



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA



TÉCNICO  
LISBOA



UNIVERSITÀ DEGLI STUDI DI NAPOLI  
FEDERICO II

**Università degli Studi di Padova**  
Centro Ricerche Fusione (CRF)

**Universidade de Lisboa**  
Instituto Superior Técnico (IST)

**Università degli Studi di Napoli Federico II**

JOINT RESEARCH DOCTORATE IN FUSION SCIENCE AND ENGINEERING  
Cycle XXIX

THESIS TITLE

# **New hardware and software technologies for real-time control in nuclear fusion experiments**

**Coordinator:** Prof. Paolo Bettini

**Supervisor:** prof. Leonardo Giudicotti

**Co-Supervisor:** Ing. Gabriele Manduchi

**Ph.D. student:** Marco Gottardo

Padova, January 2018



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA



TÉCNICO  
LISBOA

JOINT Doctorate and NETWORK in Fusion Science and Engineering.

Network Partners:

- Instituto Superior Técnico (IST) Lisboa, Portugal
- Università degli studi di Padova, Italy
- Ludwig Maximilians University Munich, Germany

In collaboration with:

- Consorzio RFX, Italy
- IPP Garching, Germany

# Abstract

*The current machines for the study of nuclear fusion does not produce energy, and their output is substantially a large amount of data.*

*The accuracy of the data collected, and their density within narrow temporal samples, can determine the effectiveness of the real time control systems to install in future reactors.*

*We set ourselves the objective to design and test a high-speed and high-density data acquisition system based on the latest generation FPGA technologies.*

*in the thesis is used the latest products released by Xilinx to design a acquire stream system of signals from generic probes (specifically magnetic probes).*

*The Zynq 7000 family is nowadays state of the art of sistemy SoC that integrating a powerful and extensive FPGA section with an ARM mullticore.*

*Of fundamental importance will be the drastic reduction of signal cables between the sensory apparatus and acquisition systems with the dual objective of eliminating the noise induced and drastically lower installation costs.*

*Magnetic field configuration in RFX is characterised by fast variations of all the three field components during the pulse, with relevant non axis-symmetry in toroidal direction. Typical spectra exhibit modes up to  $n=15$  in toroidal direction and mainly  $m=0$  and  $m=1$  in poloidal direction.*

*As a consequence, probe signals have a large dynamic (more than 60 dB), and extended frequency spectrum (several tens of kHz).*

*Therefore, a large number of probes are required to correctly identify the complex spatial structure of the plasma column.*

*To reduce shielding effects, probes must be installed inside the stabilising shell.*

*The three components of field outside the vacuum vessel can be very different in amplitude.*

*At the same time, one can reach 0.8 T and another can be typically lower than some mT. Furthermore, they vary very quickly.*

*The probes to be installed have to guarantee an uncertainty less than 1 mT to correctly reconstruct the plasma behaviour.*

*These two specifications are particularly stringent and require an accurate calibration and a careful probe alignment to minimise the spurious effect of unwanted components.*

*A further design specification for the sensors is due to the maximum operation temperature of the vacuum vessel (200 °C).*

*The analogic acquires systems must exhibit high isolation, high speed and resolution, but above all a low noise level.*

*The noise must be below minimum margins throughout the frequency spectrum contained in the signals provided by magnetic probes.*

*The main topic of the thesis is to verify the suitability of the ATCA MIMO ISOL modules in the upper and lower part of the signal spectrum of bi-axis magnetic probes in order to be able to be integrated into the new FPGA acquisition and realtime control in RFX systems.*

# Abstract

*Le attuali macchine per lo studio della fusione nucleare producono un grande ammontare di dati.*

*L'accuratezza di questi, la loro densità all'interno di stretti intervalli temporali può determinare la efficacia dei sistemi di controllo in tempo reale che dovranno essere installati nei futuri reattori.*

*Ci siamo posti l'obiettivo di sviluppare e valutare un sistema di acquisizione dati ad alta velocità basato sulla ultima generazione di FPGA.*

*In questa tesi abbiamo impiegato gli ultimi prodotti rilasciati da Xilinx per produrre un sistema in grado di acquisire segnali in streaming provenienti da sonde magnetiche generiche, installate in RFP e Tokamak.*

*La famiglia 7000 Zynq è oggi lo stato dell'arte dei sistemi SoC integrando una potente sezione FPGA con un sistema multicore ARM Cortex A9 di ultima generazione.*

*Di fondamentale importanza sarà la drastica riduzione dei cablaggi tra la macchina e l'apparato di concentrazione dei dati acquisiti con l'obiettivo di eliminare il rumore indotto e ridurre drasticamente i costi di assemblaggio.*

*La configurazione dei campi magnetici in RFX è caratterizzata da veloci variazioni nelle tre componenti di campo, nella durata dell'impulso, con rilevanti componenti non assial-simmetriche specialmente in direzione toroidale. Tipicamente lo spettro mostra dei modi superiori a  $n=15$  in direzione toroidale principalmente con  $m=0$  e  $m=1$  in direzione poloidale.*

*Come conseguenza, i segnali alle sonde hanno un largo range dinamico, superiore a 60 dB e estendono lo spettro di frequenza oltre svariati decine di kHz.*

*Quindi, molte sonde sono richieste per identificare correttamente la complessa struttura spaziale della Colonna di plasma.*

*Per ridurre l'effetto schermo, le sonde devono essere installate dentro la shell stabilizzatrice. Le tre componenti del campo fuori dalla camera da vuoto possono essere molto differenti in ampiezza.*

*Allo stesso tempo, una può raggiungere 0.8 T e un'altra può essere dell'ordine dei mT. Inoltre queste risultano essere molto veloci.*

*Le sonde installate devono garantire una incertezza minore di 1 mT per ricostruire correttamente il comportamento del plasma.*

*Queste due specifiche sono particolarmente stringenti e richiedono una accurata calibrazione e allineamento dei sensori per minimizzare gli effetti spuri e il rilevamento di componenti indesiderate.*

*Un'ulteriore specifica di progetto per i sensori è dovuta alla massima temperatura della camera da vuoto che può raggiungere i 200 °C.*

*I sistemi di acquisizione analogica devono mostrare un alto isolamento, velocità e risoluzione; ma soprattutto la qualità deve essere maggiore di quanto richiesto dal livello di rumore.*

*Il rumore deve rimanere infatti sotto una soglia minima nello spettro di uscita.*

*L'obiettivo principale della tesi è di verificare l'applicabilità del modulo ATCA MIMO ISOL nella parte alta e bassa dello spettro del segnale delle sonde magnetiche biassiali in modo da metterlo in grado di essere integrato nella nuova acquisizione FPGA e controllo in tempo reale per RFXmod.*



# Resumo

*Os dispositivos atuais para o estudo da fusão nuclear não produzem energia e tem como saída uma quantidade substancialmente grande de dados.*

*A precisão dos dados coletados e sua densidade em amostras temporais estreitas podem determinar a eficácia dos sistemas de controle em tempo real para serem instalados em futuros reatores.*

*Nós estabelecemos o objetivo de projetar e testar um sistema de aquisição de dados de alta velocidade e alta densidade baseado nas tecnologias FPGA de última geração.*

*Na tese são usados os produtos mais recentes lançados pela Xilinx para projetar um sistema de aquisição de fluxo para sondas genéricas (especificamente sondas magnéticas).*

*A família Zynq 7000 é hoje em dia o mais avançado sistema SoC que integra uma poderosa e extensa seção FPGA com um ARM “multicore”.*

*De fundamental importância será a redução drástica dos cabos de sinal entre o aparelho sensorial e os sistemas de aquisição com o duplo objetivo de eliminar o ruído induzido e reduzir drasticamente os custos.*

*A configuração do campo magnético no RFX é caracterizada por variações rápidas de todas as três componentes do campo durante o pulso, com relevante não simetria na direção toroidal. Os espectros típicos exibem modos até  $n = 15$  na direção toroidal e principalmente  $m = 0$  e  $m = 1$  na direção poloidal.*

*Como consequência, os sinais da sonda possuem alta dinâmica (mais de 60 dB) e um espectro de frequência estendido (várias dezenas de kHz).*

*Portanto, um grande número de sondas é necessário para identificar corretamente a estrutura espacial complexa da coluna de plasma.*

*Para reduzir os efeitos de blindagem, as sondas devem ser instaladas dentro do escudo estabilizador.*

*As três componentes do campo fora do vaso de vácuo podem ser muito diferentes em amplitude.*

*Ao mesmo tempo, pode-se alcançar 0,8 T e outro pode ser tipicamente inferior a alguns mT. Além disso, eles variam muito rapidamente.*

*As sondas a serem instaladas devem garantir uma incerteza inferior a 1 mT para reconstruir corretamente o comportamento do plasma.*

*Essas duas especificações são particularmente rigorosas e exigem uma calibração precisa e um alinhamento cuidadoso da sonda para minimizar efeitos espúrios de componentes indesejados.*

*Outra especificação de projeto para os sensores é devido à temperatura de operação máxima do vaso de vácuo (200 ° C).*

*Os sistemas de aquisição analógicos devem exibir alto isolamento, alta velocidade e resolução, mas acima de tudo um baixo nível de ruído. O ruído deve estar abaixo das margens mínimas em todo o espectro de frequência contido nos sinais fornecidos pelas sondas magnéticas.*



# Index

Abstract .....	iii
Resumo.. .....	iv
<b>THESIS TITLE .....</b>	<b>i</b>
<b>Chapter 1. Introduction.....</b>	<b>1</b>
1. 1. Thesis introduction .....	1
1. 2. Fusion experiment diagnostics .....	4
1. 3. A Simplified customizable DAQ .....	5
1. 4. The new proposed approach .....	8
1. 5. Daq system for array of 5 pin balanced electrostatic probe (or fast magnetics) 9	
1. 6. Daq system for general purpose magnetics, soft X-Ray detectors, PMT, CCD, arrays. ....	11
<b>Chapter 2. Definition and roles of magnetic diagnostic in a fusion machine ..</b>	<b>13</b>
<b>Chapter 3. Probe and transmission line specification .....</b>	<b>21</b>
3.1 Magnetic probe circuit model.....	21
3. 1. Probe electrical modelling .....	27
3. 2. Transmission line circuit model.....	28
<b>Chapter 4. Analog frontend specification .....</b>	<b>31</b>
4. 1. Analog frontend to Zynq, the ATCA MIMO ISOL module.....	31
4. 2. Frontend Low Pass Filter.....	35
4. 3. Frontend to FPGA connector.....	37
4. 4. Opto-isolated differential input stage .....	38
4. 5. 18 bit ADC SAR integrated circuit.....	43
4. 6. The AD7641 features .....	44
4. 7. AD7641 microprocessor interface capability.....	46
4. 8. AD7641 SPI interface capability .....	46
<b>Chapter 5. 1K5Vrms Isolated frontend .....</b>	<b>49</b>
5. 1. Power supply stage, DC/DC converter .....	49
<b>Chapter 6. Analog frontend to Pitaya interface.....</b>	<b>51</b>
6. 1. Interfacing the analog ATCA MIMO ISOL module to FPGA .....	51
6. 2. Prototype of hardware interface between ATCA MIMO ISOL module and FPGA	52
<b>Chapter 7. The ATCA MIMO ISOL module VHDL interface.....</b>	<b>61</b>
<b>Chapter 8. Zynq Architectures .....</b>	<b>65</b>
8. 1. AXI Interconnect Block .....	69

8. 2.	Block schematic and Internal architecture of Zynq 7000.....	70
8. 3.	GIC, Generic Interrupt Controller.....	71
8. 4.	Private, shared and software interrupts .....	73
8. 5.	GIC functionality .....	73
8. 6.	Interrupts, priorities and handling.....	75
8. 7.	The central interconnect module .....	77
<b>Chapter 9.</b>	<b>The AXI interface .....</b>	<b>79</b>
9. 1.	The AMBA protocol .....	79
9. 2.	The AXI interface.....	80
<b>Chapter 10.</b>	<b>AXI- Stream FIFO .....</b>	<b>85</b>
10. 1.	AXI Stream FIFO first experimental test .....	85
<b>Chapter 11.</b>	<b>ADC DMA interface.....</b>	<b>89</b>
11. 1.	The Zynq7000 DMA overview .....	89
11. 2.	The Xilinx DMA logic .....	90
11. 3.	AXI-DMA and streaming data acquire, block diagram.....	94
11. 4.	Structure of the AXI dataflow .....	94
<b>Chapter 12.</b>	<b>Final tests and results.....</b>	<b>97</b>
12. 1.	Python.....	97
12. 2.	Acquisitions test .....	99
12. 3.	Frequency response.....	103
12. 4.	The noise spectral density.....	108
12. 5.	Measurement of intrinsic noise .....	110
12. 6.	Signal numerical integration .....	115
12. 7.	Test without DC/DC converter .....	119
12. 8.	Low frequency noise noise analysis .....	122
12. 9.	Noise integration and expected integrated signals on modified ADC module 125	
12. 10.	Magnetic probe measurement test circuit .....	130
<b>Chapter 13.</b>	<b>Conclusion.....</b>	<b>135</b>
<b>A.</b>	<b>Appendix: Source codes .....</b>	<b>137</b>
<b>B.</b>	<b>Appendix: Software tools for PL section.....</b>	<b>157</b>
	The design on Vivado.....	157
	Create a new Verilog project Step by Step. ....	158
	Step 1 - Hardware definition.....	160
	Step 2 – Adding the source to be packaged in IP block design. ....	163
	Step 3 – Creating a new Verilog IP .....	166

Step 4 – Editing IP and sources add .....	169
Step 5 – Module insertion using the template .....	175
Step 6 – AXI wrapper.....	178
Step 7 – Add a new custom IP to the design.....	180
Step 8 – TOP module creation and start synthesis .....	181
Step 9 – Manually creating a test application .....	182
Step 10 – <i>Deploy on RedPitaya and test program execution</i> .....	184
<b>C. Appendix: FPGA to ARM interfacing .....</b>	<b>187</b>
The AXI write tool.....	187
<b>References .....</b>	<b>189</b>
<b>Acknowledgements .....</b>	<b>191</b>



# Index of figures

Figure 1-1A Typical approach for a diagnostic system.....	6
Figure 1-2 Current situation, typical diagnostic wiring .....	7
Figure 1-3 The new proposed approach with simplified wiring. ....	8
Figure 1-4: Triple Langmuir probe configuration.....	9
Figure 1-5: 5 pin balanced Langmuir probe configuration .....	10
Figure 2-1 inductive sensors in a toroidal machine .....	13
Figure 2-2 RFX-Mod external magnetic probes layout. ....	14
Figure 2-3 Overview of magnetic probes on RFX-Mod.....	15
Figure 2-4 Integrated System of Internal Sensors (ISIS) for RFX-Mod .....	15
Figure 2-5 View of RFX-mod first wall with probes layout.....	16
Figure 2-6 mm copper shell. Exploded view.....	17
Figure 2-7 Biaxial magnetic probes RFX installation .....	17
Figure 2-8 RFX-Mod copper shell .....	18
Figure 2-9 Magnetic probes connectors on RFX vacuum chamber .....	18
Figure 2-10 Biaxial probe LEMO connectors.....	19
Figure 3-1 Magnetic probe model typical. ....	22
Figure 3-2 <i>Two axes magnetic sensor model</i> .....	22
Figure 3-3 Derivative of the frequency response two-axis probe.....	23
Figure 3-4 Size and appearance of the magnetic probe.....	23
Figure 3-5 Torlon Vs Peek core probe capacitance and inductance.....	24
Figure 3-6 Bi-axial magnetic pick-up coils .....	25
Figure 3-7 Plasma position from magnetic measurements.....	25
Figure 3-8 RFX-mod waveform of magntic field from pick-up array.....	26
Figure 3-9 Differential acquisition of magnetic probes.....	26
Figure 3-10 Basic model of a magnetic probe .....	27
Figure 3-11 <i>Two axes magnetic sensor model connect to transmission line</i> . ....	28
Figure 3-12 <i>Two axes probe and transmission line</i> frequency response. ....	29
Figure 3-13 Capacive model of the line.....	29
Figure 3-14 Capacive <i>transmission line</i> frequency response .....	30
Figure 4-1 ATCA MIMO ISOL module, isolated differential analog fronted board.....	31
Figure 4-2 Block schematic of the ADC module. ....	32
Figure 4-3: Flow chart of data streaming acquisition to be implemented on the FPGA 34	
Figure 4-4 Low pass filter schematic.....	35
Figure 4-5 Molex 87833, probe connector.....	35

Figure 4-6 Low pass filter PSim. ....	36
Figure 4-7 Low pass filter Bode diagrams. ....	36
Figure 4-8 Analog frontend to FPGA connector. ....	37
Figure 4-9 JAE connector footprint.....	38
Figure 4-10 Differential operational amplifier THS4520.....	38
Figure 4-11 THS4520 internal configuration.....	39
Figure 4-12 Active voltage reference generator.....	39
Figure 4-13 Differential amp. op. schematic on insulated frontend.....	40
Figure 4-14 Generic voltage noise spectral density.....	41
Figure 4-15 The AD7641 functional block diagram .....	43
Figure 4-16 Input capacitors rail.....	44
Figure 4-17 The AD7641 SPI signals and wiring.....	46
Figure 4-18 The AD7641 typical application.....	47
Figure 4-19 The AD7641 pinout.....	48
Figure 5-1 DC/DC converter internal block diagram .....	49
Figure 5-2 DC/DC converter emission at 125% load.....	50
Figure 5-3 DC/DC converter emission at 8% load.....	50
Figure 6-1 Resistive net level adapter.....	51
Figure 6-2 Frontend to Pitaya adapter with channel 1 mounted.....	52
Figure 6-3 Frontend to Pitaya overview.....	53
Figure 6-4 ATCA- MIMO-ISOL to RedPitaya interface, in use version .....	53
Figure 6-5 Analog frontend to pitaya interface, Dual layer PCB.....	54
Figure 6-6 PCB Bottom side, with signals table on silkscreen.....	54
Figure 6-7 Carry board layout final revision .....	55
Figure 6-8 ATCA- MIMO-ISOL to RedPitaya interface, prototype.....	56
Figure 6-9 Carry board connectors .....	57
Figure 6-10 SNLVDS105 internal schematic.....	58
Figure 6-11 SNLVDS105 LVDS repeaters pinout.....	58
Figure 6-12 Regulated 3v3 supply and resistors interface .....	59
Figure 6-13 Carry board BNC of on board ARM ADC .....	59
Figure 6-14 RedPitaya extension connectors.....	60
Figure 7-1 Master Serial Data Timing for Reading (Read Previous Conversion During Convert).....	62
Figure 7-2 IP implementation of scc52460 module.....	63
Figure 8-1 Zynq® Architectures, simplify block schematic.....	65
Figure 8-2 PL interface to PS memory subsystem .....	67

Figure 8-3 Zynq® 7000 internal architecture.....	70
Figure 8-4 Generic Interrupt Controller, block schematic. ....	71
Figure 8-5 I/O Peripherals System Diagram and central interconnect. ....	77
Figure 9-1 The five signals evolved in the AXI memory mapped protocol .....	80
Figure 9-2 AXI data exchange .....	81
Figure 9-3 AXI data transfer FPGA to ARM .....	81
Figure 9-4 PL to PS and vice versa internal interface.....	83
Figure 9-5 Zynq-7000 AP SoC internal block diagram .....	84
Figure 9-6 Axi Interconnect block represented in HDL .....	84
Figure 10-1 AXI4-Stream FIFO Block Diagram .....	85
Figure 10-2 Module design connection .....	86
Figure 10-3 AXI Stream Complete Vivado design .....	87
Figure 10-4 scc52460 Module internal design.....	88
Figure 11-1 DMA controller interconnection. ....	91
Figure 11-2 DMA controller block diagram. ....	91
Figure 11-3 DMA access diagram.....	93
Figure 11-4 block diagram of the interfacing principle with the hardware module.....	94
Figure 11-5 AXI-DMA logic chain and HDL implementation .....	95
Figure 11-6 ATCA MIMO ISOL MODULE implemented on RFX_scc52460 logic IP with AXI interconnect to stream pipe and DMA .....	95
Figure 11-7 AXI-DMA and stream_pipe IP internal HDL Vivado representation .....	96
Figure 12-1 ADC module input voltage sensitivity .....	101
Figure 12-2 functions generator HP33120A .....	102
Figure 12-3 Frequency response of ADC module with sinusoidal input.....	103
Figure 12-4 frequency response of ADC module with white noise input.....	104
Figure 12-5 Harmonic spectrum of ADC module at 10kHz sinusoidal input.....	106
Figure 12-6 General characteristic of Op. Amp voltage noise. ....	108
Figure 12-7 Noise sampling of ADC module with 50 Ohm terminated input .....	110
Figure 12-8 Sampled Noise histogram of ADC module with 50 Ohm terminated input .....	111
Figure 12-9 Noise sampling of ADC module with 50 Ohm terminated input (Zoom)..	111
Figure 12-10 internal divider integrated on DC-DC converter.....	112
Figure 12-11 Timing sequence from converter start to DC-DC synchronization .....	112
Figure 12-12 Flip Flop divider .....	113
Figure 12-13 The SN74LVC2G74 Flip flop in the diagram context.....	113
Figure 12-14 Sampled Noise spectrum of ADC module .....	114

Figure 12-15 Resampled input and time integrated signal .....	115
Figure 12-16 Minum level reference magnetic signal. ....	116
Figure 12-17 Resampled noise compared with the simulated minum level reference magnetic signal.....	117
Figure 12-18 Resampled integrated input compared with the simulated worst case. ....	117
Figure 12-19 Typical drift error of the RFX-mod analog signal integrator. ....	118
Figure 12-20 Distribution of final integration error of the analog integrators of RFX-mod after 10 s.....	118
Figure 12-21 Noise sampling of ADC module with 50 Ohm terminated input, without DC-DC coneveter .....	119
Figure 12-22 Sampled Noise histogram of ADC module with 50 Ohm terminated input without DC-DC converter .....	120
Figure 12-23 Histogram noise input from AD7641 datasheet .....	120
Figure 12-24 Sampled Noise spectrum of ADC module without DC-DC converter....	121
Figure 12-25 Resampled input direct and time integrated signal, without DC-DC converter.....	121
Figure 12-26 Noise spectrum of the ADC module in its basic configuration, with 50Ω termination at input. ....	122
Figure 12-27 Noise spectrum of the ADC module with linear power supply and 50Ω terminator .....	123
Figure 12-28 Noise spectrum of the ADC module with analog input section bypassed and DC/DC converter power supply.....	124
Figure 12-29 Noise spectrum of the ADC module with analog input section bypassed and linear power supply. ....	124
Figure 12-30 Numerically integrated signals obtained from the ADC module in its original configuration .....	125
Figure 12-31 Histogram of the final values after 10 s of numerical integration of the sample signals from original ADC module configuration .....	126
Figure 12-32 Low level reference signal and experimental noise of original ADC module configuration.....	126
Figure 12-33 Numerical integration of low level reference signal and experimental noise of original ADC module configuration.....	127
Figure 12-34 Numerically integrated signals obtained from the ADC module in with the analog input section bypassed.....	127
Figure 12-35 Histogram of the final values after 10 s of numerical integration of the sample signals from the ADC module with the analog input section bypassed. ....	128
Figure 12-36 Low level reference signal and experimental noise of the ADC module with the analog input section bypassed.....	128
Figure 12-37 Numerical integration of low level reference signal and experimental noise of the ADC module with the analog input section bypassed.....	129



Figure 12-38 Histogram of the final values after 10 s of numerical integration of the sample signals from the ADC module with the analog input section bypassed and linear power supply. ....	129
Figure 12-39 Test coil sketch .....	130
Figure 12-40 Resistive load circuit .....	131
Figure 12-41: 2.5 $\Omega$ resistive load. ....	131
Figure 12-42 Resistive load connect to test coil and probe .....	131
Figure 12-43 Voltage signal by a potentiometer .....	132
Figure 12-44 Test coil and magnetic probe .....	132
Figure 12-45 Acquisition system connected to coil and probe.....	133
Figure 12-46 Direct acquired signal from the probe directly connected to the ADC input and its time integrated signal .....	134
Figure B-1 Xilinx Vivado setup package.....	157
Figure B-2 Vivado Welcome Page. ....	158



# Index of tables

Table 3-1	Magnetic probe technical characteristics. ....	21
Table 4-1	Differential analog signals. ....	32
Table 7-1	Serial Clock Timing in Master Read After Convert Mode. ....	61
Table 8-1	Zynq7000 PL Interrupt Signals.....	75
Table 11-1	DMA Req/Ack signals on PL section.....	90
Table 12-1	List of ADC module input voltage sensitivity test files.....	100
Table 12-2	List of ADC module frequency response test files.....	102



# Chapter 1. Introduction

## 1. 1. Thesis introduction

The aim of the thesis is to produce a new software and hardware system for data acquisition, mainly from magnetic-type probes, for general use, simpler and more economic, for research groups involved in various experimental plasma magnetic confinement machines around the world.

The goal is to get a system that maintains the speed and isolation features of the actual one but greatly reduces the size and expense, and allows to approach the data acquisition system to diagnostic electronic as much as possible to the machine, reducing or nearly eliminating any long transmission line that may act as source of disturbance.

The research application and the experimental section took place at the laboratories of the CNR of Padua where the RFP machine named RFX operates.

Magnetic field configuration in RFX-mod is characterized by fast variations of all the three field components during the pulse, with relevant non axis-symmetry in toroidal direction.

Typical spectra exhibit modes up to  $n=15$  in toroidal direction and mainly  $m=0$  and  $m=1$  in poloidal direction.

As a consequence, probe signals have a large dynamic (more than 60 dB), and extended frequency spectrum (several tens of kHz).

Magnetic diagnostics are among the most sensitive subsystems on RFX-mod. This is both due to the nature of the signal itself (noisy and easily susceptible) and to the number of signals, which is close to a thousand of various kinds. These probes are installed both in-vessel and ex-vessel. In the present thesis we will focus on the development of the system of 48x4 bi-axial pick-up coils, measuring poloidal and toroidal magnetic field, and of the 48 toroidal windings. The study can be extended to magnetic probes of different nature.

The thesis will propose a number of modifications to an original analog system based on ATCA-MIMO-ISOL components. Such a system is already in use for real-time control on JET, RFX-mod and is a candidate for ITER. The software interface will use MARTE, a framework of growing popularity within the fusion community and which represents a promising option for real-time control on ITER. At the present time the system comprises three racks. The first computer is equipped with twelve ATCA fabric channels (x1/x4 full-duplex PCIe), RTOS multicore processor (RTAI for Linux), PCIe/PCI legacy extra slots. The second machine has the multichannel system with 32 analog input channels (digitizer/transient recorder) which are removable one by one for replacement. The third computer contains the data transfer module, reaching up to 800 Mb/s over x4 PCIe to the host processor. In the present work we propose to keep the ATCA-MIMO-ISOL analog modules, which will be made autonomous by interfacing them with a SoC embedded system (commercially sourced). The new system will use the ATCA-MIMO-ISOL module as a frontend, to which circuit modifications are proposed, an interface with the SoC system and a new framework to manage timing and possible integration.

The new proposal consists in the development of a compact system, with a reduced number of analog channels directly installable on the hosting Linux based .Acquired signals will then be transmitted to the central system via TCP/IP or UDP.

The front end and the ADC section is the same used in the original solution, but in the new system it is directly interfaced to a FPGA board. The recent ZYNQ architecture, providing on the same chip a programmable FPGA and amulticore ARM processor, represent an attractive approach for interfacing rgw ADC board, using the FPGA handling the serial line digital line from the ADC and the processor to send data over the network to the central system. The board selected for this application is Red Pitaya, hosting a Zynq FPGA with a dual core ARM processor. Even if that board hosts its own ADC, in this application the internal ADC is not used, taking input signals from the external ADC board via a serial link connected to the FPGA.

The idea is that every new electronic board assigned to the research groups will be part of all diagnostic systems and will be accessible from the control room as a simplified data acquisition system, in the next chapter identify like sDAQ.

In the first part of the research, the development of an interface board connecting the ATCA-MIMO-ISOL to the RedPitaya is presented.

Two prototypes have been obtained, a first version capable of interfacing 4 ATCA-MIMO-ISOL modules applied in the experimental phase, and a second version, with three channels, much more complete than it can be installed on RedPitaya, using the appropriate topologically coincident connectors to directly insert the daughterboard to the motherboard.

A two layer system is obtained in which there are 4 RF connectors. These 4 BNC are 2 analog inputs and 2 analog outputs that remain active in the motherboard (RedPitaya), while the daughterboard, inserted as a top floor, shows the three BNC connectors associated with their respective ATCA-MIMO-ISOL modules, and on the other side the 8 standard ADC channels, standard speed, in the ARM section.

All the digital I/O available in Red Pitaya are reported on the Daughter board connectors and it is available for other applications both in ARM or FPGA Zynq internal section.

The second version solves the problem of adapting the voltages needed at some I / O points due to constraints due to both hardware configuration and transmission protocols implemented in firmware in HDL.

The developed electrical diagrams, the layout and construction files gerber are available for those who want to continue the search at [1].

In the second part of the thesis, the software of the FPGA section has been developed which allows serialization of the output of the 18bit ADC mounted on the ATCA-MIMO-ISOL modules. The same software implements the transfer between the two internal sections of the ZYNQ using the AXI 4 lite protocol in order to send data to the processor and eventually over the network.

The FPGA program will manage a data stream acquisition and implement DMA procedures transferring acquired samples into processor memory.

The software platform used for FPGA programming is Vivado 2015-4 with which using HDL language IPs have been developed.

A dedicated driver was developed and continuous streaming acquisitions were key to collecting data in the experimental part of the thesis.

A possible application of the FPGA-based acquisition and, in particular, of the new electronic board that mounts the SoC FPGA systems is the possible integration of complex timing systems such as protocol 1588 for Ethernet synchronization.

Signal processing, like filtering or integration, could also take place in the FPGA section before sending data to the ARM system.

This opens the way for the possible implementation of complex operations performed locally and with predetermined time in the design phase of logic making intrinsically compatible operations possible for real-time use.

For example, a possible application of fundamental importance to the acquisition of signals from magnetic probes is the possibility of numerical integration in hardware and before transmission, exploiting the typical times of an FPGA architecture.

The integrated value in the FPGA section can be reprocessed in the region itself or transmitted to the ARM section without the need for further integration without the need for further post processing.

numerical integration is crucial due to the nature of probes that return a signal proportional to the derivative of the magnetic field .

During the implementation of the system, it turned out that the current front end and ADC solution are not fit for numerical integration because of the added noise, originating an unacceptable drift when integrating input signals for few seconds.

The reason for the added noise has been investigated and the problem was found in the circuit section that creates the reference for the ADC at differential inputs, an anomaly has been encountered and will be discussed in a separate paragraph.

Some hardware alternatives are proposed to solve them and will be presented in the end of this thesis. and in addition of a new software solution to be located in each single SoC (system on Chip) that corresponds to any low-cost diagnostics.

Another goal, other to maintaining an equal or greater performance of these simplified, and the low cost diagnostic, is the possibility of assigning to each individual research group although the number of captured signals greatly increases.

More in detail, the low frequency noise orived to be the cause of the drift in integration. It is noise detected under the frequency we indicate with  $F_c$ , "corner Frequency", prevents the numerical integration of the acquisition for long periods, in case of RFX over 10 seconds.

However, the inadequacy of the circuit section from the connector to the input of the ADC is identified, which contributes to introducing the signal non-integrability.

In a specific chapter, the developed hardware interface, in two versions is presented, and the constructive gerber files are available at [2].

The final part of the thesis consists in some test to measurements of intrinsic noise of the components used and the circuit configurations in which these components are inserted in the context of the ATCA-MIMO-ISOL modules. This activity has been carried out in order to better characterize the system and its limits.

In the experimental section a test case, consisting of an induction coil, was created like shown in the § 0 powered by a low noise power supply and terminated on a resistive load.

A coil voltage was supply to a modulating, well-known, harmonic signal.

In the test coil a biaxial magnetic probe was inserted whose output was sampled by our experimental system for periods ranging from 10 to 40 seconds depending on the test performed.

The tests had two purposes, the noise measurement at the top of the spectrum and the measurement at the low end of the spectrum.

