophe Bianchi**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Expert / *Experte* *(données complètes)* **Luzius Kronig** EPFL | AA CTS-GE | ELD 16 | Station 11 | 1015 Lausanne	

Titre / *Titel*

CubETH Command Data and Management System

Description et Objectifs / *Beschreibung und Ziele*

The Swiss Space Center is involved in a new CubeSat program called CubETH, whose goal is to fly four GNSS (Global Navigation Satellite System) sensors on a cubesat. The GNSS sensors will be used for orbit determination and if possible also for attitude determination. The Geodesy and Geodynamics lab of the ETHZ is responsible for the payload. EPFL is responsible for the platform.

The goal of the Bachelor Thesis is to contribute to the development of the CDMS breadboard. The tasks to realize are:

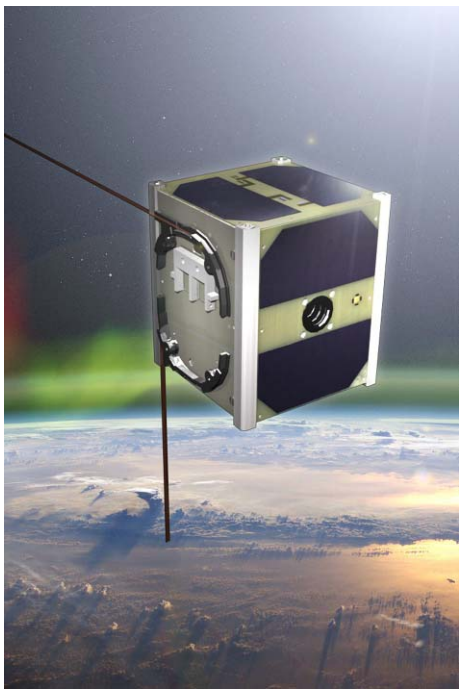
— Review of the existing CDMS design
— Design and development of the electronics of the magnetic torquer
— Design and development of the electronics for the read-out of the sun sensors
— Integration of theses electronics with the FPGA CDMS board already used during the semester project
— Propose a demonstrator to control the magnetic torquer from the sun sensors.

Signature ou visa / *Unterschrift oder Visum*	Délais / *Termine*
Resp. de la filière *Leiter des Studieng.:*	Attribution du thème / *Ausgabe des Auftrags:* 13.05.2013
[1] Etudiant/Student:	Remise du rapport / *Abgabe des Schlussberichts:* 12.07.2013
	Expositions / *Ausstellungen Diplomarbeiten:* 28 – 30.08.2013
	Défense orale / *Mündliche Verfechtung:* Semaine | Woche 36

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement le caractère confidentiel du travail de diplôme qui lui est confié et des informations mises à sa disposition.*

Durch seine Unterschrift verpflichtet sich der Student, die Vertraulichkeit der Diplomarbeit und der dafür zur Verfügung gestellten Informationen zu wahren.

Rapport reçu le / *Schlussbericht erhalten am* ……. Visa du secrétariat / *Visum des Sekretariats* ……..



Bachelor's Thesis
| 2013 |

Degree course
Systems Engineering

Field of application
Infotronics

Supervising professor
Christophe Bianchi
Christophe.Bianchi@hevs.ch

Partner
Polytechnic School of Lausanne

CubETH Command and Data Management System

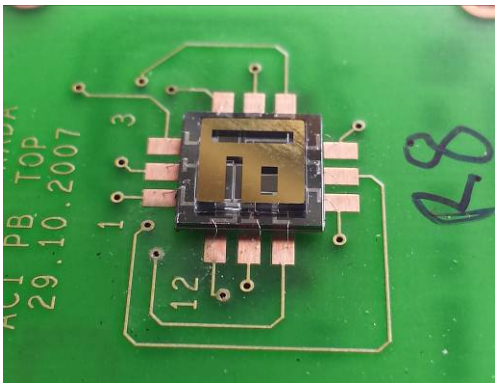
Graduate Stéphane Lovejoy

Objectives

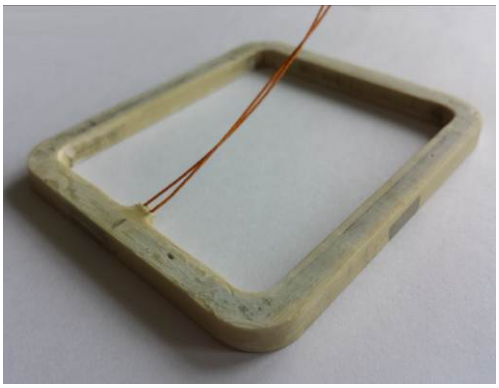
The goal of this Bachelor Thesis is to design a new board based on an FPGA to study the feasibility of this type of architecture for the attitude determination of a microsatellite.

Methods | Experiences | Results

This board will contribute to the development of the CubETH's board whose goal is to fly four GNSS sensors on a CubeSat. It commands a Magnetotorquer according to a vector indicating the direction of the sun from the satellite, based on the values returned by a Sun Sensor.
The board implements all the necessary electronics for the proper operation of the Magnetotorquer and of the Sun Sensor. This card has been designed and tested from scratch.
A PC application also has been created to establish a serial connection between the board and a computer. This application also computes the Satellite-Sun vector based on the values received from the Sun Sensor. It also allows the control of the Magnetotorquer by the use of a PWM with a configurable duty cycle. The application contains an additional feature that displays a virtual representation of the satellite that can orient itself based on the value of the Satellite-Sun vector previously calculated.
This architecture based on a FPGA works fine but needs further testing to be completely validated.



Sun Sensor wire bonded on its test board.



Magnetotorquer used to actuate the microsatellite.

Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

Degree Course Systems Engineering

Major :Infotronics

Diploma 2013

Stéphane Lovejoy

CubETH Command and Data Management System

Professeur Christophe Bianchi
Expert Luzius Kronig

Sion, le 12 juillet 2013

TABLE OF CONTENTS

LIST OF ACRONYMS

AD	Analogic to Digital
ADCS	Attitude Determination and Control System
API	Application programming interface
CAN	Controller area network
CDMS	Command and Data Management System
CRC	Cyclic Redundancy Check
FPGA	Field Programmable Gate Array
GNSS	Global Navigation Satellite System
MT	MagnetoTorquer
OpenGl	Open Graphics Library
PWM	Pulse Width Modulation
SPI	Serial Peripheral Interface
SS	Sun Sensor
UART	Universal Asynchronous Receiver Transmitter
VHDL	Hardware Description Language

1 FOREWORD

1.1 CubeSat

The CubeSat project was developed in 1999 at the California Polytechnic State University and at the Stanford University. Those two universities developed the specification of a CubeSat to allow universities worldwide to launch small satellites at reduced cost. The Cubesat must have cubic geometry, a volume of 10 cm3 and weight under 1.33 kg. [1]

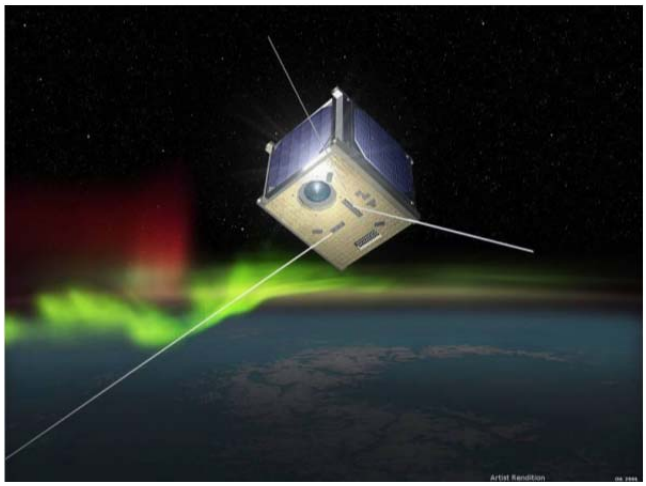


Figure 1: Artist View of the SwissCube CubeSat [2]

1.2 SwissCube

SwissCube is the first fully Swiss made satellite. It follows the CubeSat standards. The goal of this mission was to give to the EPFL students the opportunity to work on a satellite. The scientific goal of the SwissCube mission was to make optical measurements of the Nightglow phenomenon Figure 2.

Figure 2 : Nightglow phenomenon [3]

1.3 Semester project

A prototype board made to pilot a microsatellite was developed during a master's thesis project by Bastien Praplan at the HES in Sion. This board could be neither mounted nor tested. The goal of the semester project was to validate the proper functioning of this board. This board is composed of an FPGA that implements a basic architecture. This architecture allows great flexibility regarding the number and the types of peripherals that it implements.

2 BACHELOR'S THESIS DEFINITION

The Swiss Space Center is involved in a new CubeSat program called CubETH, whose goal is to fly four GNSS (Global Navigation Satellite System) sensors on a Cubesat. The GNSS sensors will be used for orbit determination and if possible also for attitude determination. This project is being developed by the ETHZ in Zurich in collaboration with the EPFL in Lausanne. The ETHZ is responsible for the payload of the mission and the EPFL is responsible for the platform. The HES-SO Valais/Wallis is working in collaboration with the l'EPFL, and therefore it will take part in the platform development.

The goal of the Bachelor's Thesis is to contribute to the development of the CDMS board (command data and management system).

The tasks to perform are:

- Review of the existing CDMS design
- Design and development of the electronics of the Magnetotorquer
- Design and development of the electronics for the read-out of the Sun Sensors
- Integration of these electronics with the FPGA CDMS board already used during the semester project
- Propose a demonstrator to control the Magnetotorquer from the Sun Sensors

3 INTRODUCTION

The present report describes the work accomplished during the Bachelor's thesis "CubETH Command and Data Management system".

As explained in the previous chapters, the EPFL is currently working on a new CubeSat project that involves the design of a new platform board for the satellite.

For redundancy purposes, such as in the case of a malfunction of the ADCS board, the CDMS board would have to take over with a simple attitude determination algorithm. Therefore it would have to be capable of reading the incoming values from the different input sensors and control the different actuators of the satellite. The EPFL's request is therefore that a study be made to determine whether, in case of malfunction of the ADCS board, it can be isolated from the different sensors and actuators so they can be accessed from and managed by the CDMS board.

In the context of this work, an ADCS/CDMS board will have to be developed. This board will be based on the FPGA board tested during the semester project. It will have to be able to handle the acquisition of the Sun Sensor's values. It will also have to be able to control the Magnetotorquer. To perform these two tasks, the necessary electronics to make the connection with the Sun Sensor and the Magnetotoquer will have to be created from scratch.

A serial communication will have to be implemented between the ADCS/CDMS board and the computer. This communication will have to be established so the attitude determination algorithm can be initially implemented on the computer and then subsequently on the board.

As the satellite's attitude determination algorithm will be implemented on a computer, a program that simulates the satellite's attitude will have to be created. This simulation will show to the user a virtual version of the satellite that can be oriented in real time with the values received from a Sun Sensor.

4 SYSTEM DESCRIPTION

4.1 Overall description of the system

This chapter gives an overall explanation of the system. A more detailed description will be elaborated in further chapters. The diagram in the Figure 3 shows the different actors that take part in the design.

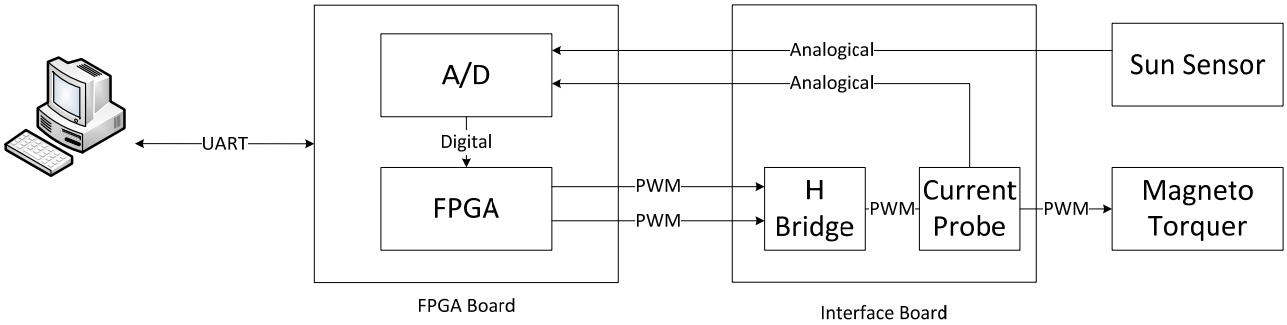


Figure 3 : Diagram of the system

The goal of the system is to read the values from a Sun Sensor, evaluate them to determine the position of the sun and drive a Magnetotorquer. This system is used to simulate the orientation of a satellite in a chosen direction.

This system comprises an application program that's running on a PC as a controller. This application commands an FPGA to obtain the values of the Sun Sensor. Once these values are obtained, a computation is performed to determine the sun's position. Then the order is given to the FPGA to drive the Magnetotorquer accordingly. A current probe is used to measure the current that is provided to the Magnetotorquer. This information is converted into a digital value and then transmitted back to the application. This principle is used to determine when the Magnetotorquer needs to be turned off accordingly to the torque provided.

This application also allows the visualization of the satellite's attitude by making a graphical simulation of the satellite. This computer simulation orients itself according to the values received from the Sun Sensor and the current measurements done on the Magnetotorquer.

The Magnetotorquer is driven with a PWM provided by the FPGA. An H-Bridge then allows for activation of the Magnetotorquer in one way or the other.

4.2 ADCS

The ADCS (Attitude Determination and Control System) is the part of the satellite that determines its attitude. To achieve that goal, the ADCS disposes of several inputs that are used to determine its orientation in space. These include: magnetometer, gyroscope and Sun Sensors. It also commands actuators that allow the control of the satellite: Magnetotorquer, reaction wheel.

4.3 CDMS

The CDMS (Command and Data Management System) manages the command and the data of the satellite. Its main objective is to compute the incoming data produced by the satellite: orientation and status of the different sub-systems and to store them before sending them to earth.

For the CubETH mission, the CDMS board is also used as a backup system. In case of an ADCS failure, the CDMS must be able to take over the basic attitude determination of the satellite. Therefore it will have to be able to process the incoming data from the Sun Sensors and also to drive the Magnetotorquers.

4.4 Sun sensors

The Sun Sensors (SS) are used for the attitude determination of the satellite. They allow the determination of the Satellite-Sun vector by giving two angles. Each SS provides two angles indicating the orientation of the sun compared to the satellite's face it is placed on.

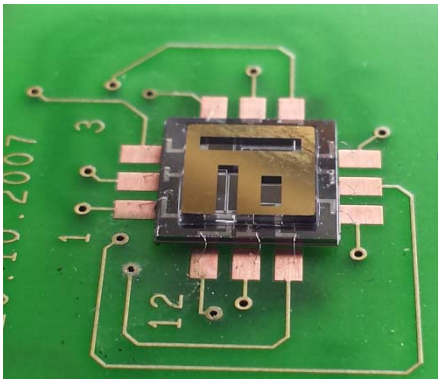


Figure 4 : DTU Sun Sensor

The SSes used for CubETH are the same as were used for the SwissCube mission. These SSes were developed at the Danish Technical University. This university also used them in a previous CubeSat mission called DTUsat. Therefore these SSes have already been proven in several space missions.

4.4.1 Description

This chapter contains a basic description of the Sun Sensors. More precise information can be found in the report in bibliography [4]. This report can be found in the DVD in annex.

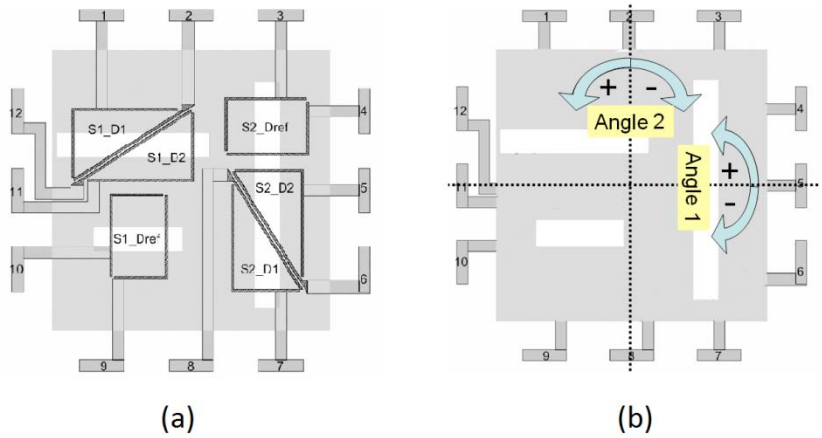
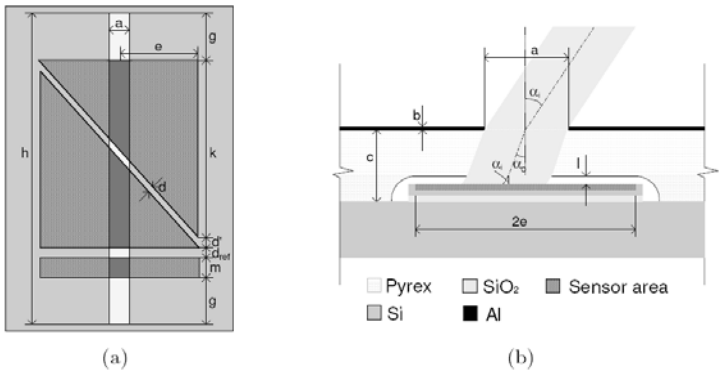


Figure 5 : (a) Top view of the sun sensor chip. (b) Axes of angle 1 and 2 [4]

For the determination of the two angles, (Figure 5b) six photodiodes are used. When these photodiodes are exposed to sunlight, they generate a current. This current is proportional to the illuminated surface, which depends on the incident angle of the sunlight as shown in Figure 6 (b).



Description		Value [µm]
Slide width	a	550
Slit extra height	g	150
Triangle height	k	2405
Triangle width	2e	1505
Reference area height	m	1190
Sensor-Pyrex distance	l	5
Pyrex thickness	c	500
Aluminium cover	b	N/A

(c)

Figure 6 : Top view (a) and front view (b) and Sensor dimensions (c) of one of the slits of the sensor [4]

4.4.2 Electronics of the Sun Sensor

This chapter explains a simplified version of the electronics of the SS. The complete and exact electrical schematics can be found in appendix 2. As shown in Figure 7, the circuit is a current-to-voltage converter with a gain set by R1 (a) and R2 (b).

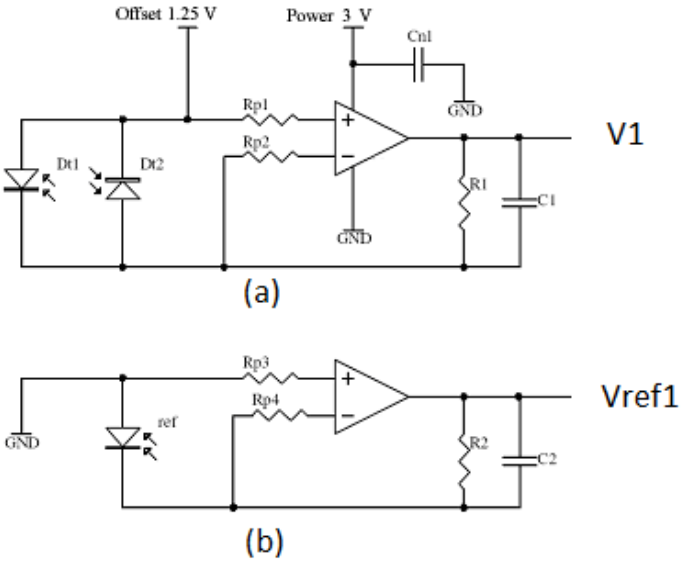


Figure 7 : Electronics for one angle. Simplified schematic. (a) Angle detection diodes, (b) Reference diode [4]

In the part (a) of the circuit, the two photo diodes are connected head to tail. This mounting type is used to have the differential current between Dt1 and Dt2 flowing in the resistor R1. An offset of 1.25 V is added to have exclusively positive tension on the output V1. The expression of V1 can be written as follows:

$$V1 = (IDt1 - IDt2) * R1 + offset$$

The expression of Vref1 is a simple ohm low and can be written as following:

$$Vref1 = R2 * Iref$$

Low pass filters R1C1 for (a) and R2C2 for (b) were added to both circuits. The constant of the low pass filters was set as explained in the report [4].

4.4.3 Typical output of the Sun Sensor

This test was performed during the "characterization and the calibration of the SS" at the EPFL in 2008 for the SwissCube mission. Further explanations and tests can be found in the report [4].

Figure 8 shows the output for the SS with the secondary angle set to 0°. The secondary angle was set to 0 because of the influence of it on the first angle. The interference between the two angles is corrected by the angle calculation algorithm, explained in chapter 8.3.5.

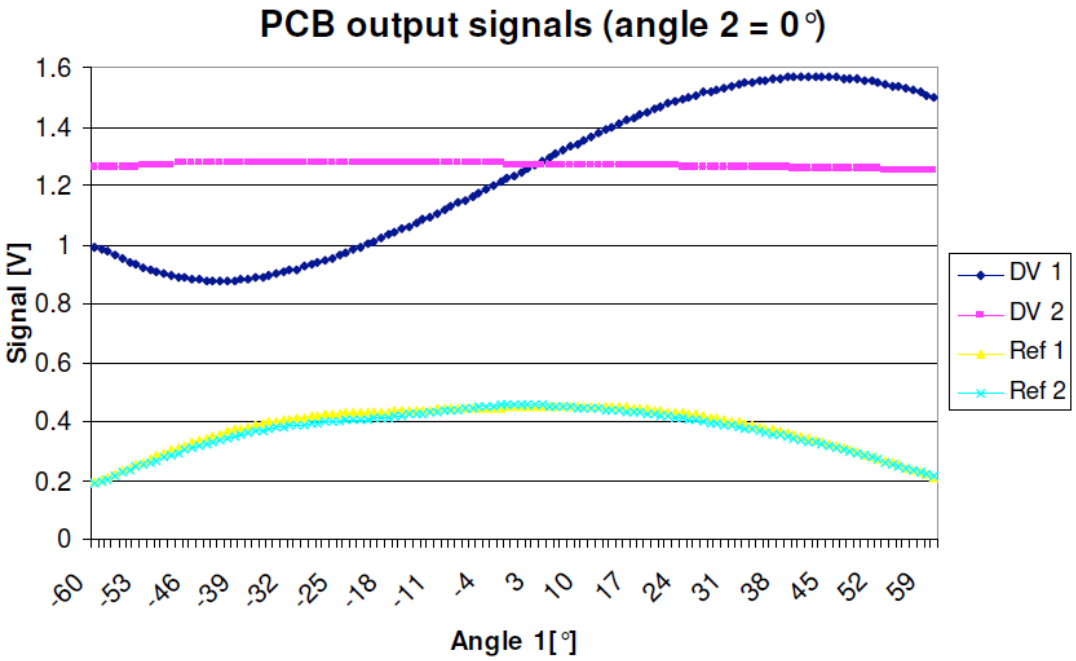


Figure 8: Output signals of a SS. Angle 2 fix at 0°. [4]

The accuracy of the SS is best for small angles, around 60°. Therefore the test was run for maximum angles of -60° to 60°. The output angles have to be computed as expressed in the following formula:

$$Angle\ 1{,}2 = \frac{\Delta\ DV_{1,2} - V_{offset}}{Ref_{1,2}} = \frac{\Delta\ DV_{1,2} - 1.25[V]}{Ref_{1,2}}$$

4.5 Magnetotorquer

As the Sun Sensors are used to define the position of the sun, the Magnetotorquers (MT) are used as actuators to orient the satellite in space. CubETH implements 3.

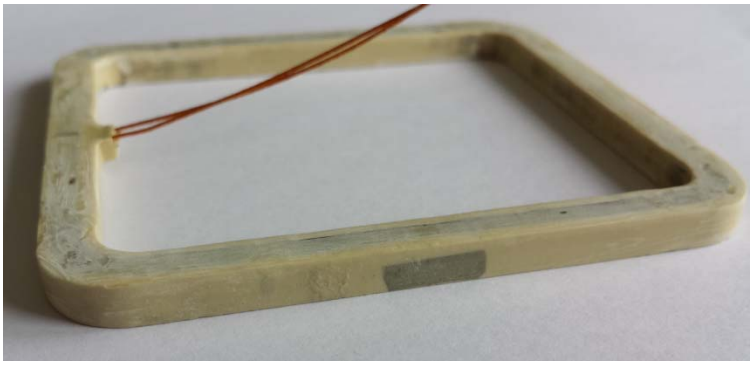


Figure 9 : Magnetotorquer

The MTs used for CubETH are the same as those used in the previous SwissCube mission.

These chapters contain a basic description of the Magnetotorqer and of its related electronics. The tests and the characterization below were performed during the Magetotorquer Impedance Characterisation at the EPFL in 2008 for the SwissCube mission. Further explanations and tests can be found in the following report [5].

4.5.1 Description

The MTs are simply coils of wire which produce a magnetic field. The interaction between the natural magnetic field of the earth and the one generated by the MT produces a torque. This resultant torque is used to orient the satellite.

The following figure shows the principle of the electronic used to command the MT.

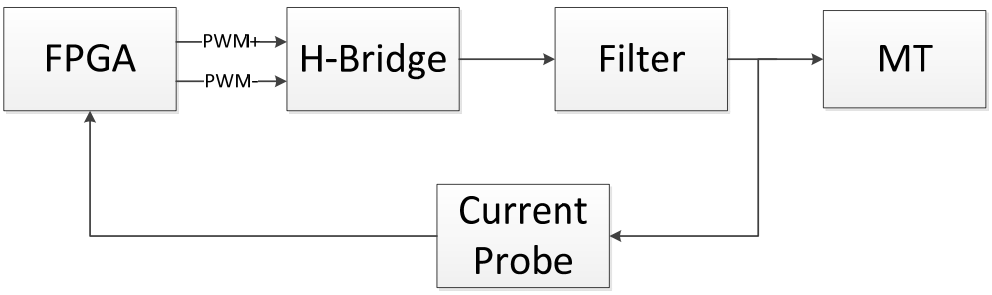


Figure 10: Principle of the MT electronics

As shown in Figure 10, an FPGA is used to generate a PWM signal. This PWM is generated on one of the two inputs of the H-Bridge. This bridge allows us to define the direction of the current that will flow in the MT. A filter is added to smooth this current. A current probe performs the measurement of the current that is actually flowing in the MT.

4.5.2 Characterization of the Magnetotorquer's Impedance

These tests were performed during the "Magetotorquer Impedance Characterisation" work at the EPFL in 2008 for the SwissCube mission. Further explanations and tests can be found in the following report [5].

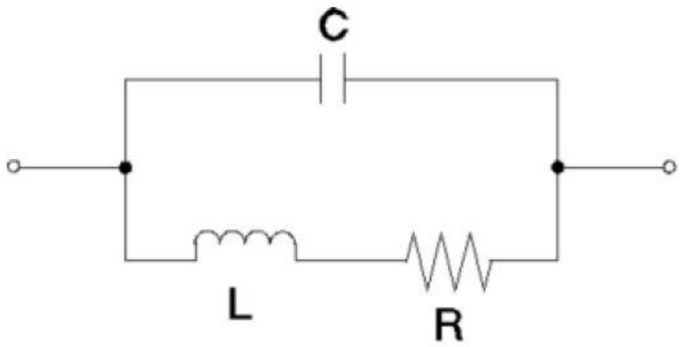


Figure 11: Magnetotorquer equivalent circuit [5]

As shown in the Figure 11, the equivalent circuit of the MT is composed of one serial resistor and of a capacitor added in parallel.

The following characteristics were measured with a spectral analyzer at ambient temperature. These measurements give the values of the equivalent circuit of the MTs.

	Unit	MT 01	MT 02	MT 03
Resistance	[Ω]	110.5	111.5	110.8
Inductance	[mH]	29.2	29.4	29.4
Capacitance	[pF]	83.4	69.3	84.3

Figure 12 : Magnetotorquer measured characteristics at ambient temperature [5]

	Unit	MT 01	MT 02	MT 03
Impedance norm |Z|	[kΩ]	6.64	6.56	6.70
Impedance angle	[°]	88.58	88.64	88.65

Figure 13: Magnetotorquer measured characteristics at 32.5 [kHz] and ambient temperature [5]

4.5.3 Current probe

The current probe measures the current that flows in the MT. This current can be either positive or negative. Therefore, a system capable of measuring bidirectional current was adopted. The following figure shows the current probe that was chosen.

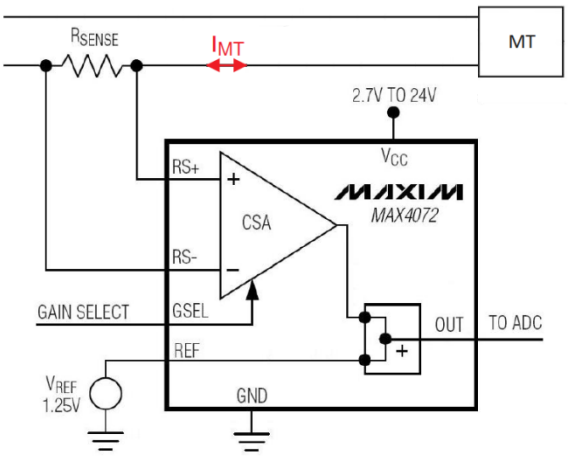


Figure 14: MAX4072 current probe

This device amplifies a differential voltage on a shunt resistor. An offset is added to this differential voltage and it is then displayed on the output. The gain can be set to 50 or 100 and the recommended Rsense value goes from 5 [mΩ] to 1 [Ω].

The configuration used for this design is:

Rsense = 1 [Ω]
Gain = 50 (GSEL set to zero)
Vref = 1.25 [V]

The following figure shows the theoretical output voltage of the current probe.

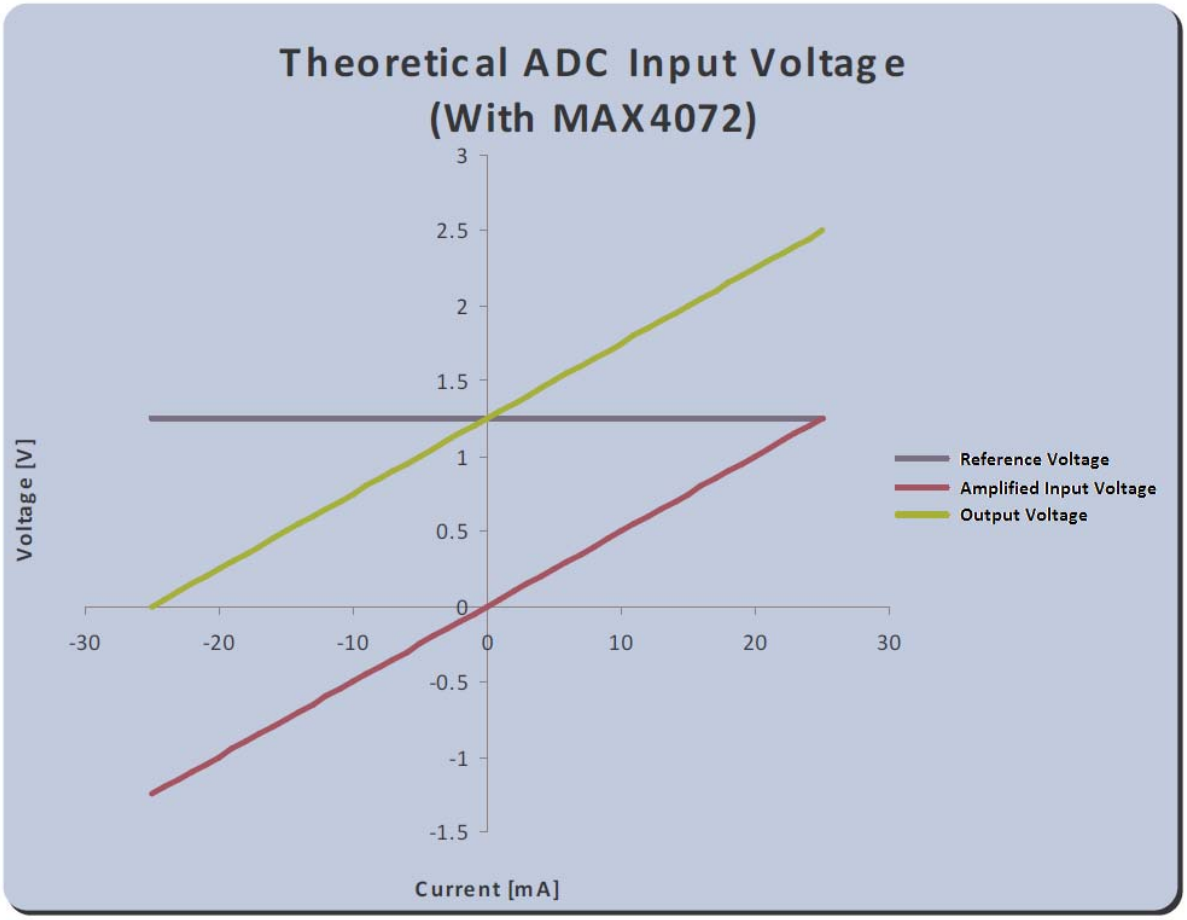


Figure 15: Theoretical output voltage of the current probe [5]

We can see that with the offset of 1.25[V], the yellow curve is always positive for a range starting at -25[mA]. This is a sufficient offset, as the current in the MT never exceeds -23.4 [mA].

4.5.4 H-Bridge and filter

The H-Bridge allows the control of the current flow direction in the MT and powers it. The PWM is generated by the FPGA and the current is amplified using the H-Bridge. The following figure shows this aforementioned schematic.

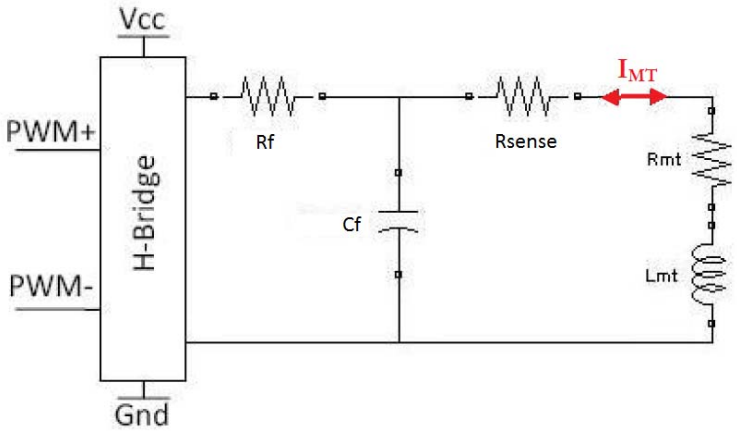


Figure 16: Magnetotorquer electronic schema

A low-pass filter is added to lower the current variation.

4.6 FPGA board

This FPGA board was developed during a master's project by Bastien Praplan at the HES in Sion. This board is composed of an FPGA that implements a basic architecture. This architecture allows a lot of flexibility in the numbers and in the types of peripherals that can be connected to it.

This board is used to make the link between the interface board and the application program on the PC. It is used to gather the data sent by the SS and the current measured on the MT and send them to the application where they are treated. The incoming data are in the form of analogical signals. Therefore, the FPGA board also implements an A/D converter that allows the digitalization of this data. This board also treats the actuation data for the MT and generates the corresponding PWM.

Further information on the FPGA board can be found in report "Cart processeur pour satellite" that can be found on DVD in annex and the rapport "Satellite navigation with GPS" that can be found in appendix 1.

4.7 AD converter

The AD converter used for this system is an `ADS8028` from Texas Instrument. It is a 12 bit resolution converter with 8 conversion channels. The communication with the AD converter is made with an SPI bus. As a mistake was made initially during the semester project, the communication between the AD converter and the FPGA will be re-explained in the following chapter.

4.7.1 Serial Interface

The Figure 17 shows the detailed serial interface timing diagram for the ADS8028. The device used a serial clock (SCLK) for internal conversion and for data transfer into and out of the device.