Several graphs have been derived, including standard noise deviation depending on the duration of the sampling, the trend of the integrated signal, the noise density histograms under various conditions of use.

## 1. 2. Fusion experiment diagnostics

In current fusion experiment machines many diagnostic equipment are involved with heterogeneous type, generating a large amount of signals that must be acquired, stored and processed. The signal acquisition and processing recently received more and more attention due to the possibilities in the machine active control that proved to be a valuable choice for long lasting pulses stabilization.

The diagnostic type is often closely related to the way the data is achieved and elaborated, passing from the sensor system through the digitalization of the signals and finally to the computing devices that generate a possible control feedback or store data in archives.

We can divide diagnostics in main categories, associated to different physical phenomena and of course to the different type of acquisition system involved. The four major families are listed below:

- Spectroscopic diagnostics, like XUV-VUV or X-Ray, or electron cyclotron emission.
- Refractometers and interferometers, which manipulate optical and laser signals by performing demodulation.
- Thomson scattering. It is an elastic scattering of electromagnetic radiation by a free and charged particle. measurement of electron temperature  $T_e$  is provided by Thomson scattering of laser light (TS).
- Arrays of other diagnostics representing many different other physical quantities acquired from the experiment.

The last type of diagnostics are the heart of the investigation for the development of new hardware and software solutions in this thesis. These are represented by broadband signals that usually extend within a 500kHz band, acquired from probe arrays. Common signals of this kind could be represented by:

- Signals from magnetic probes
- Detectors of soft x-ray, acquired by probes derived by photodiodes.
- Langmuir probes.
- photomultipliers.

In particular we will focus on the sensor chain used to acquire the magnetic field inside the experiment vessel. This specific diagnostic use to present low SNR, high dynamic range, wide bandwidth, and superimposed noise. But mainly the problem comes from the integral nature of the signal itself because the acquired quantity is proportional to the derivative of magnetic field.

Indeed the actual minimum signal detectable by the RFX magnetic probe system is:



$$\frac{dB(t)}{d(t)} = \frac{1 \text{ mT}}{1 \text{ sec}}$$

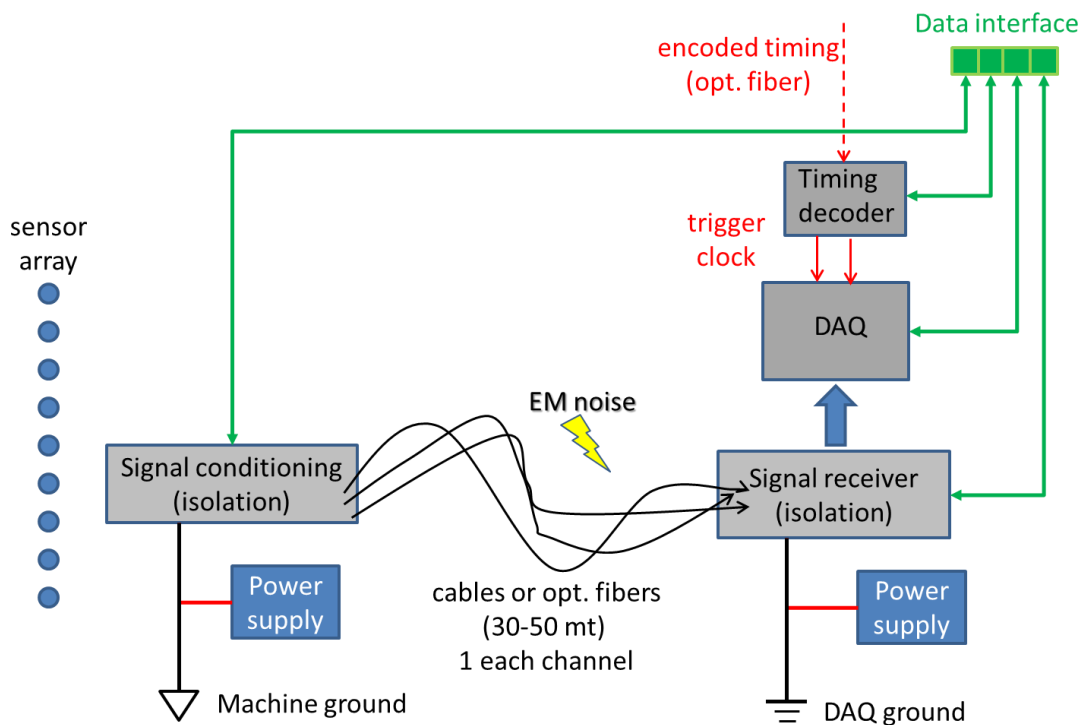
Thus it follows immediately that the magnetic field comes from an integration of the acquired signal in time.

The concept of acquisition makes desirable to design compact embedded equipment to ease the installation. It also asks for a strong insulation of the frontend with respect to the converters in order to both mitigating the possible ground loops and eddy signals coming from impulsive variations of the local magnetic field passing through the circuits.

### **1. 3. A Simplified customizable DAQ**

The aim is to realize a simple, possibly modular, system for acquiring analog signals from a plasma diagnostic in contact with or very close to the plasma. The application requires a reasonable bandwidth (typically from DC to 1 or 10 MHz, depending on the application) in an environment subject to strong EM disturbances (arc-discharges, voltage spikes, RF). Usually measurements are carried out on arrays of sensors, and number of channel to be acquired at each station is often high (say from 8 to 64).

The advent of SoC FPGA devices now allows the building of DAQ/control systems which can be connected directly close to or even embedded in plasma diagnostic systems. This approach now appear feasible and at low cost. The advantages are clear. From the technical point of view it would increases the robustness and the quality of the signals. From the management point of view it will bring a great simplification of the system cabling, overall design and maintenance, reduce the necessary documentation and deployment cost and time.



**Figure 1-1A Typical approach for a diagnostic system.**

Signals are coming from a variety of sensors like PMTs, diode detectors, probes and magnetic coils. Most of these are low voltage, high impedance outputs and require amplification and buffering to be sent over cables. In most cases their ground reference are at different potential, which at best causes noise disturbances and in worst cases can (and often did) destroy part of the electronics. Differential transmitter/receiver requires expensive balanced cables and the rejection it is not effective at high frequency. Moreover serious spike can still damage the electronics. In many cases expensive fiber optic systems have been used.

The acquisition system is to be synchronized and triggered by the general timing of the experiment and this done by using another separate module system.

Morover often an additional local PC is needed to perform simple but essential tasks or add missing interface functionalities.

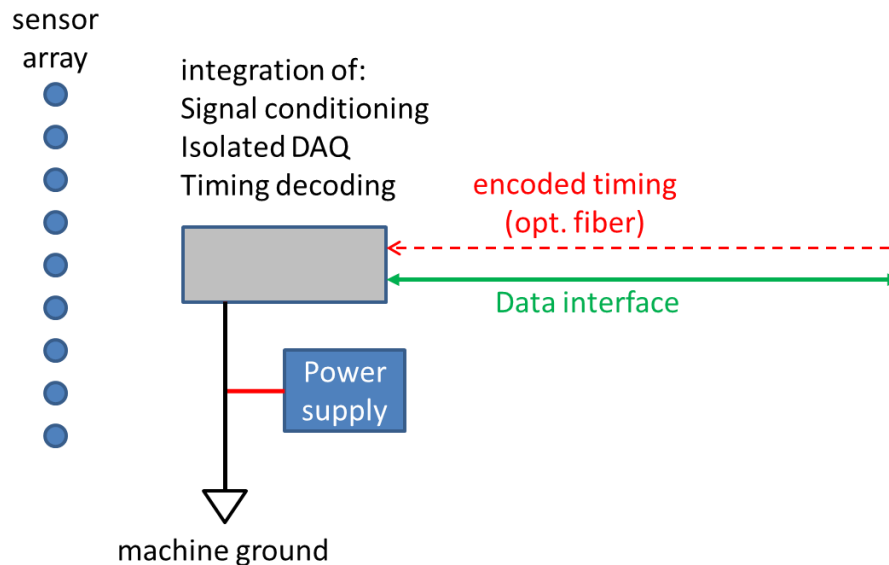


**Figure 1-2 Current situation, typical diagnostic wiring**

In Figure 1-2 Current situation, typical diagnostic wiring a real world example is shown, the cabling of fast magnetics and Langmuir probe systems (taken out from the trays for maintenance) on RFX-mod.

## 1. 4. The new proposed approach

The new proposed approach:



**Figure 1-3 The new proposed approach with simplified wiring.**

Beside the extreme simplification of the layout two other important benefits can be achieved:

- a) a short electrical connection between the transducers (typically less than 5 or 10 meters, depending on the application) signals and the Daq system;
- b) the isolation of all I/O channels, including data interface and triggering signals.

The interface and cabling requirement are drastically reduced, being needed only the power supply, the analog signal inputs/outputs, the data interface (Ethernet (100M/1G)).

For less demanding timing requirement the timing distribution can be accomplished without fiber, using PTP protocol (IEEE 1588) over the Ethernet connection.

The system should be realized in order to accommodate the additional custom electronics needed for signal conditioning, which is application dependent.

In the following the specification for two kind of system are listed.

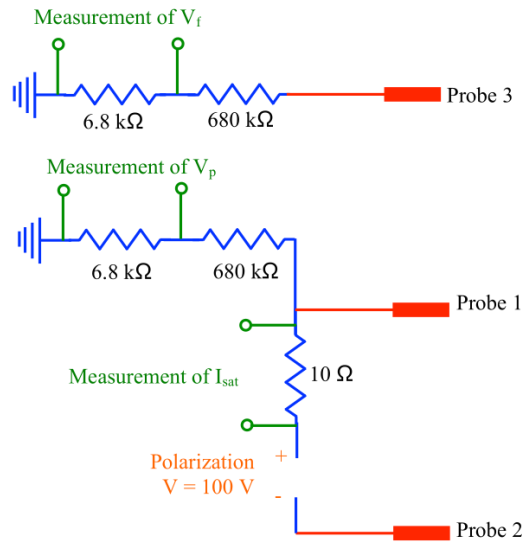
minimum requirement (optimal requirement)

Both systems should be modular and to be mounted either stand alone or inside an Eurocard 3U rack.

## 1. 5. Daq system for array of 5 pin balanced electrostatic probe (or fast magnetics)

**Specification for optimal acquisition system of one 5 pin balanced electrostatic probe.**

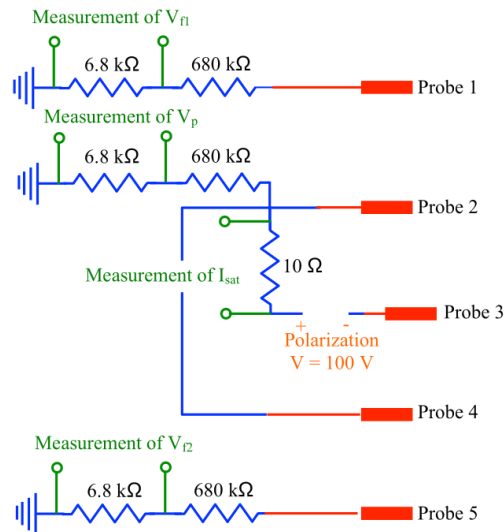
The 5 pin balanced probe is an extension of the basic triple Langmuir probe configuration. The triple probe consists of three electrodes immersed into a plasma which are polarized according to Figure 1-4 [20]



**Figure 1-4: Triple Langmuir probe configuration**

Three simultaneous measurements are taken: the current ( $I_{sat}$ ) flowing from probe 1 to probe 2, and the two voltages of probe 1 ( $V_p$ , plasma potential) and probe 2 ( $V_f$ , floating potential). From these three measurements it is possible to obtain an estimate of the local electron temperature ( $T_e$ ) and electron density ( $n_e$ ) of the plasma [21].

The 5 pin balanced probe (see Figure 1-5) is a refinement of this basic configuration:  $V_p$  and  $V_f$  are splitted onto two pins, so that their value can be on the average referred to the central spatial position, where the current measurement is taken, in order to reduce measurement errors due to non uniform distribution of the plasma density and temperature.



**Figure 1-5: 5 pin balanced Langmuir probe configuration**

Moreover the  $V_f$  are taken as two separate measurements which are also used to compute the electric field  $E=(V_{f1} - V_{f2})/d$ , which can be used in a magnetized plasma to estimate the transverse drift velocity:  $v_{\perp}=(E \times B)/B^2$ .

From the electrical point of view these measurements are not easy because very often the ground reference for the plasma is different from that of the acquisition system and in some conditions the plasma generates voltage spikes of 1 kV or more, which can damage the electronics. Thus a reliable measurement of these probes requires a galvanically insulated input.

The requirements can be summarized in the following list:

5 isolated  $\geq 2$ kV ADC channels :

1 - 10 (20) Ms/sec

14 (16) bit resolution

Sync between different devices:  $\leq 10$  ns

Direct clock / trigger recovery from single optical fiber input:

MPB RFX-mod

Asdex timing

W7-X timing

acquisition time window : 100 ms – 5 s

(repetition rate TBD, for W7-X depends on the ability to send event data)

(event recording with post trigger, for W7-X)

(real time output of reduced quantities)

Ethernet connection for data readout and device control

Possible use of additional interface for slow digital or analog signals via SPI for monitor and control of additional devices (e.g. probe biasing power supply).

## **1. 6. Daq system for general purpose magnetics, soft X-Ray detectors, PMT, CCD, arrays.**

**Specification for optimal acquisition system of small arrays (usually from one single chip sensor) of photodiodes or photomultipliers (PMT).**

4 or 8 isolated  $\geq 2\text{kV}$  ADC channels

0.1 - 1 (2) Ms/sec

18 (20) bit resolution

i.e. an interface to support for existing ADC modules designed by IST

Sync between different devices:  $\leq 200$  ns

Clock/trigger through Ethernet, using PTP Ethernet and a dedicated protocol:

Direct clock/trigger recovery from single optical fiber input:

MPB RFX-mod

Asdex timing

W7-X timing

Acquisition time window : 100 ms – 5 s

Real time Ethernet output of averaged signals.





# Chapter 2.

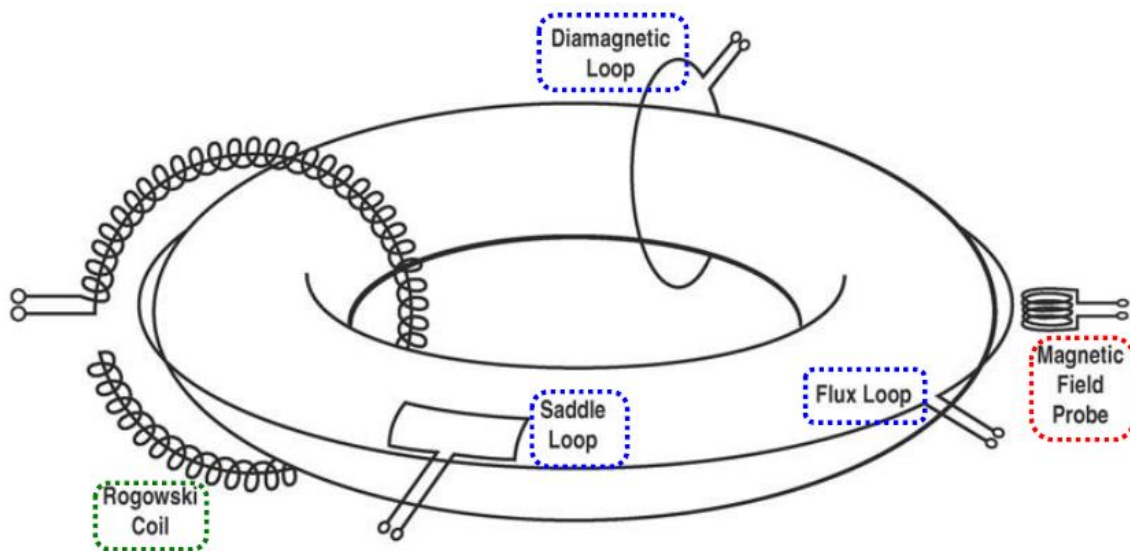
## Definition and roles of magnetic diagnostic in a fusion machine

**Magnetic diagnostics are instrumentations used for measurements of:**

- magnetic fields outside the plasma
- electric currents that are the source of the magnetic fields

**Magnetics measurements are crucial in fusion devices for:**

- Machine protection and basic plasma control 'plasma current', 'plasma position', 'plasma shape', 'magnetic field errors', 'halo currents'
- Advanced plasma control 'MHD instabilities control', 'Plasma current density profile', 'equilibrium magnetic field distribution'
- Physic studies and performance optimization 'High frequency MHD instabilities'

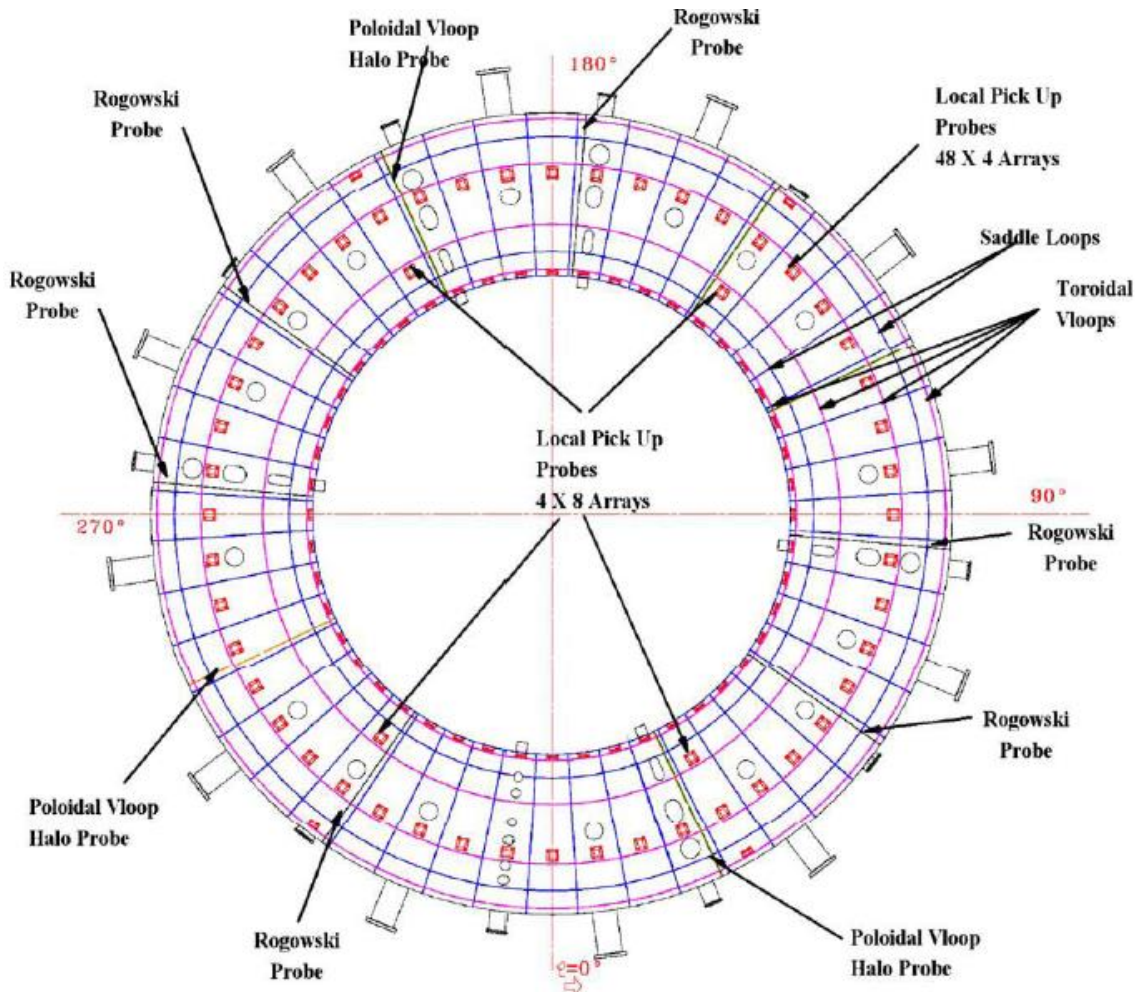


**Figure 2-1 inductive sensors in a toroidal machine**

The image shown the field of application and positioning of the probes in a toroidal machine.

Inductive sensors: (measurement)

- Extended loops ( $V$ ,  $\langle B \rangle$ )
- Localized probes ( $B$ )
- Rogowski coils ( $I$ )

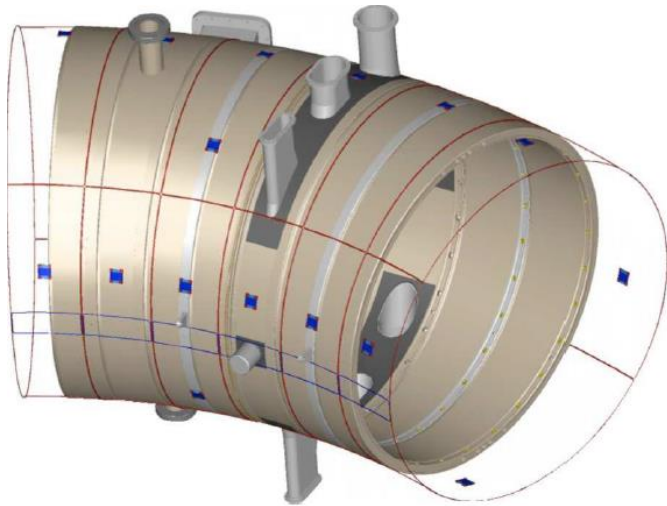


**Figure 2-2 RFX-Mod external magnetic probes layout.**

- 8 continuous flux loops
- 6 Rogowski coils (for plasma current measurement)
- 5 diamagnetic loops
- 32 partial poloidal loop voltage (Halo Currents)
- 48 x 4 saddle loops (full vessel coverage)
- 48 saddle loops (error fields at gap)
- 48 x 4 bi-axial pick-up coils (poloidal & toroidal)
- 32 bi-axial local coils (poloidal & toroidal)

In the very narrow space between the vessel and the shell, a uniformly distributed system of 220 thermocouples and 700 magnetic transducers has been designed to reconstruct the complete map of the energy deposition and of the magnetic field. The frequency bandwidth of the magnetic transducers are limited to few tens of kHz due to the shielding effect of the vacuum vessel and the sensitivity of the pick up coils is sufficient to discriminate a signal less than 1 mT whereas the maximum expected field is in the order of 0.8 T.

The magnetic transducers will also provide the feedback signals for the active control of the MHD modes.



**Figure 2-3 Overview of magnetic probes on RFX-Mod**

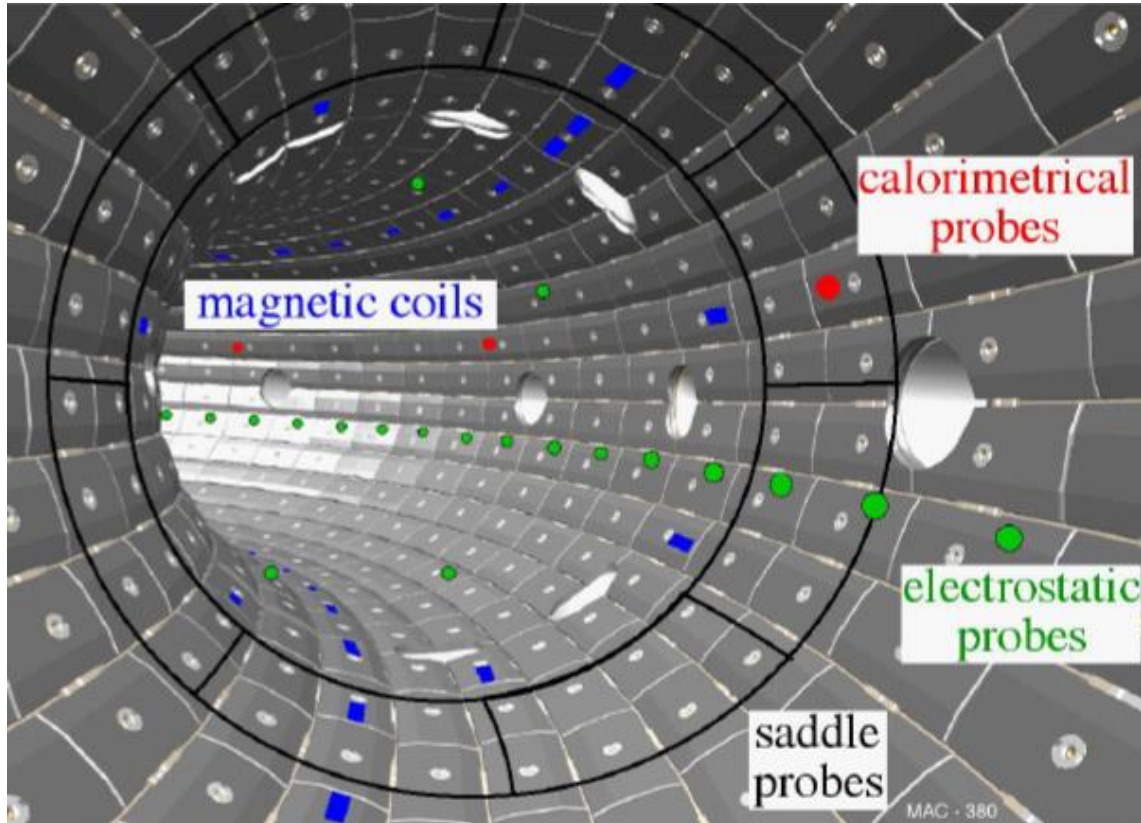
The magnetic probes, in the current RFX-mod, are 139 inside the vessel and 739 ex-vessels. A complete refurbishment/upgrade is presently under study for a major modification of the RFX machine.

In the Figure 2-3 is shown the 32 partial poloidal voltage probes for halo currents



**Figure 2-4 Integrated System of Internal Sensors (ISIS) for RFX-Mod**

In the Figure 2-4 is shown the first wall, that is, the inner surface of the vacuum chamber immediately in contact with the plasma: it is entirely covered by a 2016 graphite tile system, which resist temperatures up to 3000 °C. On the inner equatorial line, a set of magnetic probes are visible. These are a subset of the 139 inside the vessel.



**Figure 2-5 View of RFX-mod first wall with probes layout.**

- Blue: Magnetic pick-up coils: 131
- Black: Saddle flux loops: 8
- Green: Electrostatic probes: 97
- Red: Calorimetric probes: 8

During the machine upgrade in 2004 a close fitting 3 mm thick Copper shell has replaced the previous 65 mm Aluminum shell.

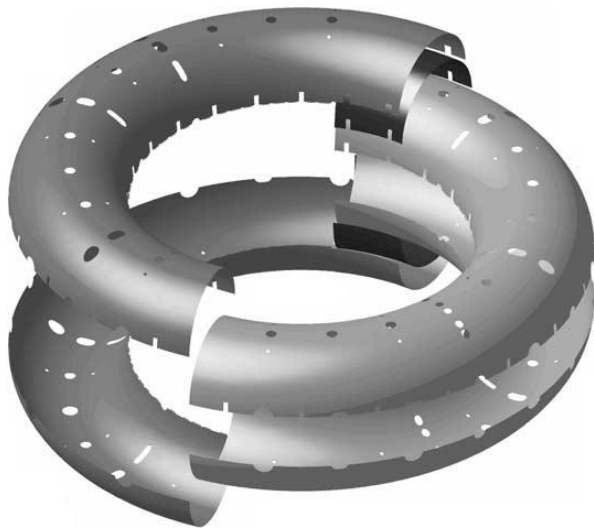
The modification is part of a set of enhancements aimed at optimizing active control of the MHD and resistive wall effect to induce plasma mode rotation and to prevent mode phase locking to prevent non axisymmetric configuration of plasma column.

The shell has one poloidal gap with a toroidal overlap of 23° (Figure 2-6).

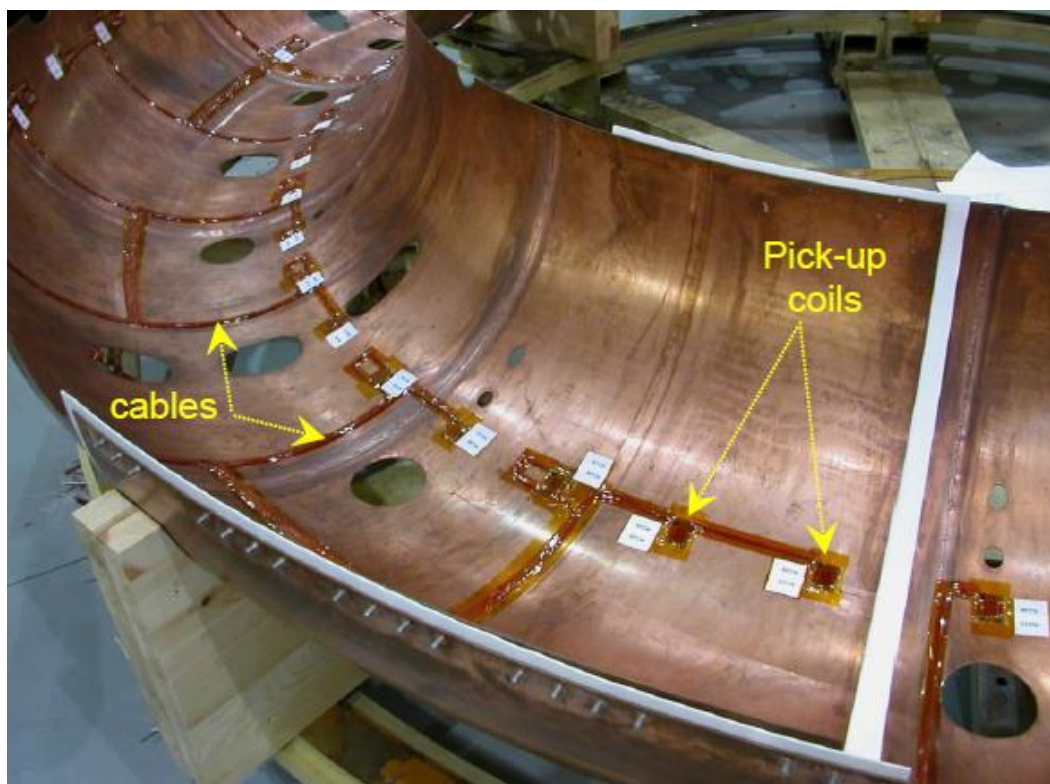
The outer equatorial gap is short-circuited with bolted copper plates. The shell is clamped to the vessel through stainless steel bands that sustain the electrodynamic forces arising in the shell.

Precise field reconstruction requires accurate calibration and accurate alignment of the probes to minimize the spurious effect of unwanted components.

The probe distribution must be chosen carefully to avoid such aliasing effect, due to conducting structures, which influence the field spatial components typical of the plasma modes.



**Figure 2-6 mm copper shell. Exploded view.**



**Figure 2-7 Biaxial magnetic probes RFX installation**

To reduce shielding effects, probes must be installed inside the stabilising shell.

The three components of field outside the vacuum vessel can be very different in amplitude.

At the same time, one can reach 0.8 T on one component and the other can be of the order of some mT. Furthermore their time variation can be very fast.

In general the probes installed have to guarantee an uncertainty less than 1 mT to correctly reconstruct the plasma behaviour.

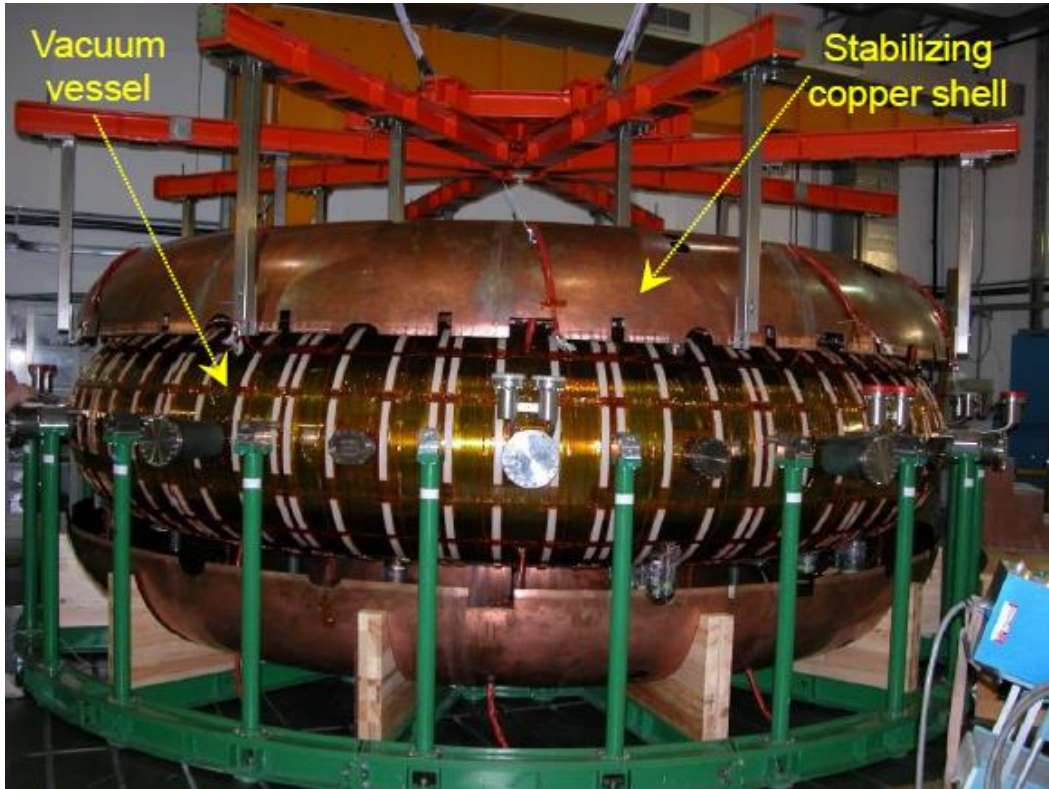


Figure 2-8 RFX-Mod copper shell



Figure 2-9 Magnetic probes connectors on RFX vacuum chamber

In the Figure 2-9 is shown the position and the size of the probe connector on the top side of the copper shell in RFX-Mod experiment.

The whole system is composed by 4 toroidal arrays of 48 bi-axial (Bt & Bp) probes each (4 x 48). Then an array of 4x48 saddle coils for the measurement of radial field component. The cables from the probes are grouped in 12 sections, each with connectors.



**Figure 2-10 Biaxial probe LEMO connectors**

Each probe uses a twisted pair and two pin on the connector.

On RFX mod there are four pairs of connectors like the ones shown in Figure 2-10 for every thirty degrees of the vessel.

The two connectors of each pair are of different diameter and different number of pin, this is not an optical effect due to the perspective of the photo.





# Chapter 3.

## Probe and transmission line specification

### 3.1 Magnetic probe circuit model

In this section is analysed a two axes magnetic sensor used on the toroidal device RFX in Padua.

The sensor was produced and tested by Laboratorio Elettrofisico s.r.l. Via G. Ferrari, 14-20014 Nerviano (MI) Italy.

A large number of probes are required to correctly identify the complex spatial structure of the plasma column.

An accurate calibration and a careful probe mounting is required.

Probes are installed inside the stabilising shell, placed very close to the vessel.

To allow a more detailed study of plasma behaviour, the electromagnetic diagnostic was split to two parts. The first is internal to the vacuum vessel and is integrated with an extended set of electrostatic and calorimetric probes. It is devoted to the study of turbulence and fast MHD phenomena not easily observable from the outside of the vessel due to its shielding effect. The second is placed between the vacuum vessel and shell. It is devoted to the study of global plasma parameters and low frequency MHD phenomena, which largely affect the plasma equilibrium.

The two parts of the diagnostics have been designed in order to be complementary and provide a complete and detailed set of information on RFX magnetic configuration.

The specific sensor is the number 244 installed.

<b>Kind: Magnetic probe</b>	<b>Sensor number on machine: 244</b>
number of internal windings	260 (270 from technical specifications)
number of external windings	214 (220 from technical specifications)
internal winding resistance	23 $\Omega$
external winding resistance	21 $\Omega$
internal winding inductance	0.23mH
external winding inductance	0.32mH
Probe area	0.021m <sup>2</sup>

**Table 3-1 Magnetic probe technical characteristics.**

Probes are realised by winding two independent and orthogonal windings on the same core, leaving four corners free (Figure 3-6).

Each winding is made with enamelled copper wires, and composed of two layers. Special high temperature Kapton<sup>®</sup> tape is used between the two windings and between layers in each windings.

This tape has also a protective function and helps to keep the winding turns in position during thermal cycles.

The core thermal expansion coefficient must to be as close as possible to that of copper, as the probes are bolted to a copper surface.

This guarantees also that the probes are subjected to low stresses due to the thermal expansion of the shell.

At the same moment must be not conductive in order not to reduce the frequency response.

A class of high performance polymers were chosen. Ceramics have been discarded due to their fragility and too low thermal expansion coefficient compared with that of copper. Two materials were selected, PEEK<sup>®</sup> and Torlon<sup>®</sup>.

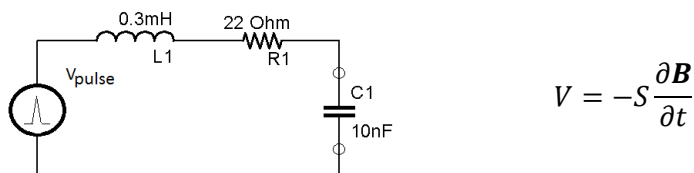


Figure 3-1 Magnetic probe model typical.

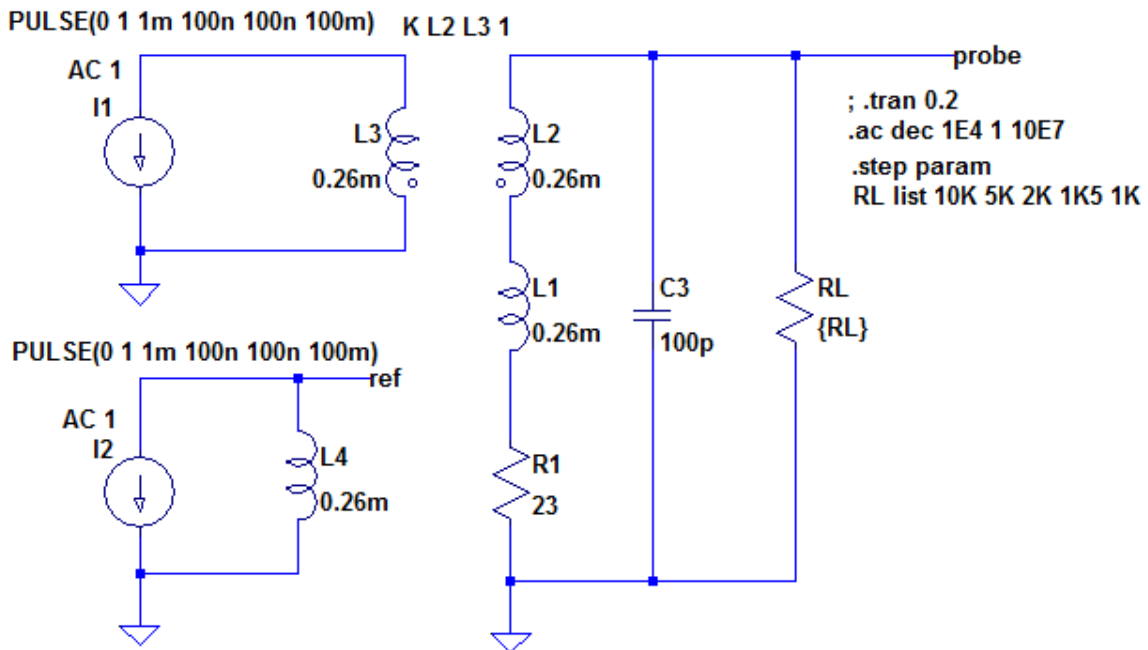
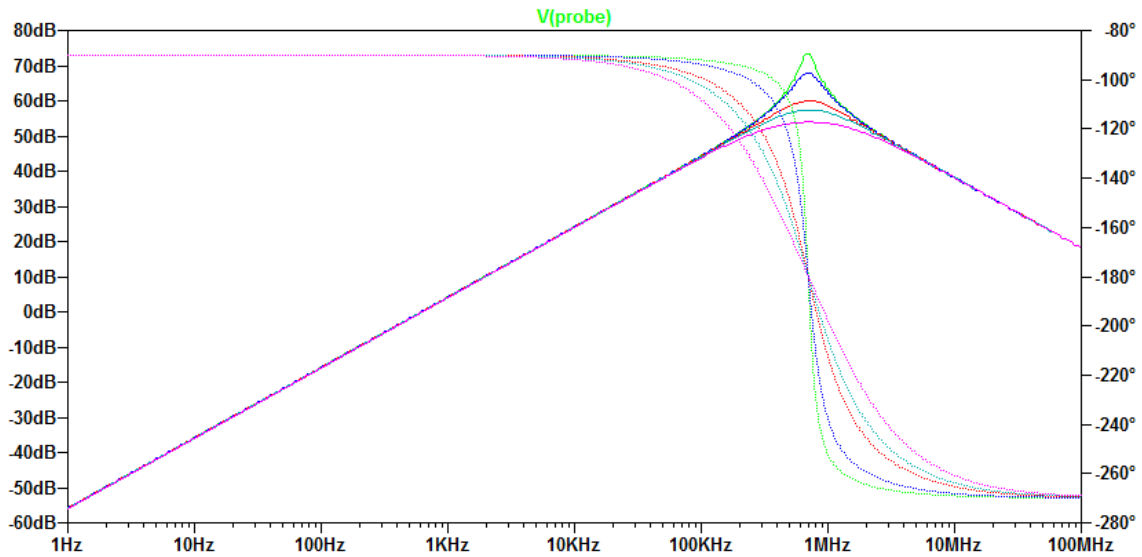


Figure 3-2 Two axes magnetic sensor model.



**Figure 3-3 Derivative of the frequency response two-axis probe.**



**Figure 3-4 Size and appearance of the magnetic probe.**

The probes must react as quick is possible to the variations of all the three components of magnetic field during the pulse, even in the presence of high spatial asymmetries in toroidal direction.

In RFX, due to modes up to  $n=15$  in toroidal direction and mainly  $m=0$  and  $m=1$  in poloidal direction the signals is typically in the dynamic range up of 60dB with a frequency spectrum of several kHz.

Probe bandwidth is limited by internal resonances of the two coils, which result around 200 kHz for both the coils.

A further design specification for the sensors is due to the maximum operation temperature of the vacuum vessel (200 °C).

The RFX probes have to guarantee an uncertainty less than 1 mT to correctly reconstruct the plasma behaviour.

The latest ones installed should allow to study local behaviour of the plasma and to operate the new saddle coils.

These specifications are particularly stringent and require an accurate calibration and a careful probe alignment to minimize the spurious effect of unwanted components.

According to the Sampling Theorem, a correct reconstruction of the magnetic field from the signals collected by the local probes could be obtained only if at least 144 toroidal X eight poloidal measurement points are used but a so large number of probes are not acceptable.

To minimize the aliasing effect the sensors spatial distribution must be chosen carefully on the machine to avoid the effect of metal structures.

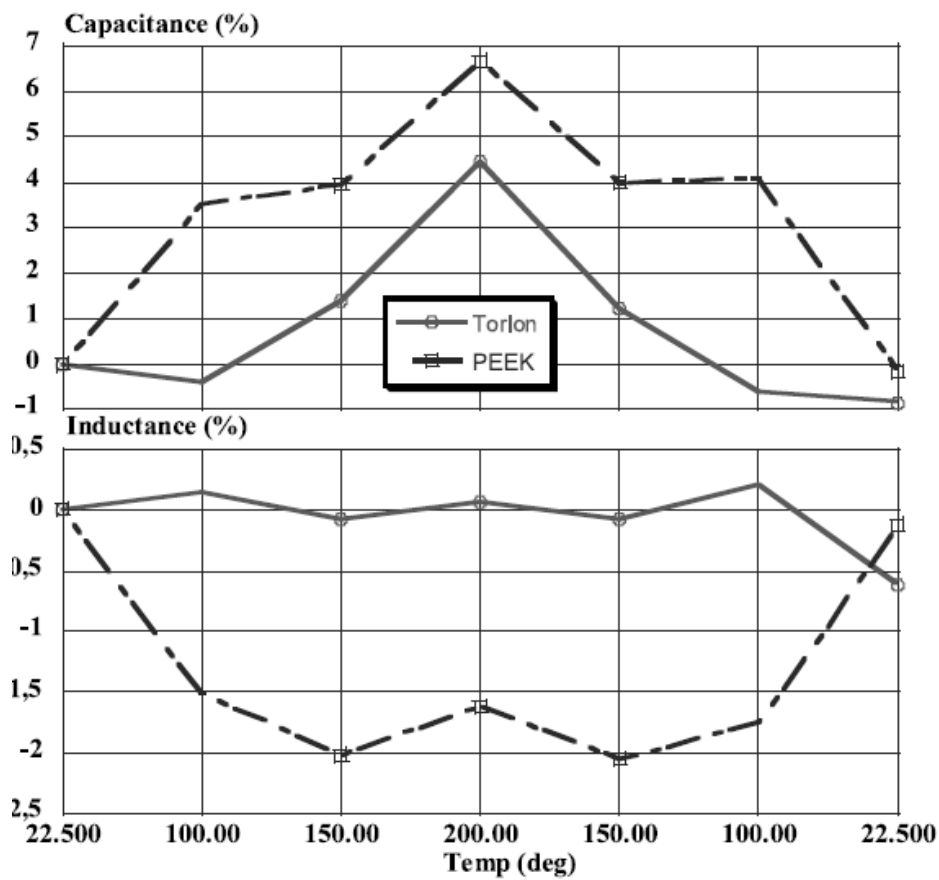
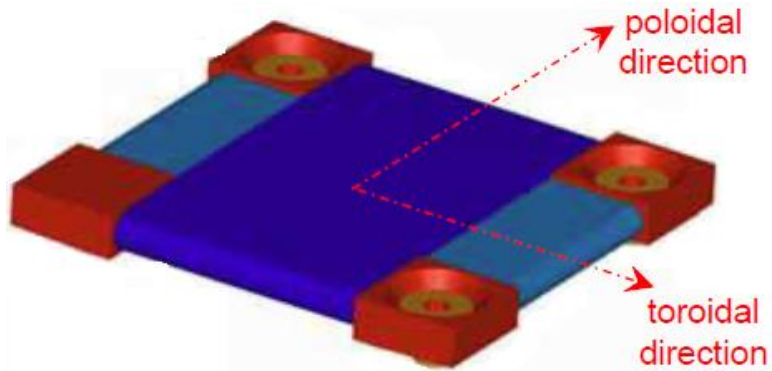


Figure 3-5 Torlon Vs Peek core probe capacitance and inductance.

In Figure 3-6 Bi-axial magnetic pick-up coils, is shown how the probes are realised by winding two independent and orthogonal windings on the same core, leaving four corners free. Each winding is made with enamelled copper wires, and composed of two layers. Three of the corners are drilled with holes, so that the probe can be bolted to the internal surface of the new shell.

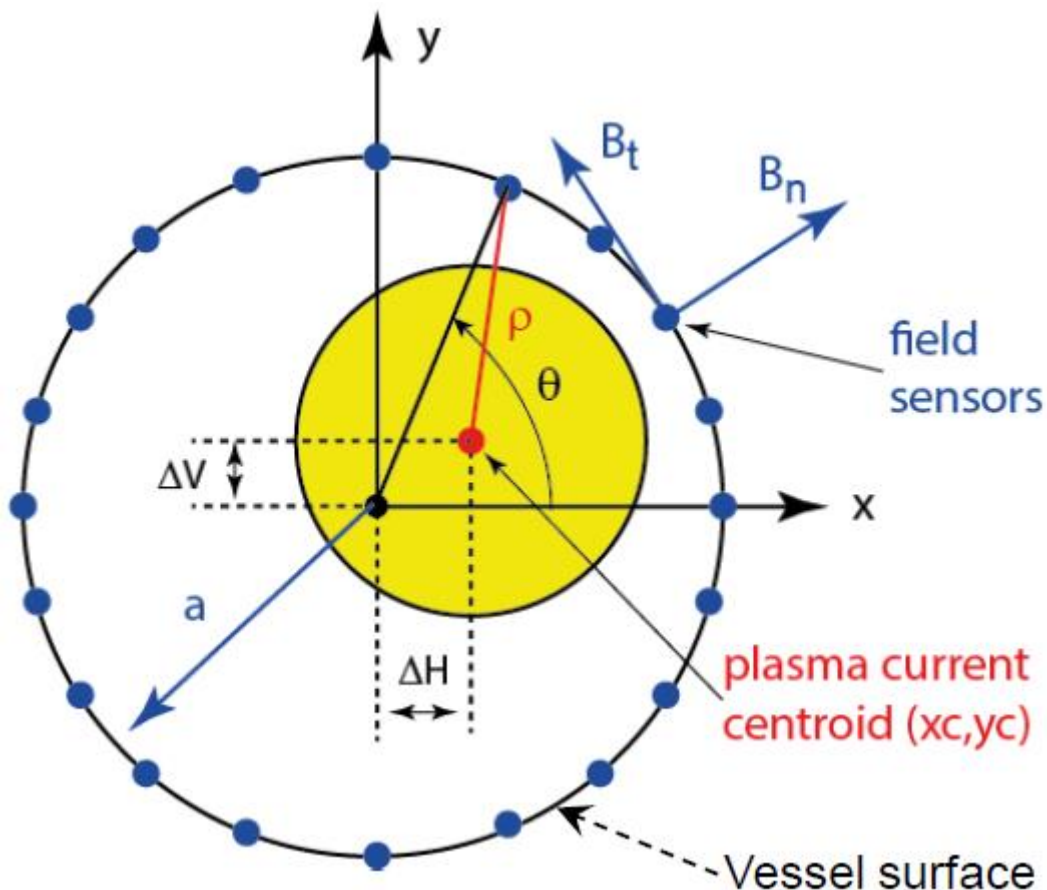


**Figure 3-6 Bi-axial magnetic pick-up coils**

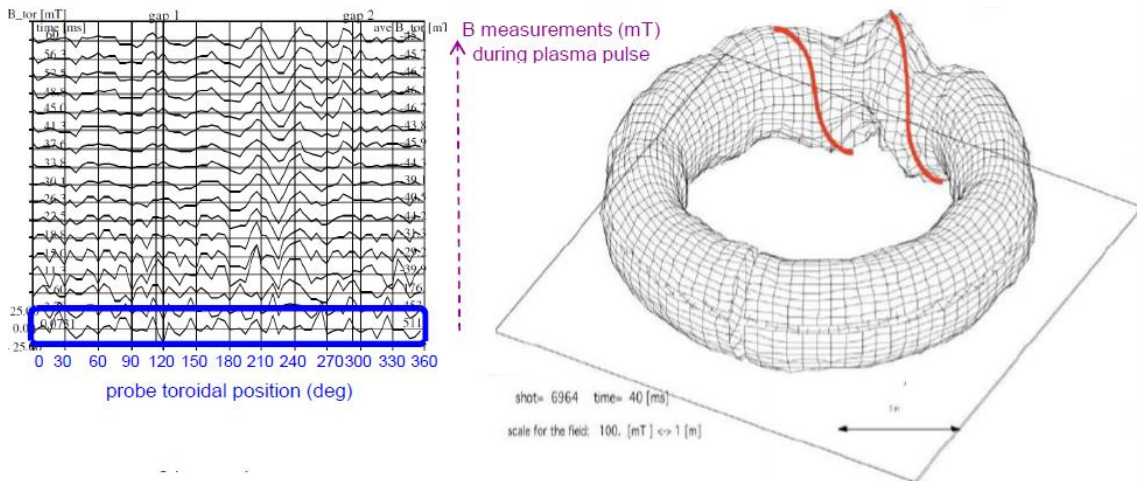
The fourth corner is not drilled and is useful to define the correct mounting orientation. In order to minimise the radial size of the probe one of the major sides of the probes is carefully machined to fit exactly with the curved surface of the vessel.

These two axis pick up probes, are placed on the machine close to the axis of saddle loops, which measure the radial field, are intent to measure local toroidal and poloidal fields.

The radial field measurement results averaged per the loop area, but this should not alter the measurement of the spatial harmonics of interest.



**Figure 3-7 Plasma position from magnetic measurements.**



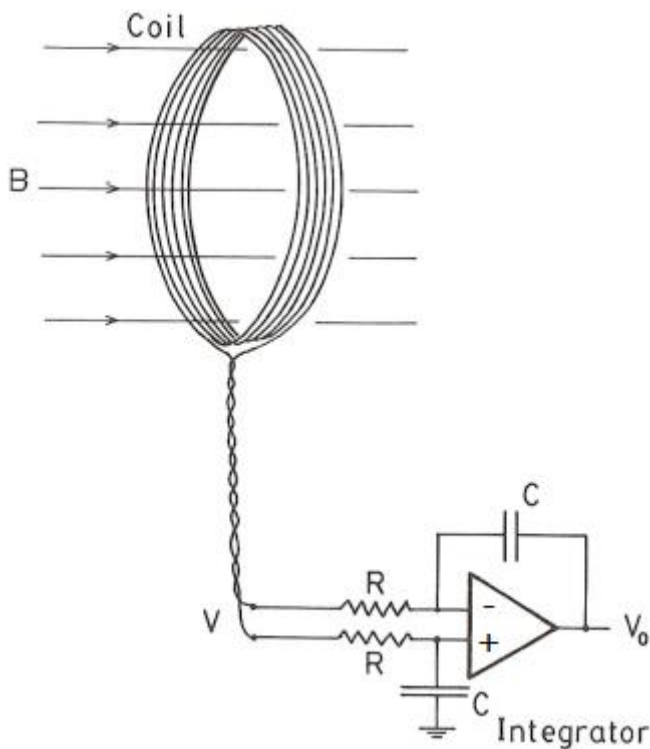
**Figure 3-8 RFX-mod waveform of magnetic field from pick-up array**

The probes acquisition typically occurs like a differential signal as shown in the following diagram.

The operational amplifier, in these principle diagram, is ideal.

In a real application it add some troubles due to the intrinsic noise and the band width.

We will see that the typical noise, present in the low end of the band, between 0.1Hz and 10Hz, or even lower, plays a fundamental role in signal acquisition, especially if extended over ten seconds.



**Figure 3-9 Differential acquisition of magnetic probes.**

From Faraday law:

$$v_{in} = -\frac{d\theta(t)}{dt} = -\frac{d}{dt} \int_A \mathbf{B} dS$$

If  $\mathbf{B}$  is uniform within the loop:

$$v_{in} = NA \frac{dB}{dt}$$

After the integrator:

$$v_{out} = \frac{NA}{RC} B(t)$$

RC is the time constant of the integrator.

A passive RC integrator, placed in the frontend, provides accurate integration of transient that are too rapid for ADC sampling rate.

For slowly varying signals the digital integral dominates and the RC term becomes negligible.

A kind of magnetic probes are the pick-up coils, that is small magnetic sensors, that locally sample  $B_\theta$ ,  $B_\phi$  and  $B_r$ .

Large arrays with hundred of coils are used in fusion machines, placed outside the vacuum vessel.

The plasma shape is reconstructed with high detail by Fourier analysis.

Pick-up coils also provide information on plasma instabilities, MHD activity and magnetic fluctuations.

### 3. 1. Probe electrical modelling

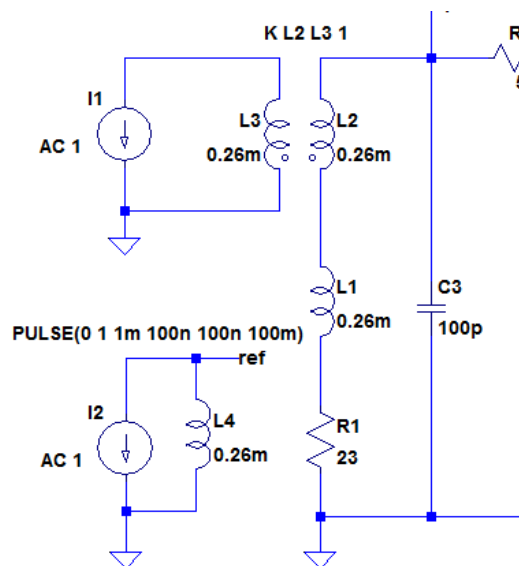


Figure 3-10 Basic model of a magnetic probe

An ideal magnetic pick-up coil is considered to have an output exactly proportional to the derivative of the applied magnetic field, However a proper description of real probe requires a more complete description to take into account its structure and the connected components.

A probe itself is basically an inductance loaded connected to cable and to a detection circuit. The probe itself have some internal capacitance which close the circuit causing

some current flowing into the probe, which reduce the signal amplitude the signal at high frequency.

A probe can be modelled as inductance, in series with a voltage generator (which is proportional to  $dB/dt$ ) a resistance, with the parasitic capacitance in parallel.

A better modelling of the voltage source can be using an idelay coupled transformer driven by a current generator. The equivalent magnetic field is directly represented by the current at the primary. The secondary when open will behave like an ideal probe giving the  $dB/dt$  term, while it can be connected to external component chain to simulate its response on a real system.

Given the circuit shown in Fig. 3-10 the basic scheme of the probe is an RLC circuit connected to a generator. The resonance frequency depends on the value of the capacitor C3, which includes the parasitic capacitance of the probe and the capacitance of the transmission line.

### 3. 2. Transmission line circuit model

Actually the sensors are connected to acquire system by a transmission line typically 40 meters long.

Being the typical impedance of the line about to 120 Ohms, it has a capacitance of 100 pF/m.

It is introduced a line capacitance equal to:

$$Cl = 100 \frac{[pF]}{[m]} * 40 [m] = 4 nF$$

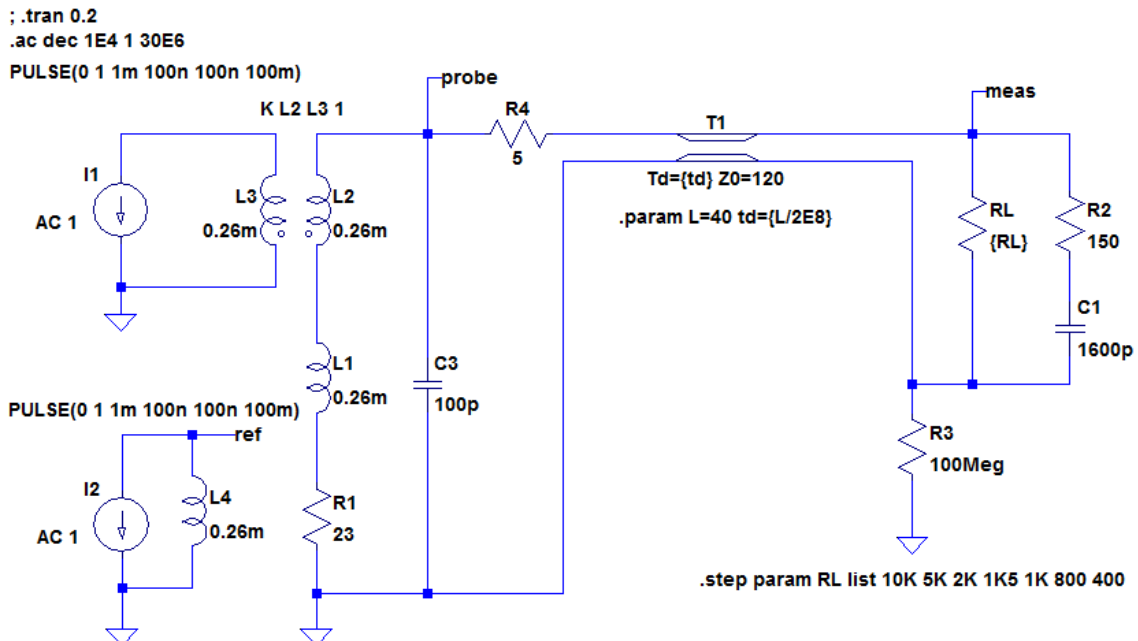


Figure 3-11 Two axes magnetic sensor model connect to transmission line.



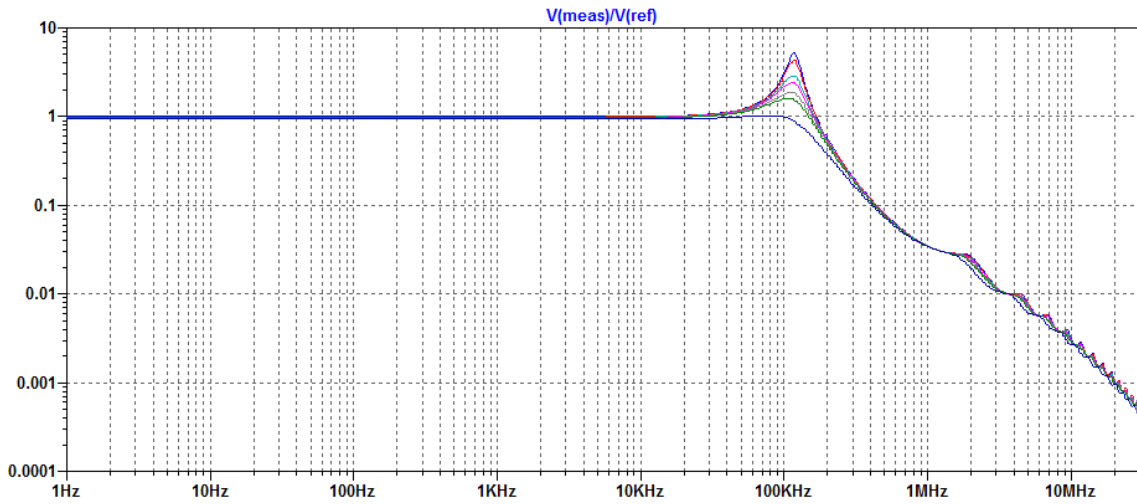


Figure 3-12 Two axes probe and transmission line frequency response.