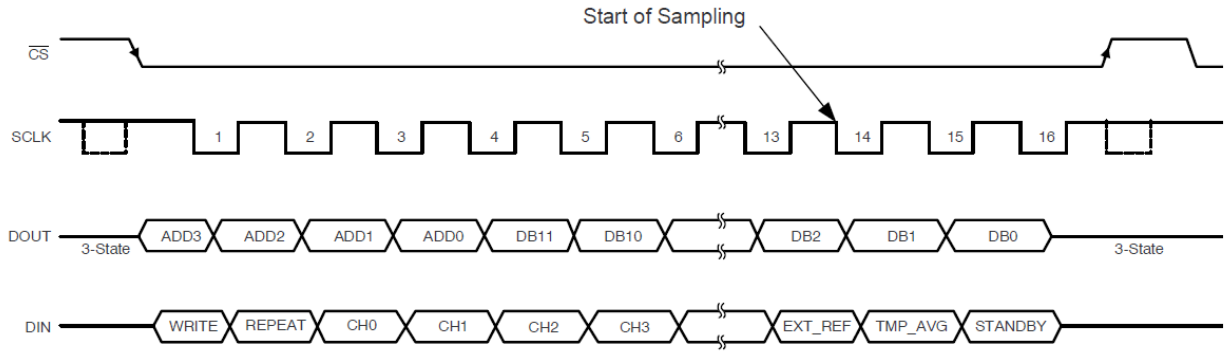


Figure 17: Serial interface timing diagram [6]

The CS signal defines one frame of conversion and serial transfer. The sequence of 16 SCLK is used for conversion and data transfer. For a valid read or write operation to the AD converter, 16 clocks must be provided on the SCLK pin between the CS falling and its rising edge.

Several modes of operation can be selected by programming the control register of the AD. For this application the single channel, one conversion mode is selected and will therefore be explained. The following figure shows the sequence that has to be applied to the converter to configure its control register and to read data from it. [6]

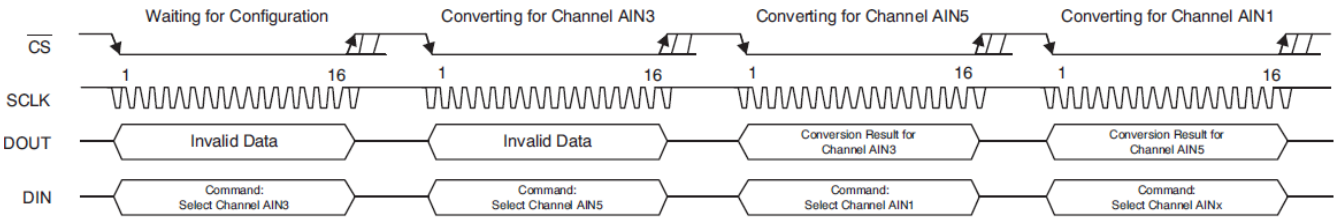


Figure 18: Configuring a conversion and read converted value from ADS8028. One channel selected for conversion, no repeat. [6]

A valid write operation must be executed on the control register to select the desired channel. The selected channel is converted in the second frame and the conversion result can be clocked out in the third frame. During the second frame, the control register can be written to select another (or the same) conversion channel, as shown in the Figure 18. [6]

4.7.2 Data write operation

The control register configures the `ADS8028` for the next conversion cycle. To configure the control register, a valid sequence of 16 bits has to be performed. The following figure shows the bit functions of the control register.

MSB 15	14	13	12	11	10	9	8
WRITE	REPEAT	AIN0	AIN1	AIN2	AIN3	AIN4	AIN5
7	6	5	4	3	2	1	LSB 0
AIN6	AIN7	T_{SENSE}	X	X	EXT_REF	TMP_AVG	STANDBY

Bit 15 — **WRITE: Write to Control Register**

Enable write operation.
0 = Write disabled; Control Register is not updated and the next 15 bits are ignored (default)
1 = Write enabled; the next 15 bits update the Control Register

Bit 14 — **REPEAT: Repeat conversion mode**

Enable conversion repeat mode (refer to the *Modes of Operation* section).
0 = Disable repeat conversion mode (default)
1 = Enable repeat conversion mode

Bits[13:6] — **AIN[0:7]: Analog input channel selection**

Each AINx bit corresponds to the associated analog input channel, AIN0 to AIN7.
0 = AINx channel is not selected for conversion (default)
1 = AINx channel is selected for conversion

Bit 5 — **T_{SENSE}: Internal temperature sensor selection**

Internal temperature sensor selection for conversion in subsequent cycles.
0 = Internal temperature sensor output is not selected for conversion (default)
1 = Internal temperature sensor output is selected for conversion

Bits[4:3] — **X: Don't care**

Bit 2 — **EXT_REF: Reference source selection**

This bit selects the reference source for the next conversion.
0 = Internal reference is used for the next conversion (default)
1 = External reference is used for the next conversion

Bit 1 — **TMP_AVG: Temperature sensor averaging selection**

This bit selects the mode of operation for the temperature sensor channel; this bit is ignored if bit 5 is set to '0'.
0 = Averaging is disabled on the temperature sensor result (default)
1 = Averaging is enabled on the temperature sensor result

Bit 0 — **STANDBY: STANDBY mode selection**

This bit sets the mode (normal or standby) for the ADS8028.
0 = The ADS8028 operates in normal mode (default)
1 = The ADS8028 goes to standby mode in the next cycle

Figure 19: Control register bit function [6]

For this system, the following configuration was chosen:

MSB 15	14	13	12	11	10	9	8
1	0	0/1	0/1	0/1	0/1	0/1	0/1

7	6	5	4	3	2	1	LSB 0
0/1	0/1	0	X	X	1	0	0

Figure 20: Configuration of the control register

As shown in the figure above, the reference source selection has been set to external. Further information on this choice can be found in the appendix 3. The temperature sensor is disabled and the normal mode is selected.

4.7.3 Data read operation

The following figure shows the `ADS8028` channel address bits.

ADD3	ADD2	ADD1	ADD0	ANALOG INPUT CHANNEL
0	0	0	0	AIN0
0	0	0	1	AIN1
0	0	1	0	AIN2
0	0	1	1	AIN3
0	1	0	0	AIN4
0	1	0	1	AIN5
0	1	1	0	AIN6
0	1	1	1	AIN7
1	0	0	0	T_{SENSE} without averaging
1	0	0	1	T_{SENSE} with averaging

Figure 21: Channel address bits [6]

As shown in Figure 17, the four first bits of the data frame specify the channel selected for conversion. The other 12 bits contain the conversion result for the selected channel.

The read operation is made as shown in Figure 18, two frames after the configuration frame. It is also clocked by the SCLK pin in the sequence of 16 clocks.

5 HARDWARE DEVELOPMENT

5.1 Interface board

In order to make the electronic link between the Sun Sensor, the Magnetotorquer and the FPGA board, it was necessary to create an interface board. The board was designed to provide the necessary electronics for proper functioning of an SS and an MT, such as a driver and a power supply.

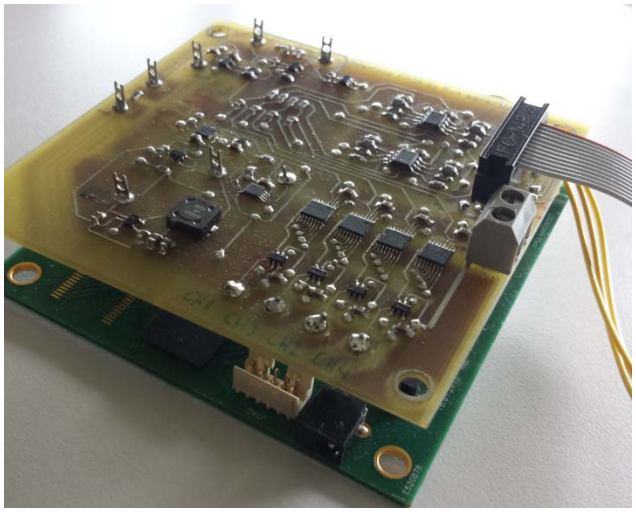


Figure 22: Picture of the Interface and FPGA board

On the figure above, one can see the interface board (yellow board) and the FPGA board (green board).

5.1.1 Description

This chapter explains the process that was followed to develop the interface board.

<u>Board specifications:</u>

For this Bachelor's Project the EPFL has provided only one SS and one MT. Therefore, the interface board was designed for one SS and one MT.

The different functions of the interface board are listed below:

- Provide the necessary power supplies for the MT and the SS.
- Convert in tension, amplify and filter the current emitted by the photodiodes of the SS.
- Implement the necessary electronics to drive the MT.
- Measure the current given to the MT when it is driven.

Schematics:

The different schematics were provided by the EPFL. They can be found in the DVD in annex. These schematics were not modified and the same components were used to provide a second validation of the design.

The interface board was initially designed for 6 SS and 3 MT (the number of SS and MT used for the satellite CubETH). However for reasons of time and complexity of routing, it was decided to make an interface for a unique SS and MT.

Schematics on PCad :

The schematic was created with PCad 2006.

In order to protect the more sensitive analogical signal, it was decided to completely separate the analogical and digital power supplies. Therefore, the two different worlds have their own DC/DC converters. The ground planes were also completely separated. There is only one single connection point.

All the schematics can be found in appendix 2.

5.1.2 Test of the card

Test of the schematic of the board

All the power sources of the board were tested. The measured values are shown is appendix 4. A control was also performed that showed that all complex components on the board were correctly powered and correctly connected together. This test can be seen in appendix 5.

These two tests validate that all the components are correctly powered and correctly linked together according to the schematic of the board "Interface FPGA to Sun sens/Mag Torquer, V1.0" that can be found in appendix 2.

Test of the current probe MAX4072

As per chapter 4.5.3, the current's value flowing in the MT was measured for various values of the PWM duty cycle at a frequency of 200 [kHz].

The following figure shows the practical output voltage of the current probe.

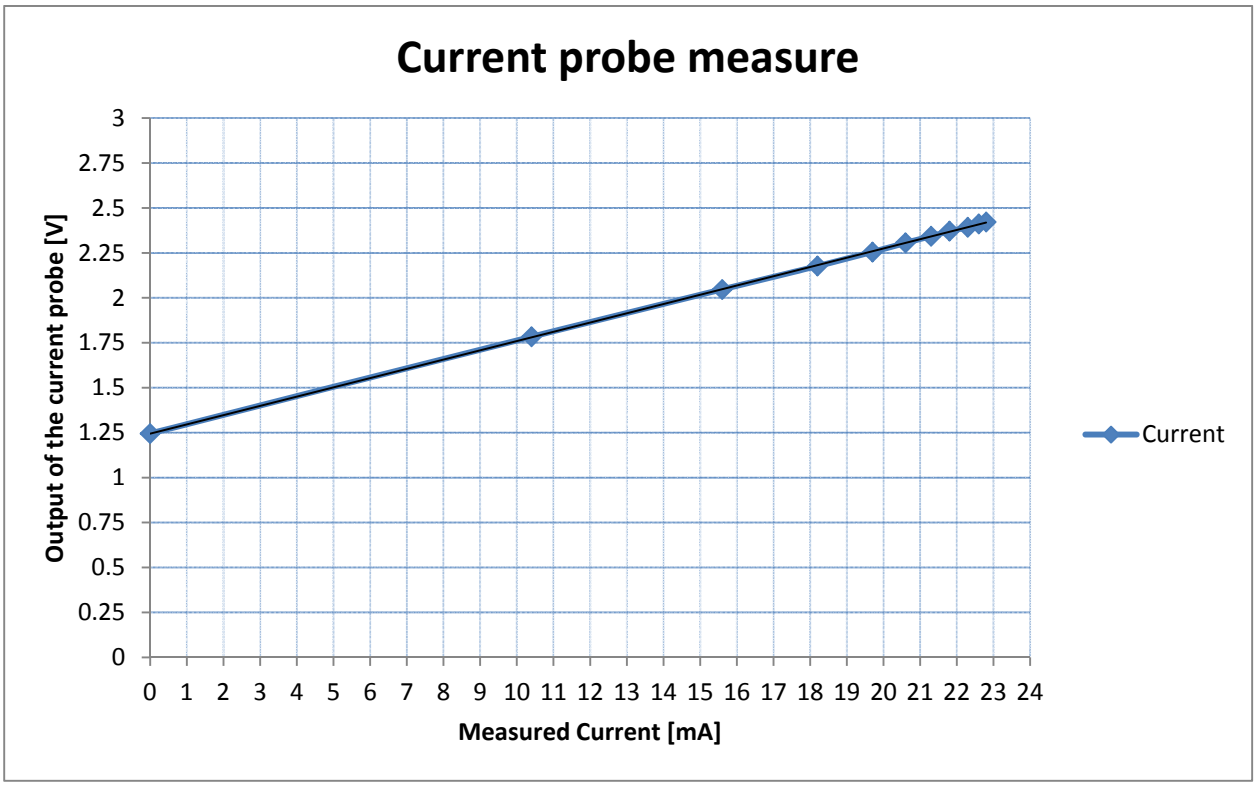


Figure 23: Practical output voltage of the current probe

As we can see on the figure above, the positive values are as expected. Unfortunately the negative measure doesn't worf. No explanation for this behavior has yet been found. The problem could come from either the current probe or the H-bridge. No other component could influence this behavior.

The complete test of the current probe can be found in appendix 3.

Test of the Sun Sensor's electronics

The output values of the SS were measured on `PT_SS_S1`, `PT_SS_R1`, `PT_SS_S2` and `PT_SS_R2` according to the schematic of the board "Interface FPGA to Sun sens/Mag Torquer, V1.0" that can be found in appendix 2, for various values of irradiation. This test is presented in the figure below.

	Irradiation of 1400 [W/m^2]		Irradiation of 1000 [W/m^2]	
	SS output Name	Measured voltage [V]	SS output Name	Measured voltage [V]
Angle of approximately 60°	S1	1.8	S1	1.7
	R1	0.8	R1	0.7
	S2	1.9	S2	1.8
	R2	0.9	R2	0.7
Angle of approximately -60°	S1	0.4	S1	0.67
	R1	0.9	R1	0.6
	S2	0.5	S2	0.8
	R2	0.8	R2	0.6
Angle of approximately 0°	S1	1.3	S1	1.2
	R1	0.9	R1	0.8
	S2	1.4	S2	1.3
	R2	0.9	R2	0.8

Figure 24: Measure of the SS's outputs values

We can see on the figure above, that the measurements don't match the intended values of Figure 8. All the reference values for R1 and R2 show higher voltage than expected. This could be modified by adapting the gain of the amplifier, as explained in chapter 4.4.2.

These measurements were taken at 1400 [W/m^2] because this is the theoretical level of irradiation in space near Earth, and at 1000 [W/m^2] because this is the irradiation level on Earth.

All those tests were performed with a halogen lamp of 1000 [W] and the irradiation was measured with a radiation meter with the following characteristics:

```
Brand: Kipp & Zonen
Type: CC20
Serial no: CC20960168
```

Global validation of the interface board

Apart from the problem of the current probe, the interface board works correctly. Several minor routing mistakes were made. No second PCB was made.Tthe routing mistakes were corrected on the board. A new version of the schematic has been made which correct those mistakes. This new schematic can be found in the DVD in annex.

6 COMMUNICATION PROTOCOL

6.1 Simple communication protocol

A simple communication protocol was elaborated to manage the flow of UART frames between the application on the PC and the FPGA. Three cases within the application use this protocol. The first one is when the application on the PC asks the FPGA to generate a PWM. The second one is when the application on the PC asks the FPGA to start an AD conversion and the last one is when the FPGA sends back to the application the result of the AD conversion. These three cases in the application are explained in details below.

6.1.1 PWM generation

This case of utilization is employed when the application on the PC needs to generate a PWM to command the MT on the Interface board. The duty cycle of the PWM can be configured along with the direction of the current that flows in the MT. The following figure shows the form of the message.

UART frame 1	UART frame 2	UART frame 3	
Start Frame	Address of the PWM block	Pos /Neg	Duty Cycle Value
10101010	10000000	x xxxxxxx	

Figure 25: Message to the FPGA containing the Duty Cycle of the PWM

Each frame is an individual UART frame of 8 bits of data plus the start and stop bits of the UART protocol. We can see on the Figure 25 that the first UART frame sent to the FPGA is the start frame. The second one contains the address of the PWM generator block (see chapter 7.3). As only one PWM generator block is present in the FPGA's design, only one address has been attributed. The last frame is separated into two parts. The first one contains the value of the current's direction that has to flow in the MT (Pos/Neg). If this value is set to one, the current will be positive and if set to zero the current generated will be negative. The second part of the last UART frame contains the value of the duty cycle that has to be generated by the block `PWM_Gen` of the FPGA. This value can be set from 0000000 to 1100100 (100).

6.1.2 AD conversion COMMAND

This case of utilization is emplyed when the application on the PC needs to obtain the values of the SS or the value of the current measure on the MT. The number of the AD channel on which the AD conversion will be performed is contained in the address of the AD_COM block.

UART frame 1	UART frame 2
Start Frame	AD conversion channel
10101010	0000xxxx

Figure 26: Message to the FPGA containing the AD channel

Each frame is an individual UART frame of 8 bits of data plus the start and stop bits of the UART protocol. We can see on Figure 26 that the first UART frame sent to the FPGA is the start frame. The second one contains the channel number on which the AD conversion must be performed. A masking of the channel numbers is performed in the FPGA so all possible channel numbers are interpreted as the AD communication block address. The list of the possible AD conversion channels and their corresponding frame values can be found below.

AD conversion channel	Value of the UART frame 2
AD channel 0	0000 0000
AD channel 1	0000 0001
AD channel 2	0000 0010
AD channel 3	0000 0011
AD channel 4	0000 0100
AD channel 5	0000 0101
AD channel 6	0000 0110
AD channel 7	0000 0111

Figure 27: List of the possible AD conversion channel

The masking is then performed on these possible UART frames with the following mask value: 0000 1111

6.1.3 AD value transmission

This case of utilization is employed when the FPGA sends back to the application on PC the value of the AD conversion. The channel on which the AD conversion was performed is sent followed by the 12 bits of the AD conversion.

UART frame 1	UART frame 2	UART frame 3		UART frame 3
Start Frame	AD conversion channel	Stuffing bits	AD MSB bits	AD LSB bits
10101010	0000xxxx	0000	xxxx	xxxxxxxx

Figure 28 : Message to the PC containing the result of the AD conversion and the channel it was performed on

Each frame is a separated UART frame of 8 bits of data plus the start and stop bits of the UART protocol. We can see on the Figure 28 that the first UART frame sent to the FPGA is the start frame. The second one contains the channel number on which the AD conversion was performed. The third frame of the message is constituted of padding bits and of the 4 MSBs of the AD conversion's result. The last frame contains the 8 last bits of the 12 bits constituting the result value of the AD conversion.

6.1.4 Weak points of this protocol

This simple protocol is used for the development phase of the system. It works correctly in a non-perturbed environment. As no control of the validity of the message is made, it can be compromised in a more perturbed environment.

Another problem of this simple protocol is that it is to linked to the code that receives the message, such that the receiving device must determine the length of the data contained in the message. For example, when the FPGA receives a message addressed to the PWM generator block (see chapter 7.3), the FPGA decodes the destination address, ascertains that it is destined for the PWM generator block and therefore assumes that the data length is of one frame. The problem lies in the fact that the addressee decoding and the data length decoding are derived from the same frame and therefore from the same data, thus strongly linking the protocol to the application. This is not an issue as long as the number of addressed block stays small, but the code would become much more complicated in the event of a large number of addressed blocks.

A solution for those two problems is proposed in the following chapter.

6.2 Final protocol proposition

The simple communication protocol described in the previous chapter was used for the development phase of the system. To address the weaknesses previously mentioned, a more complete protocol is proposed.

This protocol has been inspired by the CAN protocol.

Start Frame	Address	Length of the data (in bytes)	Data	CRC	Stop Frame

Figure 29: Proposition for the final communication protocol

The Start frame and Data could remain unchanged, but as we can see in Figure 29 the other frames would have to be modified to fit this protocol.

The `Address` frame contains the destination address. The destination could be a block on the FPGA, the computer or another device.

The next frame contains the length of the data in bytes that is contained in this message (it can also be equal to 0).

A CRC is added to verify whether the message has been compromised or not.

A `Stop Frame` could also be added, though this is not absolutely necessary, as the CRC could also act as a stop frame. The CRC could also be included in the `Stop Frame`.

This protocol ensures a safer way to exchange messages between the FPGA and the computer. It also has the advantage that the message's receiver doesn't have to decode the message's addressee to ascertain the data length. Those two tasks could therefore be separated to reduce the complexity of the code.

7 FPGA DEVELOPPEMENT

This chapter describes the different VHDL blocks that are implemented to drive the MT and to read the values from the SS from the FPGA board. The diagram in Figure 30 shows the organization of the different blocks.

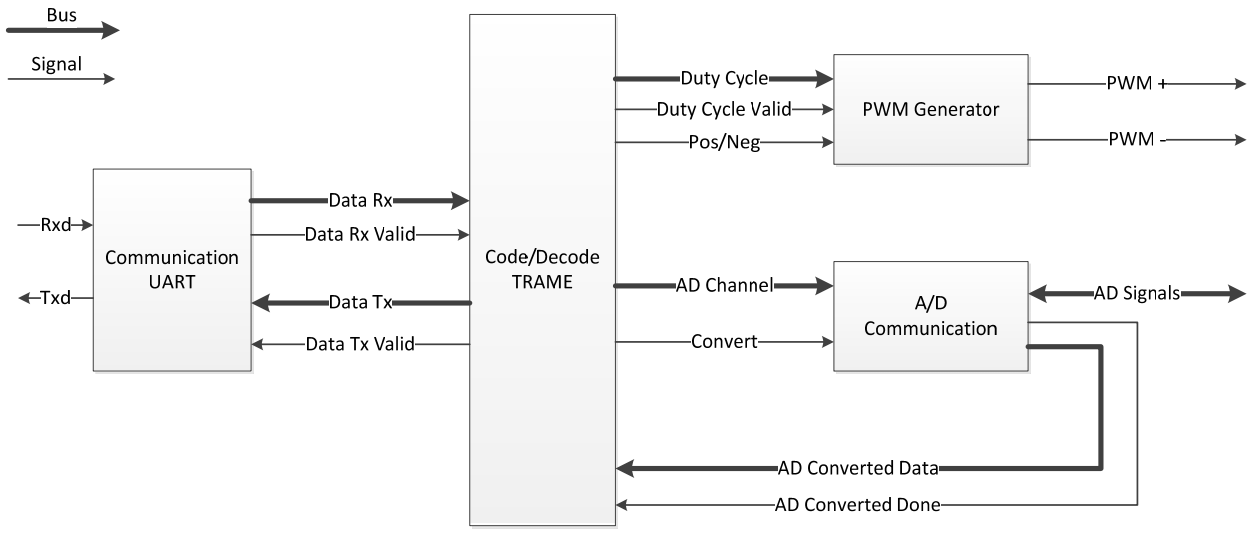


Figure 30 : Diagram of the VHDL interface code

A first block performs the parallelization of the serial data transmitted by the application on the PC.

Once these data are parallelized, a second block decodes the UART frames and extracts the useful information. This information is comprised of:

- The destination of the frame. The frame can be addressed to the PWM block or to the A/D communication block.
- The duty cycle. If the frame is addressed to the PWM block, it contains the duty cycle of the PWM that must be generated by this block. It also determines on which output the PWM is generated.
- The Channel of the A/D conversion. If the frame is addressed to the A/D communication block, it contains the number of the channel on which the A/D conversion must be done.

The PWM Generator block generates a PWM at a fixed frequency and at a variable duty cycle. The duty cycle can be modified at any time, however the PWM's frequency must be defined prior to the synthesizing of the VHDL code.

The A/D communication block commands the AD converter. It defines on which channel the conversion will be done. It also provides the necessary signals used to configure and perform the conversion.

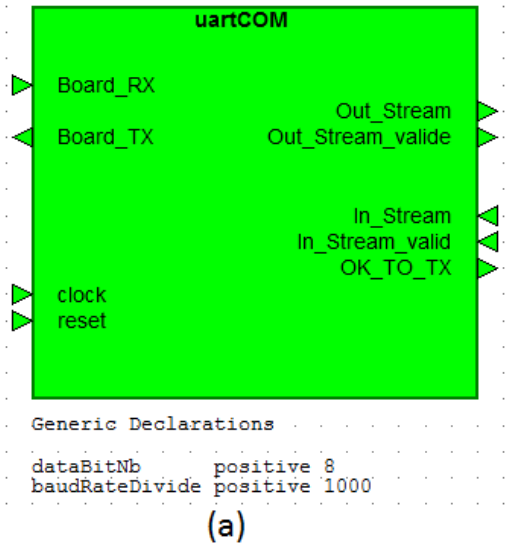
7.1 UART Communication Block

This block allows the parallelization of the incoming UART frame and the serialization of the outgoing frame.

7.1.1 Block description

This block allows the parallelization of the incoming UART frames. It also serializes the outgoing frames and clocks the UART communication.

Figure 31 (a) shows the graphical representation of the block with its generic parameters and their default values. Figure 31 (b) shows its signal table.



	A	B	C	D	E
M	Group	Name	Mode	Type	Bounds
1		clock	IN	std_ulogic	
2		reset	IN	std_ulogic	
3		Board_RX	IN	std_ulogic	
4		Board_TX	OUT	std_ulogic	
5		Out_Stream	OUT	std_ulogic_vector	(dataBitNb-1:0)
6		In_Stream	IN	std_ulogic_vector	(dataBitNB-1:0)
7		Out_Stream_valide	OUT	std_ulogic	
8		In_Stream_valid	IN	std_ulogic	
9		OK_TO_TX	OUT	std_ulogic	
10					

(b)

Figure 31 : uartCom bloc. (a) Symbol. (b) Signals table

<u>Baud Rate :</u>

The generic parameter `baudRateDivide` defines the baud rate of the UART communication. This parameter is redefined to take into account the FPGAs clock frequency, using the following formula:

$$baudRateDivide = integer\ (\frac{clockFrequency}{baudeRate})$$

The actual configuration is used for the redefinition of the generic parameter `baudeRateDivide`:

clockFrequcy : real := 100.0E6
baudeRate : real := 9600.0

<u>Incoming frames:</u>

The serial frames enter on the input `Board_RX`. The block automatically defines the start of the frame.

Once the frame has been received, it is parallelized. When this operation is done, the frame is put on the output `Out_Stream`. To indicate when the frame is valid, the output `Out_Stream_valid` is passed to high level.

Outgoing frames:

The output `OK_TO_TX` allows the block to indicate when it is ready to receive new frames that need to be sent on the UART bus. This parallel frame has to be put on the input `In_Stream`. The input `In_Stream_valid` has to be set at high level to indicate that the frame is valid. Once this step is done, the frame is serialized and sent on the UART bus.

7.1.2 Simulations

To confirm that the block is working correctly, a simulation was performed. It allows one to show the behavior of the block when an UART frame is received and sent.

Test setup:

The following figure shows the schematic and the values of the parameters that were used to perform this simulation.

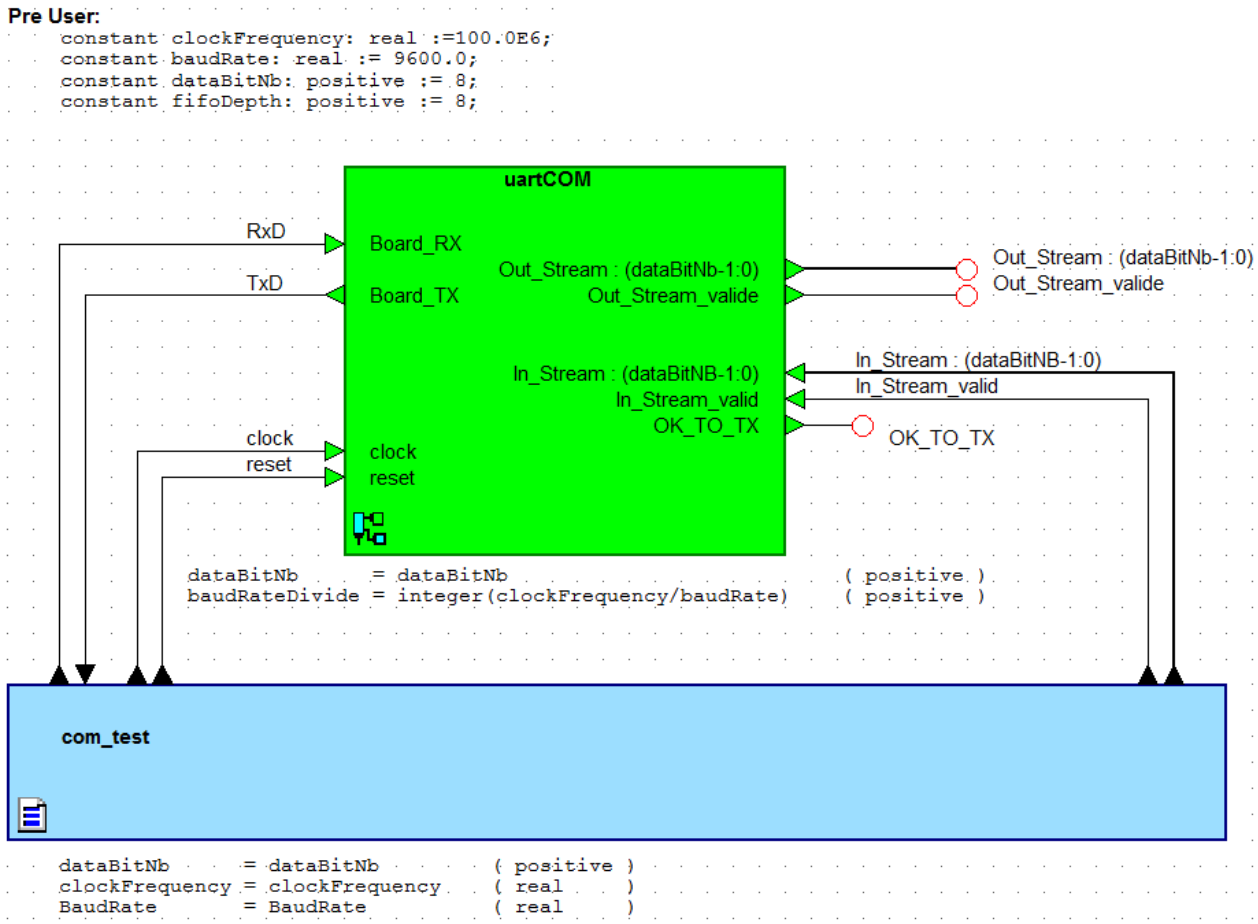


Figure 32 : Schematic of the UART communications simulation

Results of the simulation:

The Figure 33 shows the results of the simulation of the reception of an UART frame.

The following data is sent on the input `RxD` (MSB to LSB): 0111011. The frame is wrapped with a start bit of value 0 and a stop bit of value 1. The data are sent in little endian format (LSB first).

When the 8th bit is received, the output `Out_Stream_valid` is passed to high level for one clock period. At this moment, the output `Out_Stream` takes the value of the serial frame. The UART frame is correctly received in LSB first format, the start and stop bits are correctly at 0 and 1.

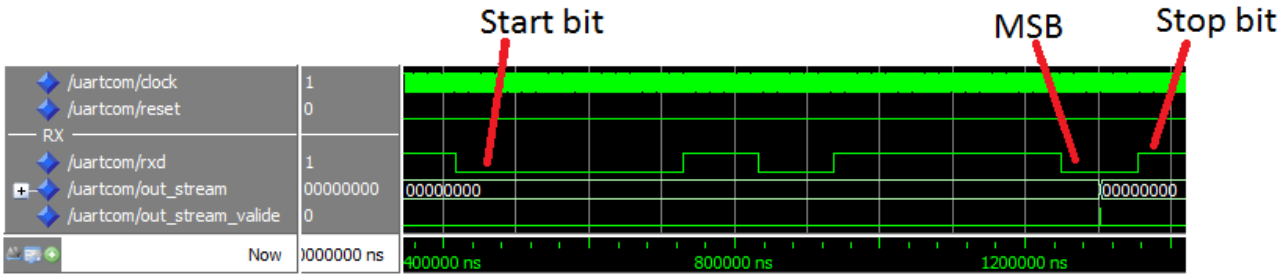


Figure 33 : Result of the simulation of the reception of an UART frame

Figure 34 shows a zoom of the figure above. It shows with greater precision what is going on when `Out_Stream_valide` is set to high level.

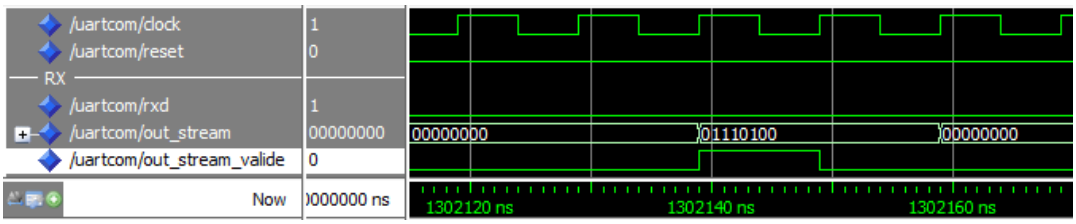


Figure 34 : Zoom of Figure 33

`Out_Stream_valid` is correctly passed at high level for one clock period. We also note that `Out_Stream` takes the value of the received serial frame on `RxD`.

Figure 35 shows the result of the simulation of the transmission of a serial frame on the UART bus.

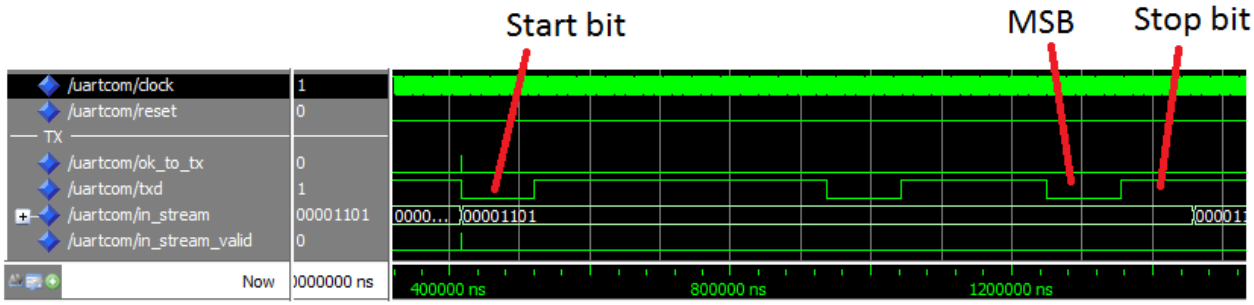


Figure 35 : Result of the simulation of the transmission of an UART frame.

The parallel data is sent on the input `In_Stream:` 01101111. We can see that the UART frame sent on `TxD` has the desired value. We can also note that `In_Stream_valid` is set to high level, prior to the transmission of the frame for on clock period. This step allows us to validate the value of the frame that has to be sent.

The Figure 36 shows a zoom of the figure above. It shows with greater precision what is going on when `In_Stream_valide` is set to high level.

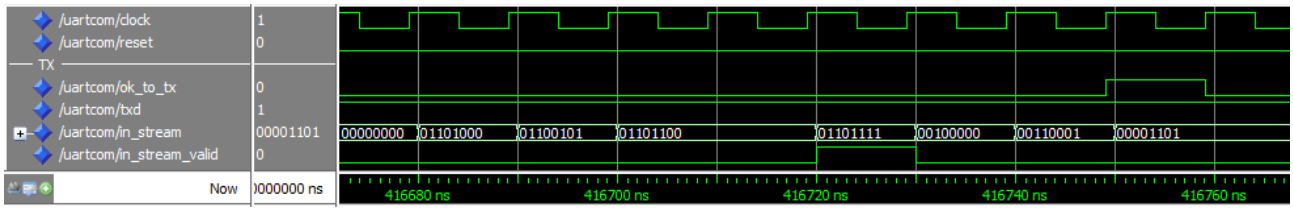


Figure 36 : Zoom of the Figure 35

Several different values are sent on the input `In_Stream`. The serial frame on `TxD` matches the data on `In_Stream` when `In_Stream_valid` is set at high level. In this case, it correctly corresponds to the desired value of 01101111.

Once the fame is validated by `In_Stream_valid`, the output `OK_TO_TX` is set at high level to indicate that the block is ready to receive new data.

The result of these two simulations shows that the different signals behave as expected. It therefore allows to synthesize this VHDL block and perform further tests on the FPGA board.

7.1.3 Test at board level

An oscilloscope measurement was made directly on the wire to visualize the passing UART frame.