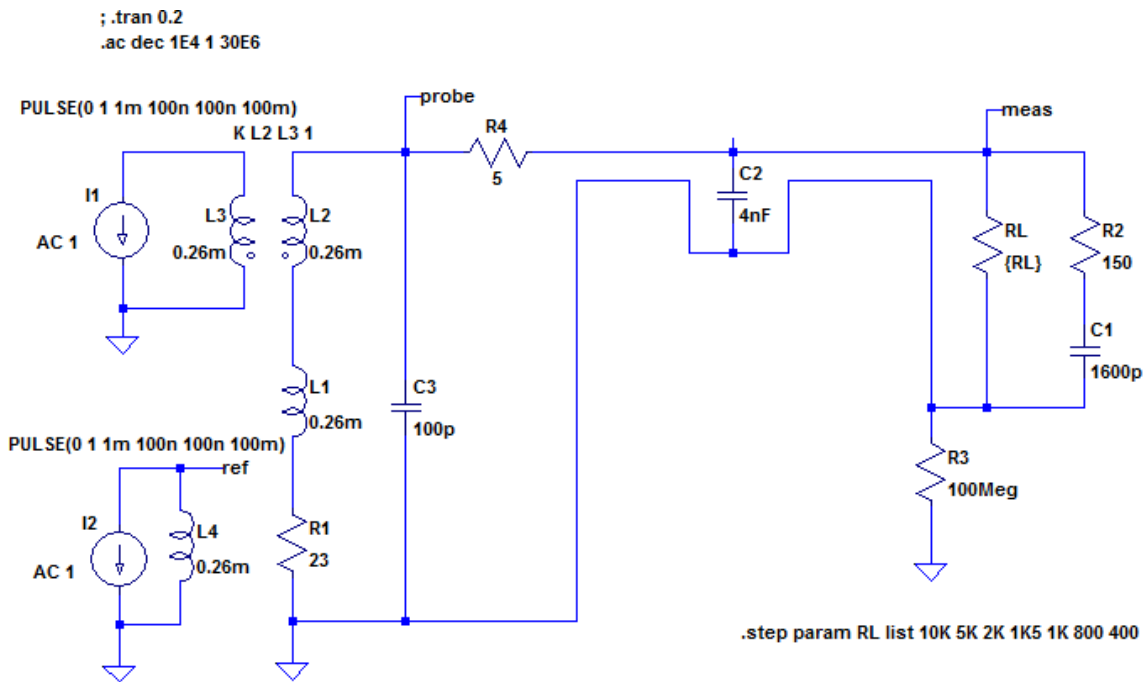
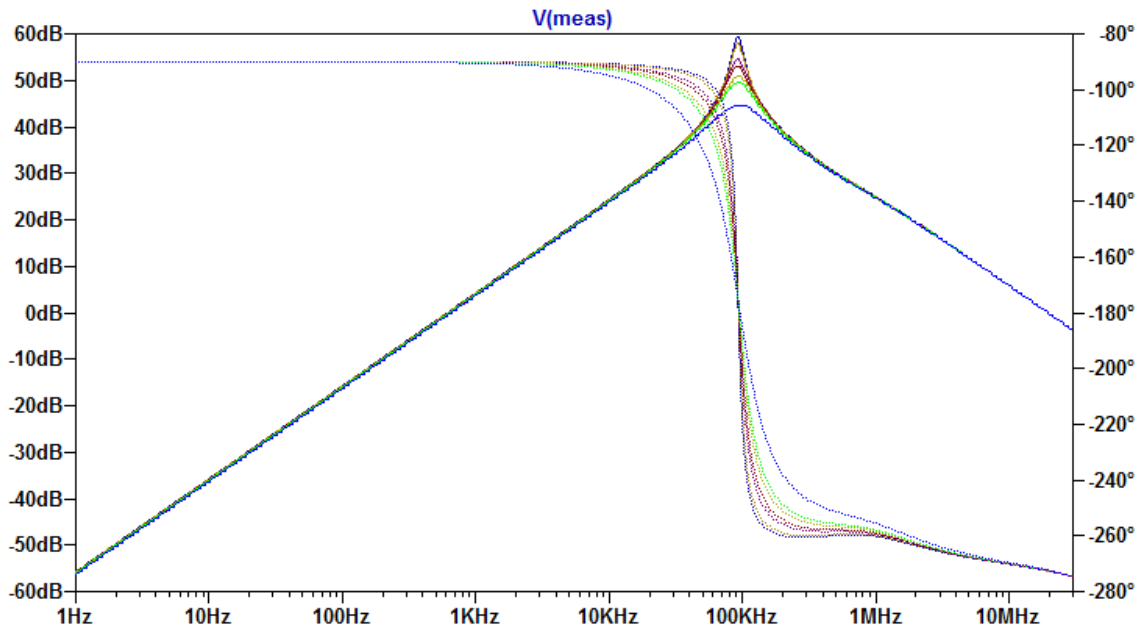


Figure 3-13 Capacitive model of the line



**Figure 3-14 Capacive *transmission line* frequency response**

Here a comparison of two models, one with a transmission line and the other with its equivalent capacitor. In the frequency range of interest (< 1 MHz), the response is the same, so both models can be considered equivalent. At high frequency the line becomes dissipative and the small peak observed can be neglected.

# Chapter 4.

## Analog frontend specification

### 4. 1. Analog frontend to Zynq, the ATCA MIMO ISOL module

The AD conversion is delegated to the chip AD7641 and occurs with a discretization in 18bit SAR converter that acquires signals from a fully differential input in the range  $[\pm 2.048]$  V at the maximum rate of 2 MSamples/s.

The board is configured for serial communication according to the SPI protocol.

The 18 bit conversion is sent to pin 21, SDOOUT.

The signal reaches the insulator IL 711-spin 3. Here is where the galvanic separation occurs from the field.

Downstream of the galvanic separation the signal passes through the SN65LVDT driver, which has an interface with other devices also outside of the board, making the system insensitive to electrostatic and electromagnetic interference.

More can be found in [15] and [16].



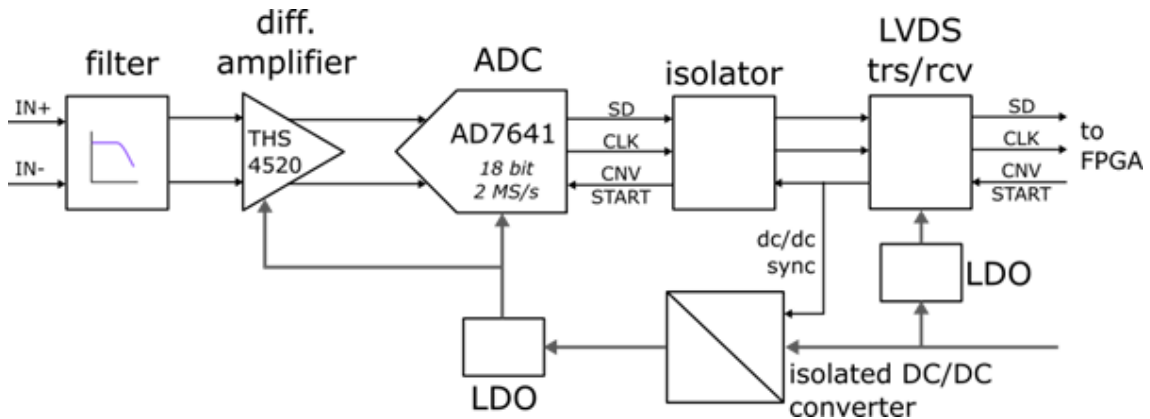
**Figure 4-1 ATCA MIMO ISOL module, isolated differential analog fronted board.**

The input of the module, Figure 4-1, so the part connected to the machine and the magnetic probes, is on the right side, on the Molex 87833 connector.

On the left side i visible the JAE connector, isolated side, which allows connection to the FPGA acquisition system.

The analog input is initially filtered by a 1 poles, 100 kHz passive component connected to a differential amplifier THS4520 used as input range adapter; the AD7641 is configured to operate using serial communication protocol and the digital signals are delivered to the FPGA by means of a digital isolator. The power is supplied through an isolated DC/DC converter which is synchronized with the ADC conversion start

command in order to minimize the noise spikes generated during power switching phase.



**Figure 4-2 Block schematic of the ADC module.**

Observing the block diagram in above figure it is noted that the external connector has two pairs of signals in two pairs of input and output connections.

All of these pass through the LVDS driver in the pin indicated in the figure, on the isolated section of the field that is the one which will refer the masses and power supplies to the Zynq.

<i>LVDS output signals from SN65LVDT</i>		<i>Input control signals SN65LVDT (connector to Zynq)</i>	
<i>SDATA +</i>		<i>CNVST+</i>	<i>K16</i>
<i>SDATA -</i>		<i>CNVST-</i>	<i>J16</i>
<i>SCLK +</i>		<i>RESET+</i>	<i>M14</i>
<i>SCLK -</i>		<i>RESET-</i>	<i>M15</i>

**Table 4-1 Differential analog signals.**

RESET+ is connected to power down signal of the ADC, and the RESET- is connected, through flip-flop, on the SYN signal of the DC/DC converter, named DCV010505, useful for the isolation:

To filter signals (getting more square) and clean them from unexpected noise, it is necessary to going through the Flip/Flop before going to DCV01 that allow the connection towards other devices even off the board.

Setting the SYNCIN pin low the oscillator stops in order to turn off the card when not in use, it reacts in about 2 microseconds.

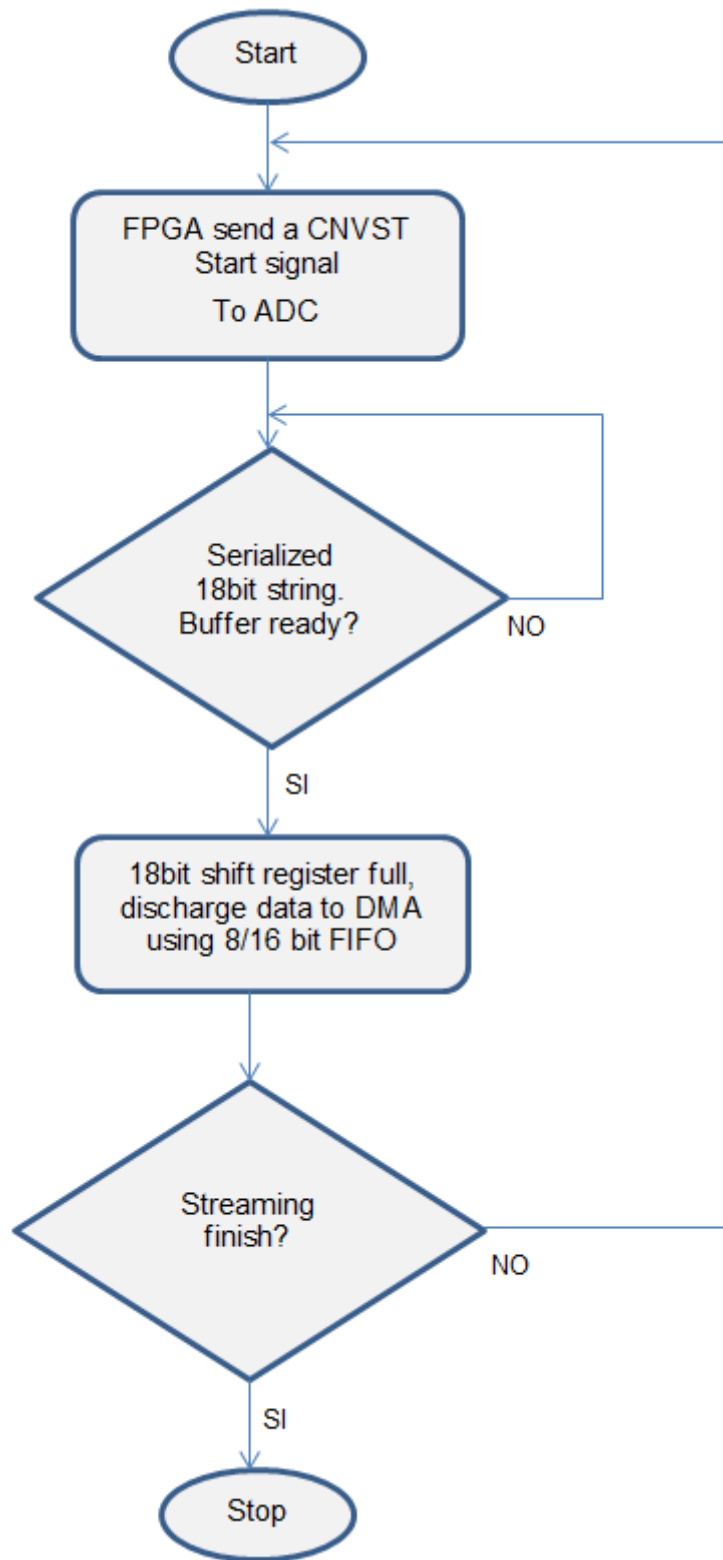
The other signals are passed in single ended and connected to the ADC through opto isolators.

The ADC 7641 is the setting in pinMode [1: 0] = 11 so, from data book, (page 22), the system is placed in the operation mode **MASTER SERIAL INTERFACE** - Internal Clock.

Other settings of the ATCA MIMO ISOL board are Pin WARP = 1 and pin \_NORMAL=0.

This configuration puts the chip at the fastest acquisition speed of 2 MS/s, called WARP mode.

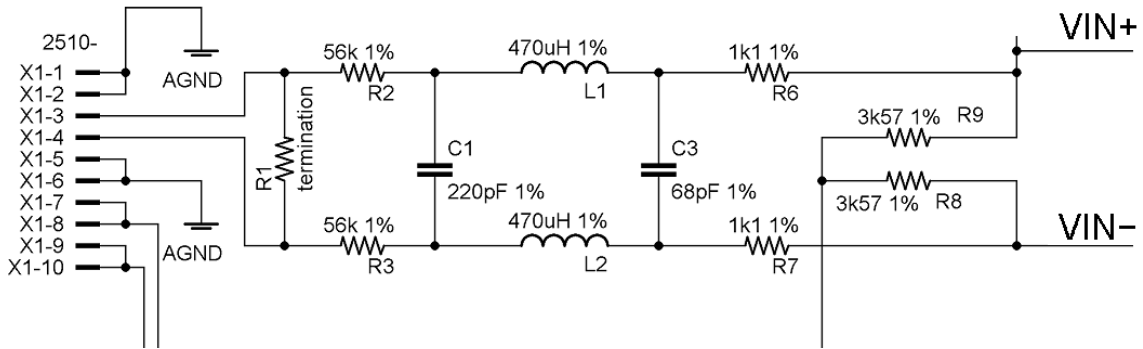
Insights are given on page 15 of the data book, chapter MODES OF OPERATION.



**Figure 4-3: Flow chart of data streaming acquisition to be implemented on the FPGA**

The serial output of the ADC requires a 18 bit shift register in the FPGA to transform the serial data in 32 bit parallel word which can be handled by the processor or DMA.

## 4. 2. Frontend Low Pass Filter



**Figure 4-4 Low pass filter schematic.**

The input stage is a passive low pass filter with a diagram in the Figure 4-4. The spice modeling leads to the tracing of the bode diagram in the Figure 4-7.

The resistive divider, composed of R9 and R8, is powered by the reference voltage generator shown in the Figure 4-12.



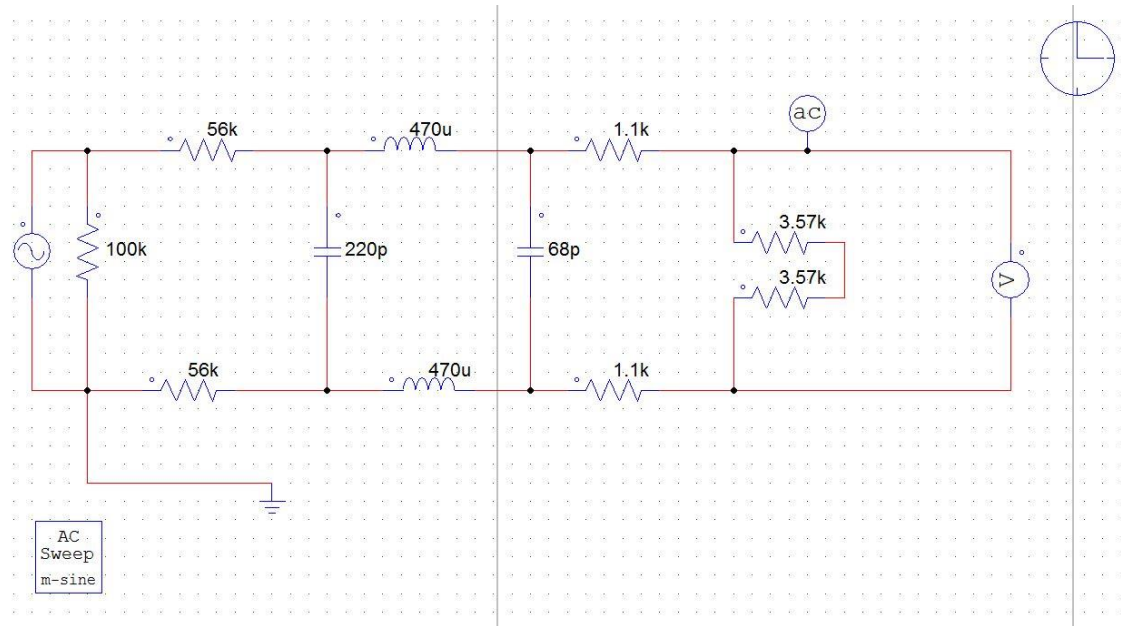
**Figure 4-5 Molex 87833, probe connector**

in Figure 4-5 Molex 87833, probe connector the input connector to which the magnetic probes are connected is shown. This is the 8-pin Molex 87833 model.

In the official diagrams, issued by the module manufacturer, it is erroneously reported to 10 pins.

A careful circuit analysis suggests that the feedback shown in fig 4-11 could be disadvantageous from the noise point of view during long-term integration processes.

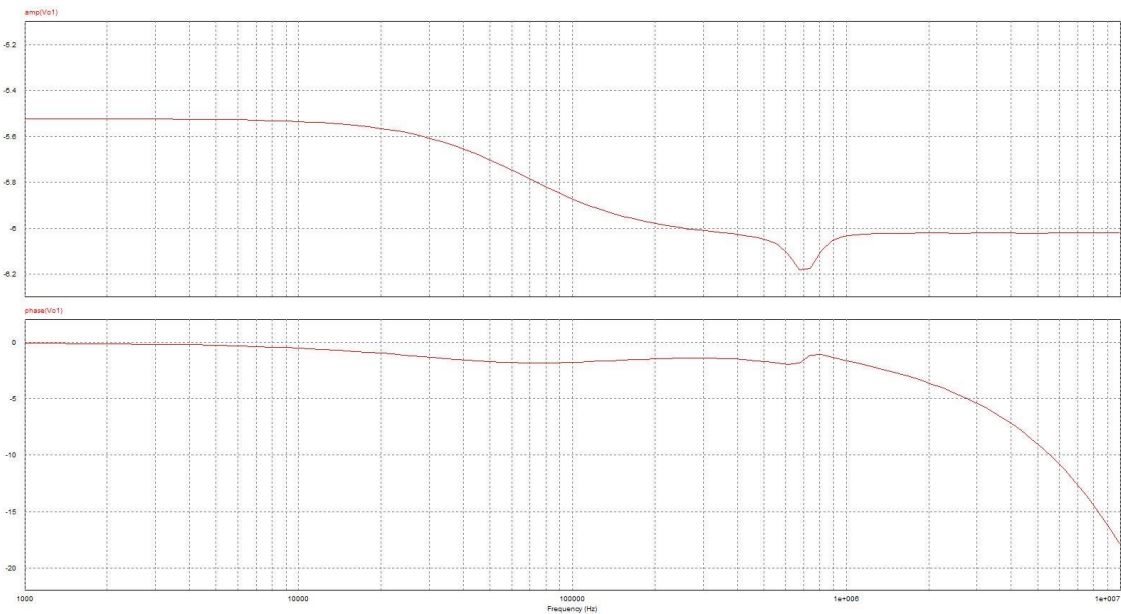
Moreover, the operational amplifier used as a differential buffer in the frontend already has a reference generation option, making it possible to eliminate all the circuit part of Figure 4-12.



**Figure 4-6 Low pass filter PSim.**

Spice model of the low pass analog filter at the ADC module input.

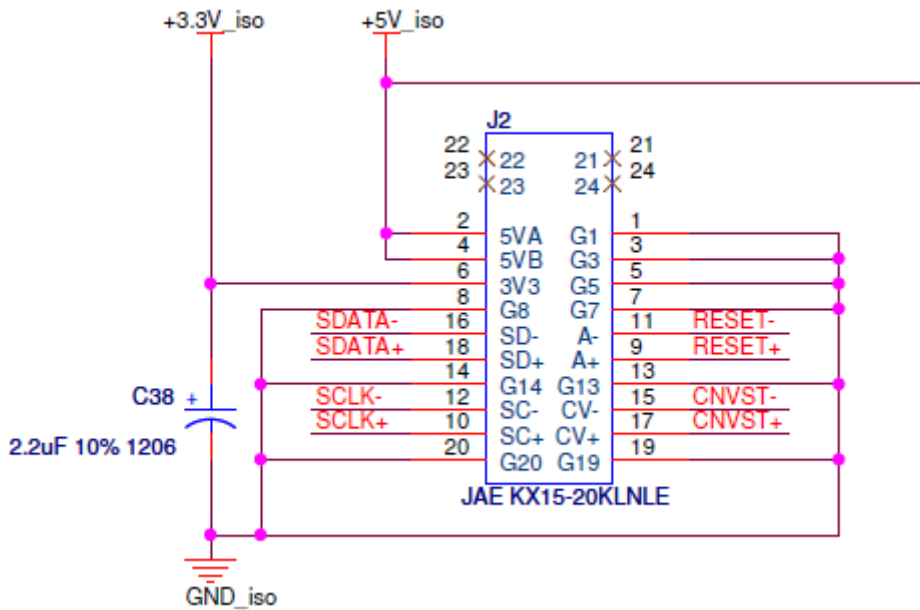
For a correct simulation of the circuit it is necessary to inject in the middle of the resistive divider -1.25V generated by the operational IC2A of Figure 4-12.



**Figure 4-7 Low pass filter Bode diagrams.**



### 4. 3. Frontend to FPGA connector

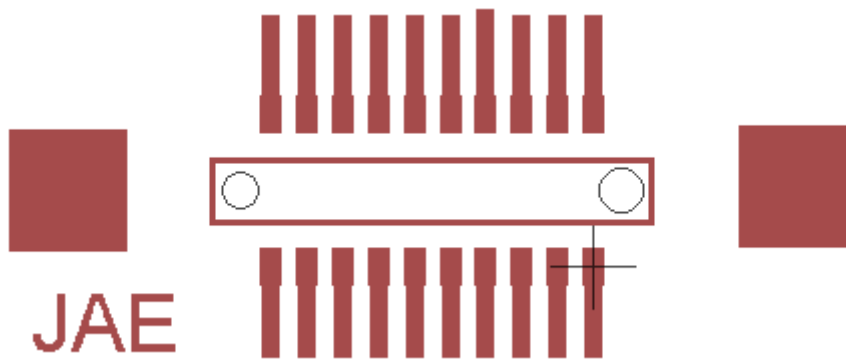


**Figure 4-8 Analog frontend to FPGA connector.**

The FPGA controller is connected to the ATCA-MIMO-ISOL module via a 20 pins connector, not 24 like shown in official schematic. The correct item signature is KX14-20K5DLE.

It is product by Japan Aviation Electronics industry.

The Eagle CAD, used to develop the interface board do not have the component in library, so it was custom drawing, the foot print is in Figure 4-9.



**Figure 4-9 JAE connector footprint.**

#### **4. 4. Opto-isolated differential input stage**

The THS4520 is a wideband, fully differential operational amplifier designed for 5-V data acquisition systems.

It has low noise at  $2 \text{ nV}/\sqrt{\text{Hz}}$ . The slew rate is  $570 \text{ V}/\mu\text{s}$ , and with a settling time of 7 ns to 0.1% (2-V step), it is ideal for data acquisition applications.

It is designed for unity gain stability.



**Figure 4-10 Differential operational amplifier THS4520.**

The THS4520 is offered in a Quad 16-pin leadless QFN package (RGT), and is characterized for operation over the full industrial temperature range from  $-40^\circ\text{C}$  to  $85^\circ\text{C}$ .

To allow for dc coupling to ADCs, its unique output common-mode control circuit maintains the output common-mode voltage within 0.25 mV offset (typical) from the set voltage.

The common-mode set point defaults to mid-supply by internal circuitry, which may be over-driven from an external source. However in the ADC module these features are not used, significantly increasing the input section noise.

The input and output are optimized for best performance with their common-mode voltages set to mid-supply. Along with high performance at low power supply voltage,

this makes for extremely high performance single supply 5-V and 3.3-V data acquisition systems.

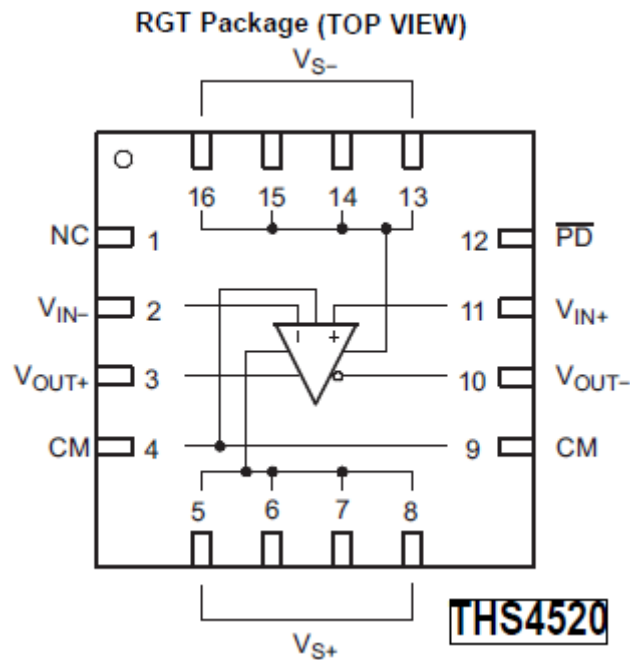


Figure 4-11 THS4520 internal configuration.

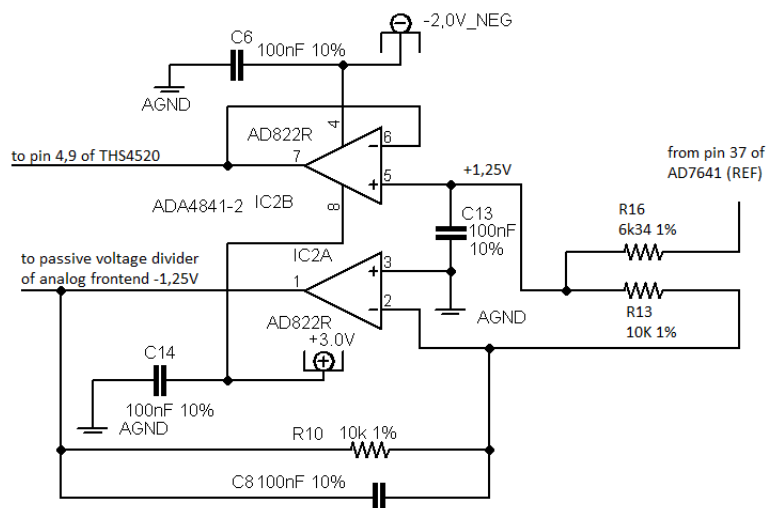


Figure 4-12 Active voltage reference generator

in the Figure 4-12 a schematic of voltage feedback is generating a reference for the THS4520.

Analyzing the circuit of the ADC it is noted that the pins 47 PDREF and pin 48 PDBUF are both grounded, the effect is to enable the voltage output to pin 37 REF.

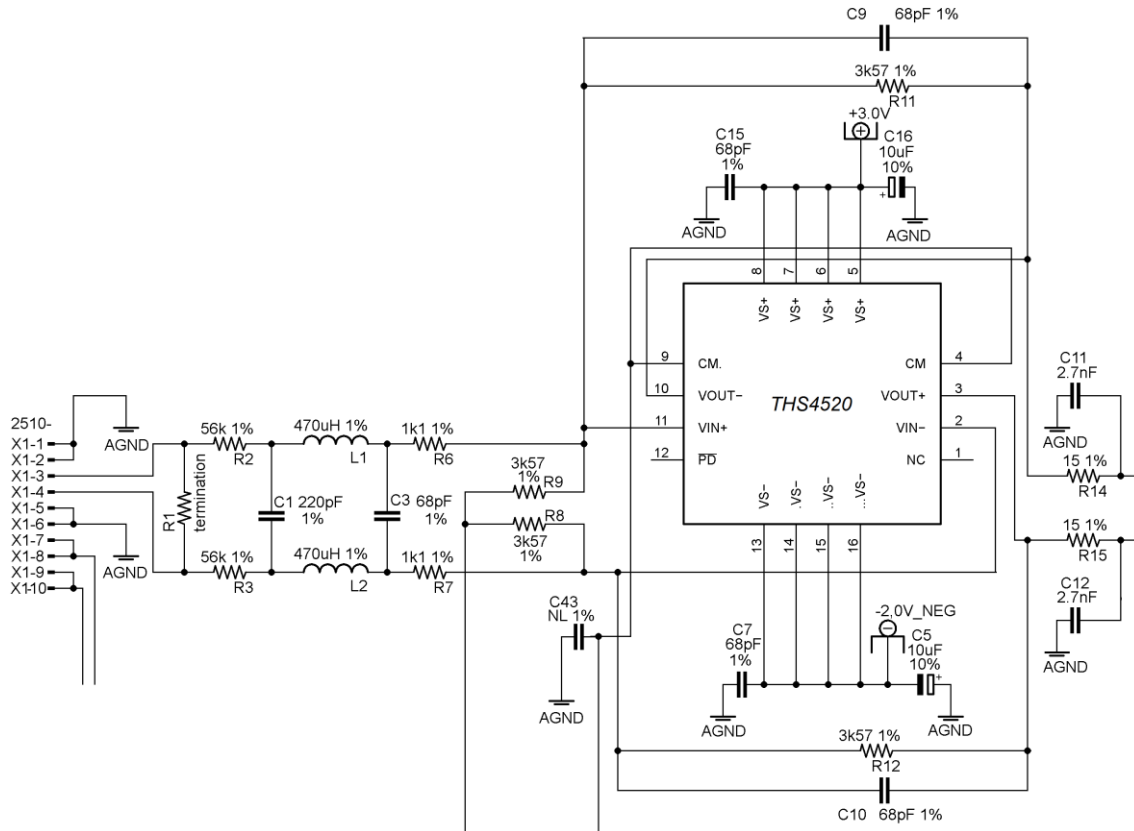
The produced voltage is equal to 2.048V with high stability.

The voltage divider composed of R16 and R13 creates a voltage of + 1.25V which is then sent to pins 4 and 9 of the differential buffer Figure 4-13.

Instead, the IC2A section is inverting and produces a negative reference at -1.25V at the output, which is used as a shifted zero reference voltage for the THS4520.

This configuration leaves free the gain setting (R11/R9 and R12/R8) for THS4520, giving some flexibility in the choice of the amplification during manufacturing phase.

However this configuration, as explained in Chapter 12, adds a significant amount of low frequency 1/f noise, which degrades the module performances when used for numerical integration of the acquired signals, as in the case of magnetic probes.



**Figure 4-13 Differential amp. op. schematic on insulated frontend.**

The isolated ATCA MIMO ISOL module, in addition to the ensuring of proper electrical separation, must fall into low-frequency noise ranges that ensure a numerical integration that is not affected by significant drifts in the length of the experiment in which it is intended to be applied.

For example, ITER is expected to have a numerical integration of 3600 seconds, while in current experiments it may be between 0.1 seconds and about 10 seconds.

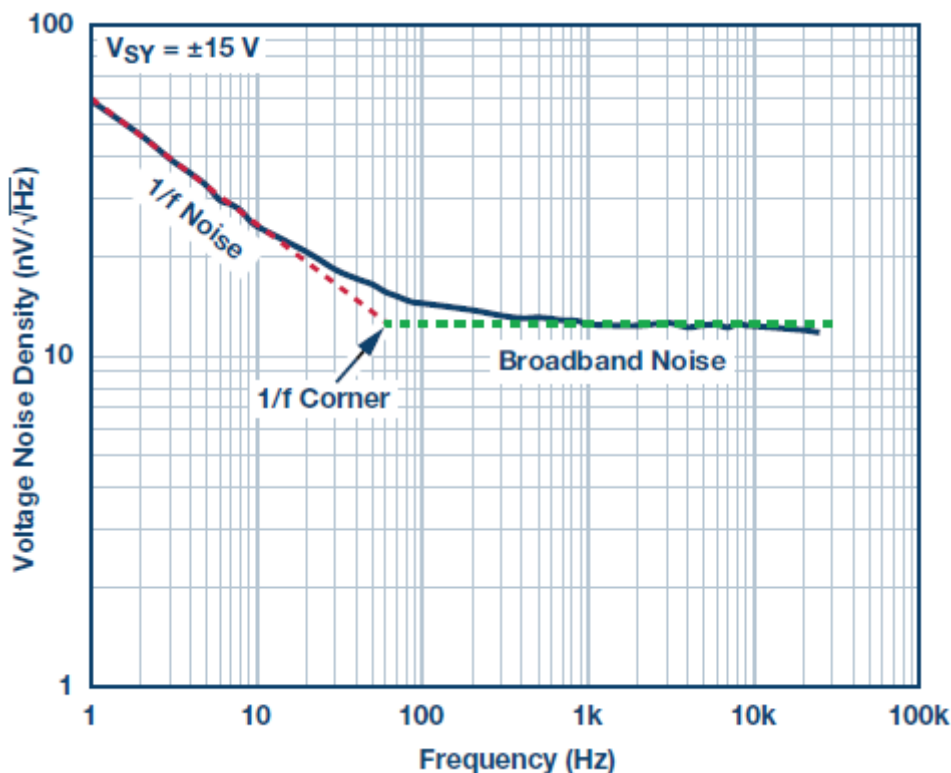
The spectral density analysis of noise shows a corner at the event known as 1/f.

Each operational amplifier shows a characteristic frequency of corner.

Some become unusable for long integration periods beyond a certain threshold.

1/f noise is low frequency noise for which the noise power is inversely proportional to the frequency as shown in the next image.

1/f noise can limit performance in any precision dc signal chain. However, it can be removed, or better considerably reduced, by using techniques such as chopping or ac excitation. There are some trade-offs to using these techniques, but modern amplifiers and  $\Sigma$ - $\Delta$  converters have addressed these issues, making zero drift products.



**Figure 4-14 Generic voltage noise spectral density.**

The 1/f corner point may vary considerably for each product and is therefore subject to analysis and testing in this thesis.

An operational amplifier that shows the corner at 50Hz, as in the picture, is unusable in any nuclear fusion experiment.

what is at the right of the crossover frequency can't influence our integration analysis because almost all the modern operational amplifiers is suitable in broadband noise.

In the left side, to estimate the low frequency voltage noise, the standard specification is 0.1 Hz to 10 Hz peak-to-peak noise.

In the particular case of experiments in which numeric integration must be particularly long while keeping the low frequency noise threshold extremely low, the 1/f corner will be kept below 1Hz.

A component with acceptable characteristics for this application will have to show a noise peak, in the range around 0.1Hz, on the scale of hundreds of nano-volts.

By anticipating the results that i will later show in the thesis, by experimental measurements, the frontend scheme is too noisy and in some cases unusable in the context.

Passive components, used in the frontend, can inject a component in a  $1/f$  noise and current noise, however, for low resistances it is in the typical application too small to be considered.

In the analysis will focus on the voltage noise analysis.

Nuclear fusion is not a typical application so voltage noise must be reduced more is possible to acquire of magnetic probe in tokamak.

## 4.5. 18 bit ADC SAR integrated circuit

The AD7641 is an 18-bit, 2 MSPS, charge redistribution SAR, (successive approximation), fully differential, analog-to-digital converter (ADC) that operates from a single 2.5 V power supply.

The part contains a high speed, 18-bit sampling ADC, an internal conversion clock, an internal reference (and buffer), error correction circuits, and both serial and parallel system interface ports.

It features two very high sampling rate modes (wideband warp and warp) and a fast mode (normal) for asynchronous rate applications.

The AD7641 is hardware factory calibrated and tested to ensure ac parameters, such as signal-to-noise ratio (SNR), in addition to the more traditional dc parameters of gain, offset, and linearity.

The AD7641 is available in Pb-free only packages with operation specified from  $-40^{\circ}\text{C}$  to  $+85^{\circ}\text{C}$ .

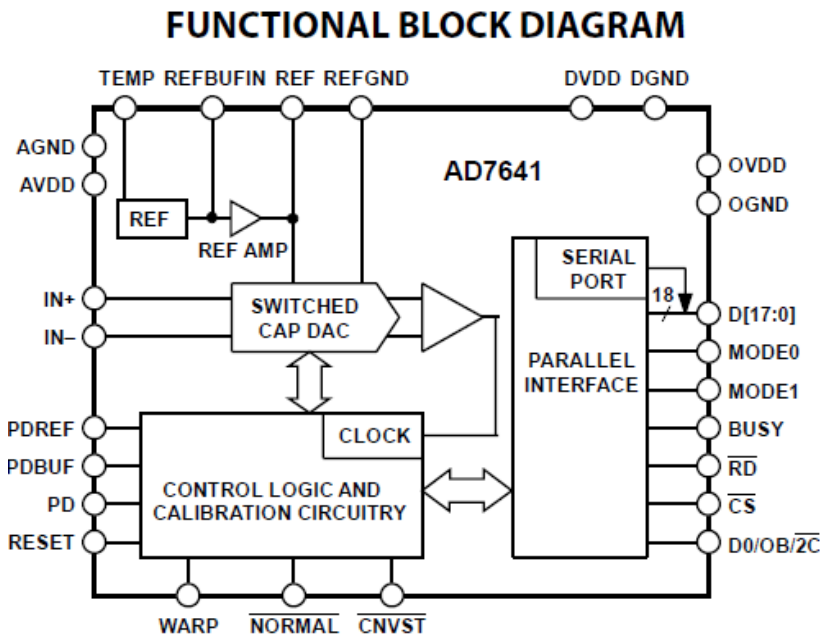


Figure 4-15 The AD7641 functional block diagram

## 4. 6. The AD7641 features

Here the basic technical characteristics are reported:

### Throughput

The AD7641 features shown different modes to optimize performances according to the applications.

In speed contest there are two modes of operations, Warp and Normal.

The AD7641 in Warp mode (Warp = high) allows the fastest conversion rate is capable of converting 2,000,000 samples per second (2 MSPS). 2 MSPS (wideband warp and warp mode).

However, in this mode, and this mode only, the full specified accuracy is guaranteed only when the time

between conversions does not exceed 1 ms.

If the time between two consecutive conversions is longer than 1 ms (e.g. after power-up) the first conversion result should be ignored.

This mode makes the AD7641 ideal for applications where both high accuracy and fast sample rate are required.

The Normal mode is the fastest mode (1.5 MSPS) without any limitation about the time between conversions.

This mode makes the AD7641 ideal for asynchronous applications such as data acquisition systems, where both high accuracy and fast sample rate are required.

### Converter operation

The AD7641 is a successive approximation analog-to-digital converter based on a charge redistribution DAC.

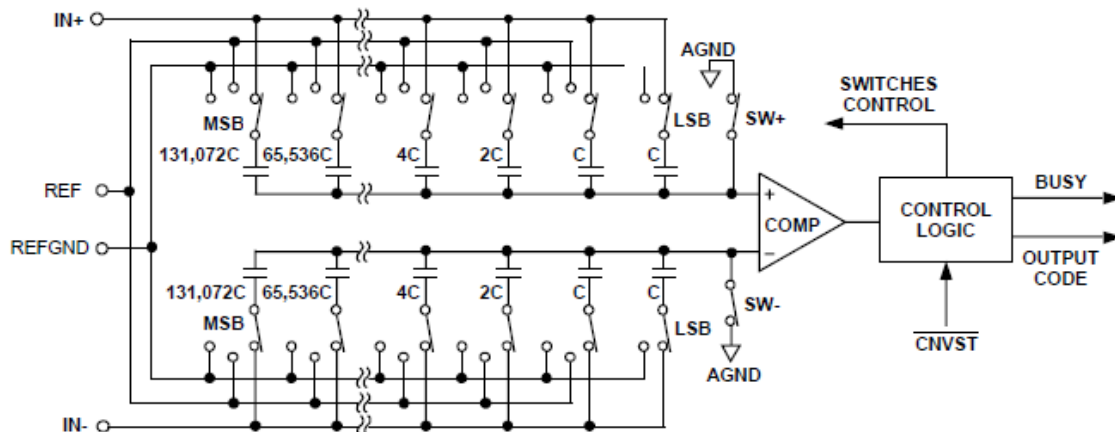


Figure 4-16 input capacitors rail

Figure 4-16 shows the simplified schematic of the ADC. The capacitive DAC consists of two identical arrays of 18 binary weighted capacitors which are connected to the two comparator inputs.

During the acquisition phase, terminals of the array tied to the comparator's input are connected to AGND via SW+ and SW-.



All independent switches are connected to the analog inputs.

Thus, the capacitor arrays are used as sampling capacitors and acquire the analog signal on IN+ and IN- inputs.

When the acquisition phase is complete and the CNVST input goes low, a conversion phase is initiated. When the conversion phase begins, SW+ and SW- are opened first.

The two capacitor arrays are then disconnected from the inputs and connected to the REFGND input. Therefore, the differential voltage between the inputs IN+ and IN- captured at the end of the acquisition phase is applied to the comparator inputs, causing the comparator to become unbalanced. By switching each element of the capacitor array between REFGND or REF, the comparator input varies by binary weighted voltage steps ( $V_{REF}/2$ ,  $V_{REF}/4$  . . .  $V_{REF}/262144$ ).

The control logic toggles these switches, starting with the MSB first, in order to bring the comparator back into a balanced condition. After the completion of this process, the control logic generates the ADC output code and brings BUSY output low.

### **Supply and package.**

The AD7641 can be operated from a single 2.5 V supply and be interfaced to either 5 V or 3.3 V or 2.5 V digital logic.

It is housed in a 48-lead LQFP or a tiny LFCSP packages that combines space savings and allows flexible configurations as either serial or parallel interface. The AD7641 is pin-to-pincompatible and is a speed upgrade of the AD7674.

## 4. 7. AD7641 microprocessor interface capability

The AD7641 is ideally suited for traditional dc measurement applications supporting a microprocessor, and ac signal processing applications interfacing to a digital signal processor.

The AD7641 is designed to interface with a parallel 8-bit or 16-bit wide interface or with a general-purpose serial port or I/O ports on a microcontroller.

A variety of external buffers can be used with the AD7641 to prevent digital noise from coupling into the ADC.

The SPI Interface (ADSP-219x) section illustrates the use of the AD7641 with the ADSP-219x SPI-equipped DSP.

## 4. 8. AD7641 SPI interface capability

Figure shows an interface diagram between the AD7641 and an SPI-equipped DSP, the ADSP-219x.

To accommodate the slower speed of the DSP, the AD7641 acts as a slave device and data must be read after conversion.

This mode also allows the daisy-chain feature. The convert command can be initiated in response to an internal timer interrupt.

The 18-bit output data are read with three SPI byte access. The reading process can be initiated in response to the end-of-conversion signal (BUSY going low) using an interrupt line of the DSP.

The serial peripheral interface (SPI) on the ADSP-219x is configured for master mode (MSTR) = 1, clock polarity bit (CPOL) = 0, clock phase bit (CPHA) = 1, and the SPI interrupt enable (TIMOD) = 00 by writing to the SPI control register (SPICLTx).

It should be noted that to meet all timing requirements, the SPI clock should be limited to 17 Mb/s, allowing it to read an ADC result in less than 1  $\mu$ s.

When a higher sampling rate is desired, it is recommended to use one of the parallel interface modes.

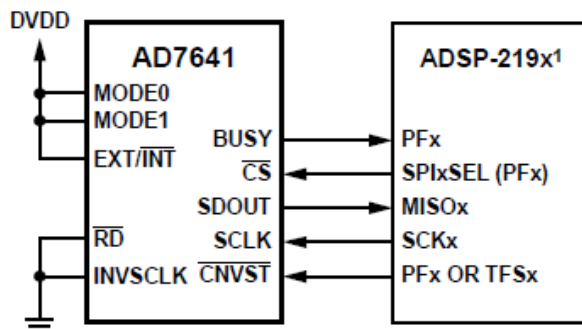
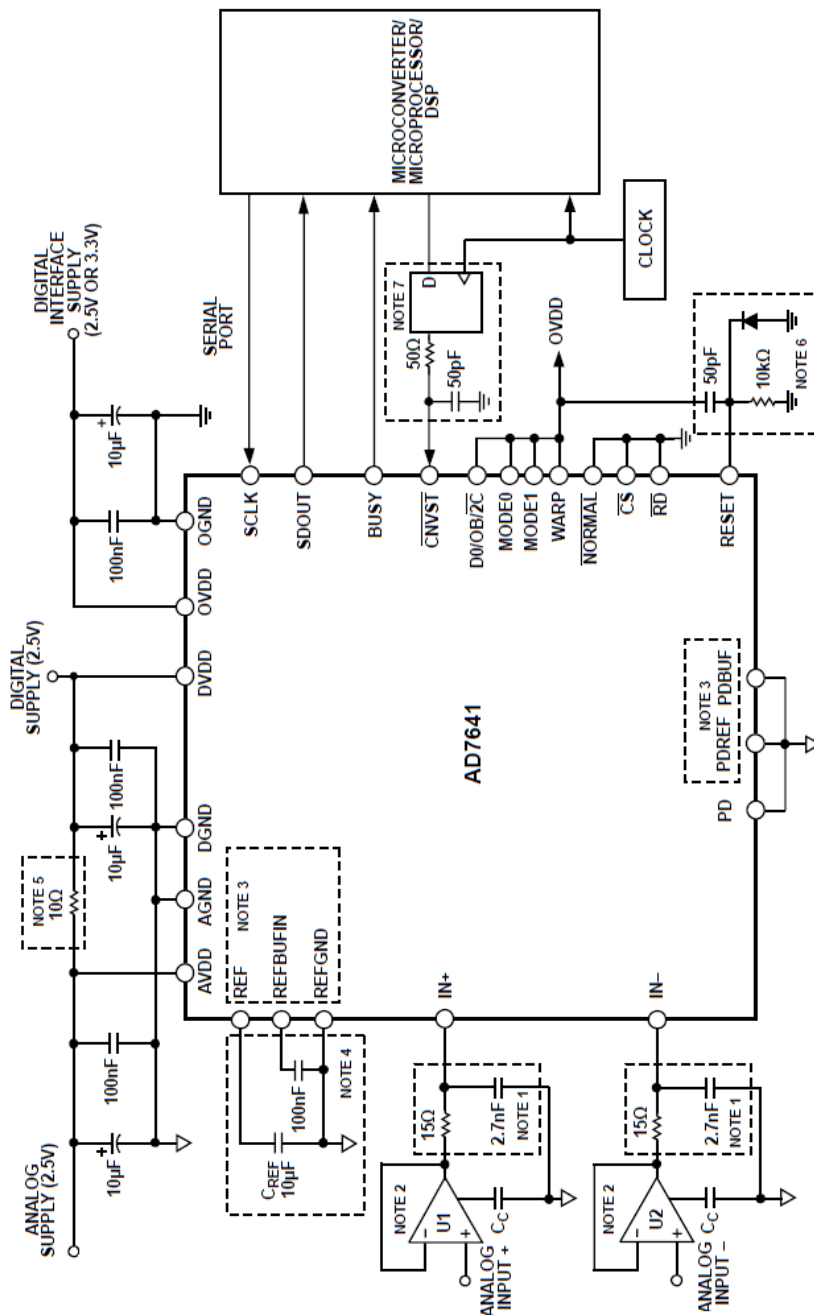
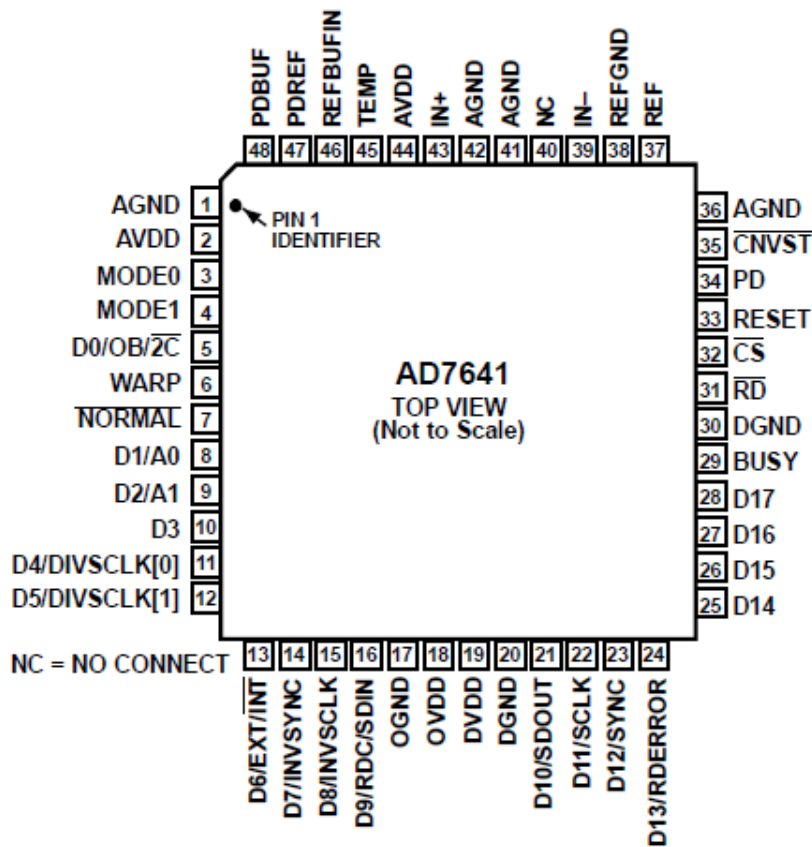


Figure 4-17 The AD7641 SPI signals and wiring.



**Figure 4-18 The AD7641 typical application.**

The IN+ and IN- inputs, in the ATCA MIMO ISOL module, are connected to the frontend circuit shown in the Figure 4-13, and the U1, U2 buffers and the U1, U2 buffers are implemented using a single chip operational amplifier THS4520.



**Figure 4-19 The AD7641 pinout.**

The pins 5 and 6 have a hardware set function, imposed by some resistors, in the ATCA MIMO ISOL.

WARP mode is imposed and MODE, in the pin 3,4 is set for 18-bit interface mode.

# Chapter 5.

## 1K5Vrms Isolated frontend

### 5. 1. Power supply stage, DC/DC converter

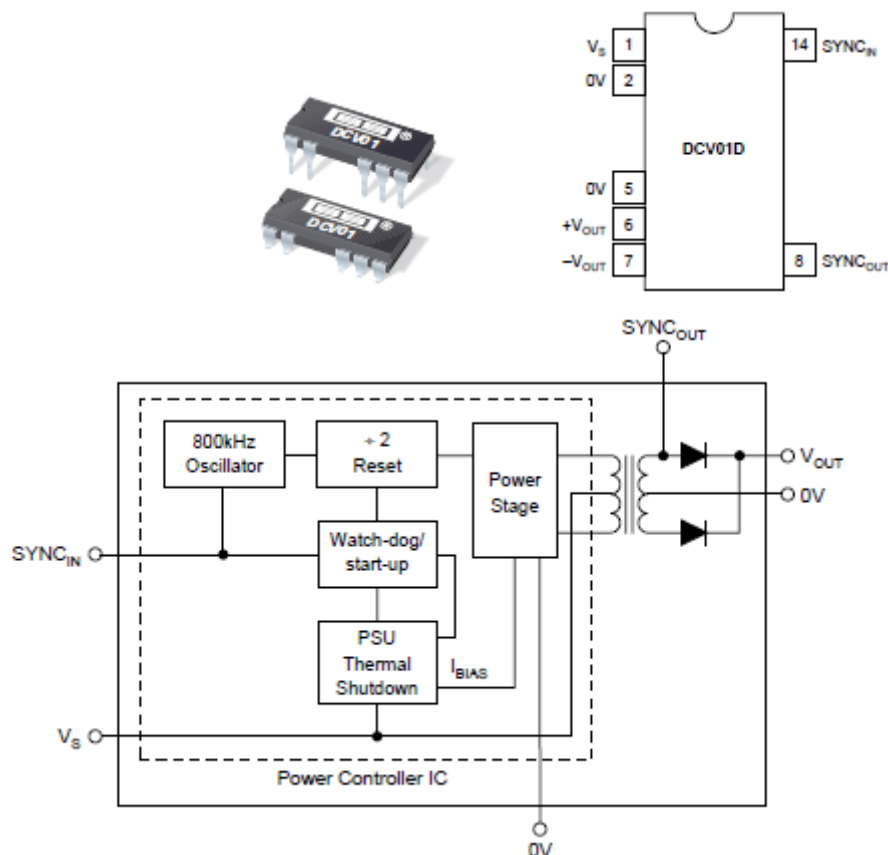
The DCV01D device is used like a solution when the application requiring a signal path isolation, like in a toroidal machines.

Inside the device there is a switching module that integrate a 800kHz oscillator.

In order to minimize possible electrical noise, the DC/DC converter is switched off during the release of the conversion phase.

The technique used in the ATCA MIMO ISOL module is to take power off at predefined moment driving specialized input pin in "SYNCin" synchronized moments.

During the shutdown period it is the task of some capacities, keep the reading active from the ADC buffer.

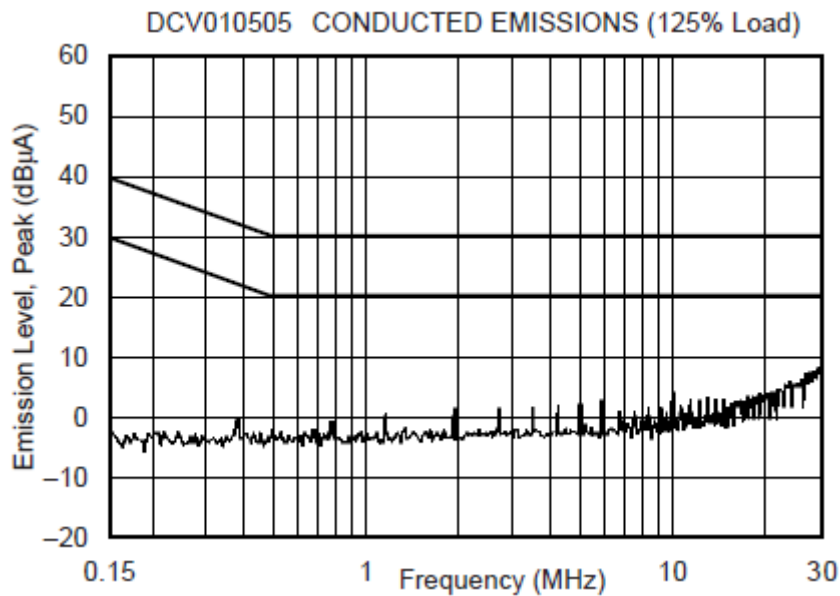


**Figure 5-1 DC/DC converter internal block diagram**

The DCV01 series is a family of 1W, 1500Vrms isolated, unregulated DC/DC converters. Requiring a minimum of external components and including onchip device

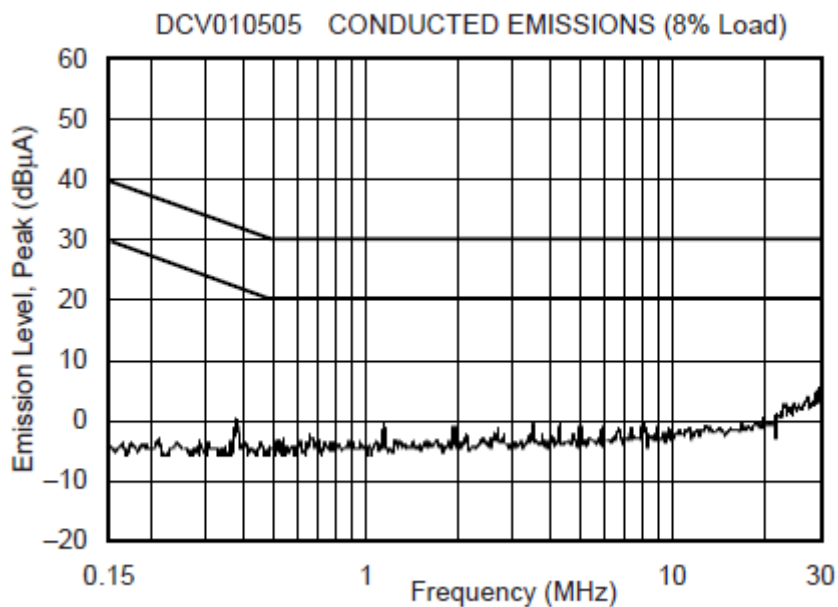
protection, the DCV01 series provides extra features such as output disable and synchronization of switching frequencies.

In Figure 5-1 SYNC<sub>out</sub> and SYNC<sub>in</sub> sync terminals are visible.



**Figure 5-2 DC/DC converter emission at 125% load**

In Figure 5-2 the trend of the electrical noise is shown when the device is loaded at 125% of the nominal value indicated in the data book.



**Figure 5-3 DC/DC converter emission at 8% load**

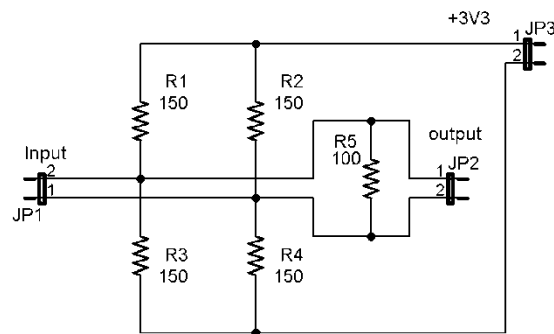
A wide range of applied load, 8% to 125%, do not influence significantly the noise emission, like is evident comparing Figure 5-2 and Figure 5-3 in both margins of the graph.

# Chapter 6.

## Analog frontend to Pitaya interface

### 6. 1. Interfacing the analog ATCA MIMO ISOL module to FPGA

To connect the analog module to FPGA it must to generate on dedicated GPIO high signal to send to CNVST command pin. the ADC performs the conversion, and responds with a serial 18bit string of SDOUT-> SDATA, synchronized by the SCLK.



**Figure 6-1 Resistive net level adapter**

If you drive more frontend modules is need that the FPGA, for each one, produces a RESET output to enable the ADC, a CNVST output to start the conversion and two LVDS SDIN and SCLK inputs to connect to a 18 bit shift register to be defined in VHDL internally of the FPGA area.

The flow chart shows the signals and timings needed to obtain the 18bit string from the ADC.

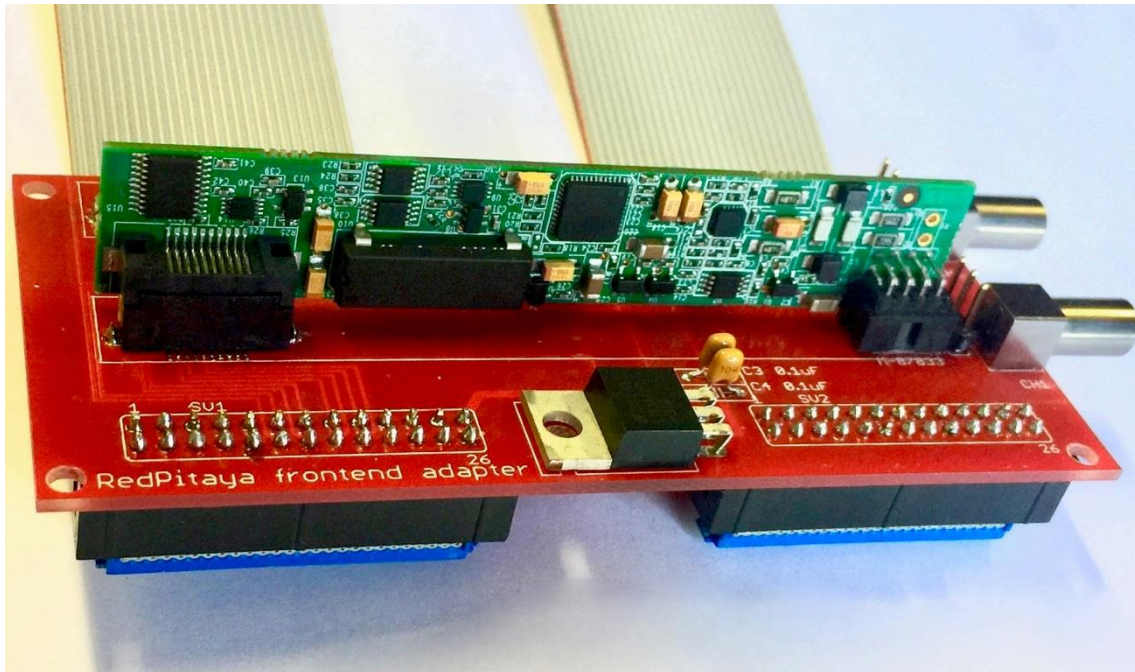
## 6. 2. Prototype of hardware interface between ATCA MIMO ISOL module and FPGA

In the first stage of the research, the necessary board for the interfacing of ATCA-MIMO-ISOL to the RedPitaya ZYNQ processor has been developed.

Two versions of the board was realized.

The first version is capable of interfacing 2 ATCA-MIMO-ISOL modules, and it is that applied in the experimental phase.

A second version, much more complete with three channels, it can be installed upon RedPitaya, using topologically coincident connectors .

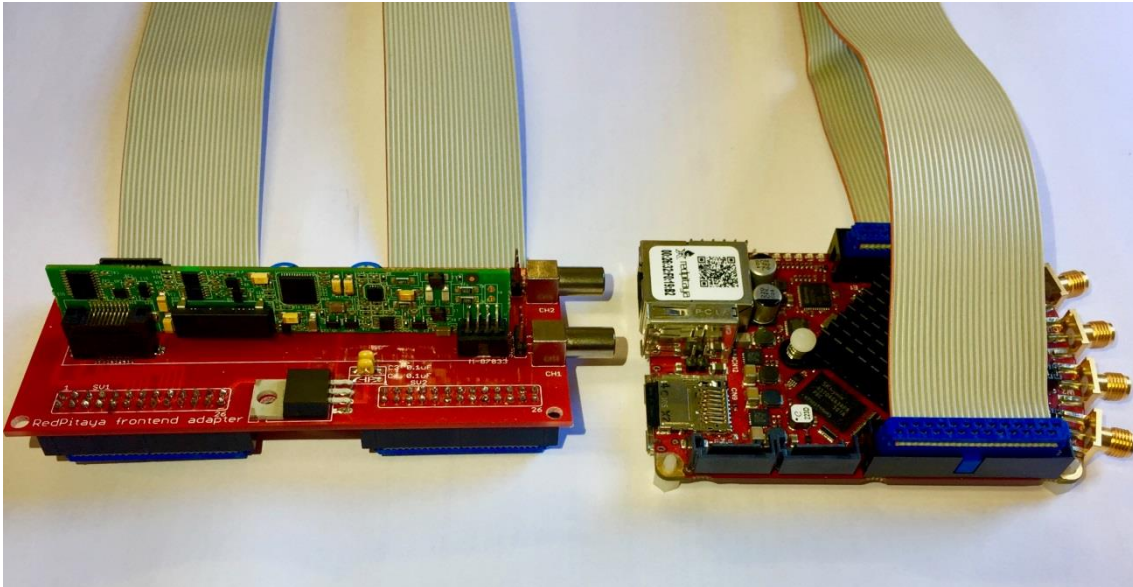


**Figure 6-2 Frontend to Pitaya adapter with channel 1 mounted.**

On the Figure 6-2, an ATCA-MIMO-ISOL channel is shown, connected to the carry board and two 26-lines flat cables to bring the signal to and from RedPitaya including power.

Although in the the RedPitaya the 3v3 voltage is present , it was preferable to install a LM1117 regulator to stepdown the 5V coming from the USB port to 3v3 usefull to energize the ADC and the electronic aboard the ATCA-MIMO-ISOL.

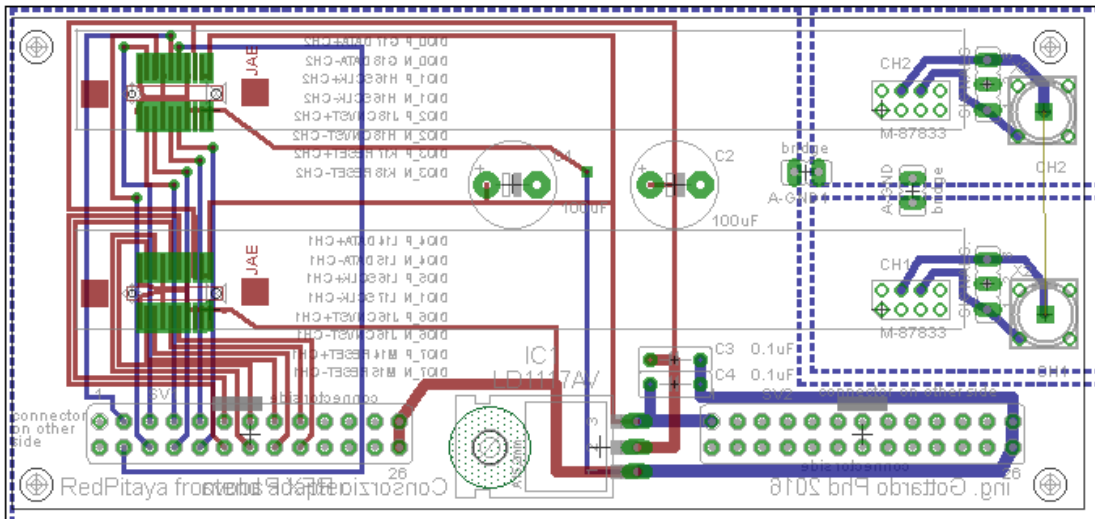




**Figure 6-3 Fronted to Pitaya overview.**

in the left side of the Figure 6-3 it is visible the carry board with one item of capture module installed, on the right side the RedPitaya board. The two devices are connected with 26-pin flat cable to interconnect the ZYNQ7000 processor FPGA section to the analog module on the carry board.