<u>Results of the test:</u>

The following figure shows an oscilloscope screen capture of several UART frames sent by the computer and several frames sent by the FPGA.

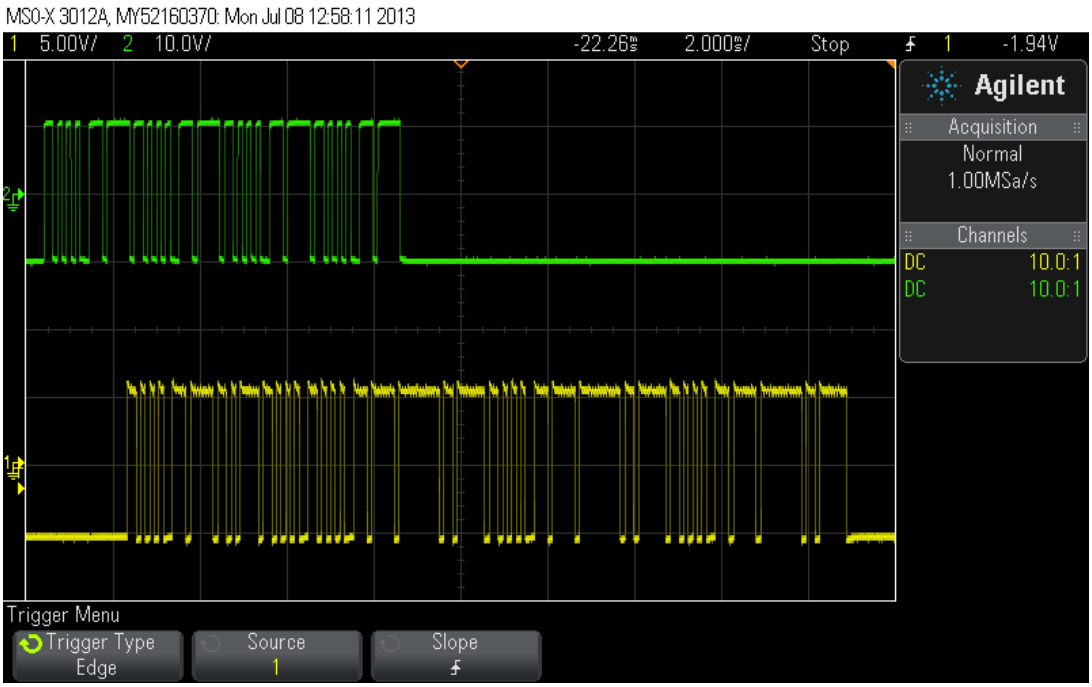


Figure 37: Oscilloscope frame capture of the UART communication

As can be seen in the above figure, frames are emitted by the computer (green trace) and received on it (yellow trace). As the decoding of these frames would have been fairly difficult in the context of this project, it was not performed. This test only validates that UART frames are emitted from the PC and from the FPGA. The correctness of these UART frames was only observed in chapter 8.1.1 when a string is sent to the FPGA board and received unchanged on the computer.

7.2 Block code_decode_Trame

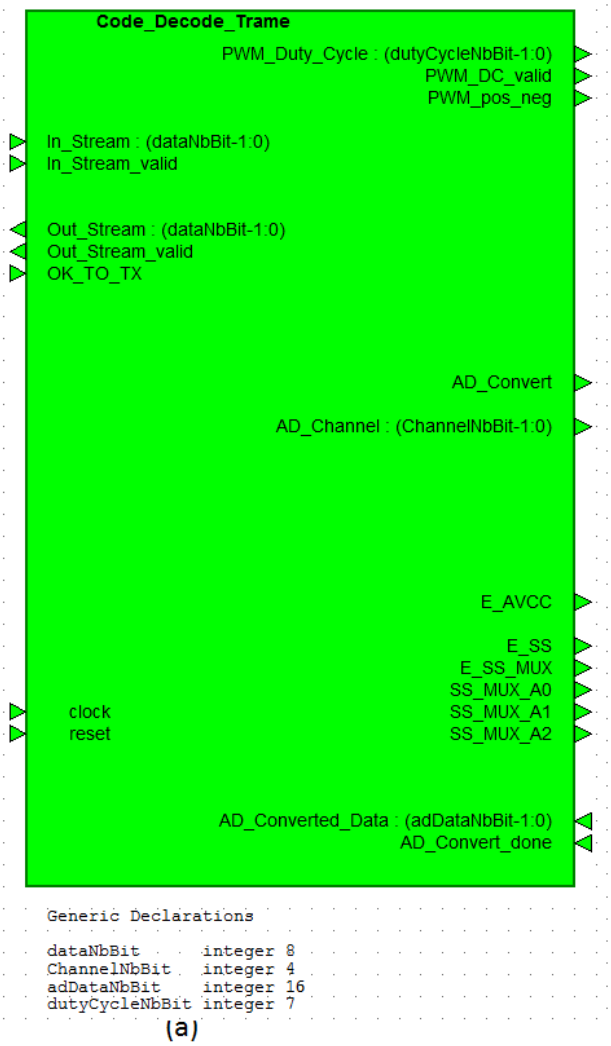
7.2.1 Block description

Block_code_decode_Trame is the decision-making block of the system. It allows the extraction of the data contained in a UART frame. In function of the data received, it commands either the PWM block or the AD communication block. It also allows coding the AD conversions result into several UART frames.

In order to communicate with the application on the PC, a simple communication protocol was developed. This protocol is described in chapter 6.1.

This block detects the start frame of the protocol and determines the number of following UART frames attached to this message. It then treats the received data to determine if they are addressed to the PWM generator block `PWM_GEN` or to the AD communication block `adCOM.`

Figure 38 (a) shows the graphical representation of the block with its generic parameters and their default values. Figure 38 (b) shows its signal table.



	Group	Name	Mode	Type	Bounds
1		In_Stream	IN	std_ulogic_vector	(dataNbBit-1:0)
2		Out_Stream	OUT	std_ulogic_vector	(dataNbBit-1:0)
3		PWM_Duty_Cycle	OUT	std_ulogic_vector	(dutyCycleNbBit-1:0)
4		E_AVCC	OUT	std_ulogic	
5		AD_Converted_Data	IN	std_ulogic_vector	(adDataNbBit-1:0)
6		clock	IN	std_ulogic	
7		reset	IN	std_ulogic	
8		AD_Channel	OUT	std_ulogic_vector	(ChannelNbBit-1:0)
9		AD_Convert	OUT	std_ulogic	
10		PWM_DC_valid	OUT	std_ulogic	
11		In_Stream_valid	IN	std_ulogic	
12		Out_Stream_valid	OUT	std_ulogic	
13		AD_Convert_done	IN	std_ulogic	
14		E_SS	OUT	std_ulogic	
15		E_SS_MUX	OUT	std_ulogic	
16		SS_MUX_A0	OUT	std_ulogic	
17		SS_MUX_A1	OUT	std_ulogic	
18		SS_MUX_A2	OUT	std_ulogic	
19		PWM_pos_neg	OUT	std_ulogic	
20		OK_TO_TX	IN	std_ulogic	
21					

(b)

Figure 38 : Code_Decode_Trame block. (a) Symbol. (b) Signals table

<u>Inputs / outputs description:</u>

The inputs/outputs `In_Stream`, `In_Stream_valid`, `Out_Stream`, `Out_Stream_valid` and `OK_TO_TX` allow us to communicate with the block `uartCOM` to transmit and receive data in the form of UART frames.

The output `PWM_Duty_Cycle` allows sending the duty cycle value that the block `PWM_GEN` will have to generate. For example if the value 50 is put on this output, the PWM generated by the block `PWM_GEN` will have a duty cycle of 50% (It is not possible to modify the frequency of the generated PWM). The output `PWM_DC_valid` defines when the duty cycle on `PWM_Duty_Cycle` is valid. `PWM_pos_neg` defines the direction of the current in the MT by controlling the H-bridge.

`AD_Channel` defines the channel on which the AD conversion will be performed. The signal `AD_Convert_valide` validates the channel and gives the order to the block `adCOM` to start the AD conversion. The input `AD_Converted_Data` contains the result of the AD conversion and `AD_Convert_done` indicates when this value is valid.

E_AVCC enables the power supply for the analogical component of the interface board. E_SS activates the power supply for the SS and its related electronics. SS_MUX_A0…2 are the control signals of the multiplexer. These signals select which SS will be converted by the AD converter. For this bachelor's thesis, only one SS is used, therefore these outputs are always set at low level. E_SS_MUX enables the multiplexer.

The following state machine defines the behavior of this block:

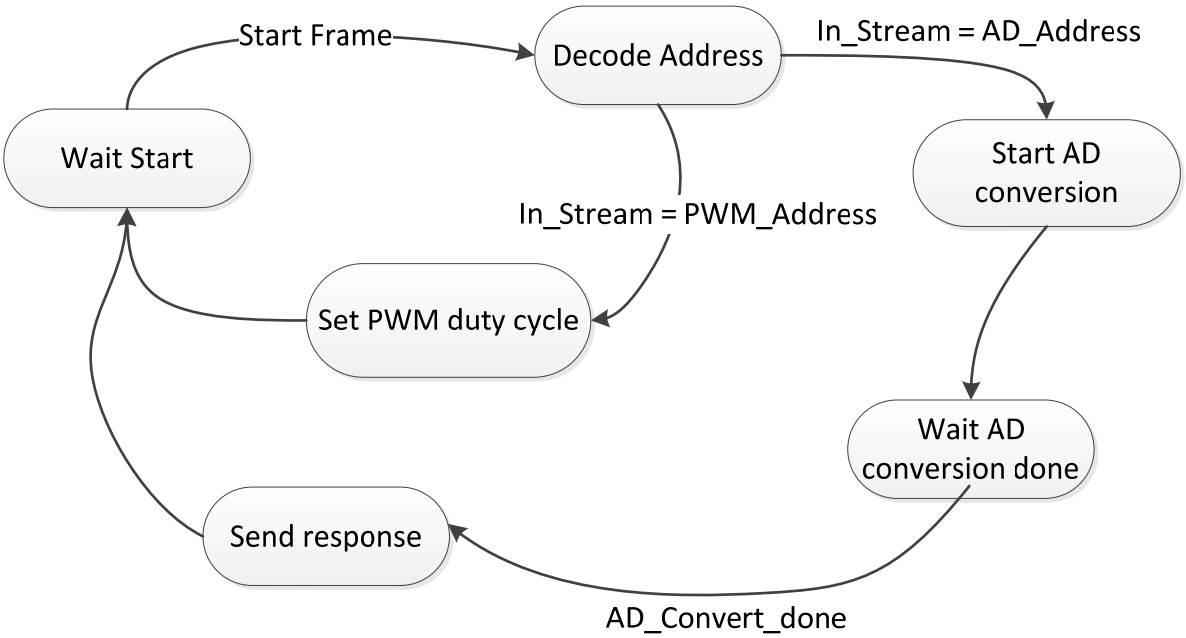


Figure 39: state machine of Code_Decode_Trame block

7.2.2 Simulation

To validate the proper operation of this block, a simulation was developed. It validates the proper behavior of the state machine (Figure 39). It also shows the behavior of the block when it receives frames form the block `uartCOM`. Finally, it shows that the blocks `PWM_GEN` and `adCOM` are properly controlled.

Test setup:

The figure below shows the schematic and the values of the parameters which were used to performed the simulation of the block `Code_Decode_Trame`.

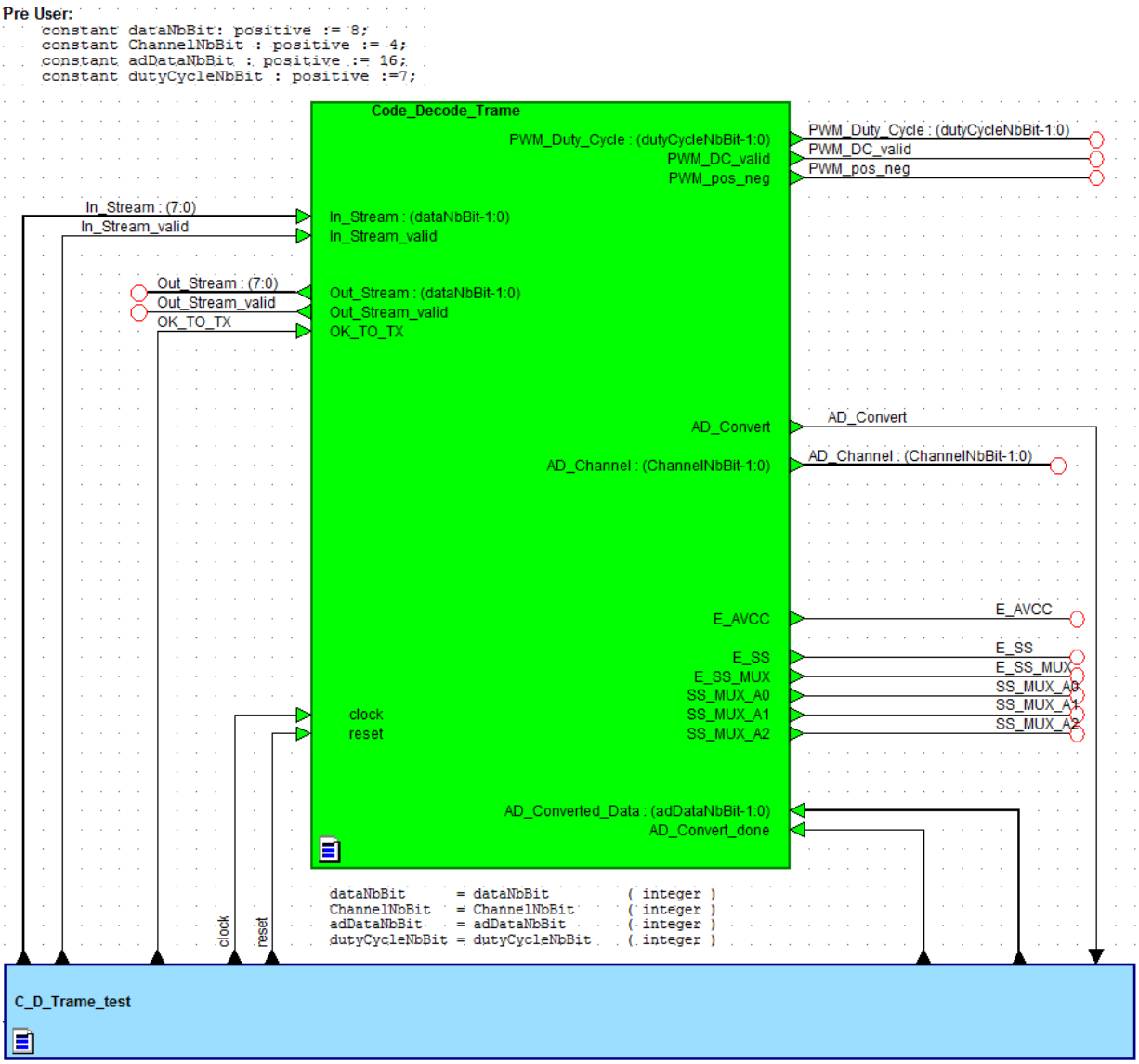


Figure 40 : Schematic of the block Code_Decode_Trame

Results of the simulation:

The following figure simulates the control of the block `PWM_GEN` by the block `Code_Decode_Trame`.

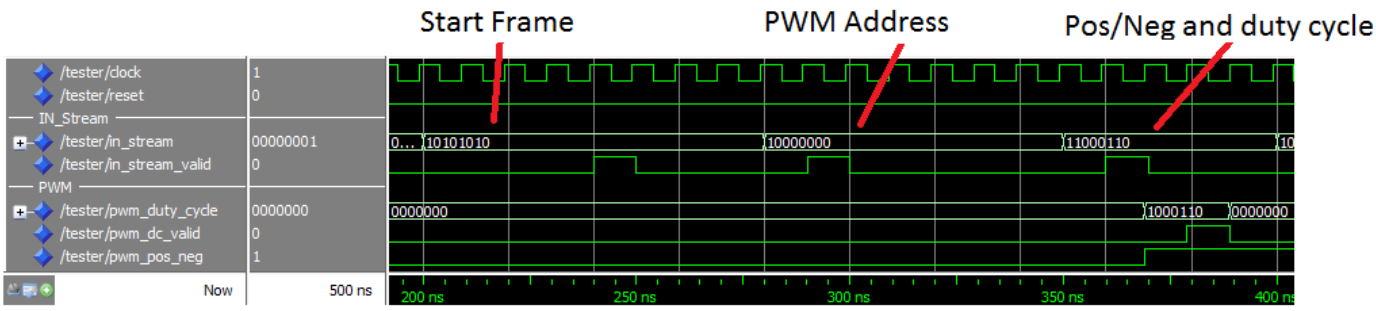


Figure 41 : Result of the simulation. Section control of the block PWM_GEN: positive current and 70% of duty cycle

Figure 41 shows the decoding of the communication protocol (described in chapter 6.1) by the block `Code_Decode_Trame`. The protocol frames are correctly received on the input `in_stream` and correctly validated by `in_stream_valid`. This test was set so that this block would have to command the `PWM_GEN` block so that it generates a PWM with a duty cycle of 70% and so that the current engendered by the PWM is positive.

Following the state machine of the block `Code_Decode_Trame` at Figure 39, we can see that when the start frame is received, the state machine jumps to the state `Decode Address`. As the received address corresponds to a `PWM_GEN` address (10000000) the state machine jumps to the state `Set PWM duty cycle`. In this last state the data frame is decoded. The MSB is used to define if the current must be positive or negative. In this case it is at 1, so the current must be positive. The next seven bits are used to define the duty cycle of the PWM. In this case they are at 70 (1000110). We can see that when `pwm_dc_valid` is set at high level, the output `pwm_duty_cycle` corresponds to 70 (1000110) and `pwm_pos_neg` is set at 1.

The following figure simulates the control of the block `PWM_GEN` by the block `Code_Decode_Trame`.

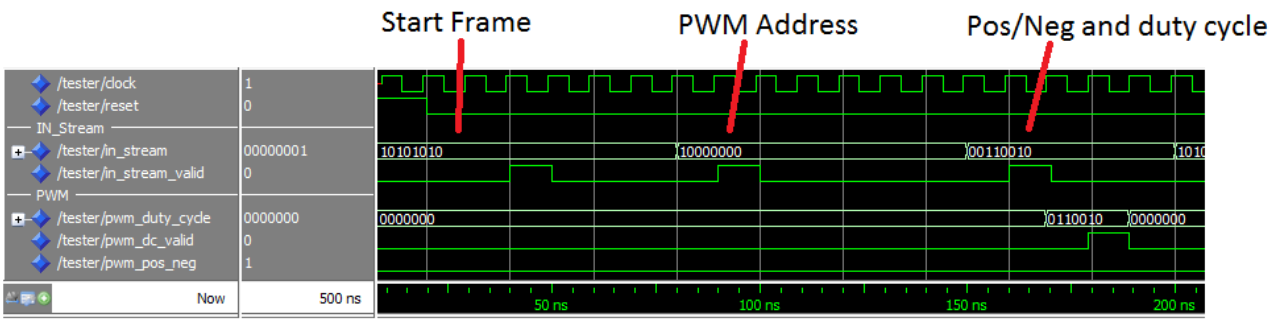


Figure 42 : Result of the simulation. Section control of the block PWM_GEN: negative current and 50% of duty cycle

Figure 42 shows the decoding of the communication protocol (described in chapter 6.1) by the block `Code_Decode_Trame`. This test is the same as the previous one except that the parameters of the PWM which have to be generated by the block `PWM_GEN` have been modified to: Duty Cycle 50% and negative current.

As the decoding approach is the same as the previous test, it will not be re-explained here.

We can see that the output `pwm_duty_cycle` has the expected value of 50(00110010) and that `pwm_pos_neg` is correctly set to 0 when the signal `pwm_dc_valid` is set to high level.

These two simulations confirm that the block `Code_Decode_Trame` correctly commands the block `PWM_GEN`.

The following figure simulates the control of the block `adCOM` by the block `Code_Decode_Trame`.

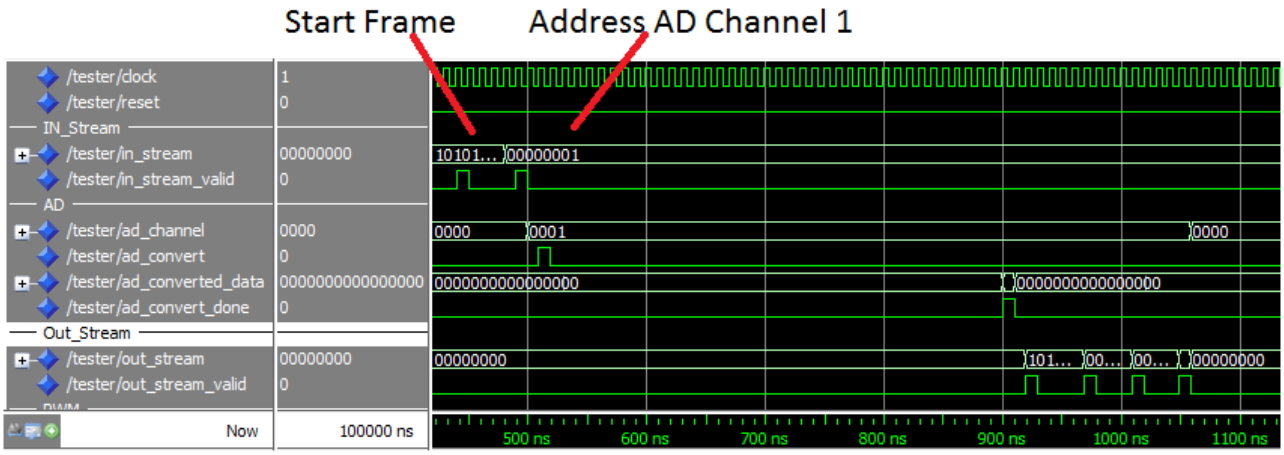


Figure 43 : Result of the simulation. Section control of the block adCOM. AD conversion on channel 1

The Figure 43 shows the decoding of the communication protocol (described in chapter 6.1) by the block `Code_Decode_Trame`. The protocol's frames are correctly received on the input `in_stream` and correctly validated by `in_stream_valid`. This test was set so that this block would have to command the `adCOM` block so that it starts an AD conversion on the channel 1.

Following the state machine of the block `Code_Decode_Trame` in Figure 39, we can see that when the start frame is received, the state machine jumps to the state `Decode Address`. As the received address corresponds to an `adCOM` address (00000001) the states machine saves the 4 LSBs of the frame and jumps to the state `Start AD conversion`.

In this state, the 4 bits previously saved are used to define on which port the AD conversion must be done. In this case the channel 1 is selected (0001). This value is then put on the output `ad_channel` and the command to start the AD conversion is sent to the block `adCOM` by setting the `ad_convert` output to high level. The data on `ad_channel` are only valid during the time `ad_convert` is at high level.

Then the state machine goes to the state `Wait AD conversion done`. In this state, the state machine waits until the AD conversion has been completed. When the signal `AD_Convert_done` indicating that

the AD conversion is done passes to high level, the data on `ad_converted_data` are saved. In this case they are 111110101010.

The state machine then jumps to the last state `Send response`. In this state the answer message is created. Figure 44 is a zoom of Figure 43 of the part where the answer message is created. The answer message follows the same communication protocol as explained in chapter 6.1. The start frame is sent first (10101010) on `out_stream` and validated by `out_stream_valid` then the AD channel on which the AD conversion was performed is sent (00000001). The third frame contains 4 padding bits followed by the 4 MSBs of the AD conversion (00001111). The last frame contains the 8 LSBs of the AD conversion (10101010).

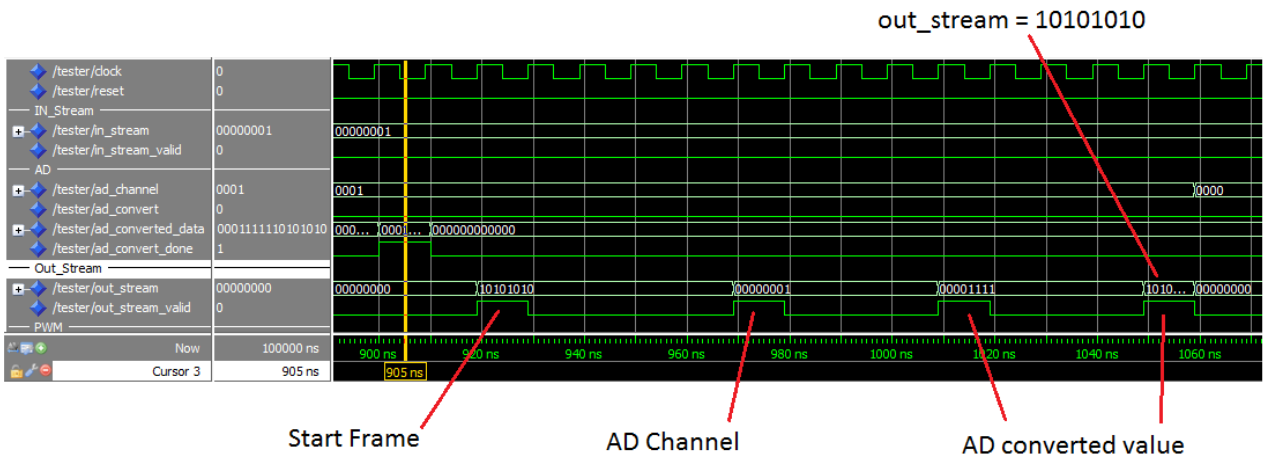


Figure 44 : Zoom of the Figure 43

This simulation confirms that the block `Code_Decode_Trame` correctly commands the block `adCOM`.

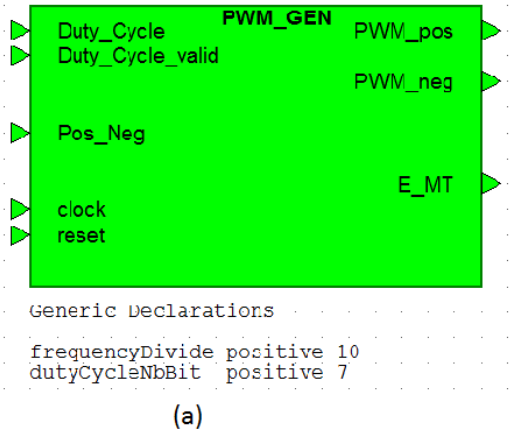
No board level tests were performed on this block. The correct results were observed through the tests at board level made on the other blocks, as this block commands the others, we can deduce that it is working correctly.

7.3 PWM generator block

7.3.1 Block description

This block is used to generate a PWM with a controllable duty cycle. The PWM can be emitted on one of the two different outputs. This mechanism is used to control the current direction in the MT by using an H bridge.

Figure 45 (a) shows the graphical representation of the block with its generic parameters and their default values. Figure 45 (b) shows its signal table.



(a)

	Group	Name	Mode	Type	Bounds
1		clock	IN	std_ulogic	
2		reset	IN	std_ulogic	
3		Duty_Cycle	IN	std_ulogic_vector	(dutyCycleNbBit-1:0)
4		PWM_pos	OUT	std_ulogic	
5		E_MT	OUT	std_ulogic	
6		Duty_Cycle_valid	IN	std_ulogic	
7		PWM_neg	OUT	std_ulogic	
8		Pos_Neg	IN	std_ulogic	
9					

(b)

Figure 45 : bloc PWM_GEN. (a) Symbol. (b) Signal table

<u>Clock frequency and PWM frequency:</u>

The generic parameter `frequencyDivide` allows the generation of a PWM with a defined frequency regardless of the clock frequency of the system, using the following formula:

$$frequencyDivide = integer\left(\frac{clockFrequency}{PWMFrequency}\right)$$

The actual configuration is used for the redefinition of the generic parameter `baudeRateDivide`:

clockFrequcy : real := 100.0E6
PWMFrequency : real := 200.0E3

The frequency of the generated PWM is set prior to synthesizing. Therefore it cannot be modified after this operation.

<u>Inputs outputs description:</u>

The desired duty cycle is applied on the input `Duty_Cycle`. It can be set with a precision of 1%. The input `Duty_Cycle_valid` defines when the duty cycle applied on `Duty_Cycle` is valid.

The input `Pos_Neg` defines on which output (`PWM_pos` or `PWM_neg`) the PWM will be generated.

`PWM_pos` and `PWM_neg` are the two outputs on which the PWM will be generated. The PWM can only be generated on one of these two outputs at one time.

Output `E_MT` enables the power supply for the MT and for it related electronics. This output is set at high level when a PWM with a duty cycle different than 0% is generated.

7.3.2 Simulation

To validate the proper operation of this block, a simulation was developed. It validates the behavior of this block when different duty cycles are applied on the input `Duty_Cycle`. It shows that the PWM is

generated with the proper frequency, duty cycle and on the proper output. Finally it shows that the output `E_MT` is correctly controlled.

Test setup:

The figure below shows the schematic and the values of the parameters which were used to performed the simulation of the block `PWM_GEN`.

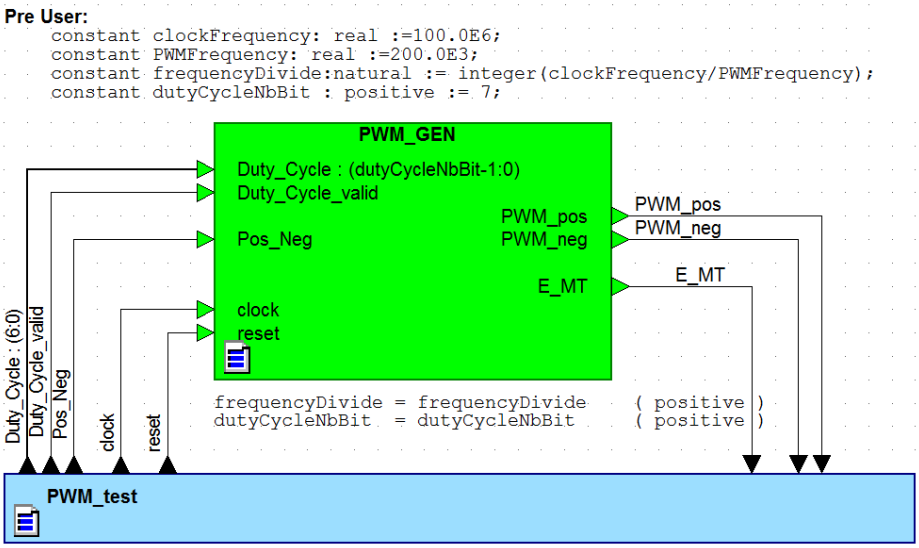


Figure 46 : Schematic of the block PWM_GEN

The parameter `clockFrequency` is defined to 100 M to match the FPGAs board clock frequency of 100 [MHz].

`PWMFrequency` is set to 200 [kHz].

The calculation of the parameter `frequencyDivide` is made during the synthesizing. That is why the parameters `clockFrequency` and `PWMFrequency` can be set to such high values.

Simulation results:

Figure 47 shows the simulation result of the generation of a PWM one the output `PWM_pos`. The duty cycle was defined to 20% and the frequency was set to 200 [kHz].

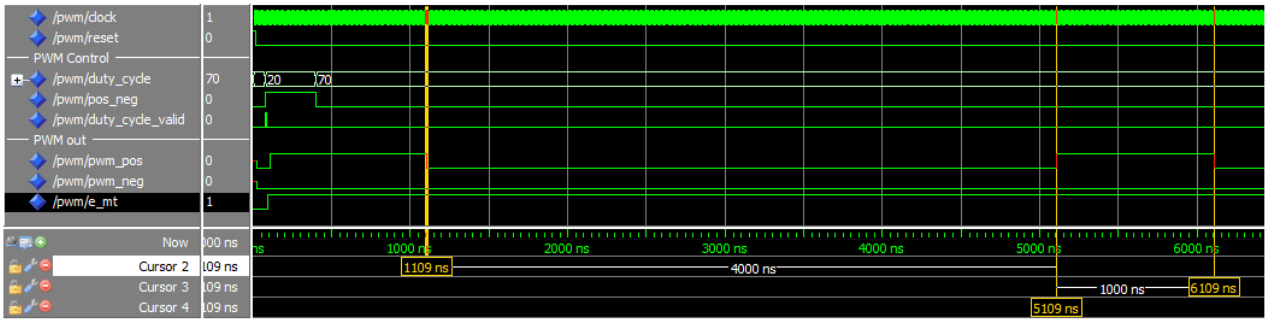


Figure 47 : PWM_test. Duty Cycle = 20%. Freq PWM = 200 kHz. Out pos

The time between the Cursor 2 and the Cursor 4 of the figure above is 5000 [ns] = 200 [kHz]. This result confirms that the PWM can be set to the desired frequency. The time between the Cursor 3 and the Cursor 4 is 1000 [ns]. This value corresponds to the desired duty cycle of 20%.

The input value of the duty cycle is first set to 20, then to 70. The duty cycle is correctly chosen when the input `Duty_Cycle_valid` is at high level, setting the output PWM duty cycle to 20% as expected. When `Duty_Cycle_valid` is at high level, the output port for the PWM is also correctly determined with the input `Pos_Neg`. The effect of this validation can be seen in the figure above by noting that the PWM is generated on the correct output `PWM_pos`.

The output `E_MT` is correctly set at high level, when the duty cycle is different than 0%.

Figure 48 shows the result of a second simulation. The PWM is generated on the output `PWM_neg`. The duty cycle was defined to 55% and the frequency was set to 200 [kHz].

This second test was performed to validate that a PWM of a precision of 1% could be generated and to validate that it could be generated on the second output.

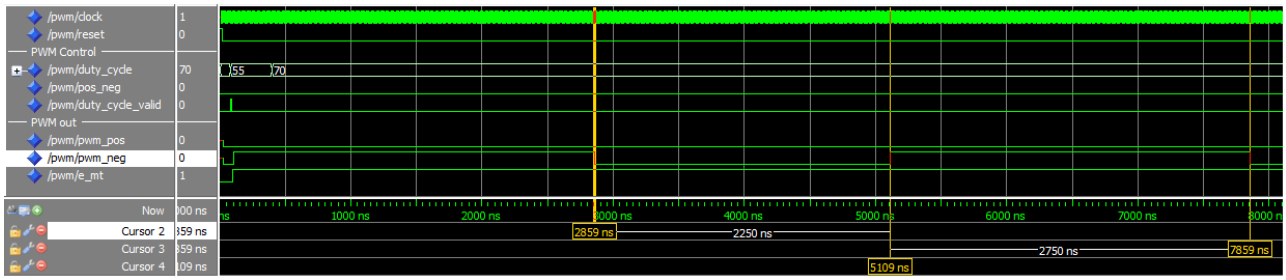


Figure 48 : PWM_test. Duty Cycle = 55%. Freq PWM = 200 kHz. Out neg

This simulation shows that the behavior of the block is correct. The PWM's frequency is correctly set to 200 [kHz] (5000 ns between cursor 2 and 4). The duty cycle is 55% (2750 ns between cursor 3 and 4) and the output is correctly generated on the output `PWM_neg`.

Figure 49 shows the last test needed to be performed in order to validate this block. It was performed to see if the output `E_MT` is set to low level when a duty cycle of 0% is validated by `Duty_Cycle_valid`.

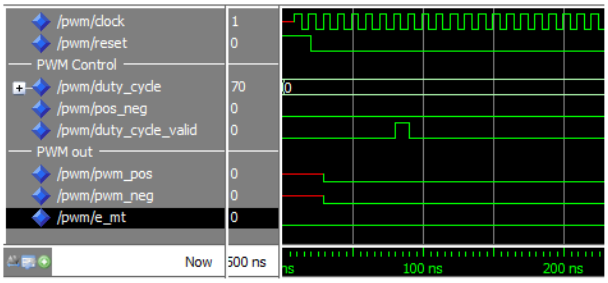


Figure 49 : PWM_test. Duty Cycle = 0%.

The figure above shows that the output `E_MT` behaves as expected. It is not set at high level when the duty cycle is validated by `Duty_Cycle_valid`.

7.3.3 Test at board level

A test was performed at board level to determine if the PWM is generated with the designed duty cycle and the correct frequency, on the desired input of the H-bridge.

Result of the test:

The following figure shows an oscilloscope screen capture of a PWM generated on the input PWM+ of the H-Bridge (see Figure 16).