The two 100uF capacitors is useful for the transitory during the power up time. the two 100nF capacitor, installed in the proximity of LM1117 regulator, is to prevent the auto oscillation of the component that can inject in the supply line spurious signal.

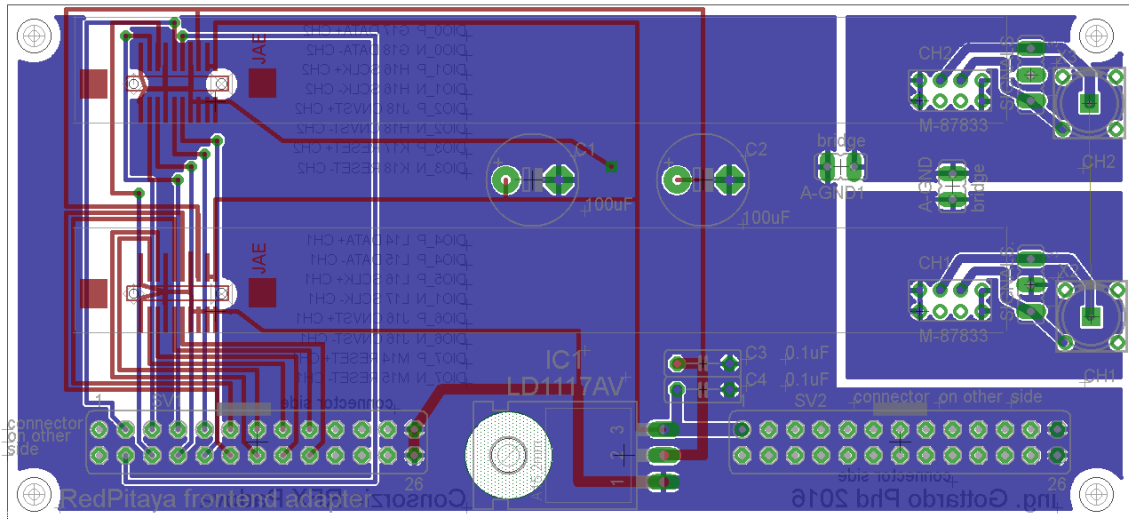


**Figure 6-4 ATCA- MIMO-ISOL to RedPitaya interface, in use version**

The dashed lines represent ground planes that are separated so as to maintain the isolation of the sections as already happens in the acquisition module.

There are some jumper that when closed can short-circuit the ground planes.

in some situations, such as comparisons, during experimental acquisition sessions are many useful.



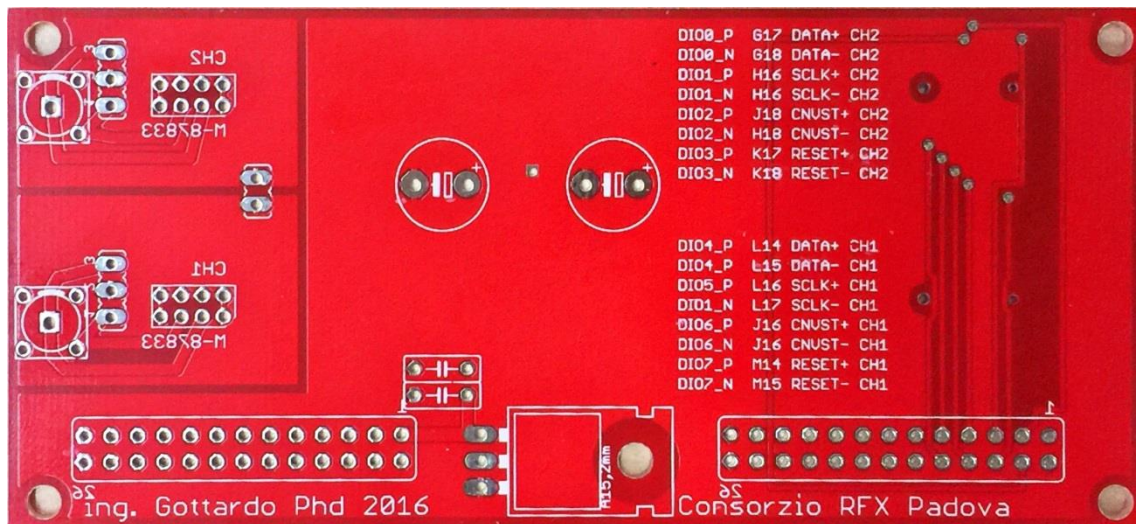
**Figure 6-5 Analog frontend to pitaya interface, Dual layer PCB.**

The JAE connector, is a 20 pins, distributed by Japan Aviation Electronics Industry, Ltd, and it is a avionic component, study for very low noise and high speed application.

The ATCA-MIMO-ISOL module use it to interconnect the signal, in the non isolate region, to the data acquire external systems.

In the Figure 6-5 it is easy to see that the left 26 pins connector, connected to the JAE, consent high speed triggers signals from the FPGA section and retrieve the data sends in stream by ADC module.

These serialized data need to be loaded into a FIFO implemented in HDL in the ZYNQ7000 side before to be send to storage systems.



**Figure 6-6 PCB Bottom side, with signals table on silkscreen.**

The Figure 6-6 highlights large ground plans, isolated between them, and jumpers to short circuit them for comparative experimental phases.

The table helps the user report the RedPitaya pinout, showing the differential pairs and their position in the connector pins.

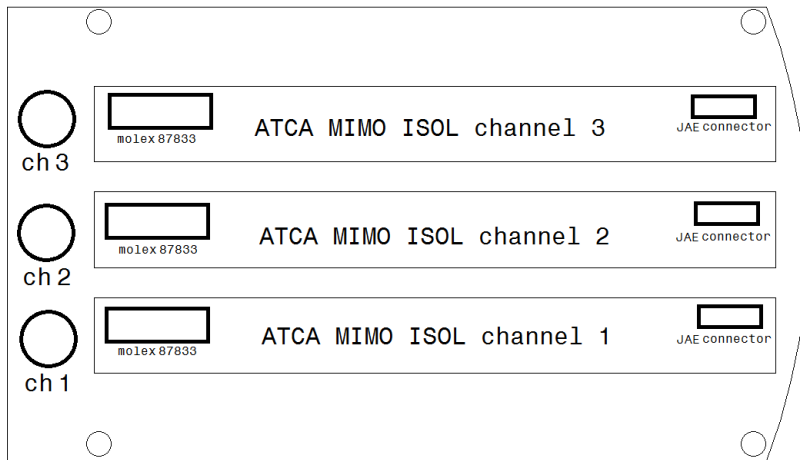
For example, starting from the top, it is evident a split in two groups.

Each group is the set of signals needed for one ATCA MIMO isol module like expose in the Table 4-1 and reporting the FPGA pin number.

A pair of signals, such as the start-up of the conversion, of the second ATCA MIMO isol channel is placed on the J18 and H18 pins, so in the Pitaya's side connectors at DIO\_2P and DIO\_2N , and are indicated by the CNVST + and CNVST- labels.

The gerber files are available at [3].

A second carry board version solves the problem of adapting the voltages needed at some I / O points due to Redpitaya's constraint due to both hardware configuration and transmission protocols.

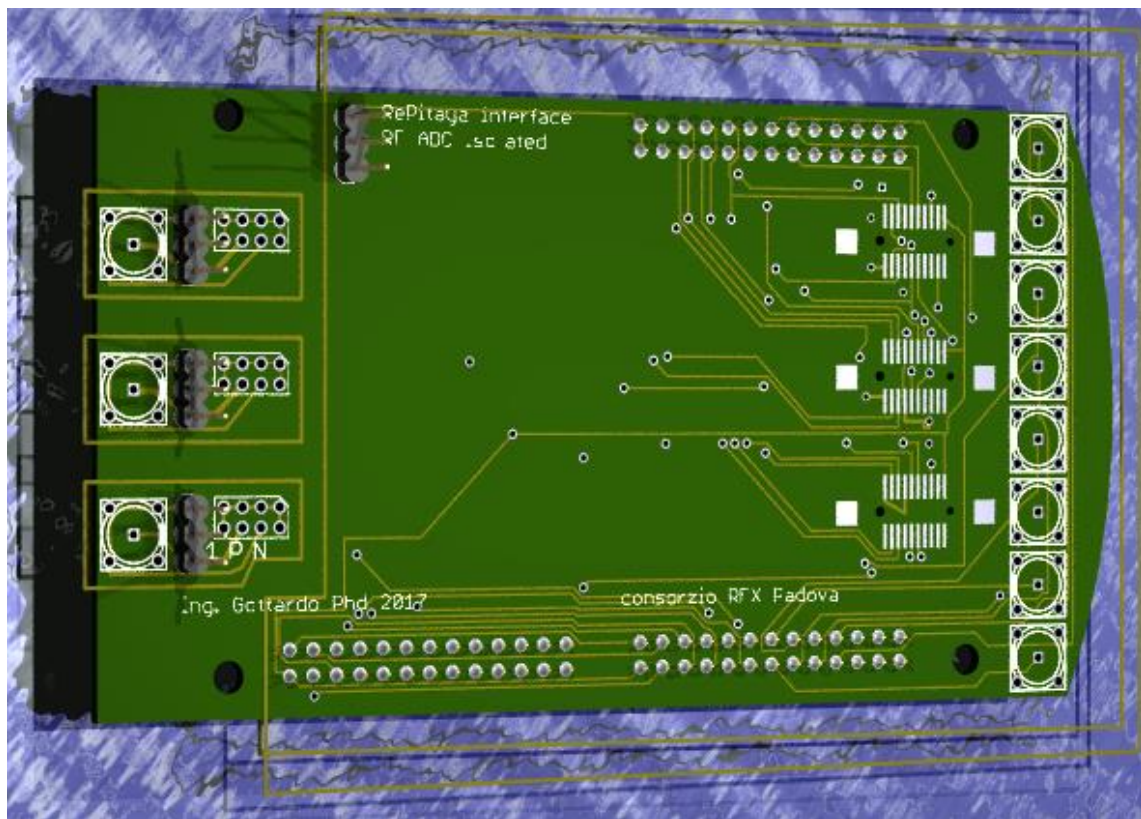


**Figure 6-7 carry board layout final revision**

This layout shows the location of the fixing holes, and the arrangement of the connectors for the insertion of the three ATCA-MIMO isolate channels.

On the left side, the position of the BNCs where connect the magnetic probes.

In order to eliminate the flat interconnecting cable, shown in Figure 6-3, a two-floor system was developed in which, in the bottom floor, the 4 RF connectors, 2 analog inputs and 2 analog outputs are active in the motherboard (RedPitaya), while the daughter board, inserted as a top floor, is present the three BNC connectors associated with their respective ATCA-MIMO-ISOL modules, and on the other side the 8 standard ADC channels available in the ARM section. All the digital I / Os available in RedPitaya are placed on the daughter board connectors for other applications.

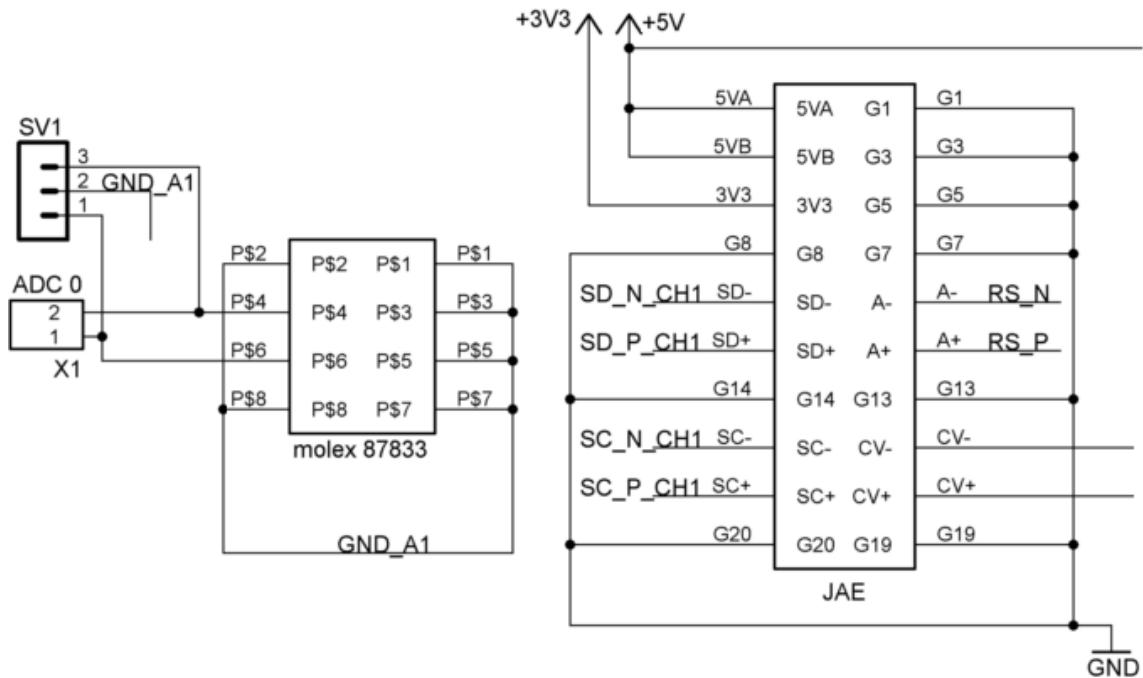


**Figure 6-8 ATCA- MIMO-ISOL to RedPitaya interface, ptototype**

The Figure 6-8 is a 3d preview of the top layers of final carry board.

In the left side shown the BNC connector for the magnetic probes signals. In the right side there are the 8 BNC directli connected to standard slow ADC available in the ARM section of the ZYNQ7000 processor.

These can be usefull for some slow experiments, not for the context of these thesis, in general not to be direct connect to magnetic probes.



**Figure 6-9 Carry board connectors**

In Figure 6-9 there is pinout of the connectors of both versions of the carry board.

The molex 87833 is in the side facing the toroidal machine and the BNC numbered X1 collects the magnetic probe signal.

The JAE connector shows which pins are connected to RedPitaya with correspondence to the signals shown in the Table 4-1.

The two connectors are referred to two different ground plane, GND and GND\_A1, isolated between them.

The circuit requires the presence of a SNLVDS105 useful like repeater, signal buffering, and sync signal generator of the three available channels.

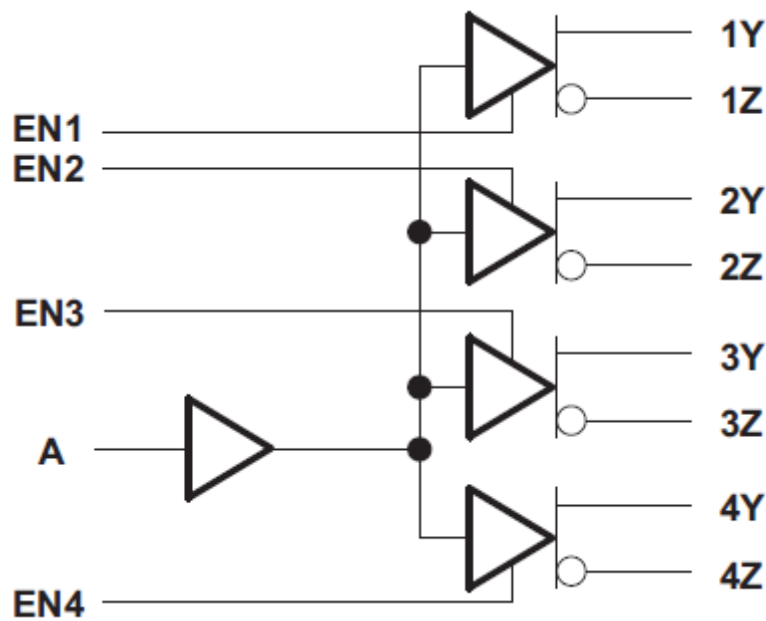
The SN65LVDS105 are a differential line receiver and a LVTTTL input (respectively) connected to four differential line drivers that implement the electrical characteristics of low-voltage differential signaling (LVDS).

LVDS, as specified in EIA/TIA-644 is a data signaling technique that offers low-power, low-noise coupling, and switching speeds to transmit data at relatively long distances.

The intended application of this device and signaling technique is for point-to-point baseband data transmission over controlled impedance media of approximately 100  $\Omega$ .

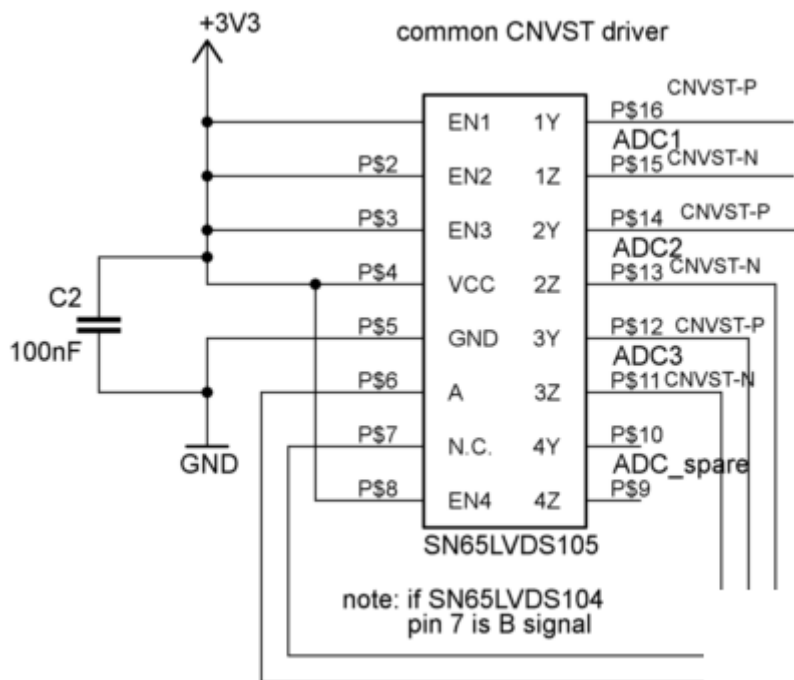
Having the drivers integrated into the same substrate, along with the low pulse skew of balanced signaling, allows extremely precise timing alignment of the signals repeated from the input.

This is particularly advantageous in distribution or expansion of signals such as clock or serial data stream.



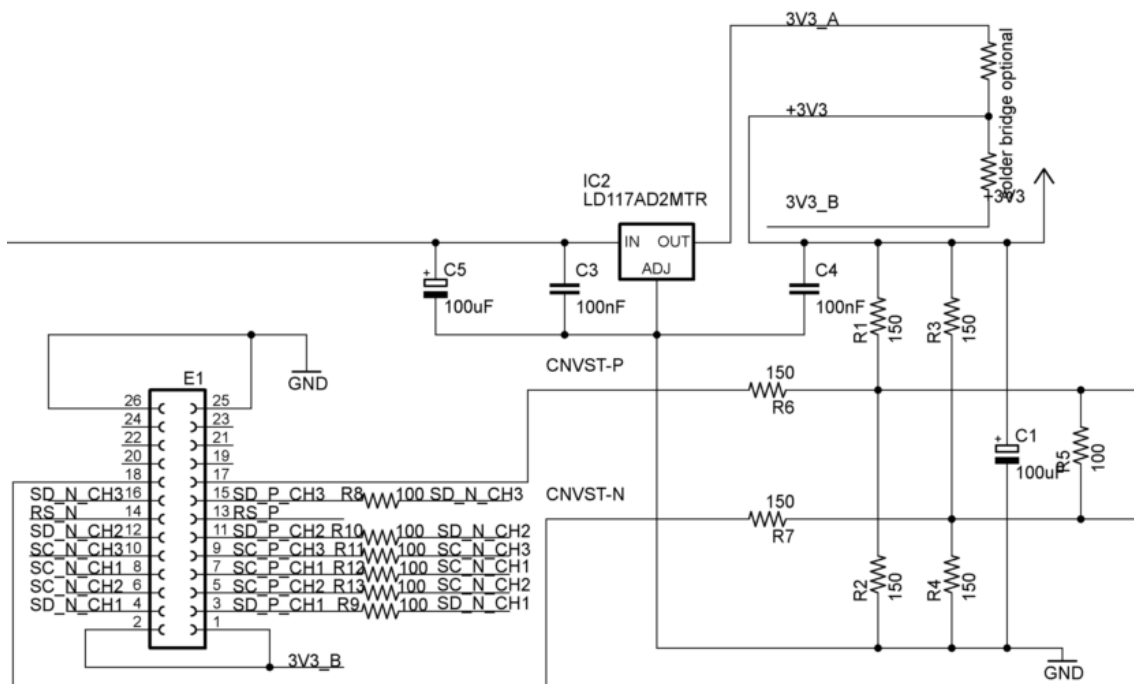
**Figure 6-10 SNLVDS105 internal schematic.**

In the circuit context of the carry board developed for this thesis, the SNLVDS105 is used to synchronize the start signal, CNVST, from the FPGA section to the three ATCA MIMO isol channels.



**Figure 6-11 SNLVDS105 LVDS repeaters pinout.**

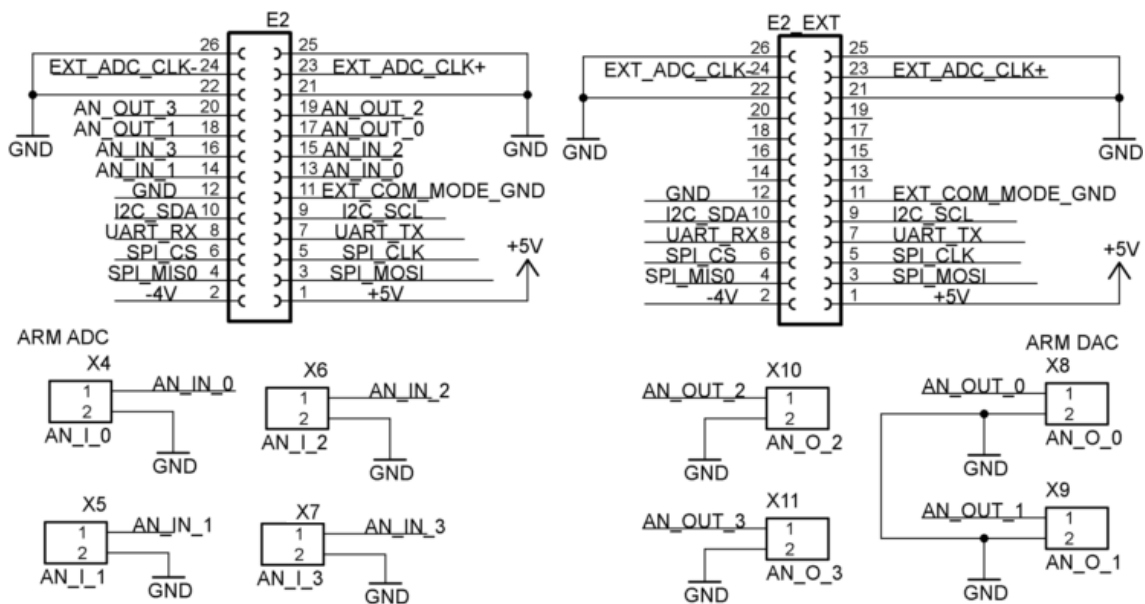
The A pin, number 6 of the SNLDS105 is connect to pin 17 in the connector E1, tat correspond to homonymous in the RedPitaya Board, where the FPGA generate the CNVST-P signal. In the case of our experimental section it is generated common mode and converted in LVDS by the SNLDS105 it self.



**Figure 6-12 Regulated 3v3 supply and resistors interface**

Looking at the diagram in Figure 6-12, can notice that the pin 17, through the resistive level adapter, converts the unipolar CNVST-P signal from 3v3 to 2v5, but this can be generated, if future research need, in differential signal using the pin 16 and making the necessary changes to the HDL code.

just in case of need to generate a differential LVDT signal of CNVST è necessario montare il chip SNLDS104 instead of SNLDS105.

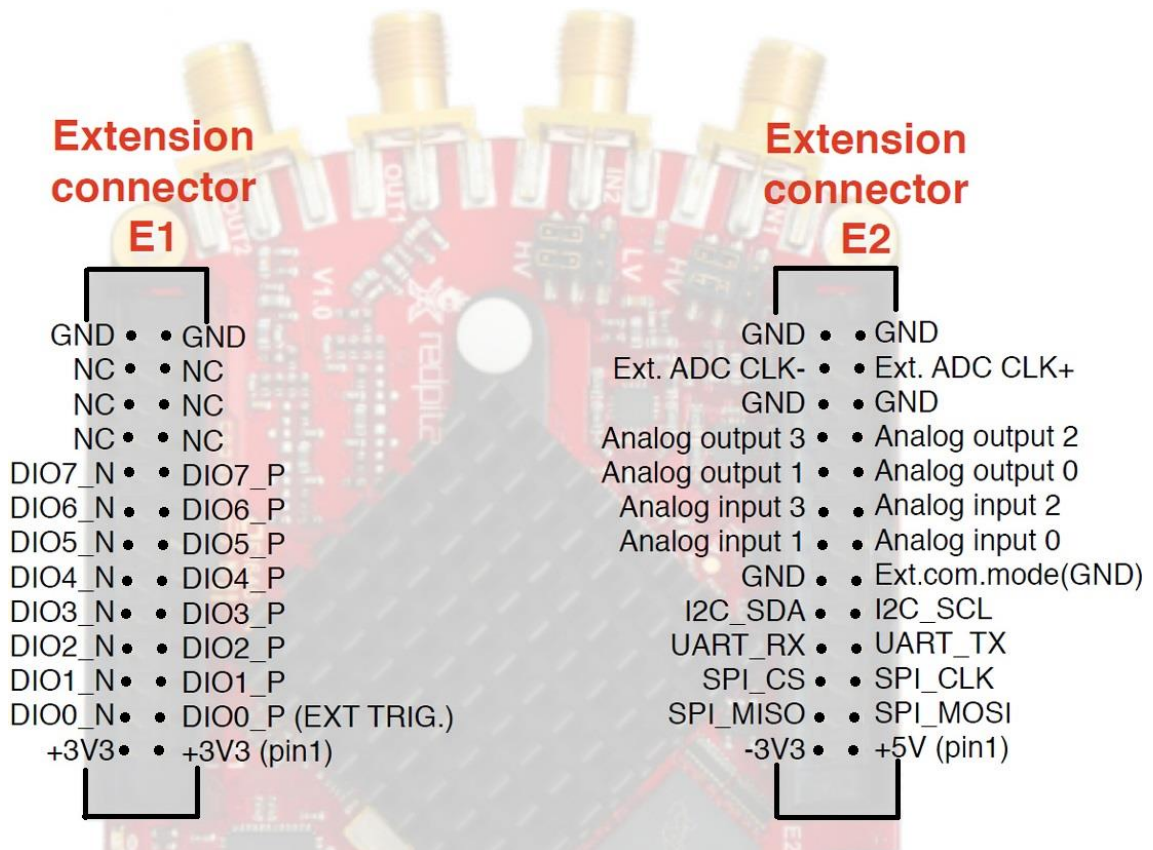


**Figure 6-13 Carry board BNC of on board ARM ADC**

Observing both Figure 6-12 and Figure 6-13 you will notice the presence of three connectors, connector E2 seems duplicated.

The E1 and E2 connectors must be welded to the carry board so that they can engage the lower part and the welds on the upper side.

The connector E2\_EXT visible in the Figure 6-13 must be mounted on the components side of the carry board so all the remaining pins are still available for experiments.



**Figure 6-14 RedPitaya extension connectors**

In Figure 6-14 the RedPitaya pinout is shown.

Electrical diagrams, layout, and gerber files are available for those who would like to continue the investigation on the same topic, at [4].

The used CAD is Eagle 6.3 version.



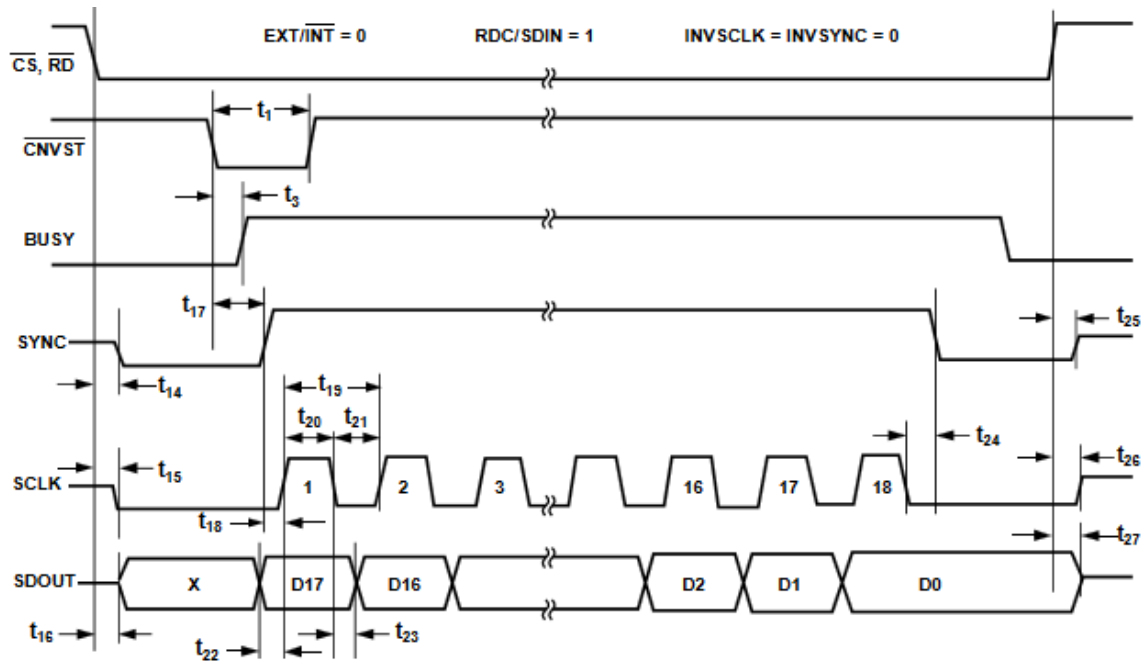
# Chapter 7.

## The ATCA MIMO ISOL module VHDL interface

The protocol selected in the schematic hardware design of ATCA MIMO ISOL module is the AD7641 WARP mode serial master connection with timing diagrams represented in figures below.

<b>DIVSCLK[0]</b> <b>DIVSCLK[1]</b>	<b>Symbol</b>	<b>0</b> <b>0</b>	<b>0</b> <b>1</b>	<b>1</b> <b>0</b>	<b>1</b> <b>1</b>	<b>unit</b>
SYNC to SCLK First edge Delay Minimum	$t_{18}$	0.5	3	3	3	<i>ns</i>
Internal SCLK Period Minimum	$t_{19}$	8	16	32	64	<i>ns</i>
Internal SCLK Period Maximun	$t_{19}$	14	26	52	103	<i>ns</i>
Internal SCLK High Minimum	$T_{20}$	2	6	15	31	<i>ns</i>
Internal SCLK Low Minimum	$T_{21}$	3	7	16	32	<i>ns</i>
SDOUT Valid Setup Time Minimum	$T_{22}$	1	5	5	5	<i>ns</i>
SDOUT Valid Hold Time Minimum	$T_{23}$	0	0.5	10	28	<i>ns</i>
SCLK Last Edge to SYNC Delay Minimum	$T_{24}$	0	0.5	9	26	<i>ns</i>
Busy High Width Maximun	$T_{24}$	0.630	0.870	1.350	2.28	<i>us</i>

**Table 7-1      Serial Clock Timing in Master Read After Convert Mode.**



**Figure 7-1 Master Serial Data Timing for Reading (Read Previous Conversion During Convert)**

In the ATCA MIMO ISOL module there is also a further flipflop connected to the clock output making the signal transaction synchronized with dual edge clock (rising and falling). This aims at achieving the same throughput with half the signal frequency (likewise the DDR protocol specification).