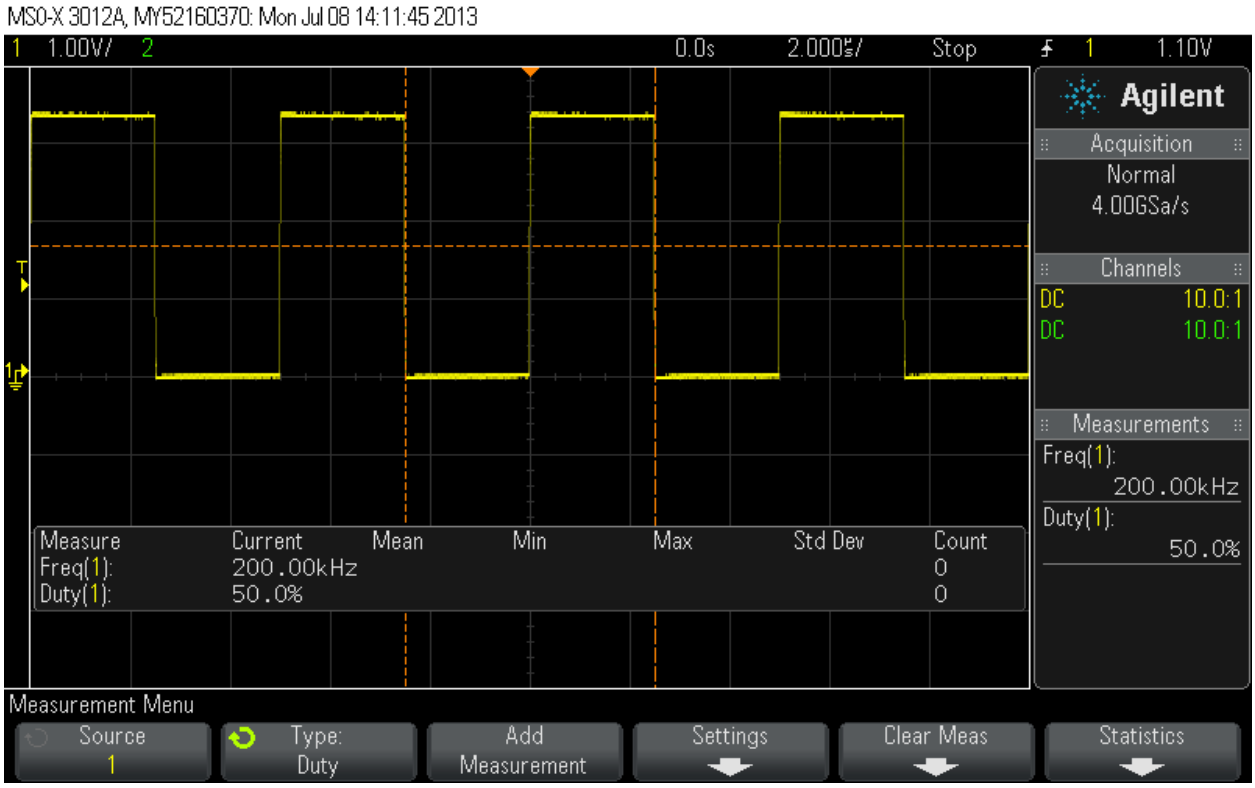


Figure 50: Oscilloscope screen capture of the PWM generated on the PWM+ input of the H-Bridge

For this first test, a PWM with the following characteristics was generated by the block `PWMgen` of the FPGA:

PWM frequency = 200 [kHz]
Duty Cycle = 50% => positive current flowing in the MT

Figure 50 shows that the generated PWM's frequency and duty cycle are correct. As the measurement was performed on the input `PWM+` of the H-Bridge, it indicates that the current that flows in the MT is positive.

The following figure shows an oscilloscope screen capture of a PWM generated on the input `PWM-` of the H-Bridge (see Figure 16).

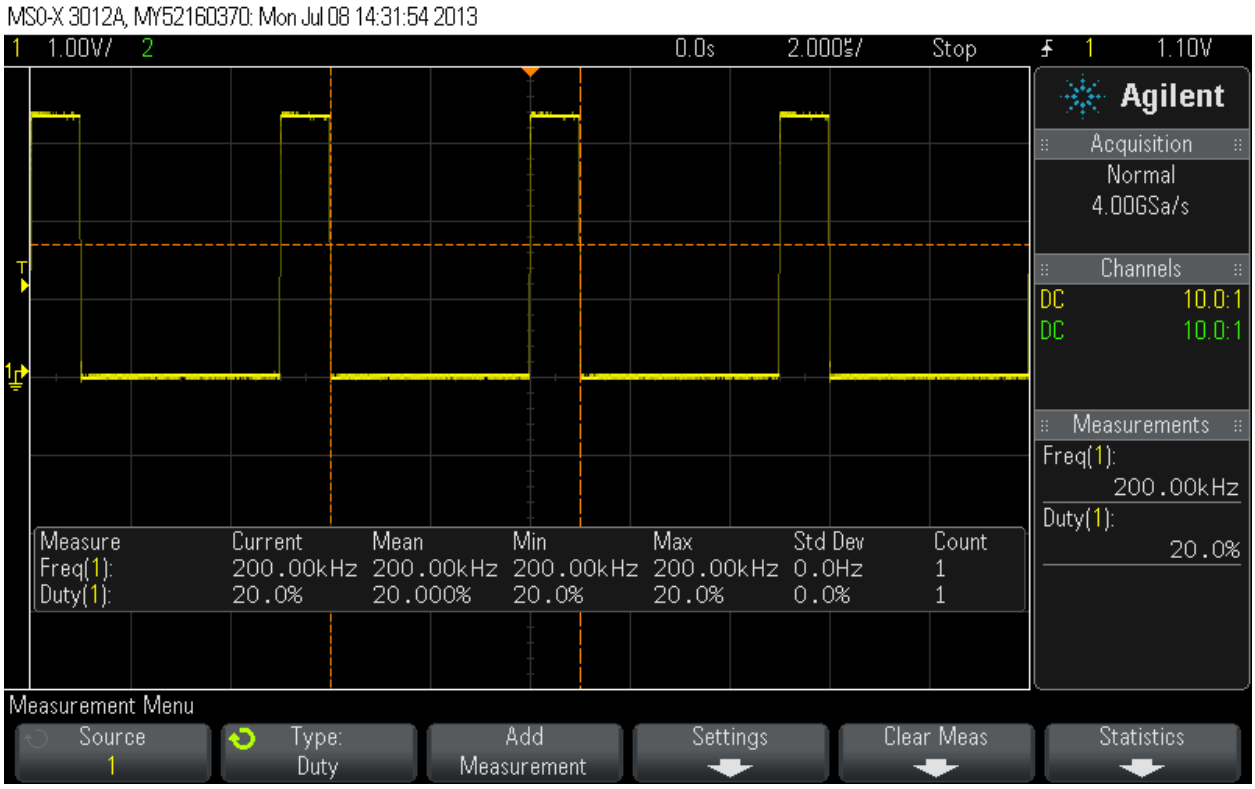


Figure 51: Oscilloscope screen capture of the PWM generated on the PWM- input of the H-Bridge

For this first test, a PWM with the following characteristics was generated by the block `PWMgen` of the FPGA:

PWM frequency = 200 [kHz]
Duty Cycle = -20% => negative current flowing in the MT

Figure 50 shows that the generated PWM's frequency and duty cycle are correct. As the measurement was performed on the input `PWM-` of the H-Bridge, it indicates that the current that flows in the MT is negative.

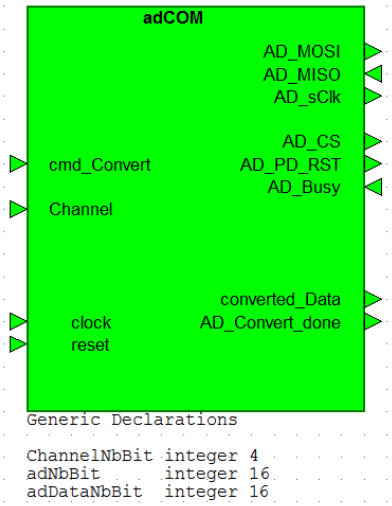
These two tests validate the correct operation of the block `PWMgen`.

7.4 AD communication block

7.4.1 Block description

This block is used to control and configure the AD converter. It establishes an SPI communication with the converter and sets its configuration register.

The Figure 52 (a) shows the graphical representation of the block with its generic parameters and their default values. Figure 52 (b) shows its signal table.



(a)

	Group	Name	Mode	Type	Bounds
1		clock	IN	std_ulogic	
2		reset	IN	std_ulogic	
3		AD_MOSI	OUT	std_ulogic	
4		AD_CS	OUT	std_ulogic	
5		AD_PD_RST	OUT	std_ulogic	
6		AD_MISO	IN	std_ulogic	
7		AD_Busy	IN	std_ulogic	
8		cmd_Convert	IN	std_ulogic	
9		Channel	IN	std_ulogic_vector	(ChannelNbBit-1:0)
10		AD_sClk	OUT	std_ulogic	
11		converted_Data	OUT	std_ulogic_vector	(adDataNbBit-1:0)
12		AD_Convert_done	OUT	std_ulogic	
13					

(b)

Figure 52 : block adCOM. (a) symbol. (b) signal list

The generic parameter `adDataNbBit` is the AD conversion number of bits and `adNbBit` is the number of bits needed to configure the AD converter's internal register.

<u>Inputs outputs description:</u>

The input `Channel` defines on which channel of the AD converter the operation will be performed on the next AD conversion.

`Cmd_Convert` validates the channel on which the AD conversion will be done and starts the AD conversion.

The input and output `AD_MOSI` and `AD_MISO` are the two data lines of the full-duplex SPI data bus. `AD_sClk` is the clock of the SPI communication that has to be generated by the master of the bus. `AD_CS` is the chip select signal to enable the AD converter. The output `AD_PD_RST` allows to reset the AD converter and `AD_Busy` indicates when it is busy and cannot be configured.

The output `AD_Convet_done` indicates when the AD conversion is done and `converted_Data` gives the result of this conversion.

Further information on how the AD conversion is performed can be found in chapter 4.7.

7.4.2 Simulation

To validate the proper operation of this block, a simulation was developed. It validates the behavior of this block when the AD converter has to be configured to perform an AD conversion on the port determined by the input address. It shows that the SPI frame to configure the AD register is correctly sent and that the different control signals are properly controlled.

Test setup:

The figure below shows the schematic and the values of the parameters which were used to performed the simulation of the block `adCOM`.

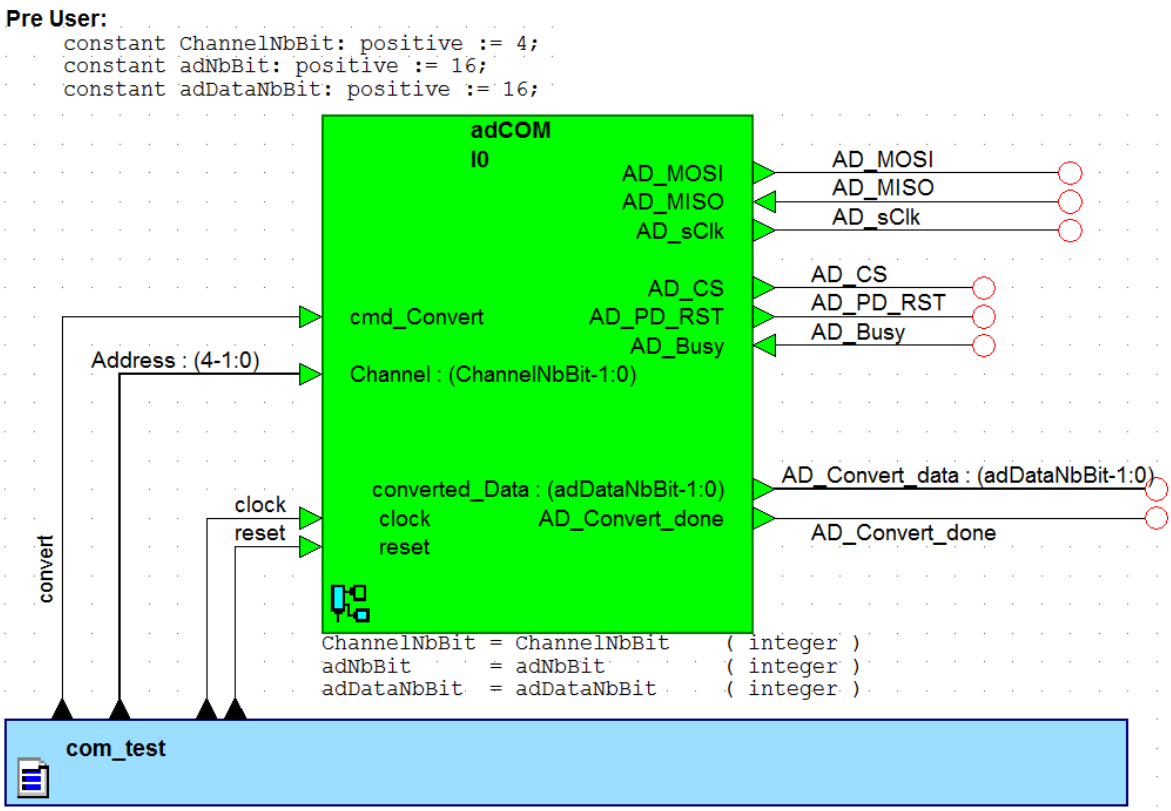


Figure 53: Schematic of the block adCOM.

Results of the simulation:

Figure 54 shows the simulation result of the configuration of the AD converter and the transmission of the AD conversion return value to the block `Code_Decode_Trame`.

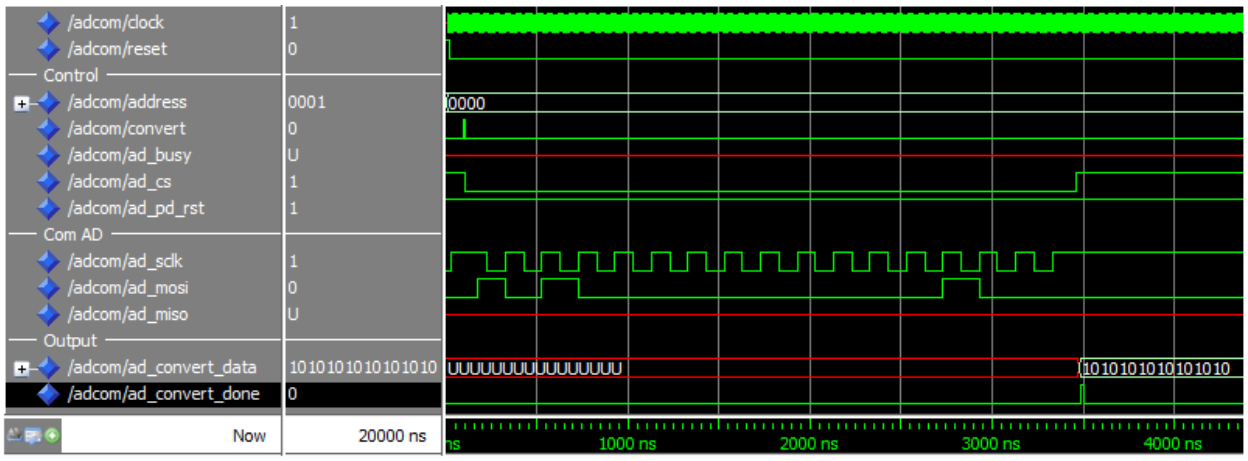


Figure 54: Result of the simulation of the block adCOM

We can see that the conversion channel is 0 (`address` = 0000). When `convert` is passed to high level, the conversion is started and `ad_cs` is passed to low level. The frame is sent on `ad_mosi` to configure the AD converter register as explained in the chapter 4.7. The 15th bit enables the conversion, the 13th sets the conversion on the channel 0 and the bit number two defines an external power source as the reference for the conversion. Once the transmission of this frame is done, the signal `ad_cs` is set to high level to indicate the end of the frame.

Then the data are received on the `ad_miso` during the next 15 `ad_sclk` periods. In this test no data are sent on the `ad_miso` input, but we can see that the `ad_cs` signal is correctly set at low level at the beginning of the 15 `ad_sclk` periods and put back a high level at the end.

As the AD conversion response contains the channel number on which it was performed, a comparison is made to determine if the received AD conversion was performed on the correct channel. In this case as no data were received on `ad_miso`, no valid channel is detected and an error frame of 1010101010101010 is sent on the output `ad_convert_data`.

The signal `ad_convert_done` is then set to high level to indicate that the value on `ad_convet_data` is valid.

This simulation does not fully validate this block as no data are received on `as_miso` and because the effects of the configuration of the AD converter are not visible. These validations are performed in the following chapter.

7.4.3 Test at board level

A test was performed at board level to determine if the AD's internal register is correctly configured, and whether the measurement is made on the correct AD channel and returned to the FPGA.

Results of the test:

The following figure shows an oscilloscope screen capture of an AD's configuration frame.

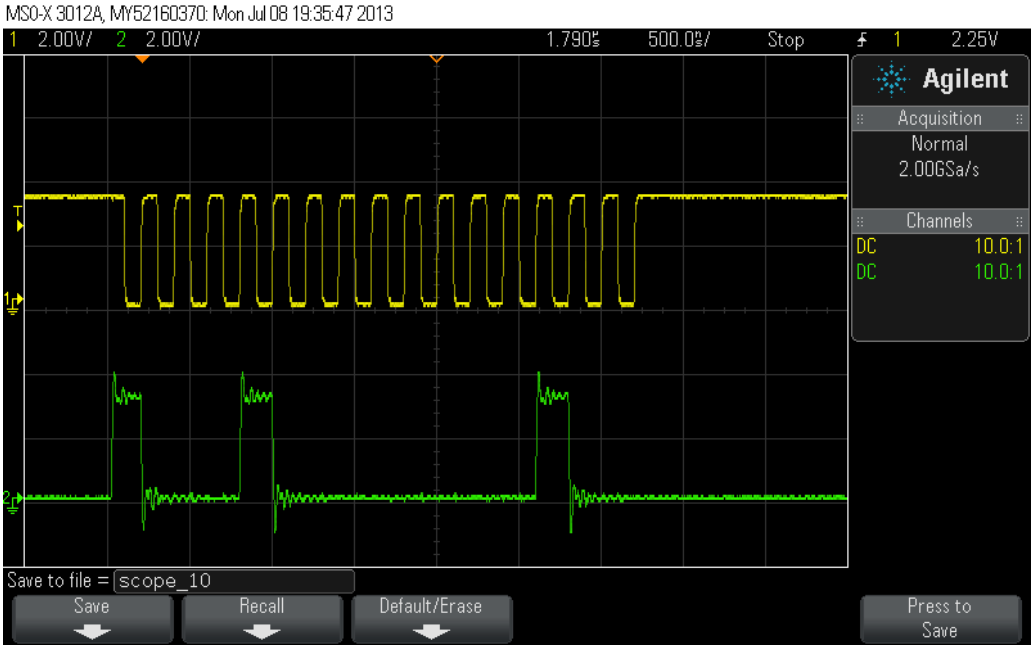


Figure 55: Oscilloscope frame capture of the AD's configuration frame

Yellow trace: SCLK measured on the pin of the AD converter
Green trace: configuration frame measured on the `MOSI` pin of the AD converter

As explained in chapter 4.7.2 the figure above shows the AD's configuration frame. This frame is formed so the next AD conversion will be performed on the channel 2 and that it uses an external power source as reference for this conversion. This figure also shows that the configuration frame is correctly clocked by `SCLK`.

The following figure shows an oscilloscope screen capture of the result of an AD conversion.

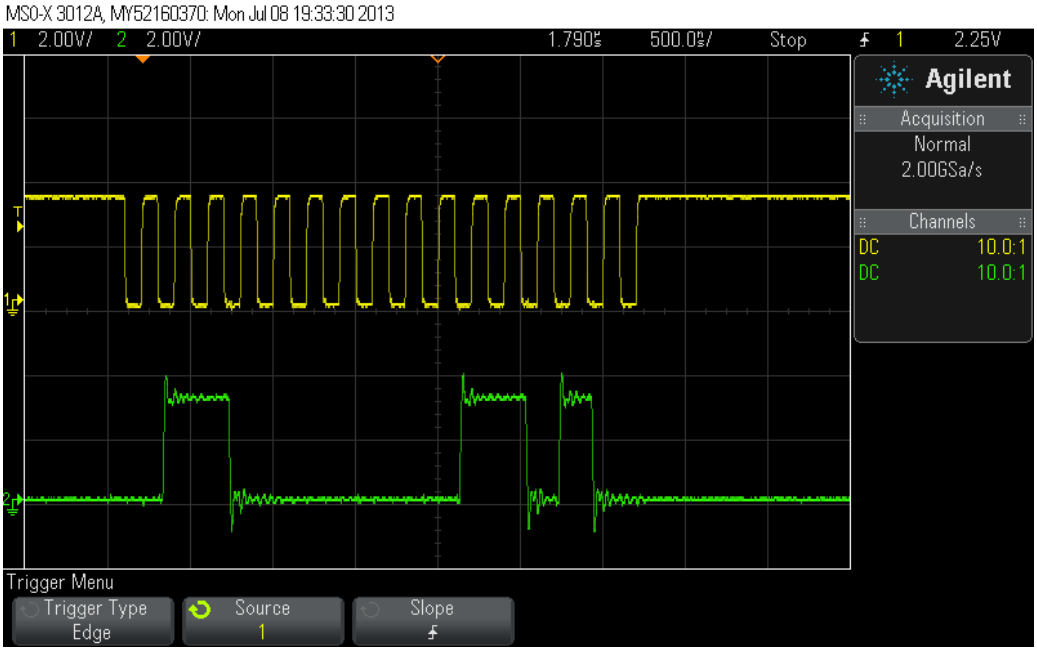


Figure 56: Oscilloscope frame capture of the AD's configuration frame

Yellow trace: SCLK measured on the pin of the AD converter
Green trace: result of the AD conversion measured on the MISO pin of the AD converter

As explained in chapter 4.7.3, the figure above shows the result of the AD conversion. It shows that the conversion was performed on the channel 2 of the AD converter. The converted value corresponds to 000000011010.

This tests shows that the AD converter can be correctly configured and that an AD conversion can be performed on the desired channel.

8 DEMONSTRATOR

The goal of this demonstration application is to read the values of an SS sent by the FPGA board. Once these values are received, a calculation determines the position of the sun relative to the orientation of the SS. Depending on the orientation of the SS, the MT will be driven to orient a virtual satellite.

To reduce the coding difficulty of the final application, different intermediary applications were designed. The first one was designed to establish an UART communication between the application and the FPGA board. The second one draws a 3D cube that can be orientated.

The different applications were designed on `QT 4.8.4` and written in `C++`.

8.1 UART communication application

The goal of this test application is to establish an UART communication between the computer and the FPGA board. The connection is made with an RS-232 cable using the UART protocol. The complete code of this application can be found in the DVD in annex.

8.1.1 Description

This application allows the sending of a character string to the FPGA board.

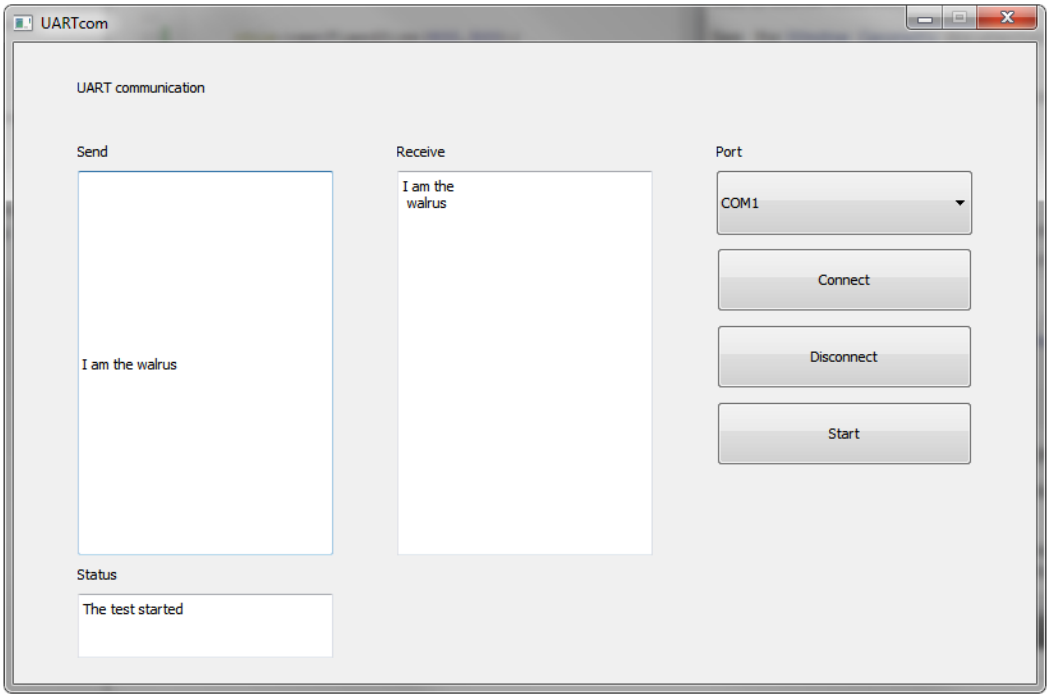


Figure 57 : UART communication application

Once the character string has been parsed into several UART frames, these are sent to the FPGA. For this test, the FPGA is configured to return without modification the received frames to the computer. The application decodes the received UART frames and displays them. The Figure 57 shows a frame exchange. On this figure we can verify that a character string is entered in the field `Send` and that the identical string is displayed in the field `Receive`.

8.1.2 Class overview

The Figure 58 shows the different classes that take part in the design of the application `UARTcom`.

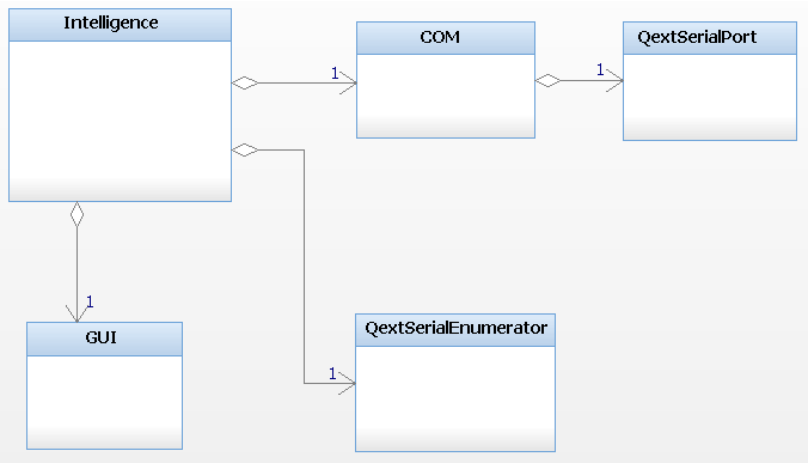


Figure 58: UARTcom class diagram

The class `Intelligence` is the decision-making class of the design. It creates the instance of the classes `COM`, `QextSerialEnumerator` and `GUI`. All the decoding of the frames and the forming of the UART frames is done in this class.

The class `COM` creates the instance of the class `QextSerialPort`. It contains the methods that allow the communication on the serial port by calling the methods from `QextSerialPort`.

The class `GUI` displays the graphical user interface on the screen.

The classes `QextSerialEnumerator` and `QextSerialPort` are two API classes used to communicate on the serial port.

8.1.3 Behavior of the code

This chapter gives a description on the behavior of the code.

Creation of a new connection:

The following figure show the behavior of the code when a new connection is opened.

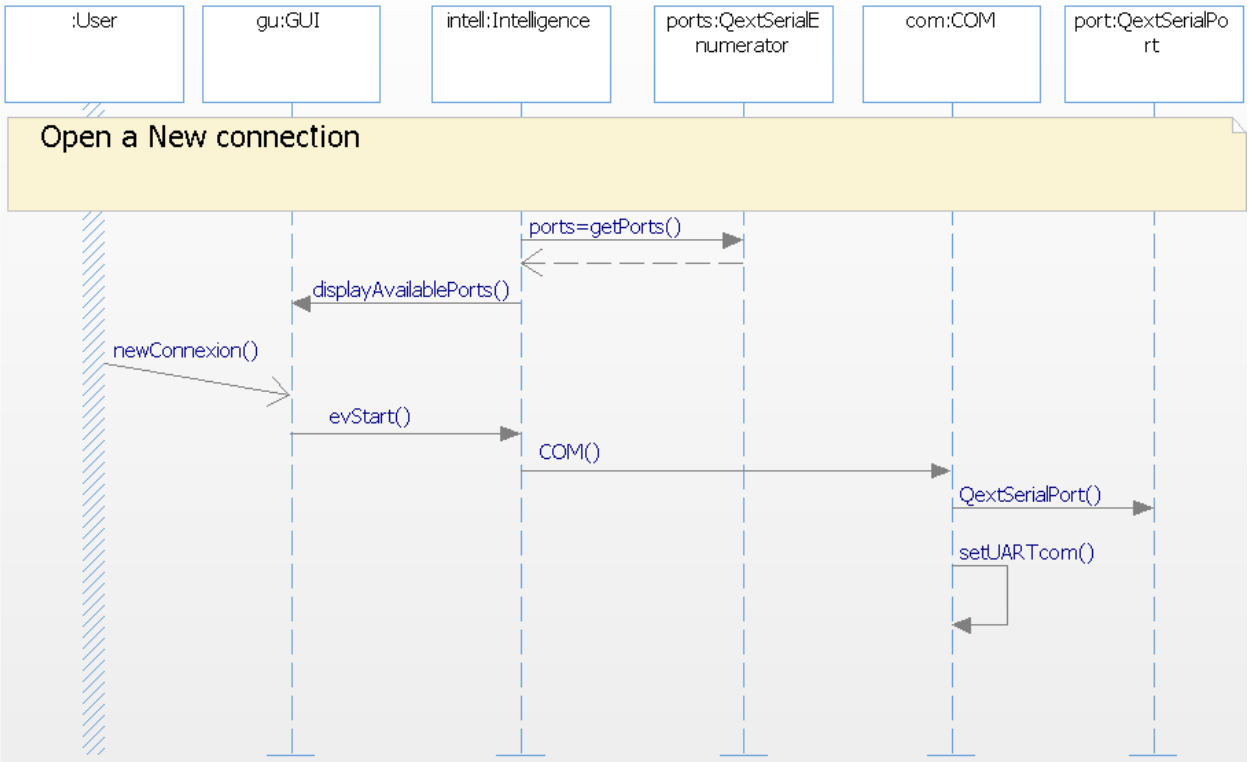


Figure 59: Sequence diagram of the opening of a new connection

When the code starts, all the available ports are scanned by the instance `ports` of the class `QextSerialEnumerator`. They are then displayed on the graphical user interface. When the user has chosen a port, clicked on the button `connect` and then `start` (see Figure 57), the object `intell` creates the object `com` that itself creates the object `port` to open a new serial communication. The settings of the UART communication are done in the constructor of the class `COM`.

Transmission of an UART frame:

The following figure show the behavior of the code when a text is entered and is sent on the UART bus.

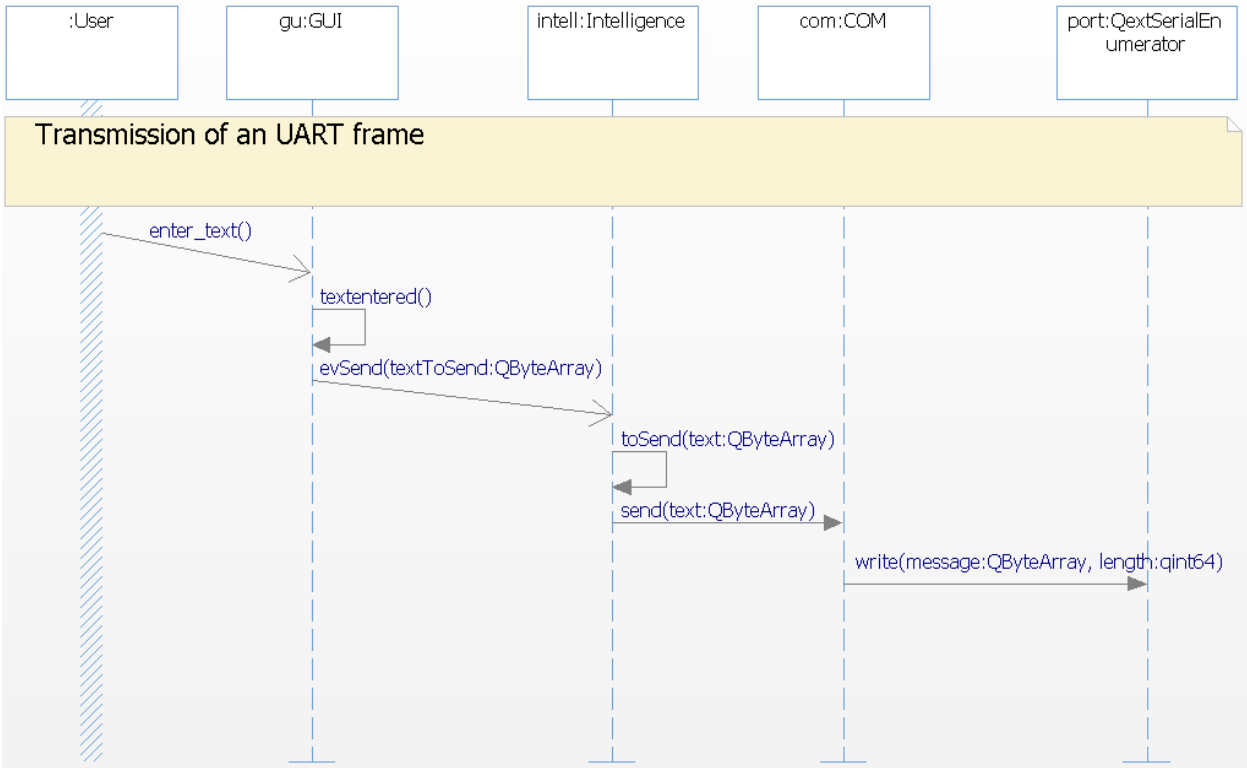


Figure 60: Sequence diagram of the transmission of an UART frame

When the user enters a text in the cell send (see Figure 57), the text is captured by the method `textentered()` of the class `GUI` and sent to the class `Intelligence` where it is separated into frames of 8 bits. Once the text has been parsed, it is send to the class `QextSerialPort` where it is then sent on the UART bus.

Reception of an UART frame:

The following figure show the behavior of the code when an UART frame is received on the opened port.

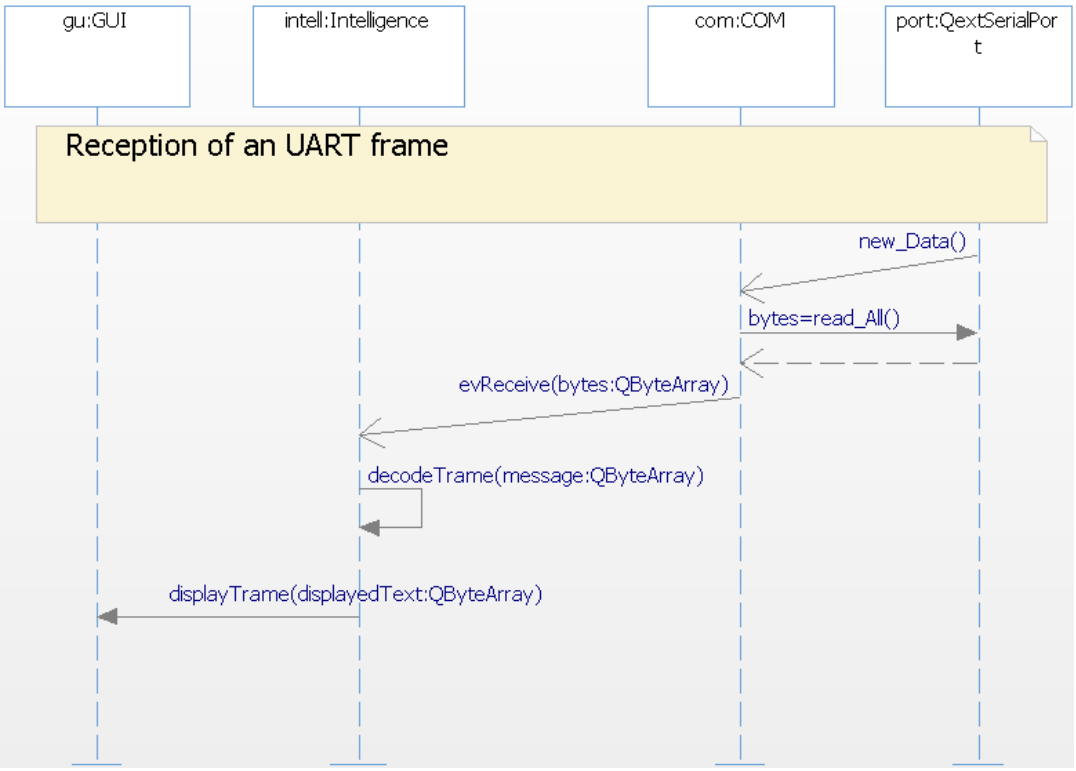


Figure 61: Sequence diagram of the reception of an UART frame

When new data are received on the opened serial port, an event is sent to the class `COM`. This class calls the method `readAll()` to read all data that have arrived on the port. These data are then sent to the class `Intelligence` where they are parsed. Once the incoming data have been parsed, they are displayed on the graphical user interface by the class `GUI` in the cell `Receive` (see Figure 57).