The read approach uses a double buffer with two process with sensibility to the respectively rising and falling edge of the  $SCLK_{in}$  port.

A set of configurable parameters are passed as register variables to make it possible to properly tune the interface with different timings with the granularity set at the rate of the FPGA clock that has been set to 125MHz.

The code used can be found in Appendix A.

The standard slave AXI\_LITE VHDL implementation from Vivado has been declared as  $s00\_axi\_*$  while the most simple version of AXI-Stream has been declared as master (it represents the stream data output from the module) with the name  $m00\_axis\_*$ . Both these bus interfaces are visible on the slave side and the master side of the ip graphical interface, as in picture below.

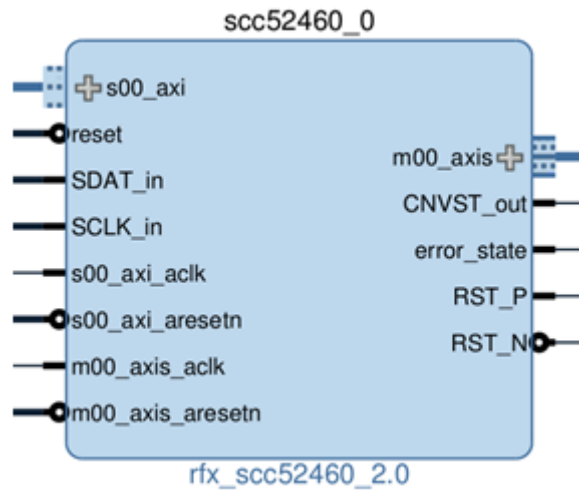


Figure 7-2 IP implementation of `scc52460` module.



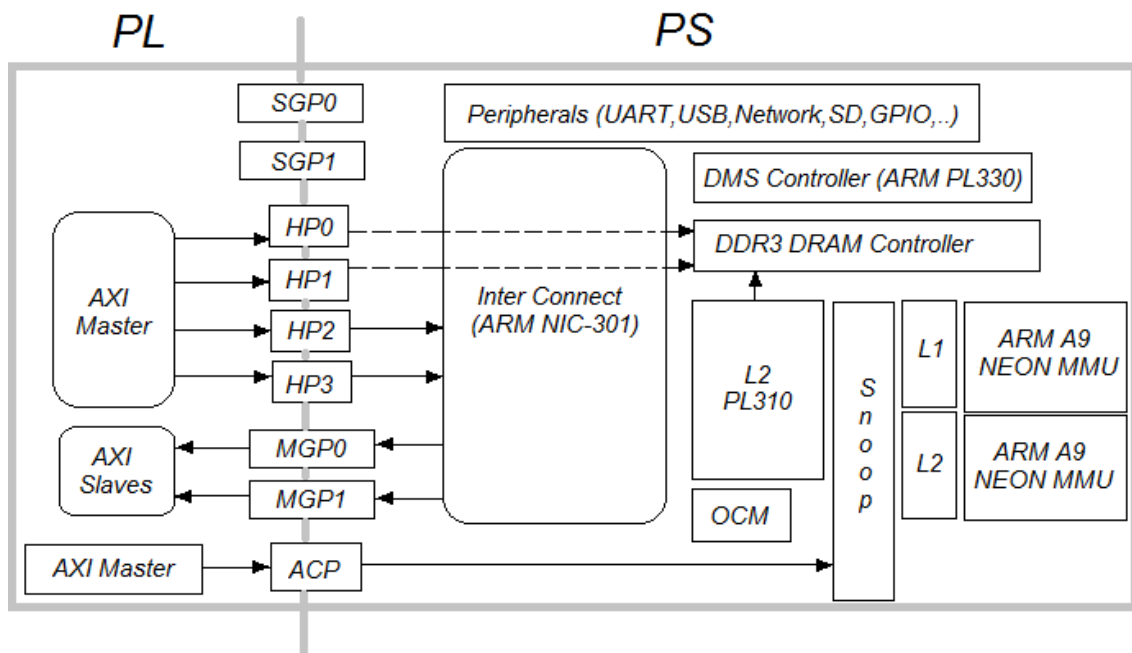
# Chapter 8.

## Zynq Architectures

The Zynq® architecture merge together the high performance on signal acquisition from the field and conditioning with the computing power and device management of ARM multicore processors.

The Zynq®-7000 family is based on the Xilinx All Programmable SoC architecture. These products integrate a feature-rich dual-core or single-core ARM® Cortex™-A9 based processing system (PS) and 28 nm Xilinx programmable logic (PL) in a single device.

The ARM Cortex-A9 CPUs are the heart of the PS and also include on-chip memory, external memory interfaces, and a rich set of peripheral connectivity interfaces.



**Figure 8-1 Zynq® Architectures, simplify block schematic.**

The PL section, in the left side of the block diagram, contains the FPGA area formed by Look-up tables, 36kb block RAM configurable as dual 18 Kb, programmable I/O Blocks supports LVCMOS, LVDS, and SSTL 1.2V to 3.3V I/O.

The Zynq-7000 architecture enables implementation of custom logic in the PL and custom software in the PS.

It allows for the realization of unique and differentiated system functions.

The integration of the PS with the PL allows levels of performance that two-chip solutions (e.g., an ASSP with an FPGA) cannot match due to their limited I/O bandwidth, latency, and power budgets.

Xilinx offers a large number of soft IP for the Zynq-7000 family.

Stand-alone and Linux device drivers are available for the peripherals in the PS and the PL.

The Vivado® Design Suite development environment enables a rapid product development for software, hardware, and systems engineers. Adoption of the ARM-based PS also brings a broad range of third-party tools and IP providers in combination with Xilinx's existing PL ecosystem.

The inclusion of an application processor enables high-level operating system support, e.g., Linux. Other standard operating systems used with the Cortex-A9 processor are also available for the Zynq-7000 family.

The PS and the PL are on separate power domains, enabling the user of these devices to power down the PL for power management if required.

The processors in the PS always boot first, allowing a software centric approach for PL configuration.

PL configuration is managed by software running on the CPU, so it boots similar to an ASSP.

In the right section, visible in Figure 8-1, there is the memory interface unit, which includes a dynamic memory controller and static memory interface modules.

The dynamic memory controller supports DDR3, DDR3L, DDR2, and LPDDR2 memories.

The static memory controllers support a NAND flash interface, a Quad-SPI flash interface, a parallel data bus, and a parallel NOR flash interface.

The multi-protocol DDR memory controller can be configured to provide 16-bit or 32-bit-wide accesses to a 1 GB address space using a single rank configuration of 8-bit, 16-bit or 32-bit DRAM memories.

ECC is supported in 16-bit bus access mode.

The PS incorporates both the DDR controller and the associated PHY, including its own set of dedicated I/Os. Speed of up to 1333 Mb/s for DDR3 is supported.

The DDR memory controller is multi-ported and enables the processing system and the programmable logic to have shared access to a common memory.

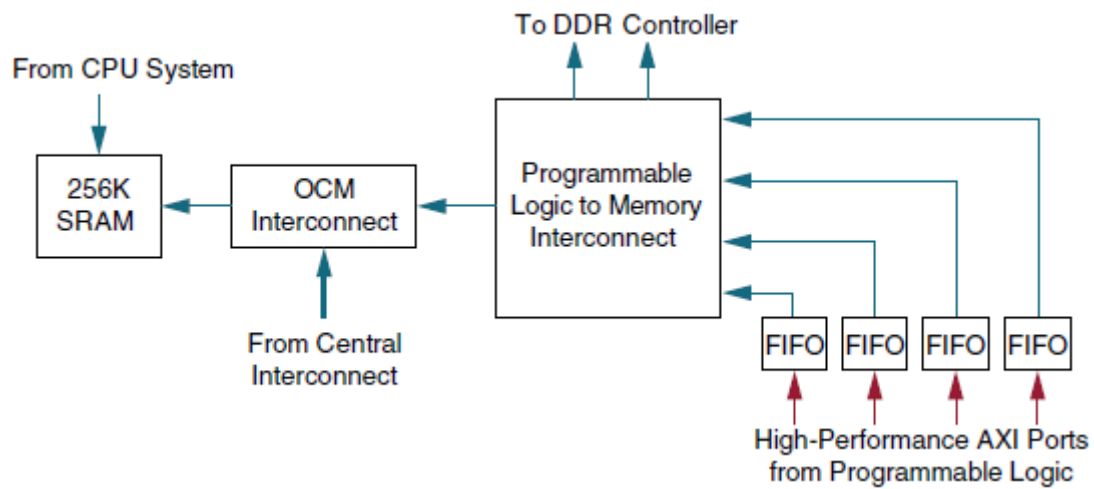
The DDR controller features four AXI slave ports for this purpose:

- One 64-bit port is dedicated for the ARM CPU(s) via the L2 cache controller and can be configured for low latency.
- Two 64-bit ports are dedicated for PL access.
- One 64-bit AXI port is shared by all other AXI masters via the central interconnect.

The high-performance AXI ports provide access from the PL to DDR and OCM in the PS.

The four dedicated AXI memory ports from the PL to the PS are configurable as either 32-bit or 64-bit interfaces. As shown in Figure 8-2 , these interfaces connect the PL to the memory interconnect via a FIFO controller.

Two of the three output ports go to the DDR memory controller and the third goes to the dual-ported on-chip memory (OCM).



**Figure 8-2 PL interface to PS memory subsystem**





## 8. 1. AXI Interconnect Block

The APU, memory interface unit, and the IOP are all connected to each other and to the PL through a multilayered ARM AMBA AXI interconnect.

The interconnect is non-blocking and supports multiple simultaneous master-slave transactions.

The interconnect is designed with latency sensitive masters, such as the ARM CPU, having the shortest paths to memory, and bandwidth critical masters, such as the potential PL masters, having high throughput connections to the slaves with which they need to communicate.

Traffic through the interconnect can be regulated through the Quality of Service (QoS) block in the interconnect.

The QoS feature is used to regulate traffic generated by the CPU, DMA controller, and a combined entity representing the masters in the IOP.

The accelerator coherency port (ACP), present in Figure 8-1, is a 64-bit AXI slave interface that provides connectivity between the APU and a potential accelerator function in the PL.

The ACP directly connects the PL to the snoop control unit (SCU) of the ARM Cortex-A9 processors, enabling cache-coherent access to CPU data in the L1 and L2 caches. The ACP provides a low latency path between the PS and a PL-based accelerator when compared with a legacy cache flushing and loading scheme.

## 8.2. Block schematic and Internal architecture of Zynq 7000

In principle, the internal architecture of a modern FPGAs produced by Xilinx, and relatively to Zynq® family SoC 7000 is shown in Figure 8-3.

It is in evidence the presence of the ARM dual core section clocked at 2x 667MHz. These parameters are intended to improve in the next years in number of core and clock speed.

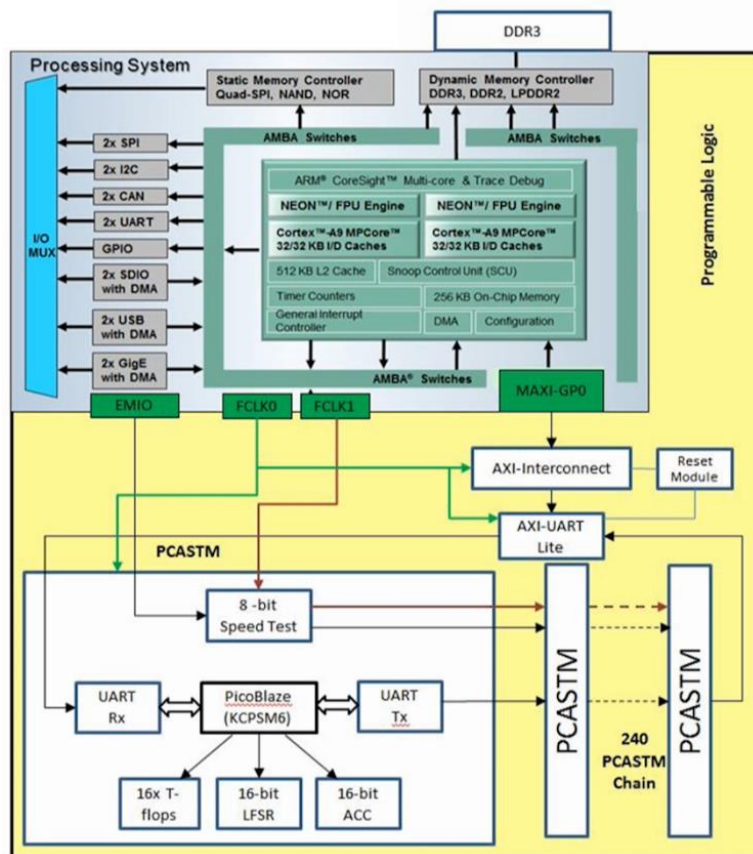


Figure 8-3 Zynq® 7000 internal architecture.

This chip version is already suitable for the realization of the controller to access on the very fast DDR3 RAM, used in standard computer.

Both ARM and FPGA sections are programmable differently and from different access points, such as the ARM can be done in C using a cross compiler of Linaro, even with Ubuntu operating system and the way to deploy firmware can be the Ethernet port.

The FPGA section can be programmed using the Verilog or VHDL. These compilers can be installed in Vivado platform distributed by Xilinx.

For uploading your design you can use the Ethernet port or the JTAG cable, or other access points.

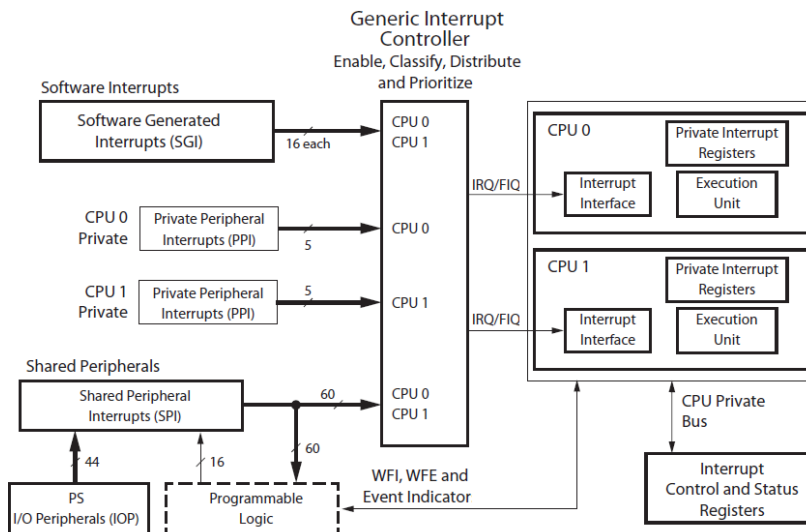
The ARM A9 RISC processor is based NEON engine with vector capabilities and multimedia.

### 8.3. GIC, Generic Interrupt Controller

The Zynq® 7000 architecture uses two Cortex A9 processors whose functionality is handled by the interrupt controller GIC pl390.

The interrupt sources can be hardware, for example generated by the I/O ports or peripherals with IOP designation, or incoming from the programmed logic section denominated PL via ARM processors.

The figure show the block diagrams of function principle.



**Figure 8-4 Generic Interrupt Controller, block schematic.**

Interrupt management will need to consider the following arguments:

- Private, shared and software interrupts
- GIC functionality
- Interrupt prioritization and handling.



## **8. 4. Private, shared and software interrupts**

Each CPU, of the multi core system, has a set of interruption signals for private peripherals said PPIs, with dedicated access and in special registers placed in suitable memory bank.

This PPIs include the general timer, a watchdog dedicated to each the core, private timers and FIQ/IRQ from PL.

The interrupts generated via software, the SGIS, are routed to one or both CPUs. The SGIS are generated writing in the registers of the generic interrupt controller (GIC).

The interrupt from distributed devices (SPIs) are generated by a lot of I/O point or by the memory controller in the PS and PL sections.

## **8. 5. GIC functionality**

The GIC (generic interrupt controller) is an integrated, centralized resource that allows the manipulation of interrupts sent from the CPU to the PS and the PL (programmed logic or the FPGA area).

The controller enable, disable, handled the interrupts in vectored mode associating a priority. GIC consider the Interrupt from various sources, and take care of sending to the correct core of the CPU .

All this happens in a way that is user-programmable manageable which will decide not only how to solve the interrupt but also those who must solve and in what sequence than other interruptions more or less simultaneous.

In addition, the controller supports security extension for implementing a security-conscious system.

The architecture of the controller is non-vectored version 1.0 (GIC v1) or an ARM Generic Interrupt Controller precisely GIC.

The access to registers is via a private bus of the CPU for extremely fast read and write that strongly limits or eliminates blockages or delays in access during the interconnections.

The interrupt distributor centralizes all sources before handing them (in the technical bibliography mentioned as "dispatch") according to the hierarchy of priorities to the respective CPU.

The GIC assures that a specific interrupt addressed to multiple CPUs will be resolved by a specific CPU at a time. All interrupt sources are identified by a unique interrupt number told ID.

All interrupt sources have a configurable priority and maintain the list of target CPU.



## 8. 6. Interrupts, priorities and handling

The interrupt controller is reset by the reset bit PERI\_RST in the subsystem setting o registry A9\_CPU\_RST\_CTRL in SLCR. The same reset signal also resets the timer associated with the specific CPU or Core, and private watchdog timer (AWDT). At reset, all interrupts pending or ongoing service are ignored.

The interrupt controller operates at a clock rate of CPU\_3x2x (half the CPU frequency).

That ones from the processing system I/O peripherals (IOP) are routed to the PL and assert asynchronously to the FCLK clocks.

In the other direction, the PL can asynchronously assert up to 20 interrupts to the PS.

Sixteen of these interrupt signals are mapped to the interrupt controller as a peripheral interrupt where each interrupt signal is set to a priority level and mapped to one or both of the CPUs.

The remaining four PL interrupt signals are inverted and routed to the nFIQ and nIRQ interrupt directly to the signals to the private peripheral interrupt (PPI) unit of the interrupt controller.

There is an nFIQ and nIRQ interrupt for each of two CPUs. The PS to PL and PL to PS.

Type	PL Signal Name	I/O	Destination
PL to PS Interrupts	IRQF2P[7:0]		SPI: Numbers [68:61].
	IRQF2P[15:8]		SPI: Numbers [91:84].
	IRQF2P[19:16]		PPI: nFIQ, nIRQ (both CPUs).
PS to PL Interrupts	IRQP2F[27:0]		PI Logic. These signals are received from the I/O peripherals and are forwarded to the interrupt controller. These signals are also provided as outputs to the PL.

**Table 8-1 Zynq7000 PL Interrupt Signals**





## 8.7. The central interconnect module

The majority of the I/O signals for PS peripherals, other than USB, can be routed to either the PS pins through the MIO, or to the PL pins through the EMIO.

Most peripherals also maintain the same protocol between MIO and EMIO, except Gigabit Ethernet. To reduce pin count, a 4-bit RGMII interface runs through the MIO at a 250 MHz data rate (125 MHz clock with a double data rate).

The route through the EMIO includes an 8-bit GMII interface running at a 125 MHz data rate. The USB, Quad-SPI, and SMC interfaces are not available to the EMIO interface to the PL.

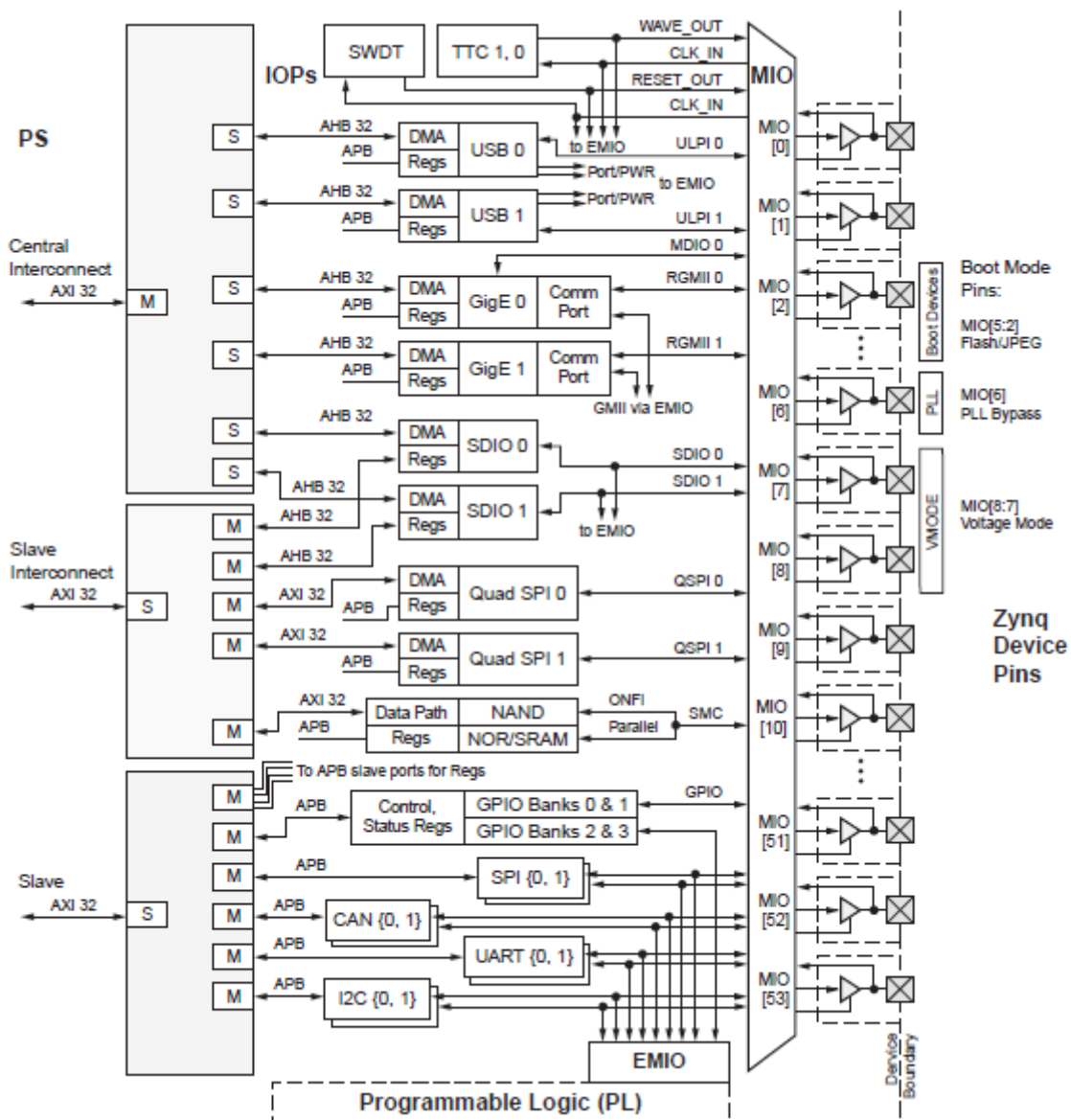


Figure 8-5 I/O Peripherals System Diagram and central interconnect.

On the interconnect side, the USB, Ethernet and SDIO peripherals are connected to the central interconnect to service the six DMA masters.

Software accesses the slave-only Quad-SPI and SMC peripherals via the AHB interconnect.

The GPIO, SPI, CAN, UART, and I2C slave-only controllers are accessed via the APB bus.

All control and status registers are also accessed via the APB interconnect except for the SDIO controllers which each have two AHB interfaces.

This architecture is designed to balance the bandwidth needs of each controller interface.

# Chapter 9.

## The AXI interface

### 9. 1. The AMBA protocol

The AXI protocol stems its root in the AMBA protocol. AMBA is an acronym for Advanced Microcontroller Bus Architecture and identifies a type of the communication bus, and the set of defined rules for its operation (protocol), initially developed for SoC systems. The developed is open source and is therefore very popular.

It is available from the year 1996, initially developed by the multinational ARM Ltd.

From AMBA Protocol arise the following bus protocols and correlated :

- Advanced System Bus (ASB).
- Advanced eXtensible Interface (AXI).
- Advanced High-performance Bus (AHB).
- Advanced Peripheral Bus (APB).

In 2003 it appears the first version of AXI protocol as Amba variant 3. This united the technical characteristics of ATB Protocol (Advanced Trace Bus) with the new high inter connectivity technologies.

Later, in 2010, from AMBA 4 was derived the specifications for the AXI4 protocol.

Subsequently, between 2011 and 2012 appears the ACE variant of AMBA 4 extending system wide coherence.

In March 2013 it makes its appearance AMBA 5 CHI (Coherent Hub Interface) in which was redesigned the system called high-speed transport layer with the aim of minimizing data congestion in transit in the bus.

Today this protocol, on the latest release, appears to be the de facto standard for 32bit processors embedded systems thanks to the fact that it is an open system that does not include royalties for its use.

## 9. 2. The AXI interface

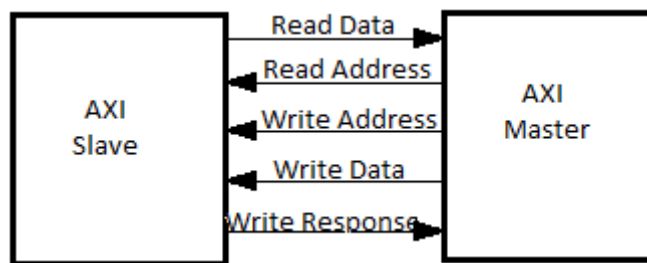
The need to interface the FPGA with faster devices of new generation, for example, to access to the current DDR3, forces the use of the technology used in the interface AXI. There are two types of AXI interface:

1. AXI stream interface.
2. AXI memory mapped interface.

There are the AXI full version, simply called AXI 4 and AXI lite version.

The difference between the two is not in the communication channel used, which in effect is the same, but in the signals which are used within it.

In the full version of AXI Memory Mapped there are a set of 5 signals exchanged between the AXI master and AXI Slave block.



**Figure 9-1 The five signals evolved in the AXI memory mapped protocol**

The first two signals representing the reading of the transaction whilst the remaining three the writing or write transaction, inside the channel.

The arrows indicate the direction of those who generate and those who receive the signal between the master and slave device.

The protocol provides that the part master to take control of the bus initializing the transaction, and the slave waiting for initialization directives from this.

The transaction may be of kind read or write, in any case the initialization and control of the AXI bus are of master competence.

In the event of a write transaction, the generation of "Write response" signal, who is competence of the AXI slave, signalling data was successfully transferred and therefore without errors.

Let's consider the write data channel and see the form and content during transmissions.

To establish communication with the master must acquire the **Ready** signal generated by the slave.

In response to the ready signal, the master generates the **Validate** signal and that expresses the need to open the communication channel for the transfer of data.

Now you can transfer the data but to do this, there are two modes, a single data or small groups, or in bursts.

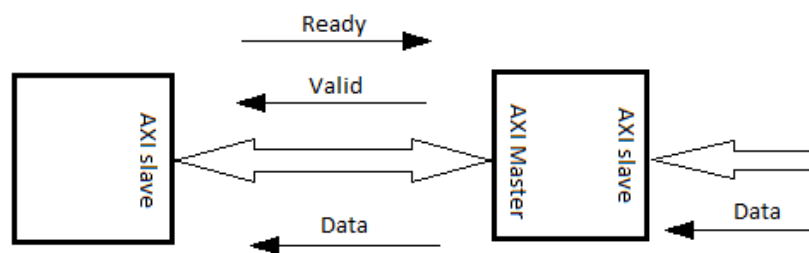
It becomes necessary to specify the following parameters:

1. Size of data.
2. Burst length.
3. QoS.

A variant of the burst mode is **streaming** in the context AXI 4 said **streaming connection**.

The necessity of using a stream is due to the fact that the data coming from the outside to an AXI block, because acquired from sensory systems or similar.

This first block performs a signal processing, often called conditioning, and then is passed to the AXI internal block which functions as a slave.



**Figure 9-2 AXI data exchange**

By analysing the internal architecture of the Zynq, and highlighting the physical and logical division of areas PL (programmable Logic) and PS (processing system), we note in the bus connection the presence of the **HP blocks**, (high performance), who handled the bus.

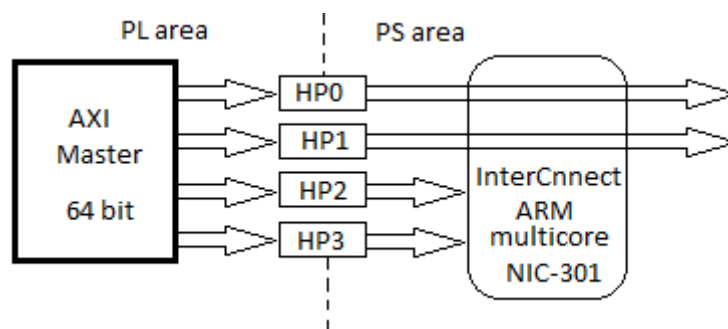
These work as slaves for AXI Master block within the FPGA.

The HP blocks are in fact the interface between the FPGA and the ARM sections.

Unfortunately there is not a direct streaming connection between the FPGA to ARM through this channel, so in case should require will need to implement the logical DMA using the Vivado IP.

The HP 0..3 ports, each 16-bit wide, can implement a 64-bit bus as required by current operating systems.

HP = AXI High performance interface



**Figure 9-3 AXI data transfer FPGA to ARM**

The DDR memories are linked to the by-passers arrows, shown in the drawing, via HP0 and HP1, and by implementing the appropriate controller using Vivado IP.

In the AXI Lite version the number of signals is lower and this also reduces the complexity of the protocol management.

In the image Figure 9-4 we see a block diagram of the integrated device in the Zynq-7000 architecture.

There is a dividing line that cut the structure in two logic part.

On the left side we see the processing system, abbreviation PS, while the right side the PL area or programmable logic of FPGA network.

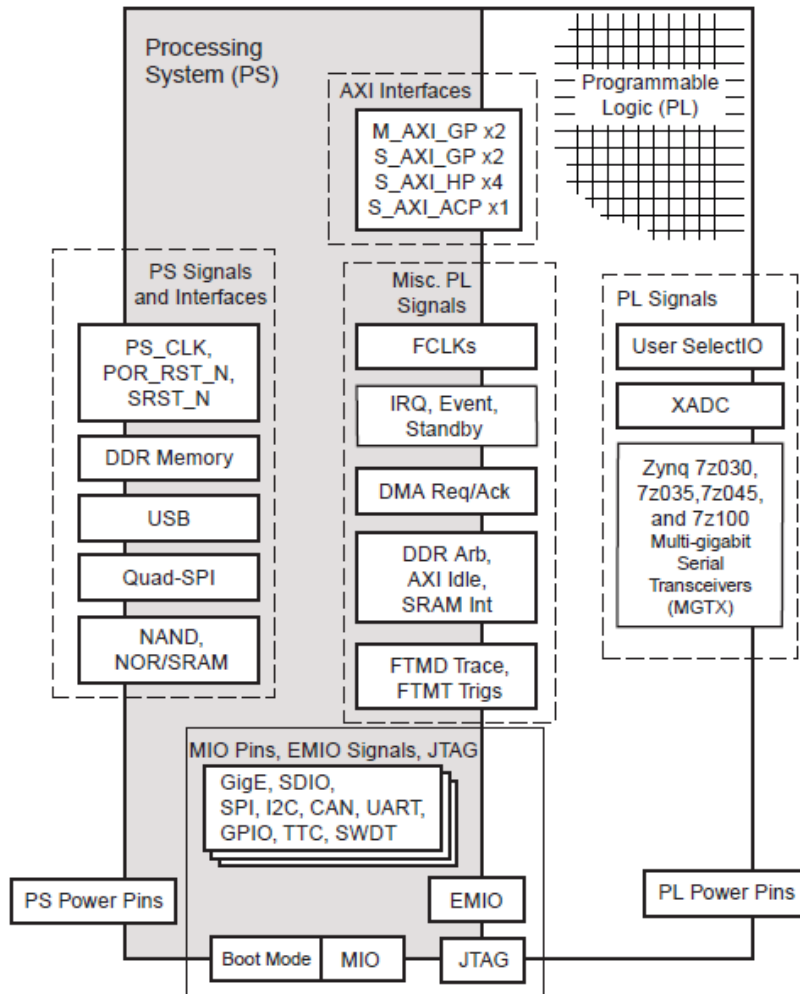
The device, represented as rectangles, are located in a realistic way, so it on the left side we see elements from the section ARM can exit through pinout to the outside of the chip, generally it is the GPIO and communication ports.