8.1.4 Validation of the communication

This application determined that a communication between the FPGA and the computer was established. It also validates the ability of the FPGA to send UART frames.

8.1.5 Further information concerning the coding of this application

Unfortunately Qt 4.8.4 doesn't have an interface library that encapsulates the UART communications. The `QextSerialPort` library was therefore used. As this library isn't native on Qt, a precompiled project had to be used and the `UARTcom` project was then added into it. This project is called `qextserialport` and the project was added to the folder `application` contained in it. This had to be done to have access to the method provided by the class `qextserialport`.

The following figure shows the location of the application inside the `qextserialport` project:

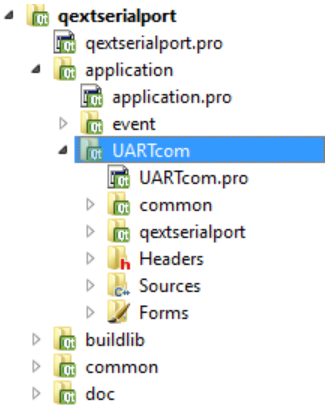


Figure 62: Location of the UARTcom project

To open the project using Qt, open the file `qextserialport.pro`. To compile and run the project, set the Run project as shown in the figure below:

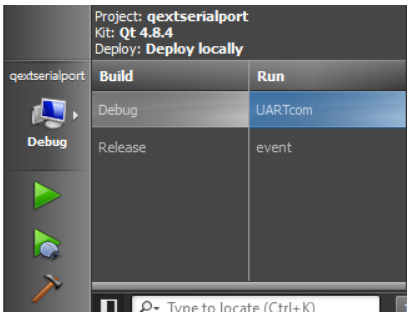


Figure 63: Run configuration of the application

8.2 Application for the Simulation of a Cube « Cube_draw »

The purpose of this application was to create a system which can simulate the behavior of a 3D cube. This was performed using the OpenGL library. The complete code of this application can be found on the DVD in annex. This application was created with the help of the tutorial in the bibliography [7].

8.2.1 Advantages of openGL

OpenGL offers efficient development tools for creating 3D applications. It easily allows the declaration of geometric shapes in the form of point or vectors. It automatically performs the projection calculation to define the image on the screen by taking into account the distance or the orientation of the object. This mode of operation allows to easily create a 3D cube and to orient it according to a given angle.

Another advantage is that Qt Creator wraps OpenGl, making it fairly easy to integrate it in an existing Qt project.

8.2.2 Key methods overview

This chapter gives an overview of the crucial methods used in this application.

glTranslatef:

```
void glTranslatef(float x,float y,float z);
```

This method produces a translation by (x,y,z). The parameters x,y and z represent the translation axis. These axes are represented in Figure 67 (c).

glRotatef:

```
void glRotatef(float angle,float x,float y,float z);
```

This method is used to orient the cube. `GlRotatef` multiplies the current matrix by a rotation matrix. It produces a rotation of an angle in degrees (angle) around the vector (x,y,z). This vector has as a base the center of the figure. This can be seen in Figure 66 (c).

glBegin / glEnd:

```
void glBegin (GLenum mode) ;
void glEnd (void);
```

`glBegin` and `glEnd` delimit the vertices that define a primitive. `glBegin` accepts a single argument that specifies in which of ten ways the vertices are interpreted. In this case the mode used is `GL_QUADS`, which interprets the vertices as a 3D quadrilateral.

glTexCoord2f :

```
void glTexCoord2f(float s,float t);
```

This method specifies the texture coordinates in two dimensions.

glVertex3f:

```
void WINAPI glVertex3f(GLfloat x, GLfloat y, GLfloat z);
```

This method specifies the vertices of a point, line or polygon. It is used in this case to define the vertices of the cube. The parameters x, y and z specify the position of one vertex.

8.2.3 Drawing the cube

This chapter gives explains how the cube is drawn. The aim is to show the relation between the different methods described in chapter 8.2.2. The following code is only a small part of the code used to draw the cube.

```
    glTranslatef(0.0, 0.0f, -7.0f);
    glRotatef(-20.0, 0.0,1.0,0.0);

    glBindTexture(GL_TEXTURE_2D, texture[0]);

    glBegin(GL_QUADS);
        // Front Face
        glTexCoord2f(0.0f, 1.0f); glVertex3f( -1.0f,  1.0f,  1.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(1.0f,  1.0f,  1.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(1.0f, -1.0f,  1.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f( -1.0f, -1.0f,  1.0f);
        // other faces
              .
              .
              .
    glEnd();
```

Figure 64 shows the texture that is applied on the faces of the cube.

Figure 64: Texture of the cube. Box.png

glTexCoord2f / glVertex3f:

The following figure shows the relation between the methods `glTexCoord2f` and `glVertex3f`.

Figure 65 : Relation between glTexCoord2f and glVertex3f

This figure was drawn by the code above. We can see that the texture is applied on the vertices as shown in the figure above. Example:

```
glTexCoord2f(0.0f, 1.0f); glVertex3f( -1.0f,  1.0f,  1.0f);
```

We can see that the top left vertex of the texture is applied to the top left vertex of the cub.

glRotatef:

The following figures show the effects of the method glRotatef.

(a) (b) (c)

Figure 66: Effects of glRotatef. (a) glRotatef (-10,0,1,0). (b) glRotatef (-20,0,1,0). (c) Rotation Axis

The cubes in Figure 66 (a) and (b) were drawn by the code above. The angle of rotation was modified from -10° in figure (a) to -20° in figure (b). Therefore, the method `glRotatef` was modified as following:

glRotatef(-10,0,1,0) => glRotatef (-20,0,1,0)

The rotation is made clockwise around the vector 0, 1, 0 in orange in Figure 67 (c). This vector has as base the center of the form and has the same direction as the y axis.

glTranslatef:

The following figures show the effects of the method `glTranslatef`.

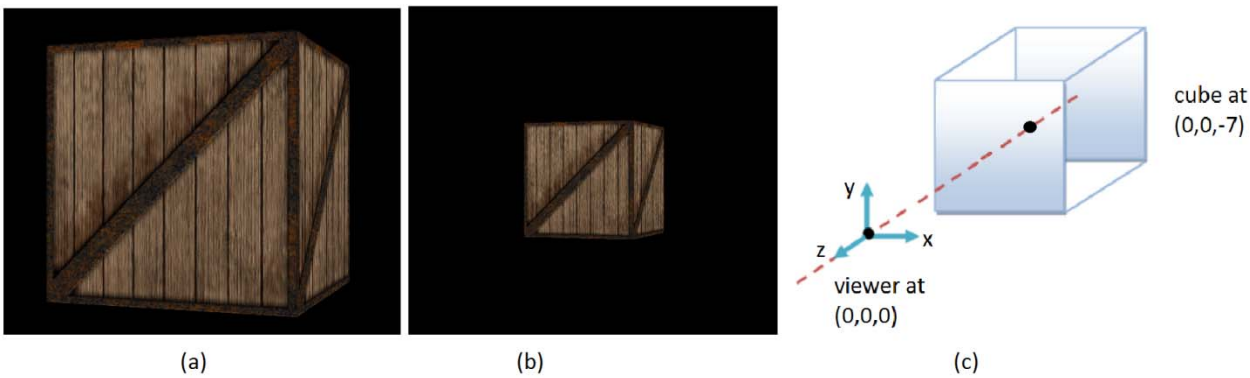


Figure 67: Effects of glTranslatef. (a) glTranslatef (0,0,-7). (b) glTranslatef (0,0,-15). (c) Translation Axis

The cubes in Figure 66 (a) and (b) were drawn by the code above. The position of the cube in the frame was modified from along the z axis from -7 in figures (a) to -15° in figure (b). Therefore the method `glTranslatef` was modified as following:

glTranslatef(0,0,-7) => glTranslatef (0,0,-15)

The Figure 67 (c) shows the translation axis. The viewer is positioned at 0,0,0 when a translation is made along the z axis, the object is redrawn further away, making it look smaller.

8.3 Demonstrator integration

8.3.1 Description

This last application asks and reads back the values of the SS to/from the FPGA board, sets the PWM's duty cycle for the MT and reads back its current. It then computes the SS values to find the two angles of the Satellite-Sun vector. A cube is then oriented according to the Satellite-Sun vector. Two reference angles can be specified. If the two angles of the Satellite-Sun vector are not the same as those two reference angles, the application commands the FPGA to start the generation of a PWM. When the Satellite-Sun vector is realigned with the two reference angle, the order is given to stop the PWM generation. The following figure shows the design of the final application.

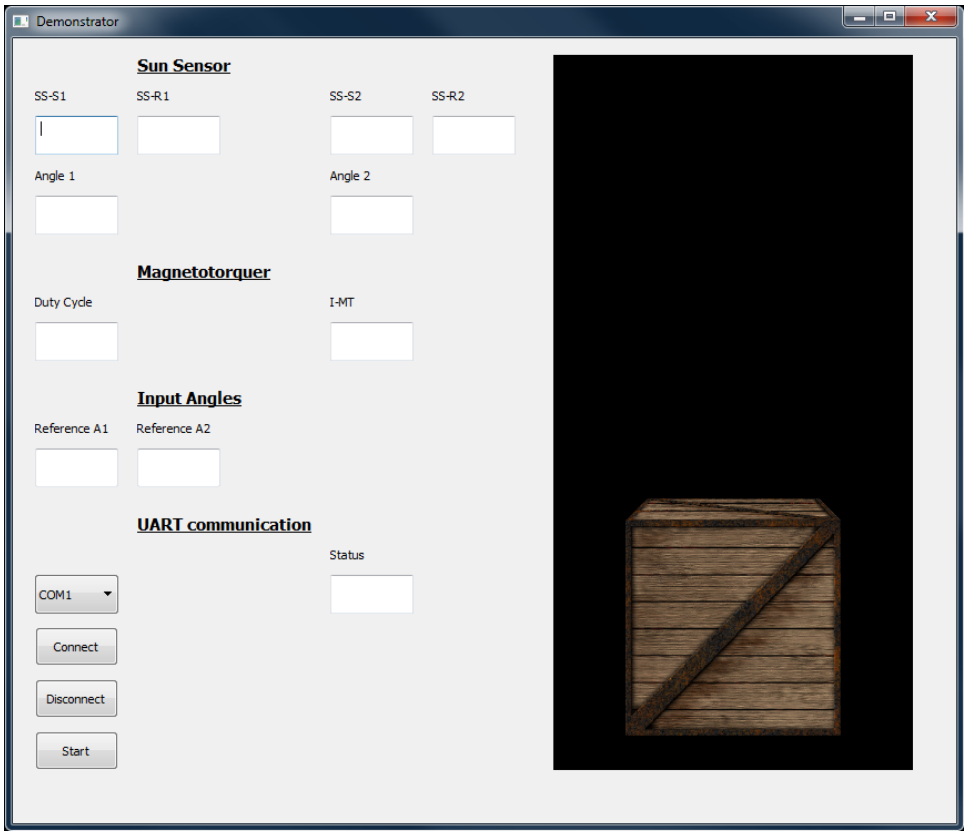


Figure 68: Demonstrator application graphical user interface view

The cells SS-S1 and SS-R1 give the voltage value of the first angle of the Satellite-Sun vector and of its reference voltage.

The cells SS-S2 and SS-R2 give the voltage value of the second angle of the Satellite-Sun vector and of its reference voltage.

The two cells Angle 1 and Angle 2 gives the angle values in degrees for the commutation of the 4 previous cells.

The cell Duty Cycle gives the value of the duty cycle that is actually generated and the cell I-MT gives the current that is currently flowing in the MT.

The two next cells Order A1 and Order A2 are where the reference angles can be set.

The last part under "UART communication" is relative to the UART communication between the PC and the FPGA board. The communication must be established before any value can be read on this GUI.

8.3.2 Reuse of previous applications

The application "UART communication application" was reused to establish the UART communication between the application on PC and the FPGA board. It was also used as a basis to begin the development of the final application "Demonstrator application". The code was reused unchanged from this application. A few classes were however renamed, as shown in the exhaustive list below:

Intelligence => Controller
Gui => Graphical

The class myglwidget was reused from the "Cube_draw" application to provide the necessary methods to draw and control a 3D cube. This class was added to the new project and renamed to `graphOpGL`.

8.3.3 Class overview

Figure 69 shows the different classes that take part in the design of the application `Demonstrator`.

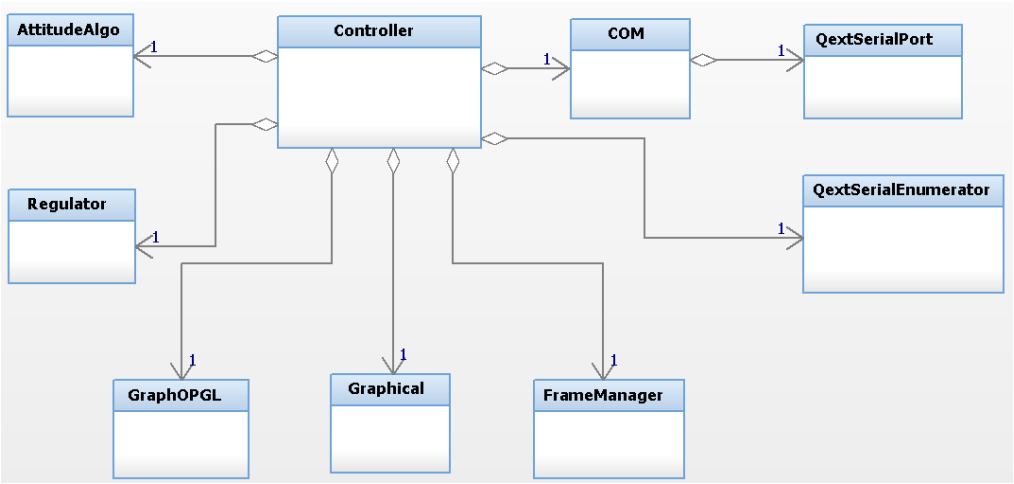


Figure 69: Class diagram of the Demonstrator

The class `Controller` is the decision making class of the design. It delegates all the tasks to the other classes.

The class `COM` creates the instance of the class `QextSerialPort`. It contains the methods that allow the communication on the serial port by calling the methods from `QextSerialPort`.

`Graphical` displays the graphical user interface on the screen. And `graphOpGL` is where the 3D cube is drawn and controlled.

The classes `QextSerialEnumerator` and `QextSerialPort` are two API classes used to communicate on the serial port.

All the UART frames are created and decoded in the class `frameManager`.

The class `attitudeAlgo` is where the angles of the Satellite-Sun vector are computed from the voltage values received from the SS.

The last class `regulator` is the class that determines if the FPGA has to generate a PWM to actuate the cube, if the Satellite-Sun vector is not aligned according to the two reference angles.

8.3.4 Behavior of the code

This chapter describes the behavior of the code. As the code for establishing a new connection, sending a UART frame and to receiving an UART is the same as in chapter 8.1, it will not be re-explained here.

The outputs of the SS explained in chapter 4.4.2 have been defined as following in the code: V1 has been defined to S1 Vref1 to R1, V2 to S2 and Vref2 to R2.

<u>Start AD conversion for SS:</u>

The following figure shows the behavior of the code when an AD conversion has to be started by the FPGA board to retrieve the values of the SS.

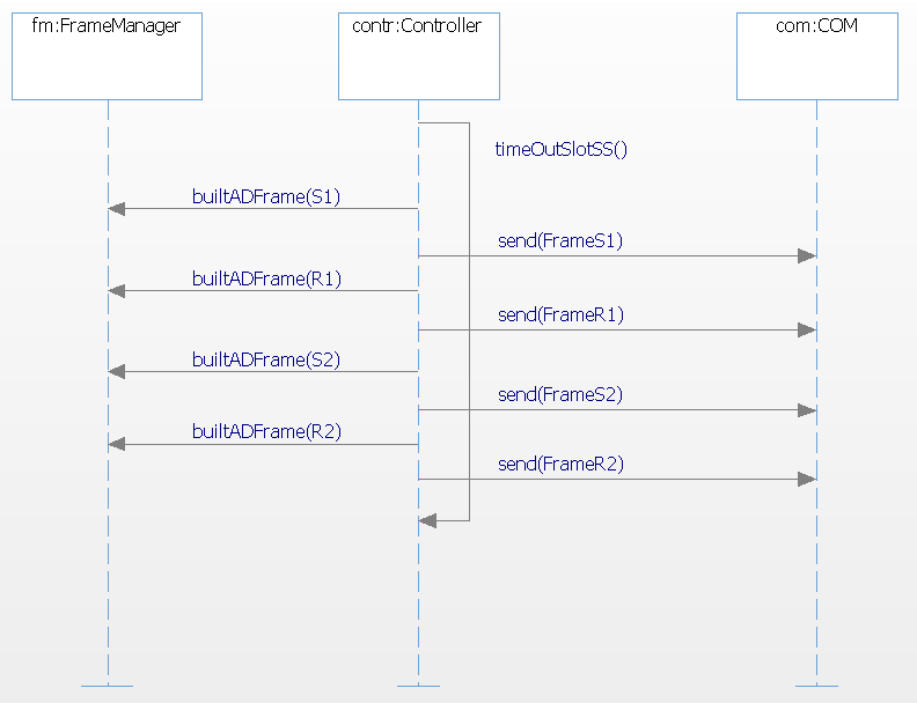


Figure 70: Sequence diagram for the start of an AD conversion for the SS

The aim of this code is to retrieve the value of the four outputs from the SS. Therefore, it commands the FPGA to perform an AD conversion on one of the outputs at a defined interval of time.

It is important that the conversions of the SS's outputs are made as closely together as possible to maintain consistency in these measurements. If his was not the case, the values would not represent the reality and the computation of the Satellite-Sun vector would be distorted. Therefore this interval of time between two AD conversions has to be as short as possible.

To perform a control when the returned values of the conversion are received, the conversions are performed in a predefined sequence that is: S1, R1, S2 and finally, R2.

This code is started when the timer `timerSS` elapses. When it elapses, the code jumps to the slot `timeoutSlotSS()`. The method `builtADFrame(AD_Channel_number)` from the class `framManager` is called to build the UART frame used to start an AD conversion on the FPGA board.

This method sends back the QByteArray `ask` containing the UART frame. This array is then sent to the class `COM` by the method `send(message)`, where it is sent via the UART bus.

This code is repeated each time the `timerSS` elapses. Each time, the AD conversion channel is changed to scan all the outputs of the SS. The method `builtADFrame(AD_Channel_number)` therefore it takes as a parameter the number of the AD channel. The possible values are listed below:

AD channel 1 =S1
AD channel 2 = R1
AD channel 3 = S2
AD channel 4 = R2

S1 is the first angle of the SS and R1 the reference for this first angle. S2 is the second angle of the SS and R2 the reference for this second angle.

Note that no verification is made as to whether the AD conversion has been performed or if the frame has been correctly received on the FPGA board. The frame to start an AD conversion is sent but the code doesn't wait for an answer from the FPGA board. This test is not made because the returned AD frame sent from the FPGA board contains the number of the AD channel on which the AD conversion was performed. So we can therefore assume the conversion has taken place.

Start AD conversion for the current flowing in the MT:

The following figure shows the behavior of the code when an AD conversion has to be started by the FPGA board to retrieve the values of the current flowing in the MT.

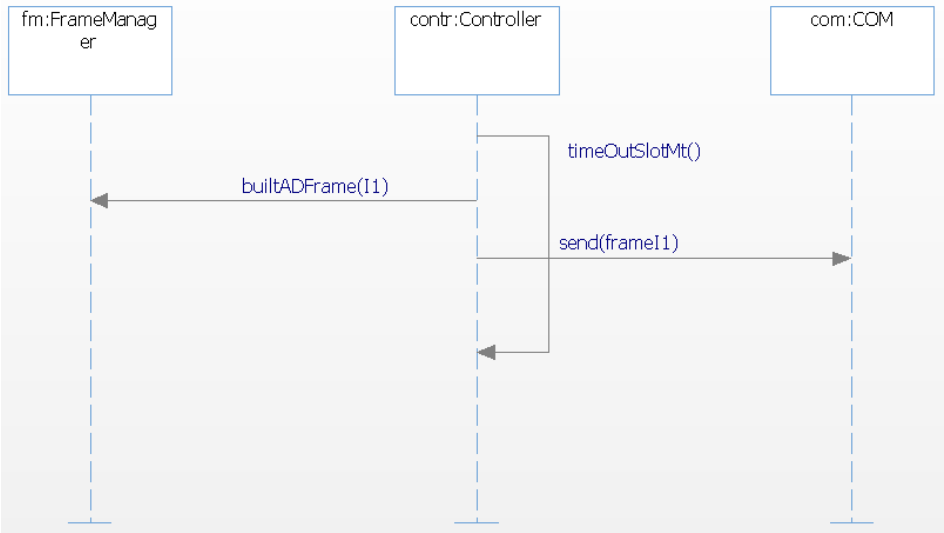


Figure 71: Sequence diagram for the start of an AD conversion for the current flowing in the MT

The aim of this code is to retrieve the value of the current that is flowing in the MT. Therefore, it commands the FPGA to perform an AD conversion on the channel 0 that is linked to the output of the current probe. This conversion is stareted at a defined interval of time.

This code is started when the timer `timerMT` elapses. When it elapses, the code jumps to the slot `timeoutSlotMT()`. The method `builtADFrame(AD_Channel_number)` from the class `framManager` is called to build the UART frame used to start an AD conversion on the FPGA board.

This methods sends back the QByteArray `ask` containing the UART frame. This array is then sent to the class `COM` with the method `send(message)`, where it is sent via the UART bus.

The method `builtADFrame(AD_Channel_number)` takes as a parameter the number of the AD channel. The AD channel corresponding to the current flowing in the MT is the number 0 = I1

Note that, as in the previous point no control is made as to whether the AD conversion has been performed or whether the frame has been correctly received on the FPGA board.

Decoding of an UART frame containing the value of an AD conversion:

The aim of this code is to receive and decode an UART message containing the value of an AD conversion and of the channel it was performed on. Then, the objective is to convert the digital value of the conversion into a voltage value. The last purpose of this code is to control whether the sequence of the AD conversion on the SS has been correctly executed (as explained in chapter "Start AD conversion for SS").

A. Explanation of the flow-chart diagram of the method `decodeTrame` and of its related methods

The following figure shows the flow-chart diagram of the method `decodeTrame`.

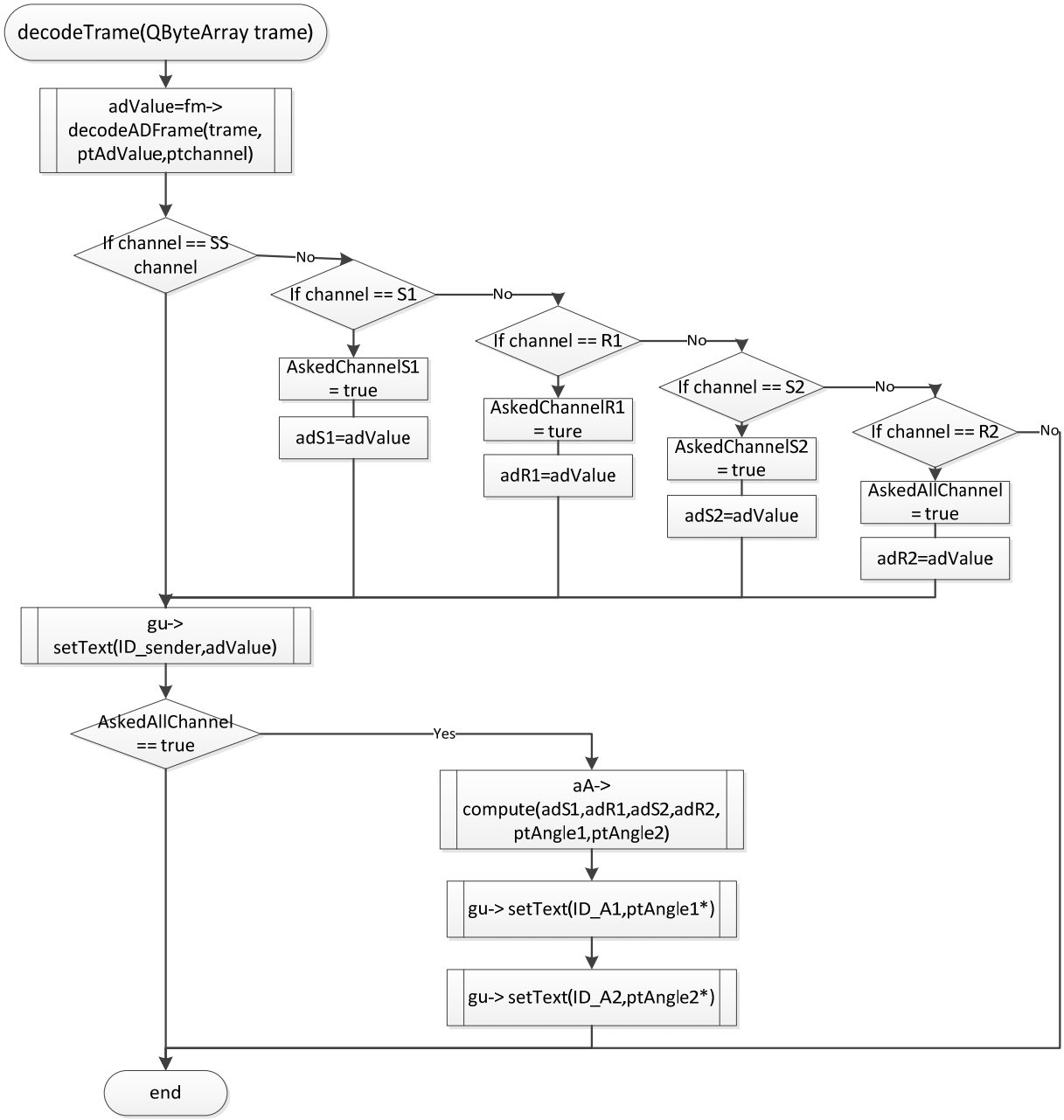


Figure 72: Flow-chart diagram of the method decodeTrame

The method `decodeADFrame(QByteArray trame, float* ptAdValue, quint8* ptchannel)` takes three arguments. The first argument `frame` passes to the method a complete UART message containing a start frame, the result of the AD conversion and the channel it was performed on. The two other parameters are pointers whose pointed values will be modified by the method.

In this method, the parameter `trame` is decoded to retrieve the AD value of the conversion and its channel. The AD value is converted from a digital value to a tension value and the content of the memory address pointed by `ptAdValue` is modified with this converted tension value. Likewise, the memory address pointed by `ptchannel` is modified with the channel value.

In short, this method allows one to retrieve the tension value and the channel contained in a UART message.

Once the AD voltage value and the channel number have been modified by the `decodADFrame` method, a test is performed in the `decodeTrame` method to determine if the channel corresponds to a SS channel or to a current measure channel.

If it's a current measurement of the MT channel, the voltage values is converted into current (see Figure 15) and displayed on the graphical user interface. In this case, the code of this method is finished.

If it's an SS channel, the voltage value is displayed on the graphical user interface and other tests are performed. These tests are explained in the following paragraphs.

As explained in the chapter "Start AD conversion for SS", it is important that the conversion of the outputs of the SS are performed as closely together as possible. Therefore, if the values are not received in the correct order of channel, this indicates that a conversion message has been lost and the whole sequence of conversion is corrupted and has to be ignored. Therefore the tests showed in Figure 72 are performed to control this sequence.

Once this sequence has been correctly achieved, the method `compute(float s1,float s2, float r2,float* angle1, float* angle2)` is used to retrieve the two angles of the Satellite-Sun vector. The two pointers `angle1` and `angle2` point to the current values of the Satellite-Sun's angles. Those two angles are updated by this method `compute` following the algorithm explained in chapter 8.3.5.

B. Explanation of the sequence diagram

The following figure show the behavior of the code when the return AD conversion UART message is received on the PC.

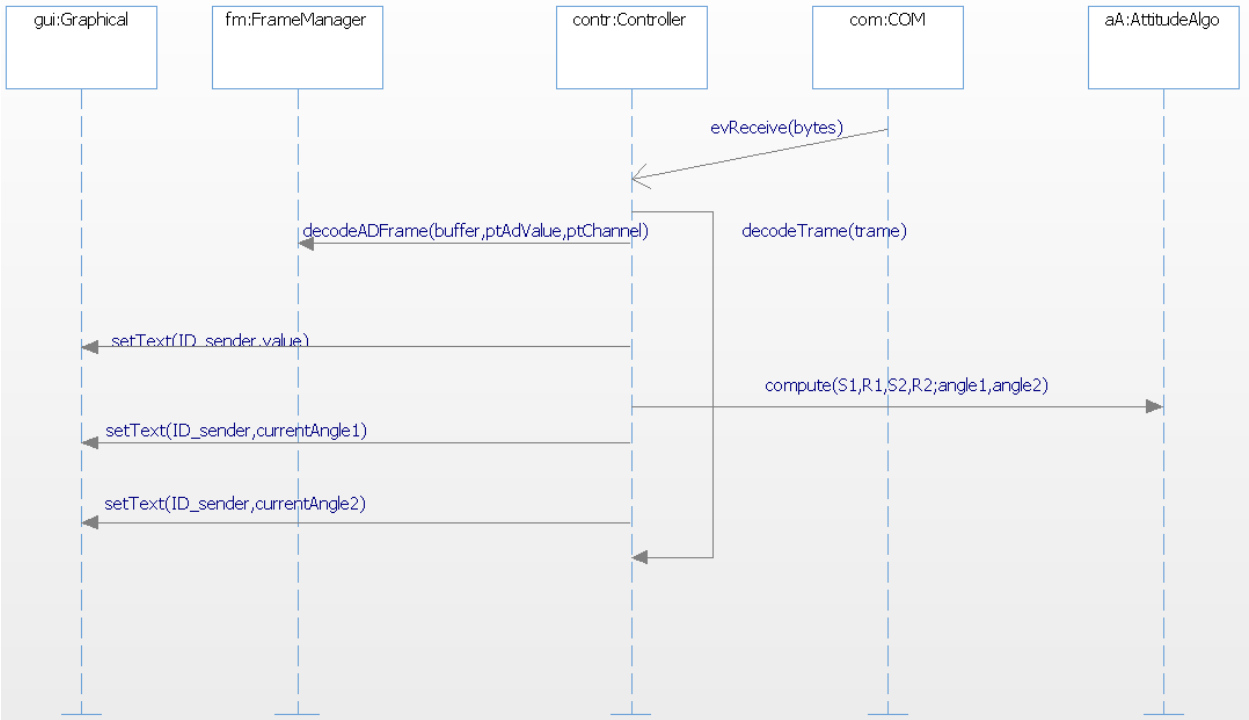


Figure 73 : Sequence diagram for the decoding of an UART message containing the value of an AD conversion of the SS

The event `evReceive(bytes)` is sent to the class `controller` when new UART message arrives on the UART bus. This event contains this received message.

The incoming message is sent to the class `frameManager` with the method `decodeADFrame(buffer,ptAdValue,ptChannel)` where it is decoded to extract the result of the AD conversion and the channel it was performed on.

The AD conversion value is also converted into a tension value. These two values are then sent back to the method decodeTrame of the class controller. Following the flow-chart diagram at Figure 72, this tension value is then displayed in the proper cell of the GUI with the method `setText(ID_Sender)` of the class `graphical`.

If all of the SS ports avec been successfully converted in the correct sequence, their tension values are sent to the class `attitudeAlgo` using the method `compute(adS1,adR,adS2,adR2, ptAngle1,ptAngle2)`. In this method, the angles of the Satellite-Sun vector are computed, as explained in the chapter 8.3.5. These two angles are then returned to the class `controller` and displayed on the graphical user interface.

Start the generation of a PWM:

The following figure shows the behavior of the code when it has to start the generation of a PWM.

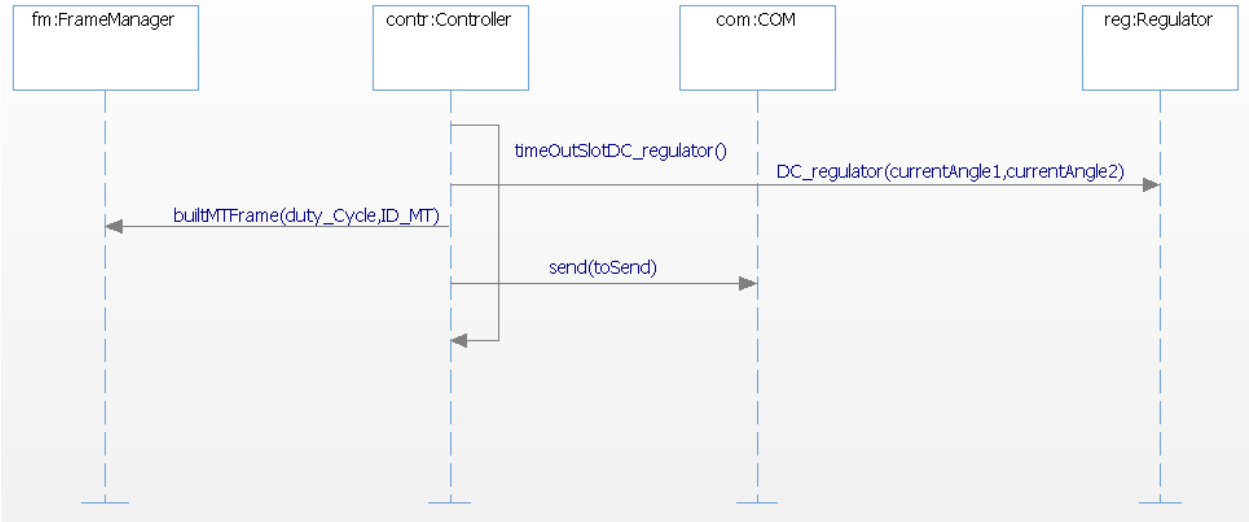


Figure 74: Sequence diagram for the generation of a PWM message

The aim of this code is to send an UART message to the FPGA containing the value of the PWM's duty cycle that has to be generated, along with the direction of the current that must flow through the MT. The value of the PWM's duty cycle is determined by a simple regulation algorithm.

This code is started when the timer `timerDC_Regulator` elapses. When this occurs, the code jumps to the slot `timeoutSlotDC_regulator()`. The method `DC_regulator(currentAngle1,currentAngle2)` of the class `regulator` is called to perform the regulation of the PWM's duty cycle.

Once the DC and the direction of the PWM have been determined, this value is sent to the method `builtMTFrame(duty_Cyle,ID_MT)` to be shaped into an UART frame and is then sent on the UART bus by the method `send(toSend)`.

8.3.5 Satellite-Sun vector determination algorithm

The attitude algorithm that returns the two angles of the Satellite-Sun vector is explained below.

This algorithm was elaborated during the "Characterization and calibration of dual axis sun sensors for ADCS" at the EPFL in 2008, see bibliographies [4].

The following flow-chart illustrates the code used for the attitude determination algorithm.

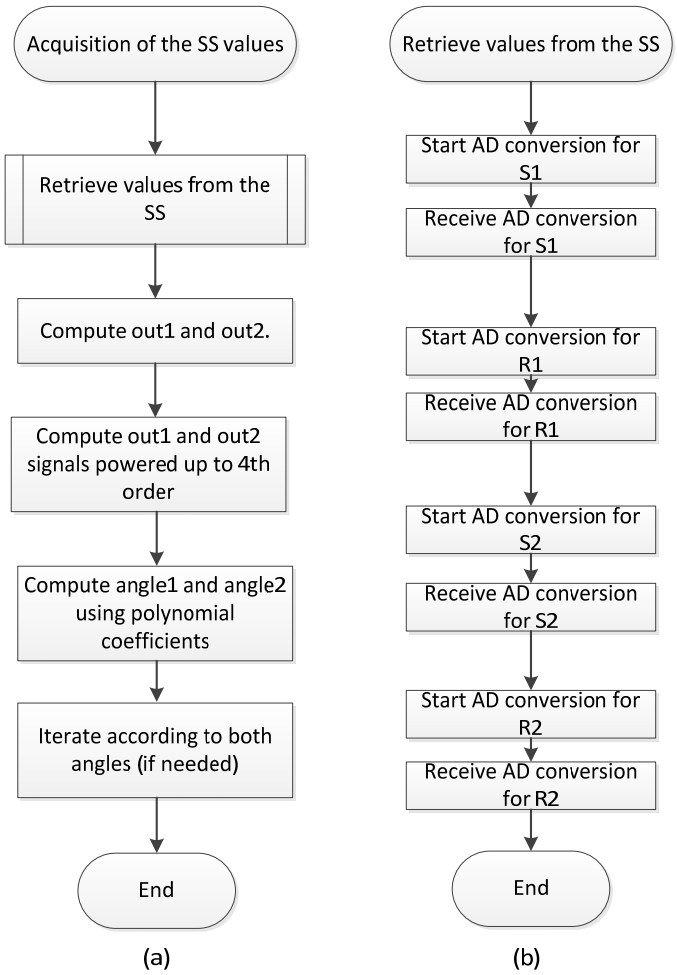


Figure 75: Flow chart of the Satellite-Sun vector computation

The computation of the Satellite-Sun vector starts by the acquisition of the values returned by the SS (see flow-chart Figure 75 (b)). Once this operation has been achieved, the algorithm computes `out1` and `out2` with the following expression:

```
out1=(s1-1.25)/r1;
out2=(s2-1.25)/r2;
```

The -1.25 in the expressions is to account for the offset added to the measurement of s1 and s2 (see chapter 4.4.2). Those two values out1 and out2 are then powered up to 4th order to fit the shape of the SS curve (see Figure 8).

```
out1m2= pow(out1,2.0);
out1m3= pow(out1,3.0);
out1m4= pow(out1,4.0);
out2m2= pow(out2,2.0);
out2m3= pow(out2,3.0);
out2m4= pow(out2,4.0);
```

The next step is to compute the values previously calculated with the polynomial coefficients as shown below:

```
angle1 = c0 + out1 * c1 + out1m2 * c2 + out1m3 * c3 + out1m4 * c4;

angle2 = d0 + out2 * d1 + out2m2 * d2 + out2m3 * d3 + out2m4 * d4;
```

These coefficients have been computed during the "Characterization and calibration of dual axis sun sensors for ADCS" at the EPFL in 2008, and can be found in the DVD in annex.

The last step is used to compensate for the influence of the second angle one the first one.

1. If the angle2 is between -20° and -40° or -40° and -60°, we change the c polynomial coefficients to take the second angle into account
2. Compute angle1 again
3. If the angle1 is between -20° and -40° or -40° and -60°, we change the d polynomial coefficients to take the secondary angle into account.
4. Compute angle2 again

9 STUDY ON THE ISOLATION OF THE ADCS BOARD'S SENSORS AND ACTUATORS

As explained in the introduction, the EPFL requested a study to determine whether, in case of malfunction of the ADCS board, it can be isolated from the different sensors and actuators so they could be accessed from and managed by the CDMS board.

Unfortunately, no time was left in this project to perform this study. A brief analysis of the tasks that need to be achieved to realize this study was performed, however. These are described below.

9.1 Components to isolate

The first task would consist of an analysis of the components that would need to be isolated in case of a failure. Which components could cause a failure? If it's a crucial sensor (SS, gyroscope…) could a similar component be placed on the CDMS board for redundancy? Those questions could be answered by the study of the existing schematic of the ADCS board that can be found in the DVD in annex and with a discussion with the EPFL on their precise requirements.

9.2 Failure analysis

To see if the isolation of the problematic components defined in step1 is possible, a second step has to be performed. This step consists of a failure analysis that would determine the possible causes of a failure of the ADCS board. To perform this task, a study of the report made by the ECSS entitled "Space Product Assurance, Failure Mode, Effects Analysis" would be essential to compile a complete list of all possible causes of failure. This report can be found on the DVD in annex.

9.3 Schematic proposition and test

A third task would consist of the creation and the test of electronics that could fill the role of isolating the problematic component(s) on the ADCS board in case of a failure. For the test of these electronics, the creation of a test board would be recommended. The isolating circuit would have to be small enough to fit on the existing ADCS board. 9.1 and 9.2

9.4 Estimated time needed to accomplish the task

As everything has yet to be done, an estimated additional work of 4 weeks would be needed, as a test board would have to be created with the necessary components to fully characterize the tasks presented in chapters

10 SYSTEM VALIDATION

This chapter combines all the validations that were performed during the previous chapters and additional validations that were done when the entire system was assembled.

10.1 Hardware validation

10.1.1 Validation of the Sun Sensor

The validation of the SS and its related electronics was achieved during the Hardware test performed in chapter 5.1.2. This test concludes that the electronics for the SS work correctly. The values received from the SS are the same has the one described in chapter 5.1.2.

10.1.2 Validation of the Magnetotorquer

The validation of the MT and its related electronics was done during the hardware test performed in chapter 5.1.2. This test concludes that the electronics for the SS don't work quite correctly. The MT can be actuated correctly in one way or the other. The current probe doesn't work correctly, as explained in the chapter 5.1.2.

10.1.3 Interface board

Apart from the current probe problem, the entire board works correctly. A version 2 of the board with the modifications made in schematics could be done easily (see DVD in annex).

10.2 VHDL validation

All the VHDL blocks are working correctly. They all have been tested and simulated.

If the final protocol described in chapter 6.2 was implemented, the block `code_decode_Trame` would need to be modified. The header block could remain unchanged.

If a new version of the interface board implementing more MTs and SSs was required, the system could handle the additional hardware.

The block `code_decode_Trame` would need to be modified.
The block `PWM_GEN` and `adCOM` could simply be duplicated.

10.3 Software validation

10.3.1 Acquisition of values from the Sun Sensor and the Current Probe

The values received from the SS on the Demonstrator are identical, within the accuracy limitations of the AD converter, to those measured on the test points of the SS (see chapter 5.1.2.). That indicates that the AD conversion is correctly done and that the conversion frame digital value to voltage value is correctly formed.

A similar observation is made for the value received from the current probe. The measured value and the converted value are similar, as showed in the appendix 3.

10.3.2 Attitude algorithm

A complementary test was run to see if the attitude determination algorithm woks correctly.

S1 [V]	R1 [V]	S2 [V]	R2 [V]	Angle1 [°]	Angle2 [°]
1.25	0.42	1.25	0.42	1.31	-0.527
0.85	0.32	1.25	0.42	-56.327	0.581
1.55	0.32	1.25	0.42	43.985	-0.5275
1.25	0.42	0.85	0.32	1.31	-60.099
1.25	0.42	1.55	0.32	1.31	42.162
1.55	0.32	1.55	0.32	43.985	42.162
0.85	0.32	1.55	0.32	-56.327	39.143
1.55	0.32	0.85	0.32	43.9854	-60.099

Figure 76: Test of the Attitude algorithm

The Figure 76 shows the test results of the attitude determination algorithm. The values for SI, RI, S2 and R2 were entered manually to be certain to test the algorithm with the desired values.

The test shows that the algorithm returns values that approach those of the Figure 8. As the coefficients for the algorithm were calculated at the EPFL during the "Characterization and calibration of dual axis sun sensors for ADCS" at the EPFL in 2008 (see bibliographies [4]), it is not possible to know if those values are precise or not.

A test of the algorithm was run from values directly retrieved form the SS. The test was run under the conditions defined in chapter 5.1.2. The results were not conclusive, as the 3D cube is not oriented correctly. The problem could lie in the fact that the measurements on the outputs of the SS are made with too much delay between one and on other.

10.3.3 Regulation algorithm

The simple regulation algorithm is functional. The PWM is driven when the first angle of the Satellite-Sun becomes smaller or larger than a defined angle and is stopped when it is equal to this defined angle. This defined angle has to be set by the user.

10.4 Test of the entire system

The entire system was tested together. Aside from the problems mentioned earlier, all the different parts that comprise this systems work correctly when connected together. The data from the SS are converted to numerical values and sent to the application, where they are computed with the regulation algorithm and displayed on the graphical user interface. The 3D cube is correctly orientated according to the Satellite-Sun vector previously calculated. A input angle parameter that defines in which orientation the 3D cube must be, can be set on the graphical user interface. From this input angle, the PWM is generated in function of the Satellite-Sun vector to orient the cube. When the Satellite-Sun vector is aligned on this Input Angle the PWM is stopped.

This test validates that the overall system works correctly together.

11 CONCLUSION

The principle objectives of this bachelor's thesis were:

- Review of the existing CDMS design
- Design and development of the electronics of the magnetic torquer
- Design and development of the electronics for the read-out of the sun sensors
- Integration of these electronics with the FPGA CDMS board already used during the semester project
- Propose a demonstrator to control the magnetic torquer from the sun sensors

Most of the primary objectives of this thesis were essentially achieved, though some points require additional work. Some remarks concerning the main achievements, difficulties and successes encountered follow.

An ADCS/CDMS board based on a FPGA has been designed and implemented. It is capable of calculating a Satellite-Sun vector based on the value retrieved from a Sun Sensor. It is also capable of controlling a Magnetotorquer. This board needs further development to reach its full potential but it shows some encouraging results.

The attitude determination algorithm has been implemented on the computer. This algorithm displays correct behavior when the expected values are entered manually but it still has to be improved to work directly with the values retrieved from the Sun Sensors.

The virtual version of the satellite was successfully implemented. It is correctly oriented by the Satellite-Sun vector, which is computed with the attitude determination algorithm.

The EPFL's request for the study of electronics capable of isolating the actuators and sensors of the ADCS board in case of its malfunction couldn't be achieved due to a lack of time. However a proposal on how the study would have to be conducted and how long it might take have been presented.

12 FUTURE ORIENTATION

To respond fully to the needs of the EPFL, the analysis and design of electronics capable of isolating the actuators and sensors of the ADCS board should be performed.

As the attitude determination algorithm doesn't work correctly, more development time should be spent to determine the exact reason for this malfunction and to correct it.

The next step for the ADCS/CDMS, once the attitude determination algorithm is fully operational, would be to translate it into assembler code to determine how many instructions are needed to perform the entire algorithm computation. This operation would help us estimate the time needed for the performing of this algorithm if it's deployed on a microcontroller of a given brand. The algorithm should also be integrated into the FPGA and tested. Once those two operations are done, a comparison of the time needed for both platforms to run this algorithm could be achieved. It would then be possible to deduce whether an FPGA could discharge the microprocessor of this task or if it's not useful.

13 BIBLIOGRAPHY

[1] Wikipedia, "CubeSat," 24 06 2013. [Online]. Available: http://en.wikipedia.org/wiki/CubeSat.

[2] H. Péter-Contesse, "ADCS System Engineer and ADCS Sytem and Hardware," Lausanne, 23.06.2007.

[3] L. Hauser, "SwissCube ADCS Hardware and Actuators," Lausanne, 2007.

[4] P. Thévoz, "Characterization and calibration of dual axis sun sensors for ADCS," Lausanne, 2008.

[5] L. Hauser, "SwissCube ADCS Harware an Actuators," Lausanne, 2008.

[6] Texas-Instruments, "datasheet ADS8028," may 2011 - revised march 2012.

[7] F. Halgand, "Intégtation d'OpenGL dans une interface Qt," 12 septembre 2010. [Online]. Available: http://fhalgand.developpez.com/tutoriels/qt/integration-opengl/01-premiere-application/.

Date **Signature**

14 APPENDIX

APPENDIX 1 :

Semester project

Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

Filière Systèmes industriels

Infotronics

Projet de semestre 2013

Stéphane Lovejoy

Satellite navigation with GPS

Professeur Christophe Bianchi

Sion, le 27 avril 2013

HES-SO Valais-Wallis • rte du Rawyl 47 • C.P. • 1950 Sion 2
+41 27 606 85 11 • info@hevs.ch • www.hevs.ch

TABLE DES MATIERES

1 Contexte

Actif depuis de nombreuses années dans le domaine de l'aérospatiale, le département Systèmes Industriels de la HES-SO Valais collabore avec l' EPFL sur plusieurs missions spatiales. Afin de diminuer les coûts de la conception de l'électronique embarquée, qui occupe une grande partie du budget alloué au développement, un projet d'approfondissement du Master (SatProc) a été réalisé. Il devait mettre en évidence les besoins communs de plusieurs missions spatiales et développer une architecture matérielle polyvalente. Un second travail de Master réalisé par Bastien Praplan a permis d'élaborer une carte de développement utilisable pour plusieurs missions.

2 Introduction

Une carte prototypage de commande de microsatellite a été développée dans le cadre d'un projet de master. Cette carte, composée d'une FPGA, peut être connectée à plusieurs circuits GPS afin de tester leur utilité dans le contrôle d'orientation d'un satellite. Cette carte contient aussi de la mémoire pour le stockage d'information liée à la mission et des liens de communication permettant de dialoguer avec les sous-systèmes. Cette carte n'a cependant pas pu être montée. Le but de ce projet de semestre est donc de valider le bon fonctionnement de cette carte.

Le but de cette carte de démonstration est d'avoir une architecture de base qui permet une grande flexibilité dans le nombre et les types de périphériques qu'elle implémente. Cette base permettra de déterminer non seulement l'architecture, mais également les caractéristiques matérielles utiles à de futures missions.

3 Cahier des charges

Le but de ce projet est de poursuivre le développement et la mise en œuvre de cette carte de démonstration afin de réaliser une première application dans le cadre du projet CubETH.

Cahier des charges initiales.

- Mettre en œuvre la communication avec un GPS
- Transmettre les données vers un PC via un lien série
- Représenter le trajet effectué pendant la période d'acquisition de position sur l'écran du PC
- Définir, puis mettre en œuvre une technique de stockage des données sur la carte
- Documenter les développements et tests réalisés

Cependant, comme la carte de démonstration n'est pas encore fonctionnelle, le cahier des charges a du être modifié en conséquence.

Cahier des charges pour le travail de semestre :

- Monter la carte de démonstration
- Réalisé les tests nécessaires a sa validation au niveau électrique.
- Etablir une communication entre le convertisseur A/D et le module UART
- Documenter les développements et tests réalisés

4 Hardware

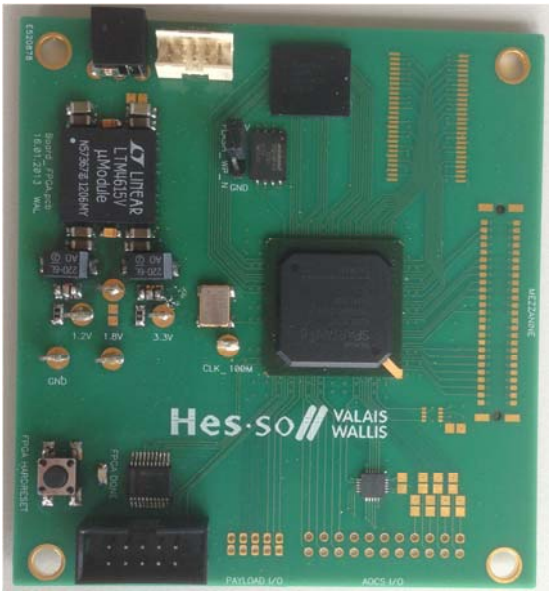


Figure 1 : Carte de prototypage

L'architecture de la carte Prototypage se présente sous la forme suivant:

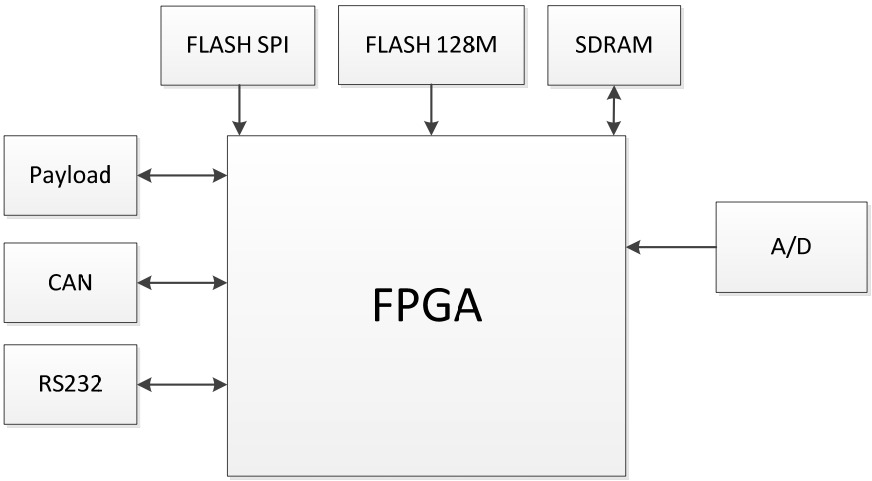


Figure 2 : architecture carte FPGA

La carte a pour élément de contrôle une FPGA. Celle-ci permet de piloter les périphériques de communication ainsi que le convertisseur A/D. Elle implémente également sa propre FLASH ainsi qu'une seconde FLASH et une RAM.

4.1 Eléments de la carte :

Les quelques points suivants offrent une brève description des différents composants de la carte FPGA utilisée dans ce rapport.

4.1.1 FPGA

Pour la carte de démonstration, une FPGA de grande taille a été choisie. Ce choix d'utiliser une telle FPGA a été fait pour différentes raisons.

La première étant, que sa taille ne devait pas être un facteur limitant sur le nombre de périphériques implémentés sur la carte. C'est pour cette raison que le choix s'est porté sur la plus grande FPGA de la série Spartan 6, la `XC6SLX150` de Xilinx bien qu'elle ne soit pas compatible spatial.

La FPGA est cadencée par un oscillateur externe de 100 MHz

4.1.2 A/D

Model : `ADS8028` de Texas-instruments

Le convertisseur A/D utilisé sur cette carte est un convertisseur de 12-bit possédant 8 canaux de conversions. Il est accessible à l'aide du bus de données SPI.

4.1.3 RS232

Le tranceiver `MAX3224` permet d'adapter les niveaux de tension, afin qu'une communication RS-232 puisse être établie entre la FPGA et un UART.

4.1.4 FLASH SPI

Model : `W25Q128FV` de winbond

Cette flash série a une capacité de 128Mbits et est capable de fonctionner jusqu'à 104MHz.

4.2 Test de la carte :

La carte FPGA a été préalablement testée avant son montage afin de vérifier son routage. Ces tests se trouvent dans l'annexe 1. Les tests effectués, démontrent que le routage est correctement réalisé. Il est cependant vérifié à l'aide de la schématique. Si ce dernier est faux, le test ci-dessus le sera également.

Ce premier test prouve que la carte correspond bien à sa schématique et, qu'aucune erreur de routage n'a été faite. Cependant si ce dernier s'avère faux, la carte sera tout de même non fonctionnelle.

Une fois la carte testée, elle a été partiellement montée suivant le rapport de mesure défini à l'annexe 2. La carte n'est pas entièrement montée pour limiter les sources d'erreurs durant la phase de validation.

Ce second test démontre que les éléments montés sur la carte FPGA, ont été correctement soudés et qu'ils sont correctement alimentés. Comme la consommation de courant est acceptable, la supposition est faite : qu'il n'y a aucun cour-circuit. Il démontre également que les composants montés sont correctement alimentés.

La liste des composants suivants montrent lesquels sont montés sur la carte avant de commencer les tests software :

- FLASH SPI
- FLASH 128M
- FPGA
- MAXs 3051 et 3224
- Convertisseur AD

Un liste de commande complètent de tous les composants nécessaire à la réalisation de cette carte ce trouve à l'annexe 3

5 Software

5.1 Environment de travail:

Outils de développement :

- HDL Designer version 2009.2 (Build 10)
- Xilinx ise version 14.4
- iMPACT version 14.4

5.2 Organigramme des projets test1 et test_com_AD_UART

Ce chapitre montre l'organisation des fichiers des projets test1 et test_com_ad_UART. Ces deux organigrammes montrent également l'emplacement des différents fichiers utilisés dans l'intégralité du chapitre 5

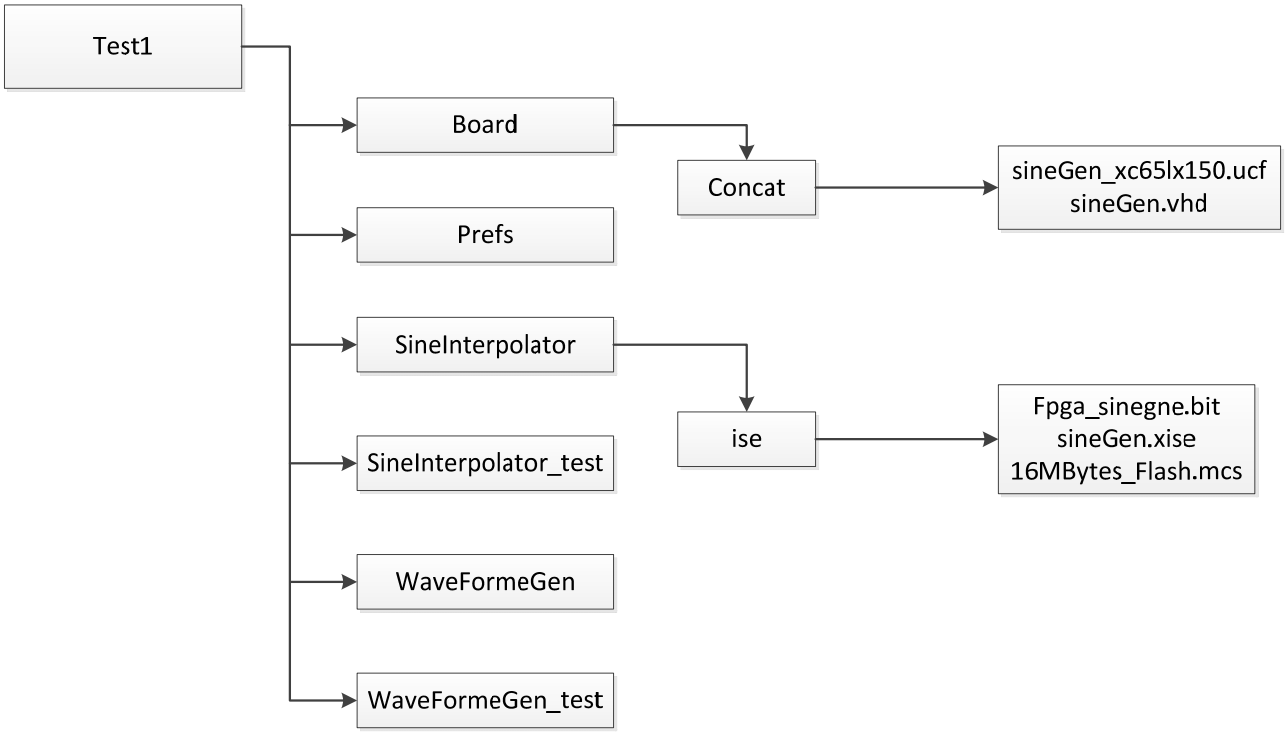


Figure 3 : Projet test1

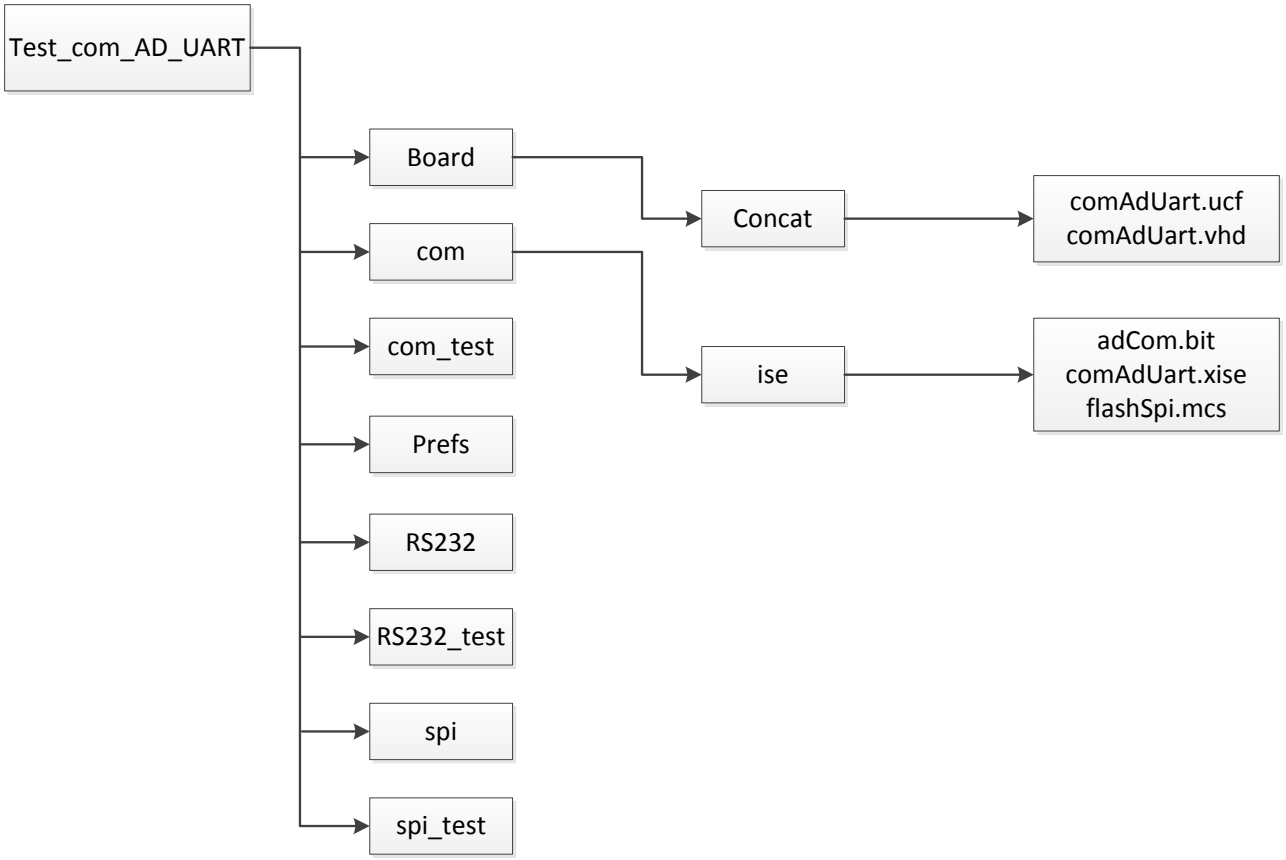


Figure 4 : Projet test_com_AD_UART

5.3 Architecture AMBA

La carte étant une unité de démonstration, le code se doit non seulement d'être capable de communiquer avec tous les périphériques implémentés sur celle-ci, mais également d'être extrêmement flexible. Cette flexibilité est demandée vis-à-vis du nombre de périphériques qui pourraient être rajoutés ou enlevés dans le cadre d'une quelconque future mission. Pour ces raisons, une architecture basée sur le bus AMBA a été choisie.

Cette architecture offre les avantages suivants :

1. Aucune limitation quant au nombre de périphériques que contient le bus.
2. Un périphérique implémenté peut être aisément dupliqué.

D'autres bus similaires existent. Par exemple, le bus Avalon ou le bus Wishbone chacun d'entre eux offrants différents avantages. Un projet précédent a déjà été réalisé à l'aide du bus AMBA et pourra donc être en partie réutilisé, ce qui réduit grandement le temps de développement.

5.4 Validation de la carte FPGA :

Avant d'arriver à cette architecture finale, une plus simple est implémentée. Cette dernière est utilisée dans le but de gagner du temps dans la phase de validation de son fonctionnement. L'architecture AMBA étant plus longue à mettre en place et plus complexe, elle sera donc utilisée une fois cette phase de validation terminée.

Figure 5 : Architecture simple

Avec cette architecture, aucun bus n'est utilisé. La FPGA est directement connectée au RS 232 ainsi qu'au convertisseur AD.

5.4.1 Validation de la FPGA et de la FLASH SPI:

Dans un premier temps, il est indispensable de valider le bon fonctionnement de la FPGA ainsi que celui du tool chain utile à la programmation de ladite carte. Dans le but de gagner du temps, mais également celui d'écarter le code de la liste des erreurs potentielles, un code VHDL déjà existant et validé est utilisé.

Le code utilisé pour ce test est celui conçu durant le cours SEm.

Information relative au projet :

- Emplacement : `test1`
- Projet : `test1\Prefs\hds-lissajous`
- Composant compilé : `FPGA_sineGen`

Ce circuit combine deux sinus à des fréquences différentes, pour permettre de dessiner les courbes ci-dessous à l'aide du mode XY de l'oscilloscope.

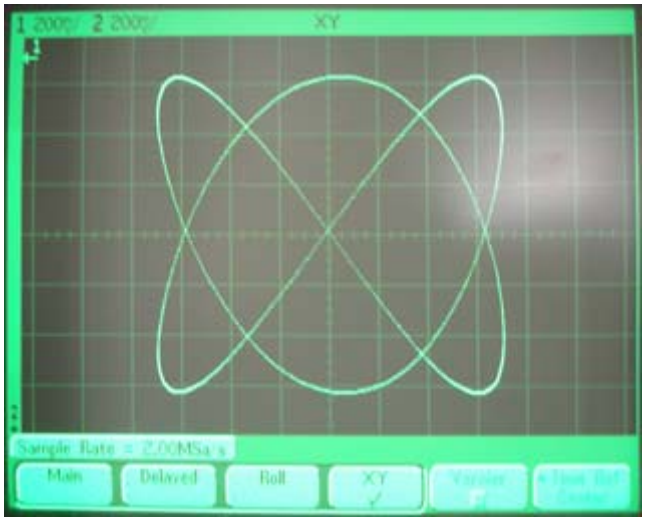


Figure 6 : Courbe de Lissajous [1]

Réalisation du circuit sur HDL Designer :

Le schéma utilisé est le suivant :

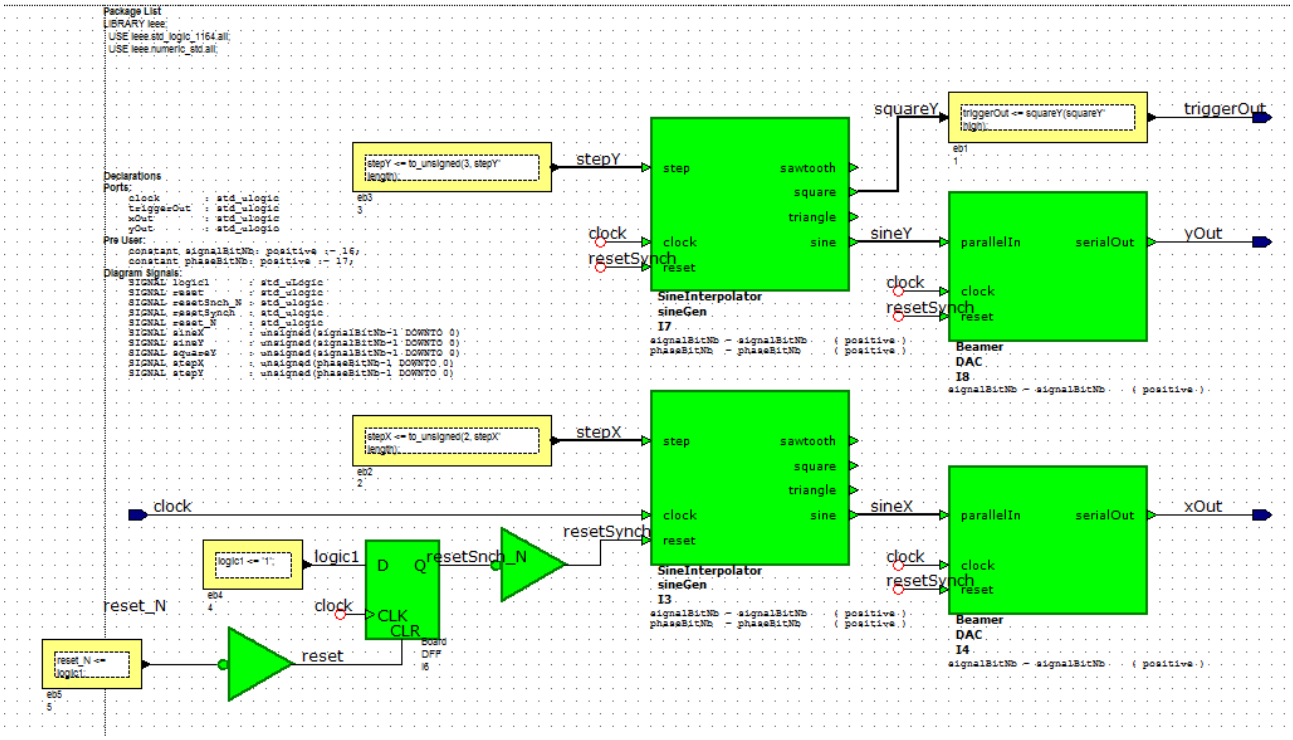


Figure 7 : schéma sinGen

Les deux sorties xOut et yOut permettent de sortir deux sinus de deux fréquences différentes. La sortie triggerOut, signal carré, fournit un signal de synchronisation pour l'oscilloscope.

Une fois ce schéma simulé, il permet d'obtenir le fichier sineGen.vhd.

Localisation de sineGen.vhd :

`test1\Board\concat`

Configuration de la FPGA avec Xilinx Project Navigator :

A l'aide des fichiers `sineGen.vhd` et `sineGen_xc6slx150.ucf` xilinx synthétise le code pour en sortir un fichier `fpga_sinegen.bit`

Ce fichier est ensuite utilisé pour programmer la FPGA.

Localisation des fichiers :

- sineGen.vhd :

  `test1\Board\concat`

- sineGen_xc6slx150.ucf :

  `test1\Board\concat`

- fpga_sinegen.bit :

  `test1\Board\ise`

Programmation de la FPGA avec iMPACT :

A l'aide du fichier `fpga_sinegen.bit` , iMPACT permet de programmer la FPGA.

Impacte a également permis de programmer la FLASH SPI. Pour ce faire, le fichier `16MBytes_Flash.mcs` a été utilisé.

Localisation de 16MBytes_Flash.mcs :

`test1\Board\ise`

Résultat :

Une fois l'architecture implémentée sur la FPGA, les résultats suivants sont obtenus en utilisant la fonction XY de l'oscilloscope avec le signal carré, utilisé pour synchroniser l'oscilloscope.

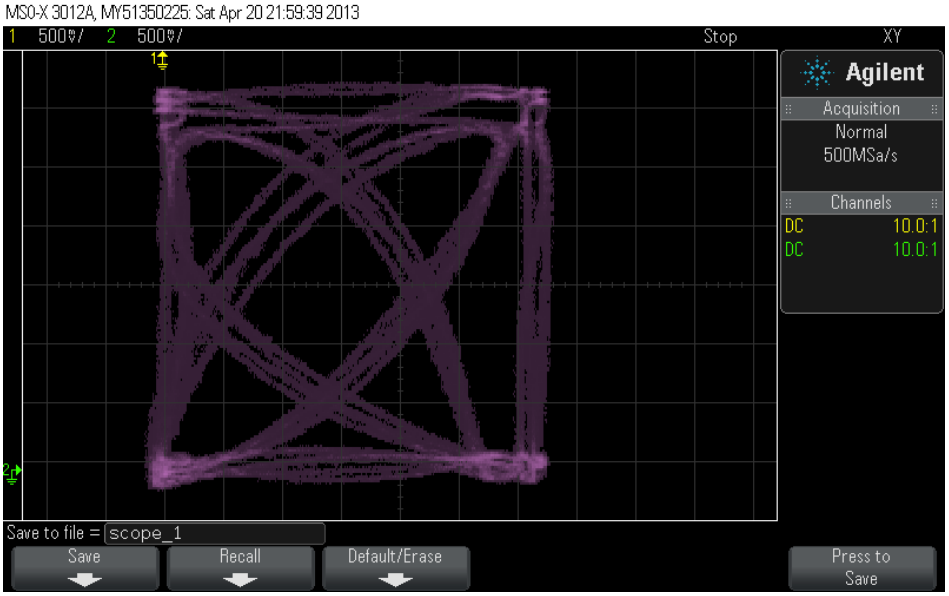


Figure 8 : Résultat du projet test1

Ces résultats permettent de valider le bon fonctionnement de la FPGA et du tool chain `hdl designer-xilinx-iMPACT`

L'alimentation de la carte FPGA a été coupée. Une fois l'alimentation rétablie, le code est correctement chargé sur la FPGA depuis la FLAH SPI ce qui confirme le bon fonctionnement de cette dernière.

5.4.2 Validation du convertisseur AD

Généralités :

Afin de valider le fonctionnement du convertisseur AD `ADS8028`, un nouveau projet a été créé. Il possède les caractéristiques suivantes :

- Emplacement : `test_com_AD_UART`
- Projet : `test_com_AD_UART\Prefs\hds-adCom`

Pour établir une communication SPI avec le convertisseur AD, les librairies SPI et SPI_test de l'école sont utilisées. Ces dernières sont employées afin de gagner du temps dans la phase de test. Ces librairies fournies par l'école assurent également l'utilisation de block VHDL, dont le comportement a été préalablement testé et validé.

Afin de ne pas détruire le convertisseur AD, la simulation d'une trame d'écriture sur l'AD a été préalablement effectuée.

L'architecture suivante a été utilisée pour cette simulation :

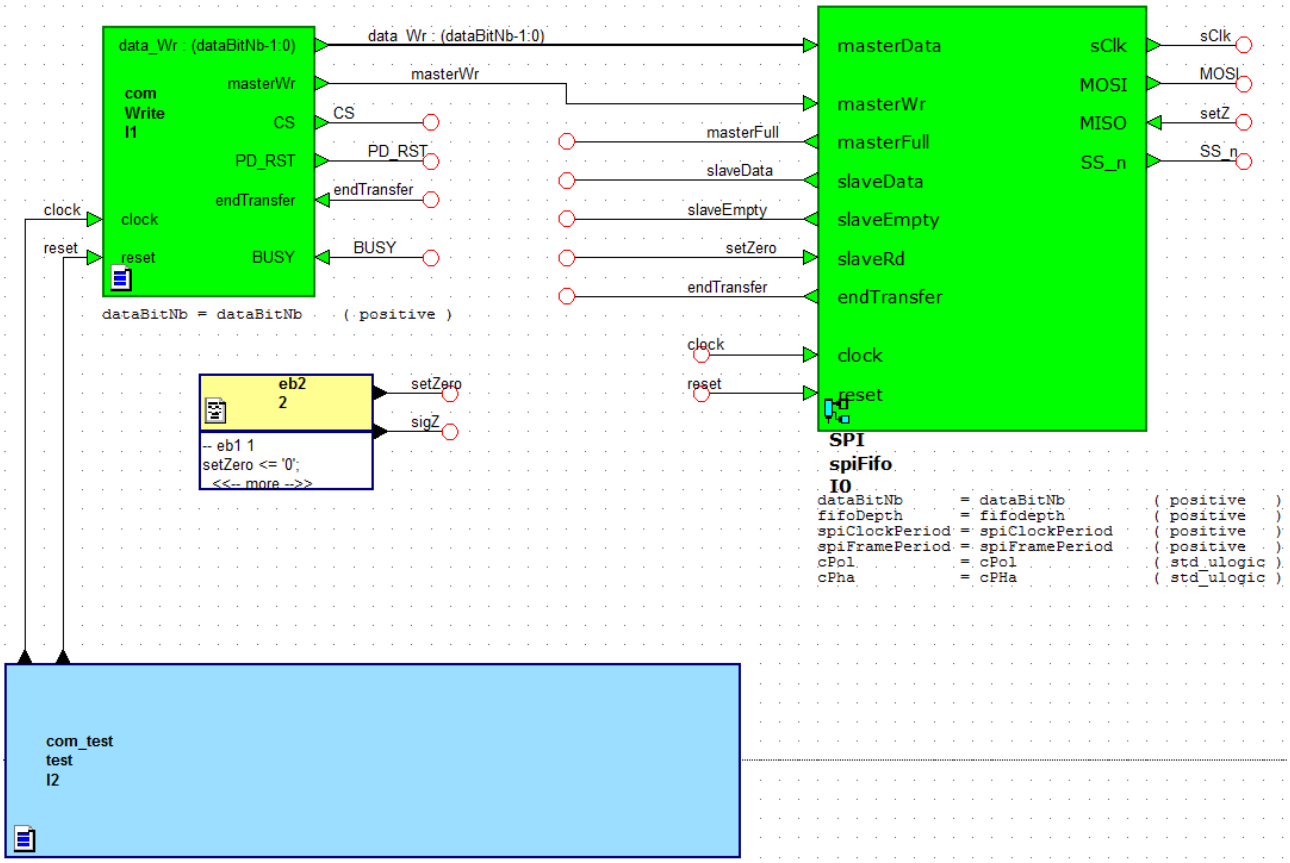


Figure 9 : Architecture com_test

Dans l'architecture ci-dessus, l'écriture de la trame SPI est effectuée par le block `com Write`. Ce dernier, envoie la trame désirée au block `spiFifo` (block spiFifo fourni par l'école `test_com_AD_UART\spi`). Le block bleu `test` permet uniquement de simuler le clock des 100MHz utilisé sur la carte FPGA ainsi qu'un signal de reset.

Le signal sClk qui permet de cadencer l'échange de trame entre l'FPGA et l'AD, est généré par spiFIFO.

Le paramétrage de la communication SPI ainsi que la mise en forme de la trame désirée, sont expliqués dans les chapitres suivants.

Paramétrage de la communication SPI :

L'image ci-dessous contient les valeurs des différents paramètres utilisés ainsi que le type des signaux employés pour cette simulation :

Declarations
Ports:
Pre User:

```
--constant cPol: std_ulogic := '0';
constant cPHa: std_ulogic := '0';
constant cPol: std_ulogic := '1';
--constant cPHa: std_ulogic := '1';

constant dataBitNb: positive := 16;
constant fifoDepth: positive := 16;
constant spiClockPeriod: positive := 10;
constant spiFramePeriod: positive := 1;
```

Diagram Signals:

```
SIGNAL clock        : std_ulogic
SIGNAL reset        : std_ulogic
SIGNAL data_Wr      : std_ulogic_vector(dataBitNb-1 DOWNTO 0)
SIGNAL masterWr     : std_ulogic
SIGNAL sClk         : std_ulogic
SIGNAL MOSI         : std_ulogic
SIGNAL SS_n         : std_ulogic
SIGNAL slaveData    : std_ulogic_vector(dataBitNb-1 DOWNTO 0)
SIGNAL endTransfer  : std_logic
SIGNAL CS           : std_ulogic
SIGNAL masterFull   : std_ulogic
SIGNAL setZero      : std_logic
SIGNAL setZ         : std_logic
SIGNAL sigZ         : std_logic
SIGNAL slaveEmpty   : std_ulogic
```

Figure 10 : paramètre et type des signaux de la simulation com_test

On constate que la communication SPI est configurée avec `CPHA` = 0 et `CPOL` = 1. Et que la trame comporte 16 bits, donc dataBitNb est initialisé à 16. En mettant la constant `spiClockPeriod` à 10, la fréquence du `sCLk` est définie à 10 MHz.

Ces paramètres sont initialisés comme tels, pour satisfaire à la configuration SPI utilisée par le convertisseur AD illustré ci-dessous.

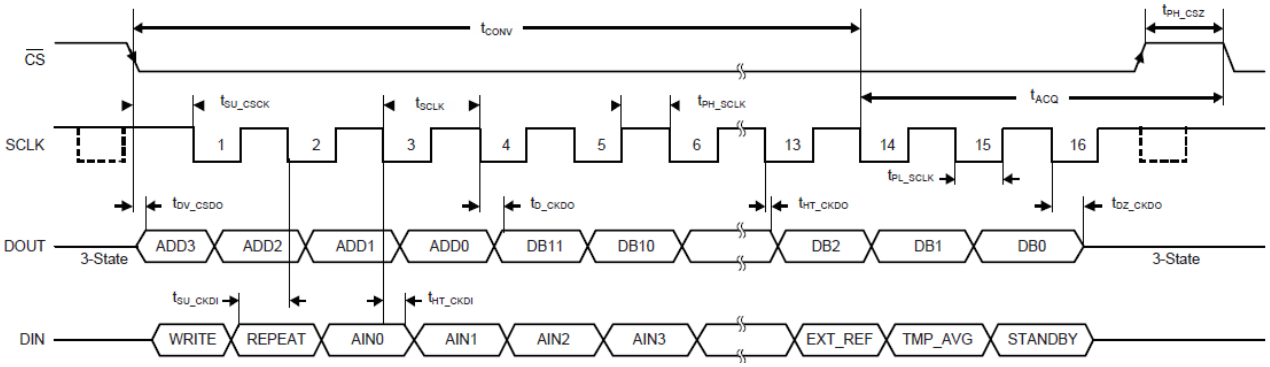


Figure 11 : Timing diagramme du convertisseur AD [2]

PARAMETER	DESCRIPTION	TEST CONDITIONS	ADS8028 MIN	ADS8028 TYP	ADS8028 MAX	UNIT
f_{SCLK}	External serial clock frequency				20	MHz
t_{SCLK}	External interface clock time period		50			ns
t_{PH_SCLK}	SCLK high pulse width		0.4 t_{SCLK}		0.6 t_{SCLK}	ns
t_{PL_SCLK}	SCLK low pulse width		0.4 t_{SCLK}		0.6 t_{SCLK}	ns
t_{CONV}	Conversion time:				t_{SU_CSCK} + 13 t_{SCLK}	ns
	For channels AIN0 to AIN7	f_{SCLK} = 20 MHz			700	ns
	For internal temperature sensor measurement				100	µs
t_{PH_CSZ}	$\overline{CS}$ high pulse width		6			ns
t_{ACQ}	Acquisition time (for channel AINx)	f_{SCLK} = 20 MHz	100			ns
t_{SU_CSCK}	Setup time: $\overline{CS}$ to SCLK falling edge		10			ns
t_{DV_CSDO}	Delay time between $\overline{CS}$ falling edge to DOUT enabled				17	ns
t_{D_CKDO}	Delay time between SCLK falling edge to (new) data available on DOUT	DVDD = 1.65 V to 3 V			32	ns
		DVDD = 3 V to 3.6 V			29	ns
		DVDD = 3.6 V to 5.25 V			28	ns
t_{HT_CKDO}	Hold time: SCLK falling edge to (previous) data valid on DOUT		10			ns
t_{DZ_CKDO}	Delay time between 16th SCLK falling edge to DOUT going to high-impedance		12		27	ns
t_{DZ_CSDO}	Delay time between $\overline{CS}$ going high to DOUT going to high-impedance				26	ns
t_{SU_CKDI}	DIN setup time before SCLK falling edge		5			ns
t_{HT_CKDI}	DIN hold time after SCLK falling edge		4			ns
t_{ACQ_TMP}	TM_BUSY falling edge to $\overline{CS}$ falling edge		100			ns
$t_{PU\_STANDBY}$	Power-up time after coming out of STANDBY mode				1	µs
t_{POWER_UP}	Power-up time after coming out of power-down mode	Internal reference mode, 10-µF capacitor on REF pin			6	ms

Figure 12 : Timing a respecter pour la figure 11 [2]

- CPOL est initialisé à 1 afin que la valeur de base du clock SCLK soit de 1.
- CPHA est quant à lui initialisé à 0 dans le but de capturer la valeur des données au flanc descendant de SCLK comme spécifié dans le diagramme ci-dessus figure 11.
- Pour finir, la fréquence de SCLK est définie à 10 MHz afin d'être inférieure à sa fréquence maximale de 20 MHz spécifiée dans la figure 12.

Trame d'écriture sur le convertisseur:

Afin de configurer le convertisseur AD, une trame d'écriture doit être envoyée au convertisseur. Cette dernière permet de configurer le registre de contrôle du convertisseur. Cette trame définit sur quel(s) port(s) la conversion AD sera effectuée.
Elle permet également de définir en quel mode d'opération fonctionnera le convertisseur.
L'utilisation d'une référence interne ou externe pour la conversion peut être paramétrée à l'aide de ce registre.
Une dernière fonctionnalité peut être paramétrée à l'aide de cette trame. Il s'agit de l'utilisation d'un capteur de température interne au convertisseur.

Dans le cadre de la phase de validation, l'utilisation du capteur de température n'est pas utile. Le mode de fonctionnement par défaut est employé car simple, il convient parfaitement à la validation du convertisseur.

La figure ci-dessous montre la configuration du registre de contrôle de l'AD :

MSB 15	14	13	12	11	10	9	8
WRITE	REPEAT	AIN0	AIN1	AIN2	AIN3	AIN4	AIN5
							LSB
7	6	5	4	3	2	1	0
AIN6	AIN7	T_{SENSE}	X	X	EXT_REF	TMP_AVG	STANDBY

Bit 15 — **WRITE: Write to Control Register**
Enable write operation.
0 = Write disabled; Control Register is not updated and the next 15 bits are ignored (default)
1 = Write enabled; the next 15 bits update the Control Register

Bit 14 — **REPEAT: Repeat conversion mode**
Enable conversion repeat mode (refer to the *Modes of Operation* section).
0 = Disable repeat conversion mode (default)
1 = Enable repeat conversion mode

Bits[13:6] — **AIN[0:7]: Analog input channel selection**
Each AINx bit corresponds to the associated analog input channel, AIN0 to AIN7.
0 = AINx channel is not selected for conversion (default)
1 = AINx channel is selected for conversion

Bit 5 — **T_{SENSE}: Internal temperature sensor selection**
Internal temperature sensor selection for conversion in subsequent cycles.
0 = Internal temperature sensor output is not selected for conversion (default)
1 = Internal temperature sensor output is selected for conversion

Bits[4:3] — **X: Don't care**

Bit 2 — **EXT_REF: Reference source selection**
This bit selects the reference source for the next conversion.
0 = Internal reference is used for the next conversion (default)
1 = External reference is used for the next conversion

Bit 1 — **TMP_AVG: Temperature sensor averaging selection**
This bit selects the mode of operation for the temperature sensor channel; this bit is ignored if bit 5 is set to '0'.
0 = Averaging is disabled on the temperature sensor result (default)
1 = Averaging is enabled on the temperature sensor result

Bit 0 — **STANDBY: STANDBY mode selection**
This bit sets the mode (normal or standby) for the ADS8028.
0 = The ADS8028 operates in normal mode (default)
1 = The ADS8028 goes to standby mode in the next cycle

Figure 13 : Registre de contrôle [2]

La configuration suivante est choisie :

MSB

15	14	13	12	11	10	9	8
1	0	0	1	0	0	0	0

LSB

7	6	5	4	3	2	1	0
0	0	0	X	X	0	0	0

Figure 14 : Configuration du registre de contrôle

On constate que le mode de fonctionnement par défaut sera sélectionné. Ce mode permettra d'effectuer une unique conversion. La conversion s'effectuera sur `AIN1` et la référence interne au convertisseur sera utilisée. Comme mentionné précédemment, le capteur de température ne sera pas utilisé pour effectuer la conversion.

Echange de trame entre FPGA et AD:

Ce chapitre illustre l'échange de trames entre le convertisseur et la FPGA.

La FPGA est le maitre de la communication SPI, c'est donc elle qui fournit le signal de cadencements `sCLk`. La valeur d'un bit est déterminée sur le flanc descendant de `sCLk`. La FPGA pilote également le chip select `CS`.

Le chronogramme suivant démontre l'écriture et la lecture sur l'AD :

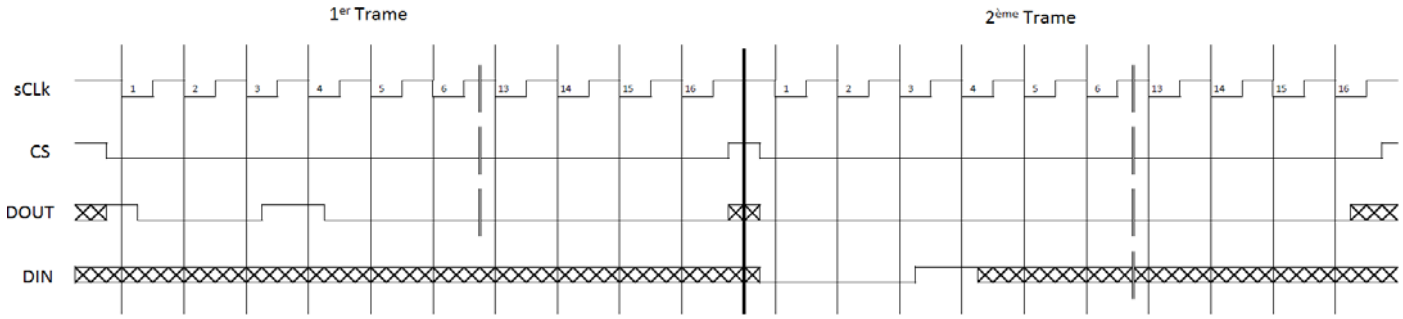


Figure 15 : Chronogramme d'écriture/lecture sur l'AD

Le chronogramme ci-dessus se sépare en deux parties bien distinctes. La première, consiste à écrire sur le registre de contrôle du convertisseur AD. La seconde consiste à lire les données depuis ce dernier.

1ère partie, Trame d'écriture:

Pour qu'une trame d'écriture soit validée, 16 `sCLk` doivent avoir eu lieu entre le moment ou `CS` est mis à l'état bas et le moment où il est remis à l'état haut. La trame est envoyée en big-endian sur la pin DIN du convertisseur.

Déroulement temporel de la trame d'écriture

- Le `CS` est mis à l'état bas.
- Le premier bit de la trame est mis à 1 afin de sélectionner le mode d'écriture.
- Le quatrième bit est mis à 1 pour définir une conversion AD sur le port N°1 `ANI1`
- Les bits suivant son mis à l'état bas afin de ne pas activer le capteur de température, d'utiliser la référence interne pour la conversion et d'opérer en mode normale.

2ème partie, lecture de la conversion :

Une fois le registre de contrôle configuré, les 16 prochains bits correspondent à la réponse du convertisseur. Cette réponse est lue en commençant par le MSB et elle est lue sur la pin `DOUT`.

- Les 4 premiers bits conformément à la figure 16 donnent sur quel chanel a été réalisée la conversion.
- Les 12 derniers bits donnent le résultat de la conversion AD.

ADD3	ADD2	ADD1	ADD0	ANALOG INPUT CHANNEL
0	0	0	0	AIN0
0	0	0	1	AIN1
0	0	1	0	AIN2
0	0	1	1	AIN3
0	1	0	0	AIN4
0	1	0	1	AIN5
0	1	1	0	AIN6
0	1	1	1	AIN7
1	0	0	0	T_{SENSE} without averaging
1	0	0	1	T_{SENSE} with averaging

Figure 16 : Channel Address Bits [2]

<u>Simulation d'envoi d'une trame SPI d'écriture sur le registre de contrôle du convertisseur:</u>

L'exécution de la simulation avec les paramètres définis au chapitre `Trame d'écriture sur le convertisseur` donne les résultats ci-dessous :

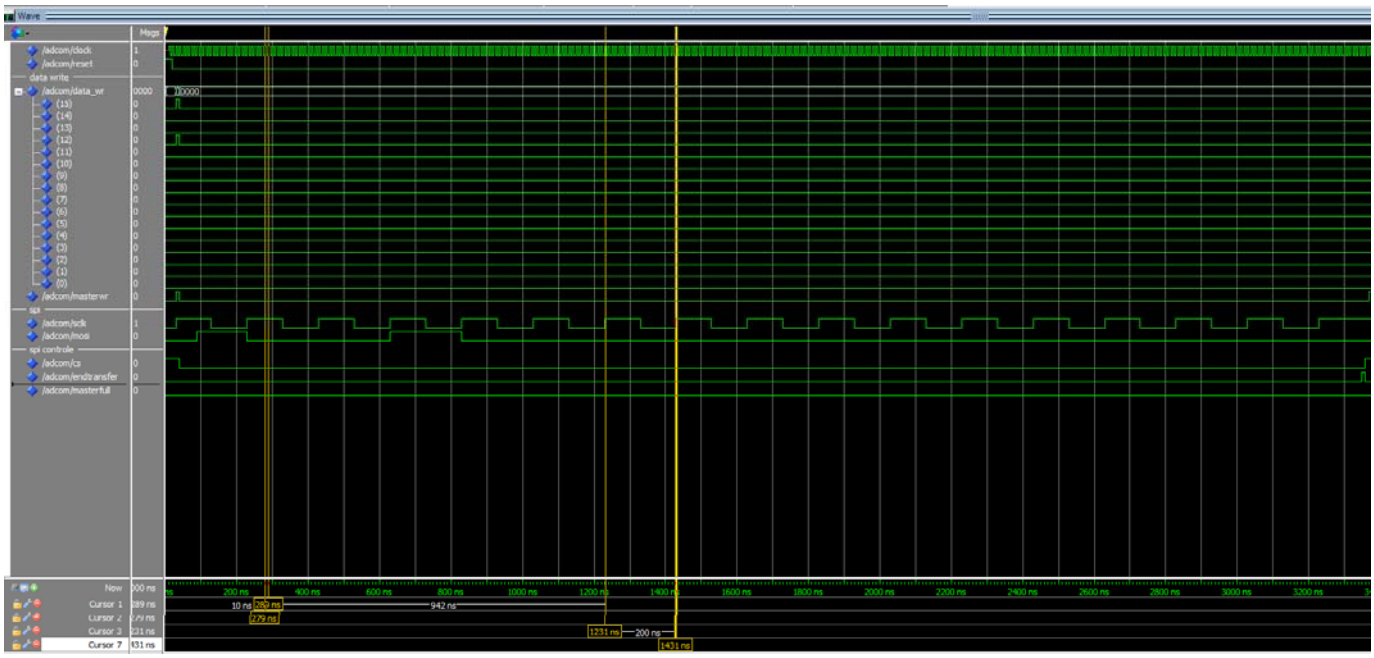


Figure 17 : Simulation d'une trame d'écriture sur l'AD

On constate que la trame est effectivement d'une longueur de 16 bits entre le moment où CS passe à zéro et l'instant où il remonte au niveau haut. L'ordre de ces derniers correspond à la configuration du registre de contrôle désiré. La différence entre les curseurs 1 et 2 est de 10ns, ce qui prouve que la fréquence de clock est bien de 100 MHz. 200 ns s'écoulent entre les curseurs 3 et 4. C'est 200 ns correspondent au 10MHz déterminé dans le chapitre précédent comme fréquence désirée pour SCLK.
Cette simulation confirme donc que com_test est correctement réalisé. Le code peu donc être utilisé afin de programmer une communication SPI entre la FPGA et le convertisseur.

Programmation sur la FPGA de la communication avec l'AD :

Une fois la trame d'écriture validée, elle est testée sur le convertisseur AD.
L'architecture suivante a été utilisée pour ce test:

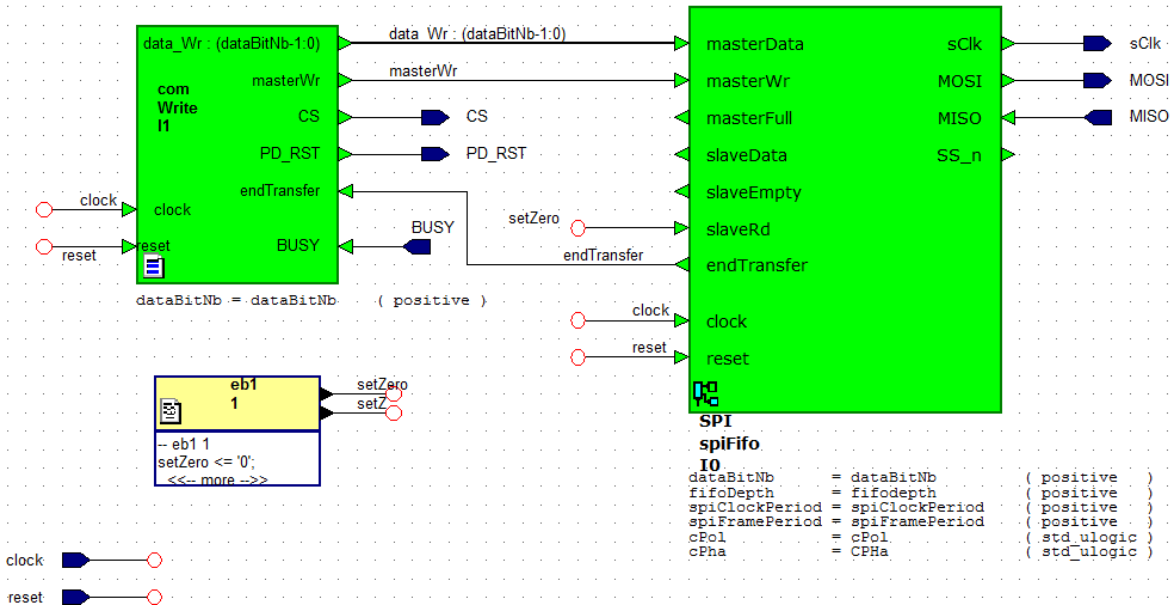


Figure 18 : Architecture com

Le schéma ci-dessus montre les ports qui sont utilisés en sortie de la FPGA et également ceux qui sont connectés à la FPGA. La figure suivant démontre sur quels pin du convertisseur AD ces derniers sont connectés.

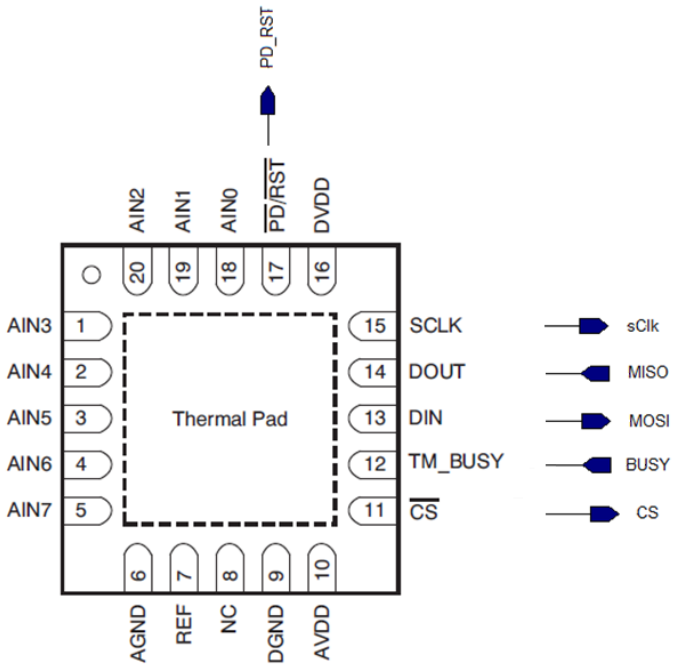


Figure 19 : Pining du convertisseur AD avec connexion avec la FPGA [2]

Une fois les ports de la FPGA correctement connectés au convertisseur AD, l'architecture de la figure19 est programmée dans cette dernière.

La trame d'écriture sur les registres du convertisseur est dans un premier temps envoyée. La figure ci-dessous montre ladite trame, mesurée à l'oscilloscope sur la pin DIN du convertisseur.

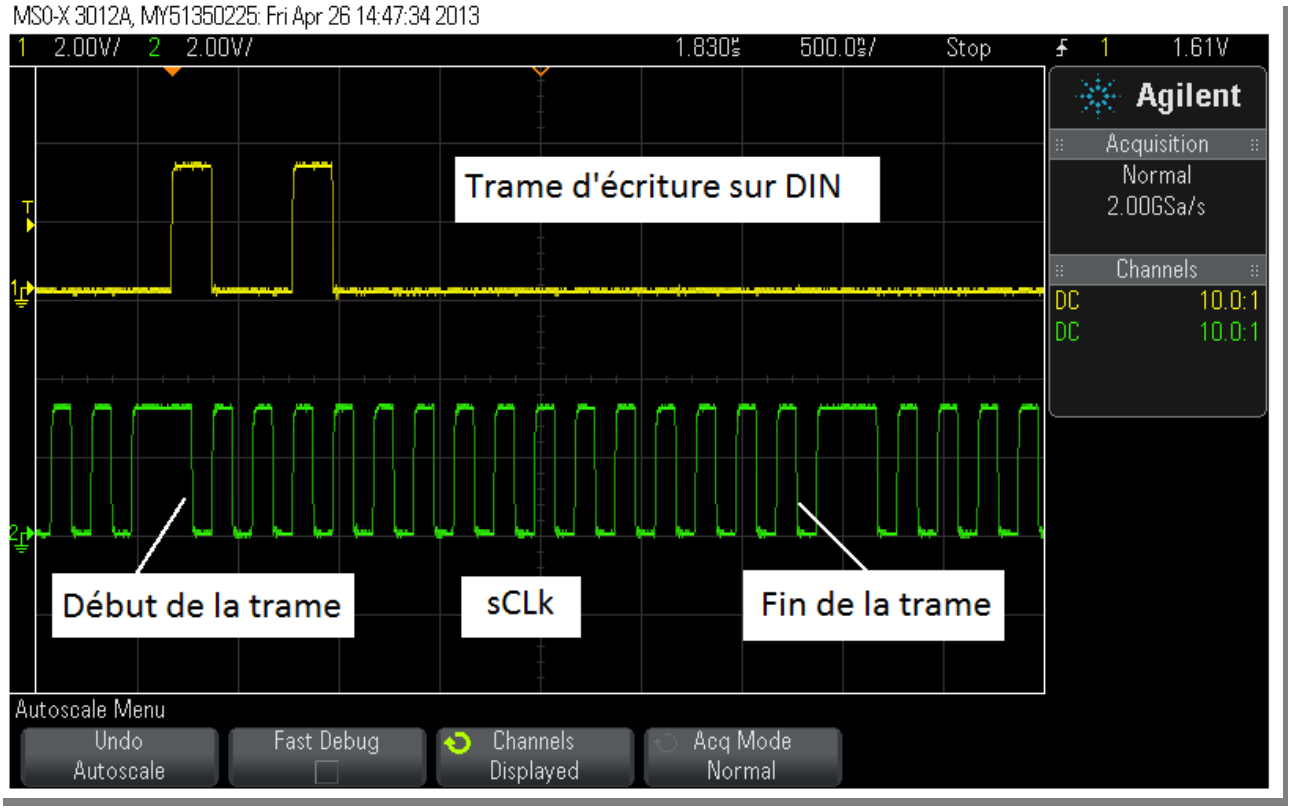


Figure 20 : Trame d'écriture sur le registre de contrôle de l'AD

On constate que la trame d'écriture a bien la configuration désirée. Le premier bit est bien à 1, ce qui implique que le mode d'écriture est activé. Le quatrième bit est également de niveau haut, ce qui signifie que la conversion sera effectuée sur le channel `ANI1`.

La capture d'oscilloscope suivant, montre la trame de réponse du convertisseur AD suite à la configuration de son registre de contrôle ci-dessus:

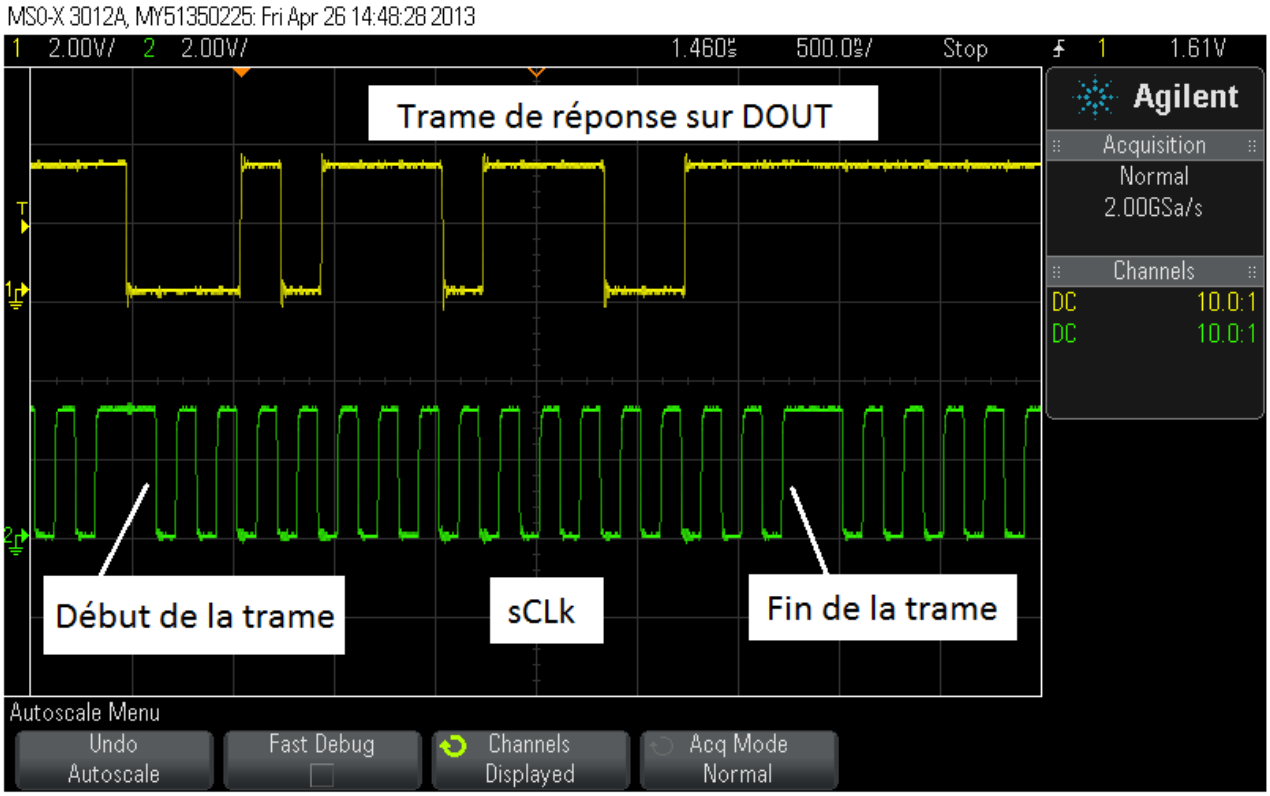


Figure 21 : Trame de réponse du convertisseur AD

On constate que le quatrième bit est comme attendu a 1. Selon la figure 16, la conversion est bien effectuée sur la channel `ANI1`. Les 12 bits suivants correspondent au résultat de cette conversion. La valeur de cette conversion n'est pas contrôlée, car le but de ce test est d'uniquement constater un échange de trame entre l'AD et la FPGA.

Les résultats obtenus ne valident pas totalement le bon fonctionnement du convertisseur AD. Cependant une communication est établie entre ce dernier et la FPGA. La conversion est bien effectuée sur le channel désiré et ce résultat est retransmis. Pour le valider complétement, des tests supplémentaires sont nécessaires. La justesse de la valeur de conversion retournée doit être vérifiée, différents modes de fonctionnement doivent être testés ainsi que l'emploie du capteur de température interne.

6 Conclusion

La validation hardware de la carte de démonstration est terminée pour tous les composants montés. Tous les différents niveaux d'alimentations ont été validés. Un rapport de mesure a été établi.

Les outils de développement nécessaires à l'élaboration de ce projet ont pu être installés. La FPGA et la FLASH SPI peuvent être programmés selon les besoins. La communication entre le convertisseur AD et la FPGA est fonctionnelle.

7 Orientation future

Pour la suite de ce projet, il est dans un premier temps nécessaire de terminer la validation de la carte. Le transceiver RS232, la SDRAM ainsi que la FLASH de 128 M doivent être testées. La fin de la validation du convertisseur AD doit également être achevée.

Une fois la validation de la carte terminée, le bus AMBA doit être implémenté. Les différents périphériques doivent également être adaptés pour fonctionner avec ce bus. Il serait également important de créer l'architecture la plus souple possible afin de pouvoir aisément dupliquer le nombre de périphériques sur le bus.

Une fois les étapes précédentes réalisées, le cailler des charges initiale pourra être réalisé.

8 Bibliographie

[1] François Corthay, Synthèse automatique,
`R:\Modules\SI\225_Sem\Digital\Laboratoires\04 synthese.html`

[2] Texas Instruments, datasheet ADS8028, may 2011- revised march 2012

9 Annexes

Annexe 1 : test hardware

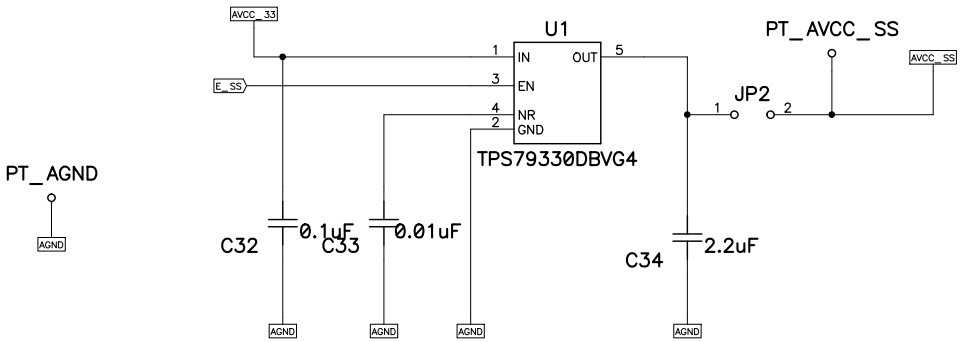
Annexe 2 : rapport de mesure

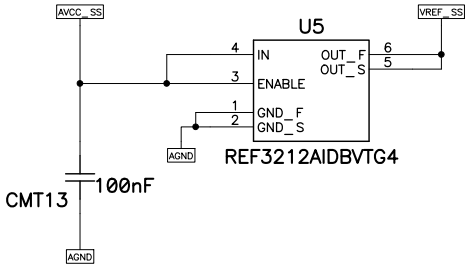
Annexe 3 : liste de commande

APPENDIX 2 :

Schematics of the interface board

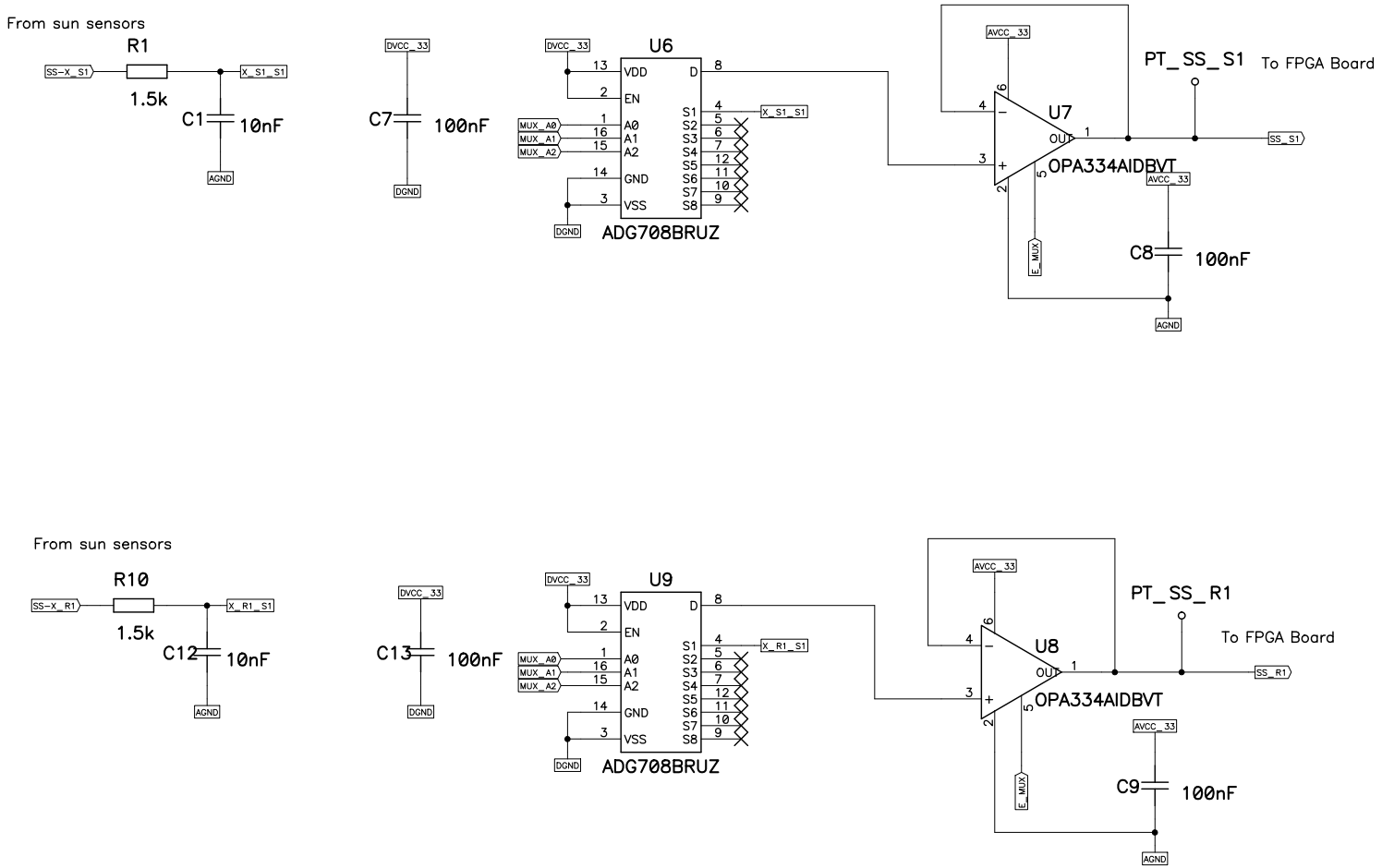
Alimentation for Sun Sensor





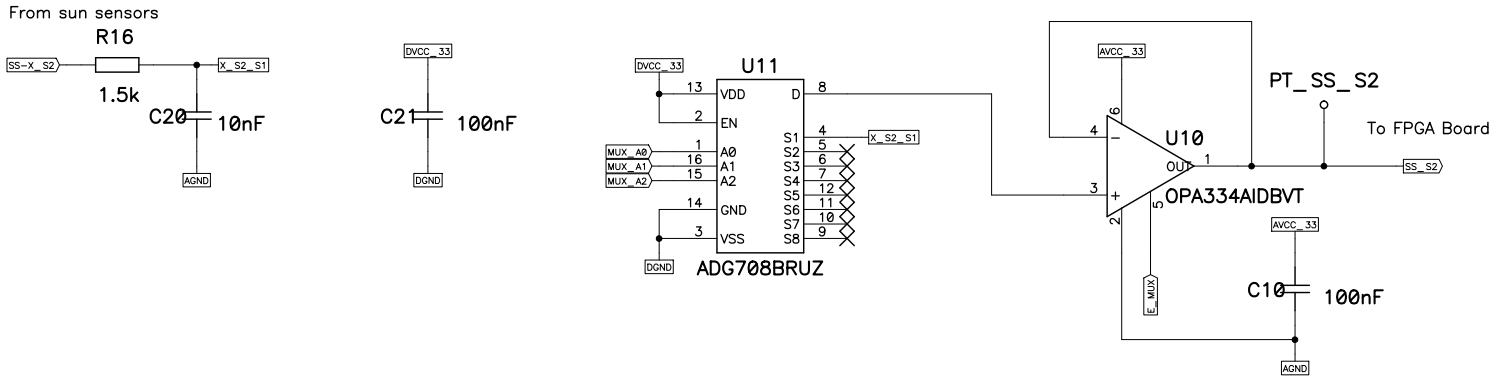
CubETH CDMS Interface FPGA to Sun sens/MagTorq Sun Sens alim	DES	{Date} lovs
	REV	V1.0
HAUTE ECOLE VALAISANNE	1/6	{Path} Interface_FPGA_sunSens_magTorq.sch

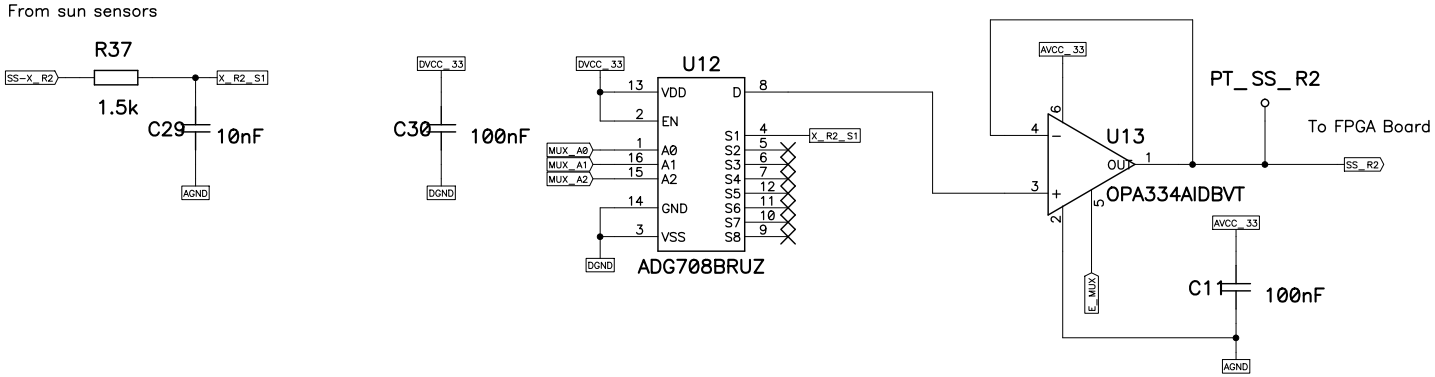
Sun Sensors electronics 1



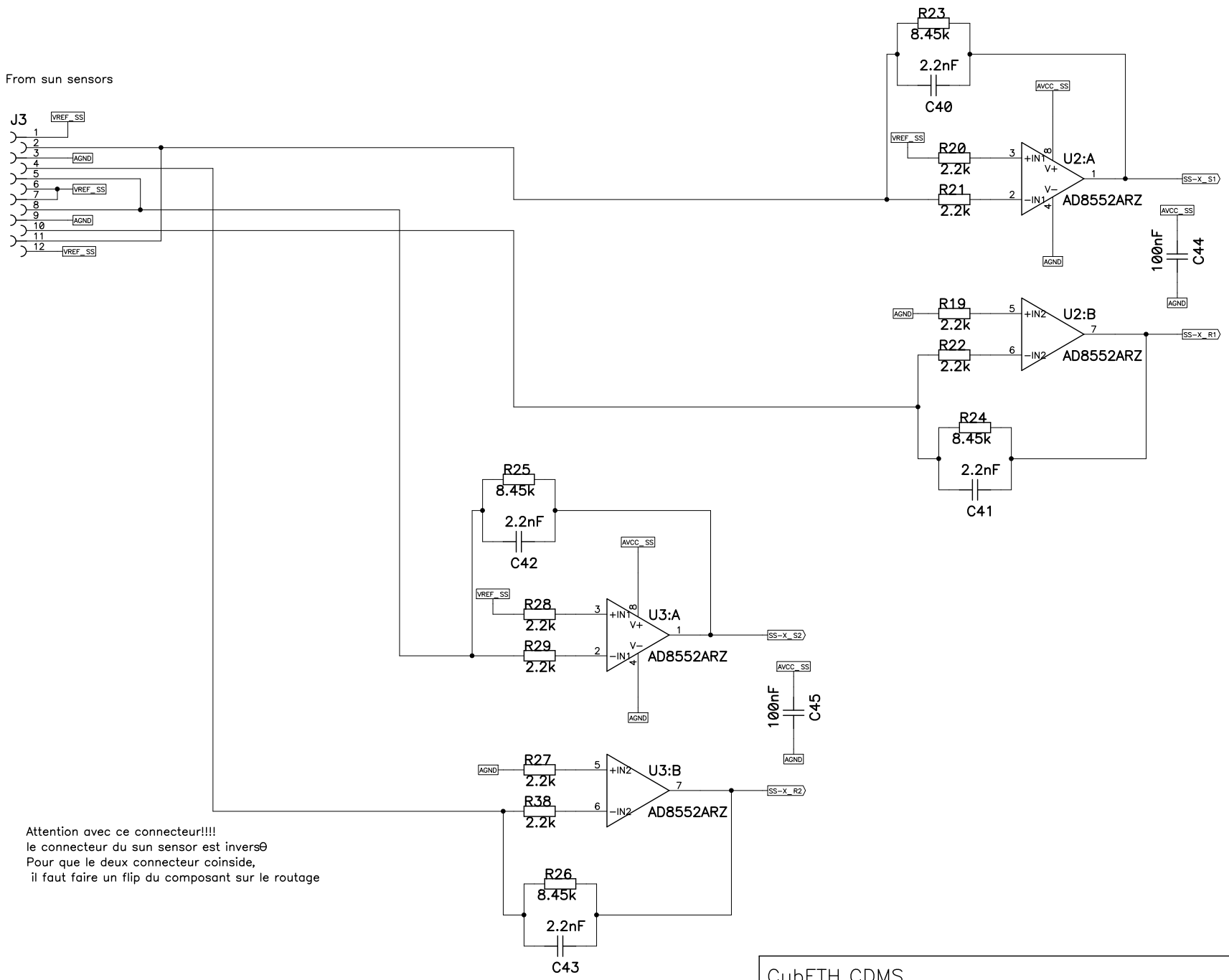
CubETH CDMS Interface FPGA to Sun sens/Mag Torq Sun Sens 1	DES	{Date} lovs
	REV	V1.0
HAUTE ECOLE VALAISANNE	2/6	{Path} Interface_FPGA_sunSens_magTorq.sch

Sun Sensors electronics 2





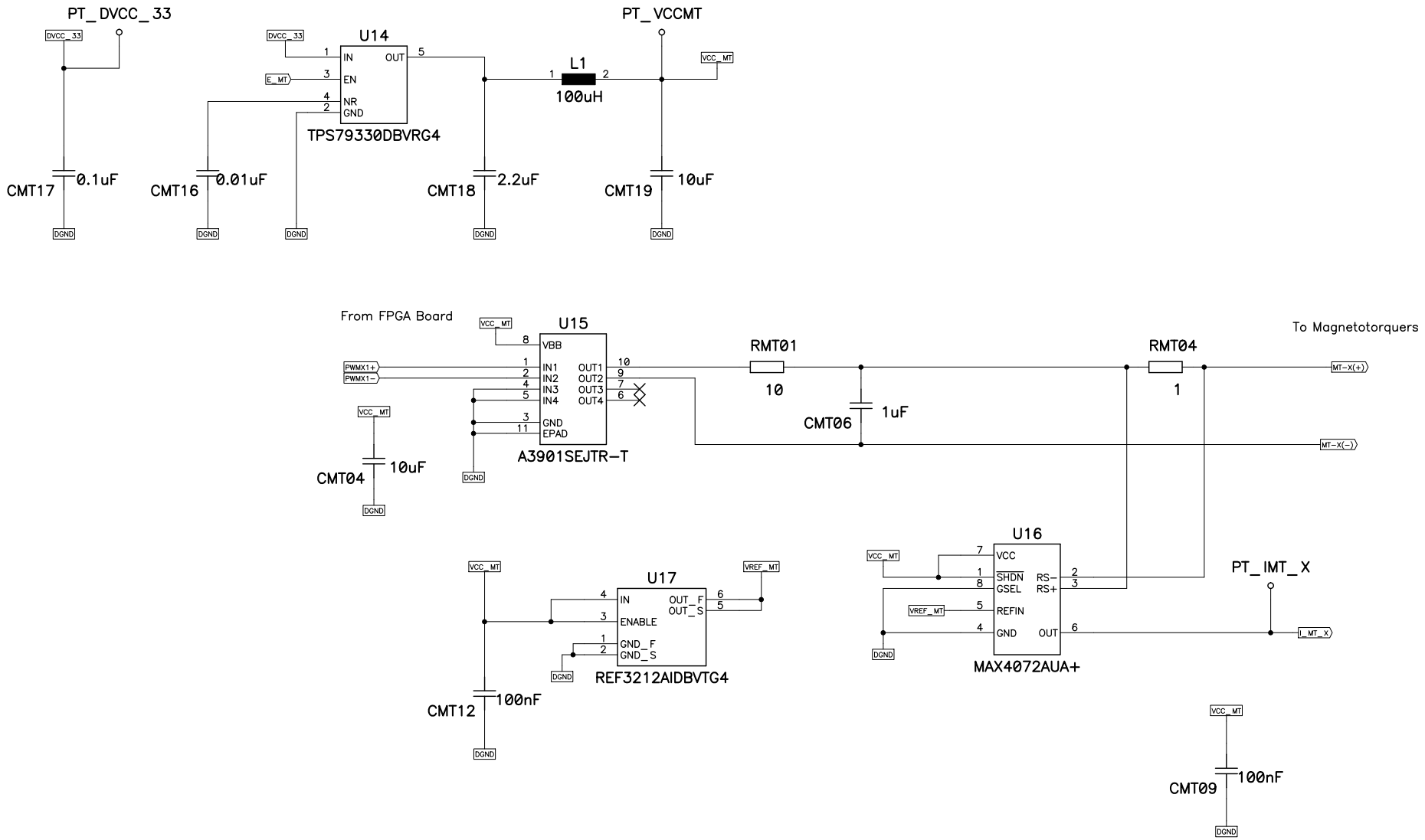
CubETH CDMS Interface FPGA to Sun sens/Mag Torq Sun Sens 2	DES	{Date} lovs
	REV	V1.0
HAUTE ECOLE VALAISANNE	3/6	{Path} Interface_FPGA_sunSens_magTorq.sch



Attention avec ce connecteur!!!!
le connecteur du sun sensor est inversé
Pour que le deux connecteur coinside,
il faut faire un flip du composant sur le routage

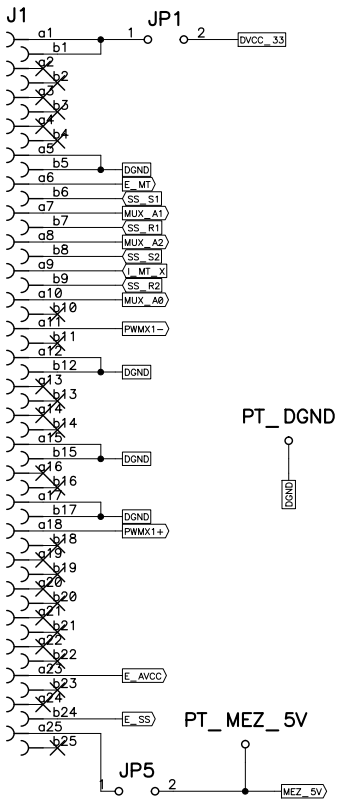
CubETH CDMS Interface FPGA to Sun sens/MagTorq sun sensor1	DES	{Date} lovs
	REV	V1.0
HAUTE ECOLE VALAISANNE	4/6	{Path} Interface_FPGA_sunSens_magTorq.sch

Magneto Torquers electronics

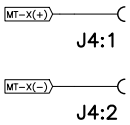


CubETH CDMS Interface FPGA to Sun sens/Mag Torq Mag Torquer	DES	{Date} lovs
	REV	V1.0
HAUTE ECOLE VALAISANNE	5/6	{Path} Interface_FPGA_sunSens_magTorq.sch

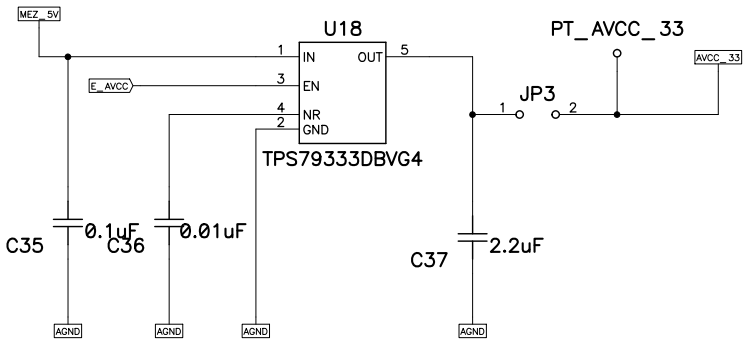
From FPGA Board



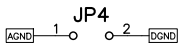
To magneto torquers



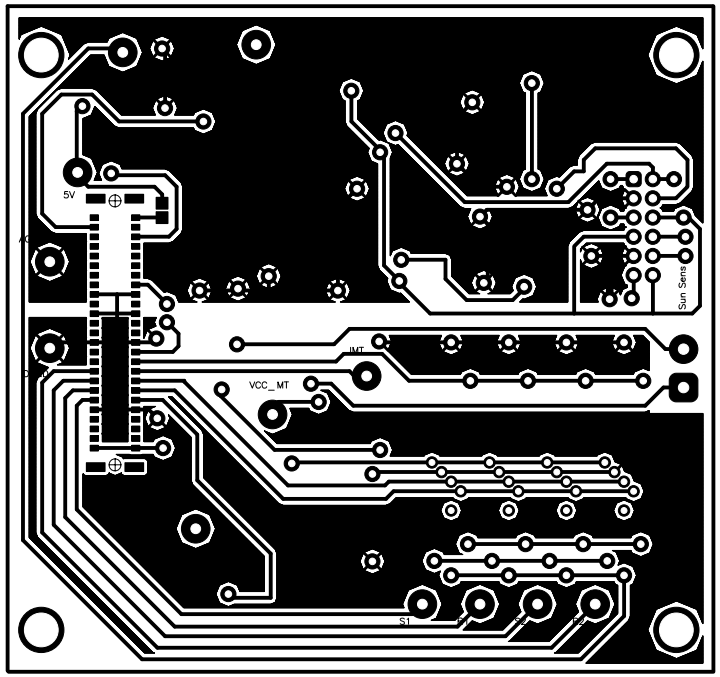
Analogique alimentation



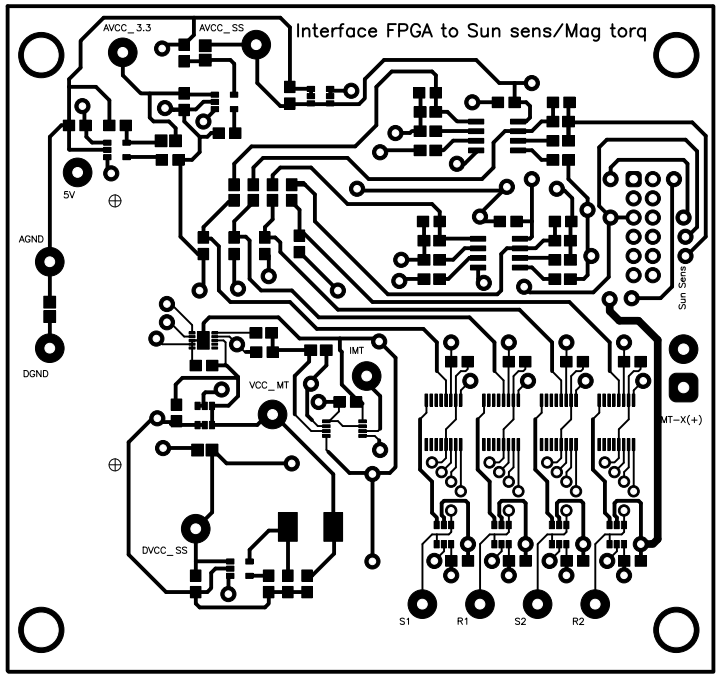
Connection between AGND and DGND



CubETH CDMS Interface FPGA to Sun sens/Mag Torq To FPFA	DES	{Date} lovs
	REV	V1.0
HAUTE ECOLE VALAISANNE	6/6	{Path} Interface_FPGA_sunSens_magTorq.sch



CubETH CDMS Interface FPGA to Sun sens/Mag Torq	DES	06.06.13 Lovejoys
	REV	V1.0
HAUTE ECOLE VALAISANNE	Interface.PCB	



CubETH CDMS Interface FPGA to Sun sens/Mag Torq	DES	06.06.13 Lovejoys
	REV	V1.0
HAUTE ECOLE VALAISANNE	Interface.PCB	

APPENDIX 3 :

Measure of the current probe

Measure of the current probe

< **Internal Vref=2.5[V] used as reference for the AD converter** **Frequency PWM = 200 [kHz]**

Positive current

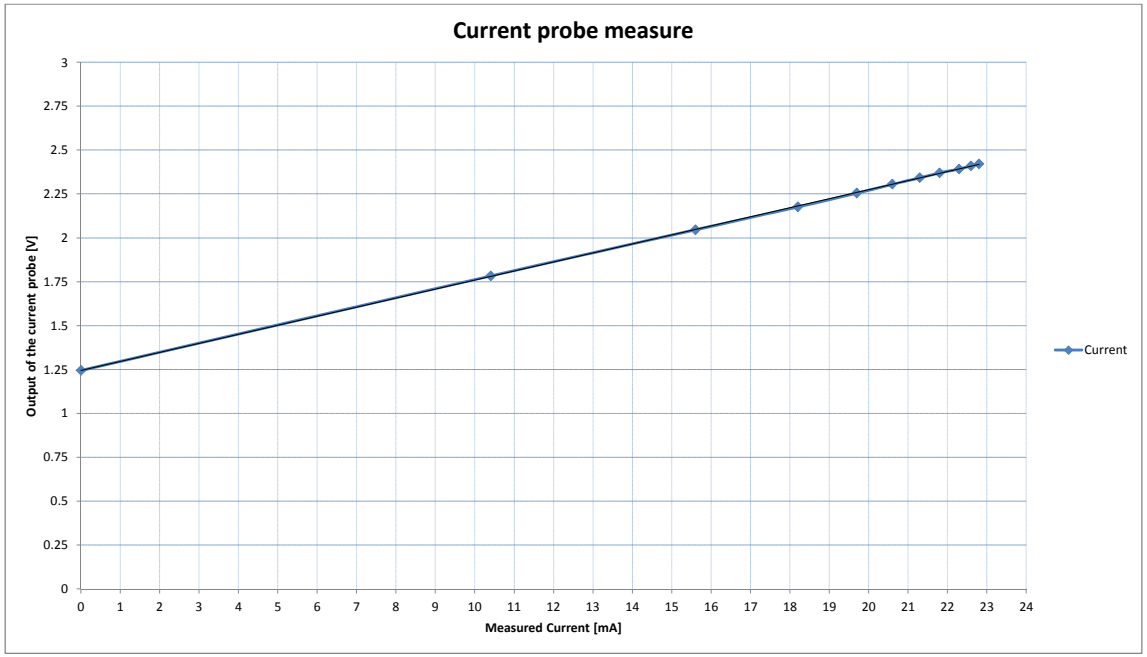
PWM Duty Cycle	Voltage on output of	Mesured Current[mA](1)	ADConv1 HEX	ADConv2 HEX	ADConv3 HEX	ADConv4 HEX	ADConv5 HEX
1%	1.245	0	888	7e5	888	8cc	767
10%	1.784	10.4	777	c8c	9de	b33	b12
20%	2.046	15.6	b33	bbc	e65	e33	bb4
30%	2.177	18.2	e65	f33	cee	bfc	ccc
40%	2.255	19.7	bbc	dba	e65	e67	e33
50%	2.306	20.6	f33	e33	dcc	e65	d9a
60%	2.343	21.3	f33	d9a	ee5	dde	dde
70%	2.371	21.8	e77	e65	e65	ecc	f33
80%	2.393	22.3	ee5	dde	dde	ee5	f33
90%	2.411	22.6	f61	f11	ee5	f31	e65
99%	2.422	22.8	f33	ecc	ee5	f1a	dde

External Vref = 3.3 [V] used as reference for the AD converter **Frequency PWM = 200 [kHz]**

Positive current

PWM Duty Cycle	Voltage on output of	Mesured Current[mA](1)	ADConv1 HEX	ADConv2 HEX	ADConv3 HEX	ADConv4 HEX	ADConv5 HEX	Average of the 5 AD	Voltage conversion [V]	Current AD [mA]
1%	1.245	0	604	607	605	603	605	604	1.241025641	0
10%	1.784	10.4	8cd	8ce	8d0	8bb	8b9	8C6	1.80996337	11.15
20%	2.046	15.6	a13	9ef	a23	9e6	9eb	9FE	2.061391941	17
30%	2.177	18.2	ae2	a84	aae	ad5	a7e	AAE	2.203223443	20
40%	2.255	19.7	b41	b05	b17	ada	ae6	B05	2.273333333	21.4
50%	2.306	20.6	b7c	b22	b77	b13	b67	B4F	2.332967033	22.3
60%	2.343	21.3	b9d	b69	b81	b45	b91	B79	2.366813187	23.3
70%	2.371	21.8	ba8	ba4	b77	b9a	b7c	B91	2.386153846	23.5
80%	2.393	22.3	b93	bc9	b9b	bbb	bca	BB2	2.412747253	24
90%	2.411	22.6	bd2	bdb	bb7	bb2	bb7	BC2	2.425641026	24.3
99%	2.422	22.8	bda	bd0	bd3	bd1	bcd	BD2	2.438534799	24.6
-1%	0.6	0	This testes were not performed as the current probe gives rubbish values for negative current							
-10%	0.94	-10.3								
-20%	0.59	-15.5								
-30%	0.4	-18								
-40%	0.4	-19.4								

(1) the current measur was performed with an ampermeter on the MT


Current probe measure
Output of the current probe [V]
Measured Current [mA]
Current
0
0.25
0.5
0.75
1
1.25
1.5
1.75
2
2.25
2.5
2.75
3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

APPENDIX 4 :

Measure of the interface board’s power sources

Mesure des allimentation de la carte d'interface

Composant	Type de test	Endroit de la mesure	Valeur mesurée	validé
Test des alimentations du regulateur (U14)	Tension d'alimentation 3.3 [V] Tension de sortie 3.0 [V]	PT_DVCC_33 PT_VCCMT	3.27 [V] 3.0 [V]	ok ok
Test des alimentations du regulateur (U17)	Tension d'alimentation 3 [V] Tension de sortie 1.25[V]	Sur pin 4 de l'IC Sur pin 5-6 de l'IC	3.01[V] 1.25 [V]	ok ok
Test des alimentations du regulateur (U18)	Tension d'alimentation 5 [V] Tension de sortie 3.3[V]	PT_MEZ_5V JP3	5.1 [V] 3.31 [V]	ok ok
Test des alimentations du regulateur (U5)	Tension d'alimentation 3[V] Tenison de sortie 1.25[V]	PT_AVCC_SS pin N 6 de cette IC	3.01 [V] 1.25 [V]	ok ok
Test des alimentations du regulateur (U1)	Tension d'alimentation 3.3 [V] Tension de sortie 3[V]	JP3 JP2	3.31 [V] 3.01 [V]	ok ok

Ces tests ont été réalisés suivant la schematics intitulée "Interface FPGA to Sun sens/Mag torqu V1.0

Les courants et les tensions ont été mesurées avec le multimeter agilent (A203/8400.6).
L'alimentation utilisée pour ces testes est l' Alimention de laboratoir : power supply A304 / 8399.03

APPENDIX 5 :

Test of the interface board’s connection

Test des connexions de la carte d'interface

Composant	Type de test	PIN testée	connectée à	validé
GND	connecté à tous les GND			ok
	pas de cout-circuit avec 3.3 V			ok
	pas de court-circuit avec 1.8 V			ok
	pas de court-circuit avec 1.2 V			ok
3.3V	connecté à tous les 3.3V			ok
	pas de court-circuit avec GND			ok
	pas de court-circuit avec 1.8 V			ok
	pas de court-circuit avec 1.2 V			ok
1.8V	connecté à tous les 1.8V			ok
	pas de court-circuit avec GND			ok
	pas de court-circuit avec 3.3 V			ok
	pas de court-circuit avec 1.2 V			ok
1.2V	connecté à tous les 1.2V			ok
	pas de court-circuit avec GND			ok
	pas de court-circuit avec 3.3 V			ok
	pas de court-circuit avec 1.8 V			ok
U15 (A3901SEJTR-T)	connecté	PWMX1+	PWMX1+ (J1)	ok
	connecté	PWMX1-	PWMX1- (J1)	ok
	connecté	DGND	DGND (U14)	ok
	connecté	VCC-MT	VCC-MT (PT_VCCMT)	ok
	connecté	MT-X(-)	MT-X(-) (CMT06)	ok
	connecté	NET017	NET017 (RMT01)	ok
	court-circuit entre les pins de l'IC			ok
U1 (TPS79330DBVG4)	connecté	DVCC_33	DVCC_33 (J1)	ok
	connecté	DGND	DGND (J1)	ok
	connecté	E_MT	E_MT (J1)	ok
	connecté	NET006	NET006 (L1)	ok
	connecté	NET016	NET016 (CMT16)	ok
	court-circuit entre les pins de l'IC			ok
U16 (MAX4072AUA)	connecté	VCC_MT	VCC_MT (PT_VCCMT)	ok
	connecté	MT-X(+)	MT-X(+) (RMT04)	ok
	connecté	NET004	NET004 (RMT04)	ok
	connecté	DGND	DGND (J1)	ok
	connecté	I_MT_X	I_MT_X (J1)	ok
	connecté	VREF_MT	VREF_MT (U17)	ok
	court-circuit entre les pins de l'IC			ok
U17 (REF3212AIDBVTG4)	connecté	DGND	DGND(J1)	ok
	connecté	VCC_MT	VCC_MT (PT_VCCMT)	ok
	connecté	VREF_MT	VREF_MT (U16)	ok
	court-circuit entre les pins de l'IC			ok
U18 (TPS79333DBVG4)	connecté	AGND	AGND(J1)	ok
	connecté	MEZ_5V	MEZ_5V (J1)	ok
	connecté	E_AVCC	E_AVCC (J1)	ok
	connecté	NET003	NET003 (JP3)	ok
	court-circuit entre les pins de l'IC			ok
U1 (TPS79330DBVG4)	connecté	AGND	AGND(J1)	ok
	connecté	AVCC_33	AVCC_33 (PT_AVCC_33)	ok
	connecté	E_SS	E_SS (J1)	ok
	connecté	NET002	NET002 (JP2)	ok
	court-circuit entre les pins de l'IC			ok

APPENDIX 6 :

Command list

Gestion des composants

Type	Valeur	Caractéristique spéciale	Boîtier	Fabricant	Fournisseur	Référence fabricant	Code commande	Prix unitaire (fr)	Délai d'approvisionnement	Nbr de pièces/carte
Multiplexeur		8 canaux	TSSOP-16	ANALOG DEVICES	Mouser	ADG708BRUZ	584-ADG708BRUZ	3.41	de stock	4
Ampli op			SOT-23-6	Texas instruments	Mouser	OPA334AIDBVT	595-OPA334AIDBVT	3.09	de stock	4
Regulateur LDO	3.3V		SOT-23-6	Texas instruments	Farnell	TPS79333DBVRG4	1287689	0.863	de stock	1
Regulateur LDO	3V		SOT-23-6	Texas instruments	Farnell	TPS79330DBVRG4	1287688RL	0.58	de stock	2
Pont H			10-DFN/MLP	Allegro Microsystems	Farnell	A3901SEJTR-T	1651947RL	2.05	de stock	1
Amplificateur bidirectionnelle de détection de courant			uMAX-8	MAXIM	Mouser	MAX4072AUA+	700-MAX4072AUA	2.35	de stock	1
Référence de tension	Vref =1.25V		SOT-23-6	Texas instruments	Farnell	REF3212AIDBVTG4	1180175	7	de stock	2
Ampli Op			SOIC	ANALOG DEVICES	Farnell	AD8552ARZ	9425845	4.7	de stock	2
Condo Céramique	2.2 uF		0805	AVX	Farnell	08053C225KAT2A	1657931RL	0.846	de stock	3(commande min 10)
Résistance shunt	1 ohm	1%	0805	Panasonic	Farnell	ERJ6BQF1R0V	1717826RL	0.177	de stock	1(commande min 5)