The blocks in which the peripheral hardware are implemented are all contained in the chip, for example, we can see the SPI.

On the right side we see the interfacing section of the FPGA with the outside world.

The parties of direct interest right now are in between the PL section and the PS. They are largely the AXI interface with all its subsystems.

Are connected at the same level in order to interconnect the DMA controller, managed AXI, the JTAG debug functions and the interrupt controller.



**Figure 9-4 PL to PS and vice versa internal interface**

Internal connections and their associated bus are best exposed in the block diagram on the next Figure 9-5 in which are expressed the bus extensions, the points of departure and arrival of these.

It is highlighted also extensions of the cache areas.

We also note that for each ARM section is available a coprocessor unit based on the NEON 128bit architecture.

There are 8 DMA controllers, the first four channels are associated with the PS (processors system) unit to make copies of memory areas to and from each section of the Zynq.

The remaining four channels will be used for transfers from the PL (programmable logic alias the FPGA section) to memory and back again from the memory to PL unit.

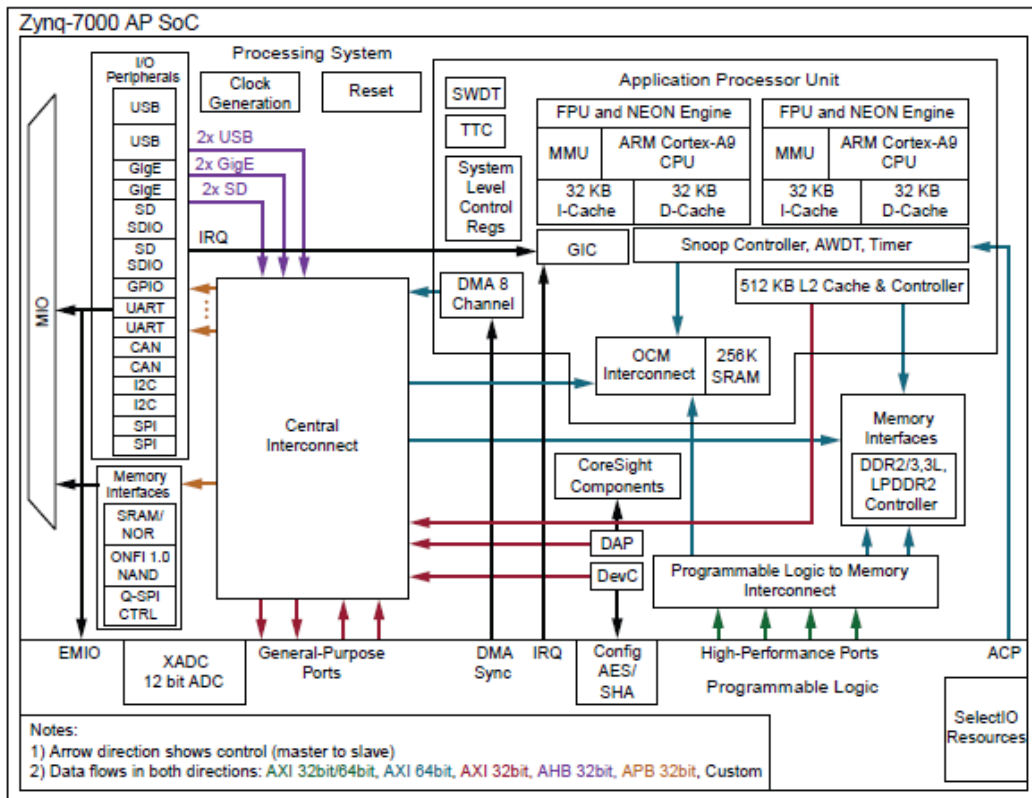


Figure 9-5 Zynq-7000 AP SoC internal block diagram

One of the most important sections for AXI operation is the central interconnect units.

In the chapter devoted to the development of the HDL language program, the interconnect block is of equal importance to the block representing the ZYNQ.

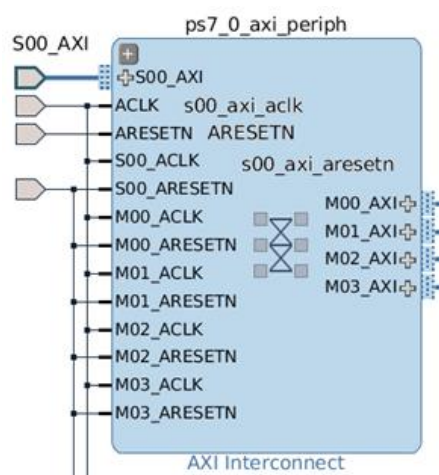


Figure 9-6 Axi Interconnect block represented in HDL



# Chapter 10.

## AXI- Stream FIFO

### 10. 1. AXI Stream FIFO first experimental test

For the actual module acquisition of the outputted stream a Xilinx AXI-Stream FIFO Ip has been exploited as a first experimental level before approaching the less trivial DMA implementation. More information on this data transfer approach are available from the Xilinx product guide [5].

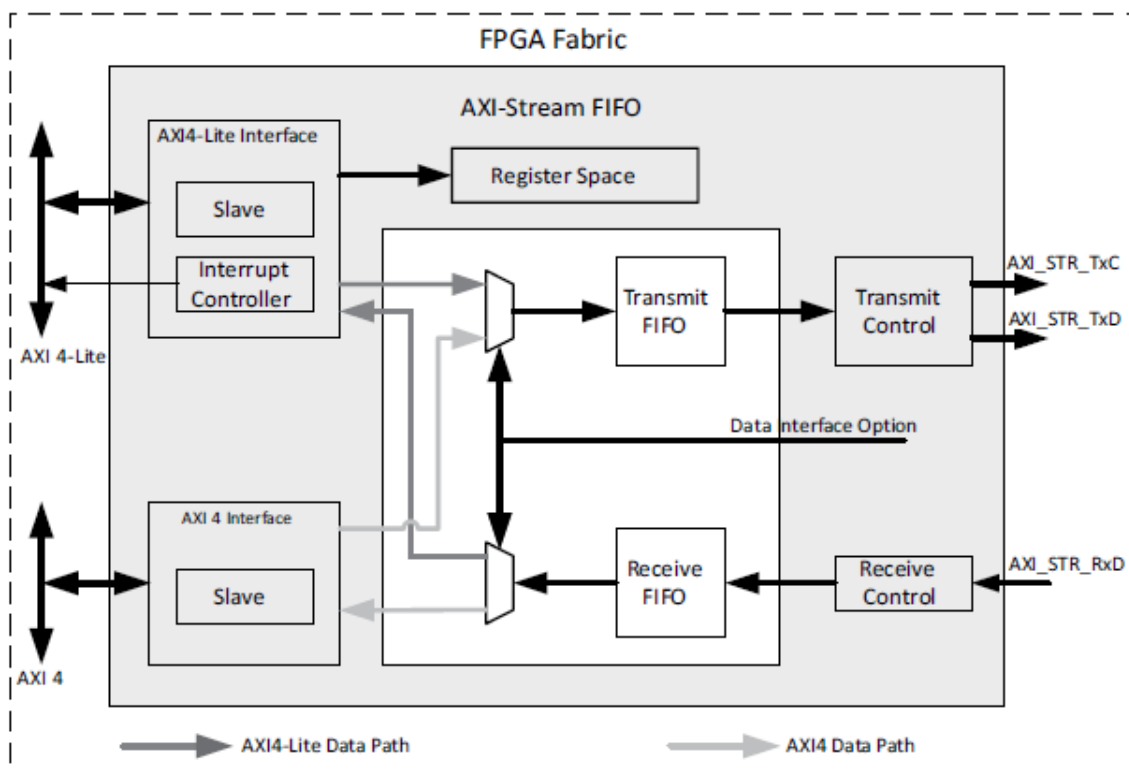
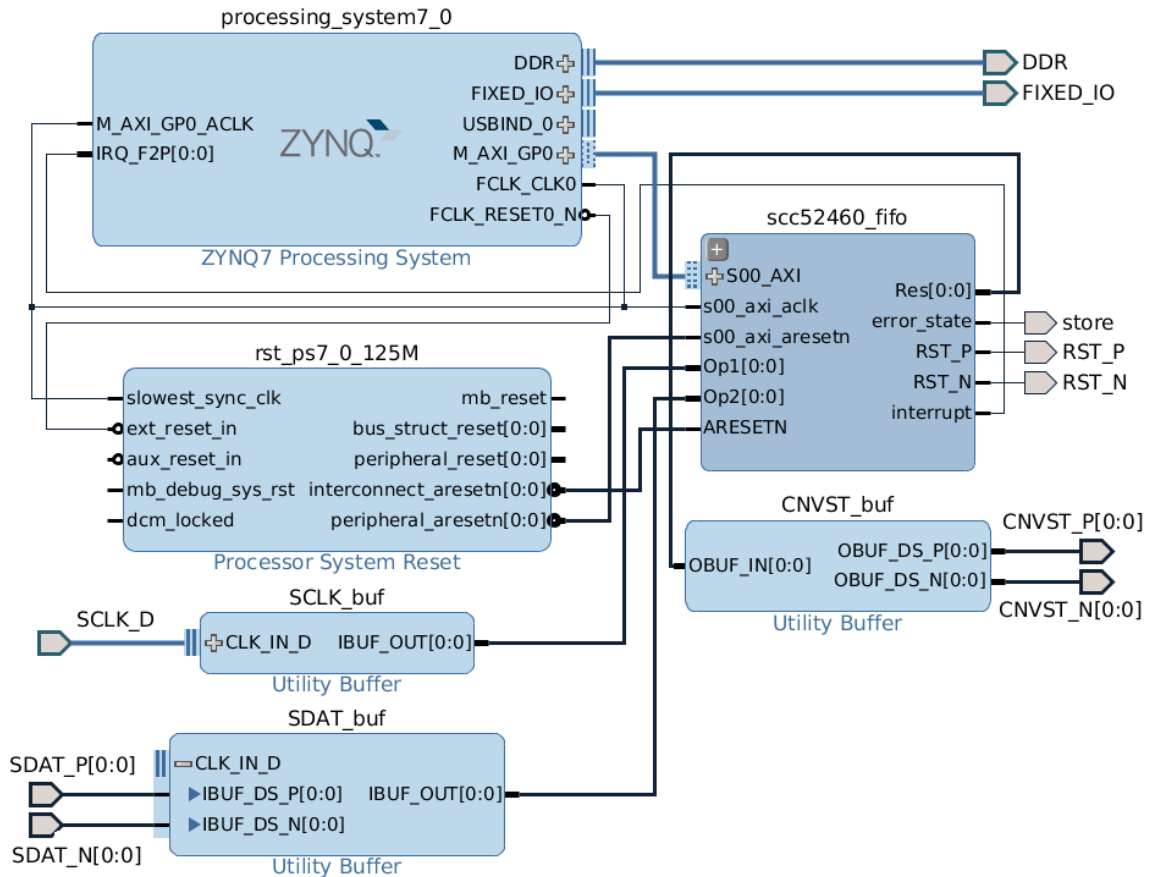


Figure 10-1 AXI4-Stream FIFO Block Diagram

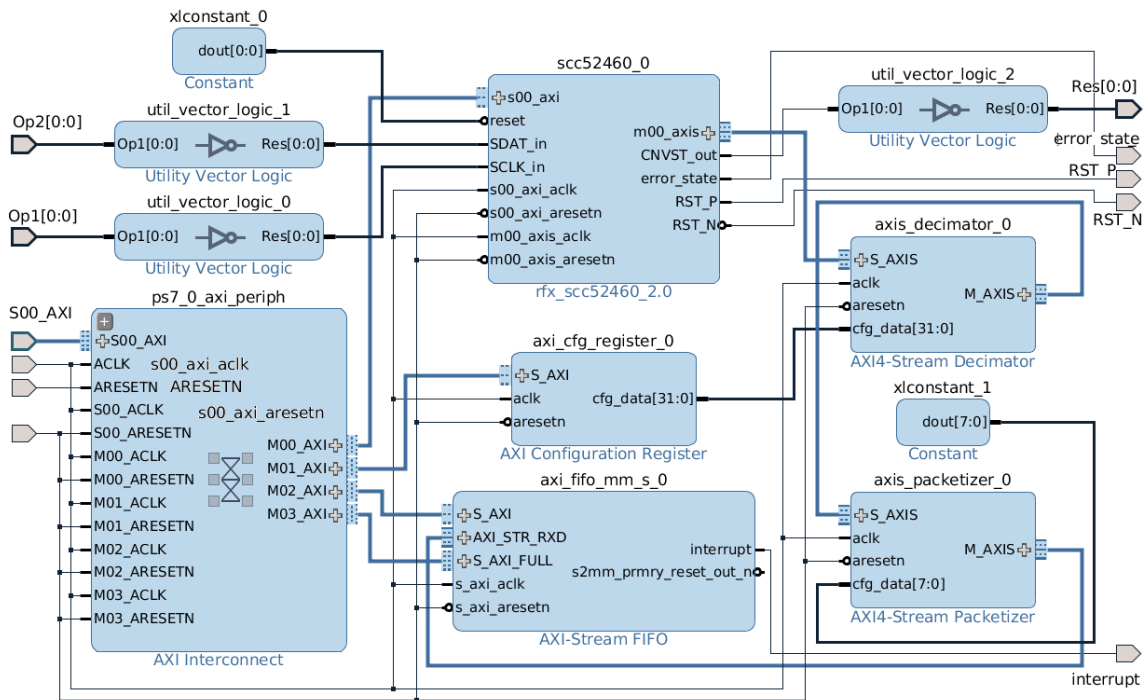
The major components in the AXI4-Stream FIFO core are an AXI Interface block with an AXI4/AXI4-Lite Slave interface, Interrupt Controller, a Registers module, a Receive Control Module, a Transmit Control Module, a Receive FIFO for the receive data and length, and a Transmit FIFO for the transmit data and the length.



**Figure 10-2 Module design connection**

The IP AXI4-Stream FIFO, integrated in the custom IP scc52460, allows memory mapped access to a AXI4-Stream interface. The core can be used to interface to AXI Streaming IPs, similar to the LogiCORE IP AXI Ethernet core, without having to use a full DMA solution. The principal operation of this core allows the write or read of data packets to or from a device without any concern over the AXI4-Stream interface signaling. You can easily manage the AXI4-Stream interfaces as they are transparent.

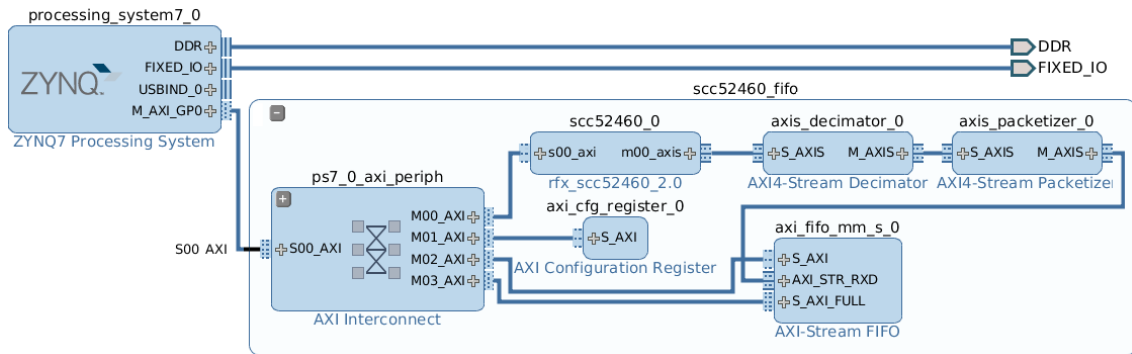
Configurable data width of 32, 64, 128, 256 or 512 bits (AXI4 Data Interface only). For AXI4-Lite, the FIFO data width is 32 bits and for AXI4, it is identical to AXI4 data width.



**Figure 10-3 AXI Stream Complete Vivado design**

The module has been subsequently interfaced with a standard Xilinx Zynq FPGA board, developing a custom HDL code with the purpose of streaming data output to a proper embedded GNU Linux driver. The Figure 10-3 shows the schematic design of the bus connections of the FPGA logic. Two main ip components are visible: the first one from the left of the sketch represents the processor entity, the second is related to the module digital acquisition. The processor unit presents two bus connections, externally linked to the DRR and fixed IO hardware components, and one internally connected by the AXI\_GP0 general purpose bus to the other custom logics. Here all the IRQ and clock connections have been hidden to improve the schema readability but all the components are clocked by 125Mhz Zynq internal multiplier and the FIFO drives an interrupt of the PS to inform the driver of the status changes. The custom logic named scc52460\_fifo encloses all the implemented design exploiting a 32-bit Xilinx AXI-Stream FIFO core with the input connected to a chain of custom cores performing in sequential order: the acquisition from module LVDS serial port, a configurable decimation of acquired data, and a configurable length packetization of the output stream. The communication protocol selected in the schematic hardware design of ADC module is the AD7641 serial master WARP mode connection with a further external flipflop connected to the clock output that makes the signal transaction dual edge synchronized (rising and falling). This aims at achieving, likewise the DDR protocol specification, the same throughput with half of the frequency that in this way matches the one of the signal data. The read implementation approach is then a double buffers with two process that have the sensibility to the respectively rising and falling edge of the SCLK\_in clock input port. On the other side a proper kernel module has been developed implementing the Xilinx FIFO core communication protocol, in this way the kernel spools data from the FIFO buffer to the user allocated memory. Other solutions could be certainly applied to read data from the device, for example the Zynq is equipped with two high speed embedded DMA channels that are able to write data directly into DDR relieving the CPU of the mem-copy load. The proper transfer architecture should be always chosen in relation with the overall project complexity and

goals. The system has been then tested for long lasting acquisition pulses at full speed 2 MSPS output and it resulted to be able to work properly in continuous mode.



**Figure 10-4 scc52460 Module internal design**

The code which implements the GNU Linux driver to drive the FIFO is reported in Appendix A.

# Chapter 11.

## ADC DMA interface

### 11. 1. The Zynq7000 DMA overview

The DMA controller (DMAC) uses a 64-bit AXI master interface operating at the CPU\_2x clock rate to perform DMA data transfers to/from system memories and PL peripherals.

The transfers are controlled by the DMA instruction execution engine.

The DMA engine runs on a small instruction set that provides a flexible method of specifying DMA transfers.

This method provides greater flexibility than the capabilities of DMA controller methods.

The program code for the DMA engine is written by software in to a region of system memory that is accessed by the controller using its AXI master interface.

The DMA engine instruction set includes instructions for DMA transfers and management instructions to control the system.

The controller can be configured with up to eight DMA channels. Each channel corresponds to a

thread running on the DMA engine's processor. When a DMA thread executes a load or store instruction, the DMA Engine pushes the memory request to the relevant read or write queue.

*The DMA controller uses these queues to buffer AXI read/write transactions. The controller contains a multi-channel FIFO (MFIFO) to store data during the DMA transfers.*

The program code running on the DMA engine processor views the MFIFO as containing a set of variable-depth parallel FIFOs for DMA read and write transactions.

The program code must manage the MFIFO so that the total depth of all of the DMA FIFOs does not exceed the 1,024-byte MFIFO.

The DMAC is able to move large amounts of data without processor intervention. The source and destination memory can be anywhere in the system (PS or PL).

The memory map for the DMAC includes DDR, OCM, linear addressed Quad-SPI read memory, SMC memory and PL peripherals or memory attached to an M\_GP\_AXI interface.

The flow control method for transfers with PS memories use the AXI interconnect. Accesses with PL peripherals can use the AXI flow control or the DMAC's PL Peripheral Request Interface.

There are no peripheral request interfaces directed to the PS I/O Peripherals (IOPs). For the PL peripheral AXI transactions, software running on a CPU is used in a programmed IO method using interrupts or status polling.

The controller has two sets of control and status registers. One set is accessible in secure mode and the other in non-secure mode.

Software accesses these registers via the controller's 32-bit APB slave interface.

The entire controller is either operated in secure or non-secure mode; there is no mixing of modes on a channel basis.

Security configuration changes are controlled by slcr registers and require a controller reset to take effect.

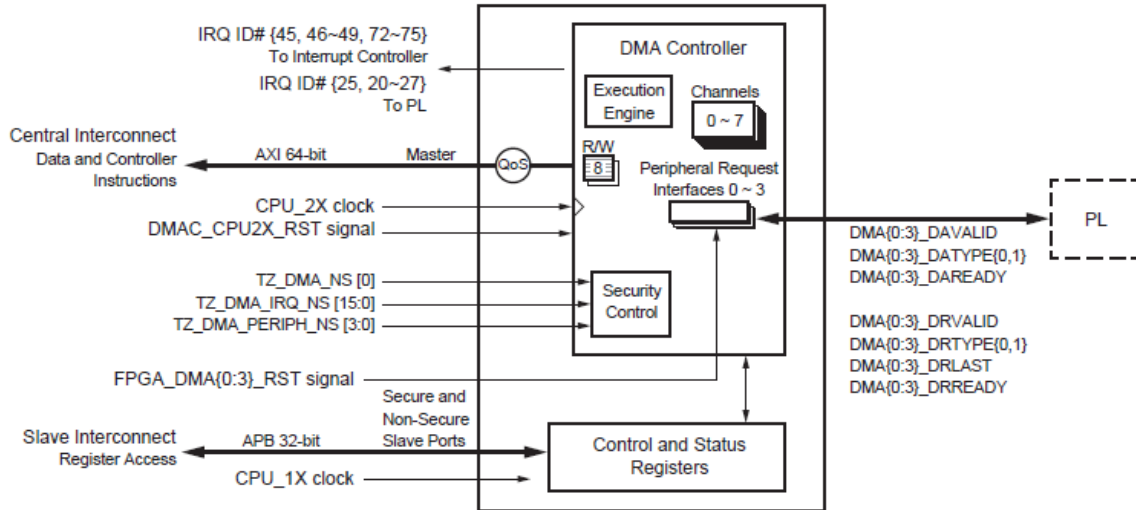
## 11.2. The Xilinx DMA logic

On Zynq7000 architecture there are four sets of DMA controller flow control signals for use by up to four PL slaves connected via the M\_AXI\_GP.

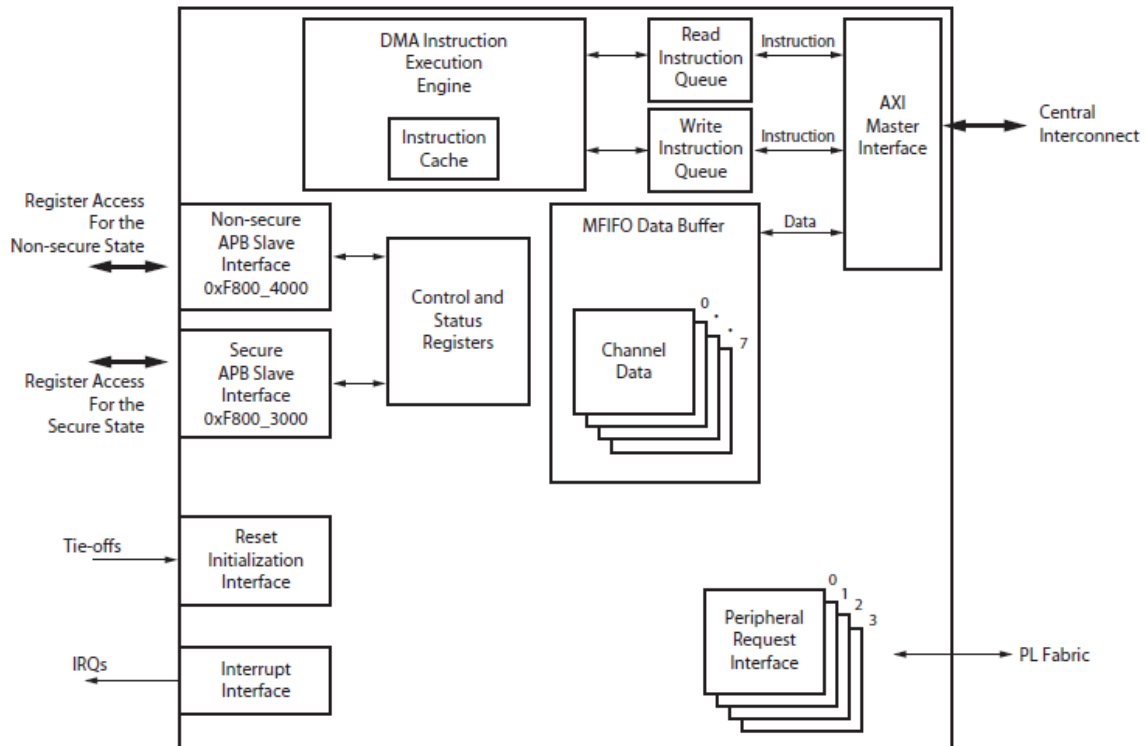
These four sets of flow control signals correspond to DMA channels 4 through 7.

Type	Signal	PL Signal Name	I/O
Clock and Reset	Clock	DMA[3:0]ACLK	I
Request	Ready	DMA[3:0]DRREADY	O
	Valid	DMA[3:0]DRVALID	I
	Type	DMA[3:0]DRTYPE[1:0]	I
	Last	DMA[3:0]DRLAST	I
Acknowledge	Ready	DMA[3:0]DAREADY	I
	Valid	DMA[3:0]DAVALID	O
	Type	DMA[3:0]DATYPE[1:0]	O

**Table 11-1 DMA Req/Ack signals on PL section**



**Figure 11-1 DMA controller interconnection.**



**Figure 11-2 DMA controller block diagram.**

The DMAC contains an instruction processing block that enables it to process program code that controls a DMA transfer. The DMAC maintains a separate state machine for each thread.

- Channel arbitration
  - Round-robin scheme to service the active DMA channels
  - Services the DMA manager prior of servicing the next DMA channel
  - Changes to the arbitration process are not supported
  
- Channel prioritization
  - Responds to all active DMA channels with equal priority
  - Changes to the priority of a DMA channel over any other DMA channels are not supported



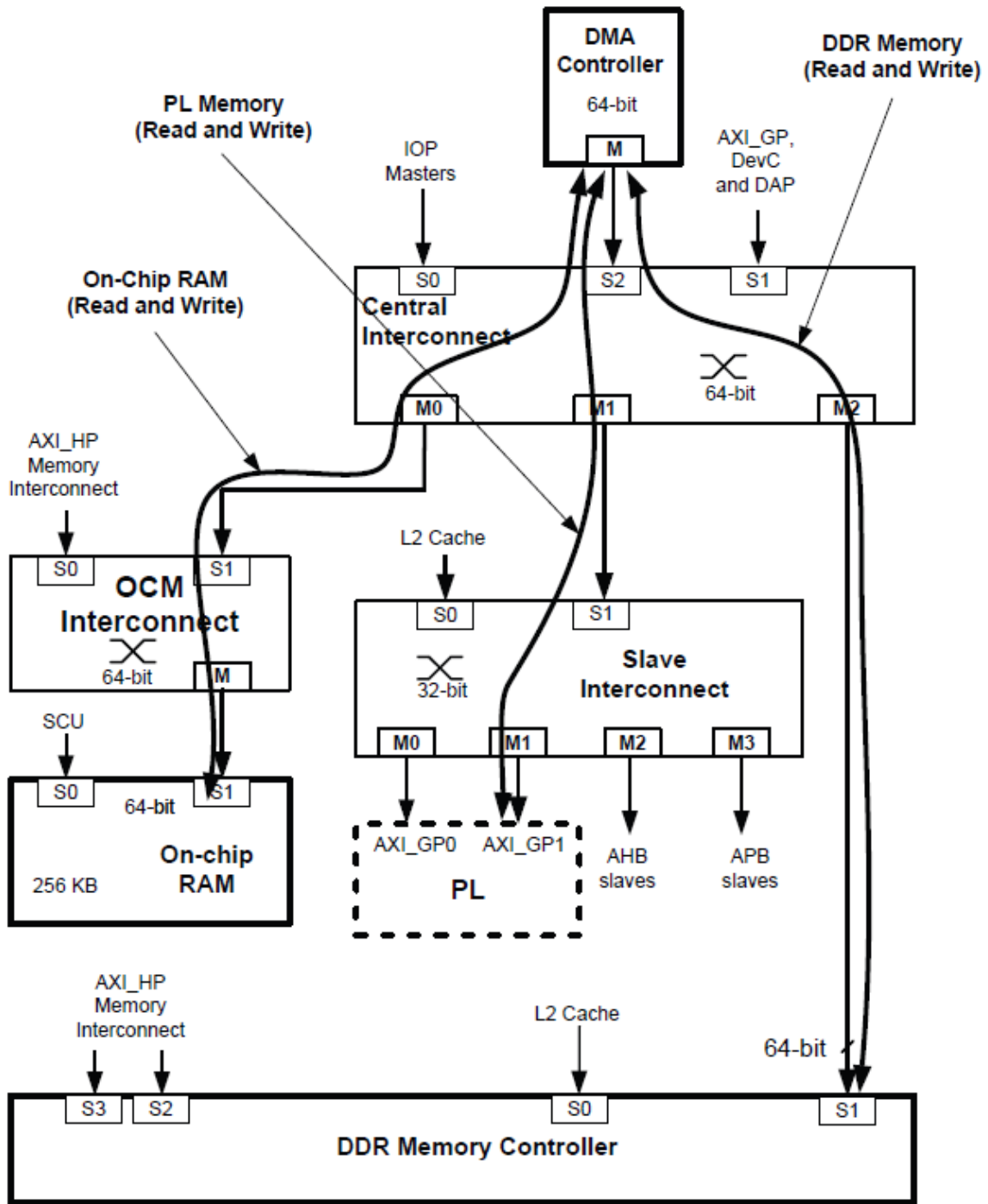


Figure 11-3 DMA access diagram.

### 11.3. AXI-DMA and streaming data acquire, block diagram

Standard way to controlling the movement of data between the memory controller and other peripherals is to route all data transfers via the processor. This method of memory transfer is known as programmable I/O, and allows the system to process memory transfers with a minimum amount of resources.

This approach requires that the peripheral and the processor are situated on the same bus, and the processor acts as a central point of communication between all peripherals and the memory.

If the number of memory transfer requests between peripherals and memory is high, the processor could spend a lot of time performing memory transfers and less time performing other computations.

One way of reducing the burden on the processor is to use Direct Memory Access (DMA) to perform memory transfers.

Using this approach, the processor issues a memory transfer request to the DMA controller, which will then perform the memory transaction.

This allows the processor to perform other tasks while the DMA controller performs the transfer.

In this situation, the DMA controller acts as both a bus master and a bus slave.

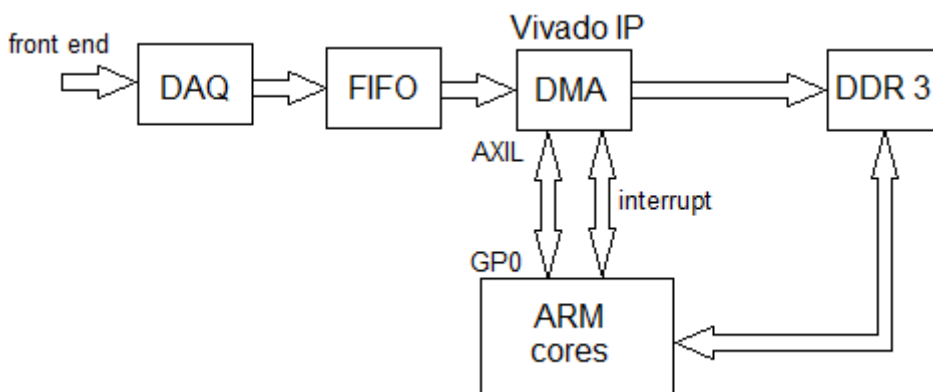
As a master, the DMA controller communicates with the memory controller whilst also arbitrating for the bus.

While acting as a slave, the DMA controller sets up memory transfers by responding to requests from bus masters (the processor, in most cases).

In order to initiate the transaction, the DMA controller must be provided with the following information:

- **Source address**, the address where the data will be read from.
- **Destination address**, the location where the data should be written to.
- **Transfer length**, the number of byte that could be transferred

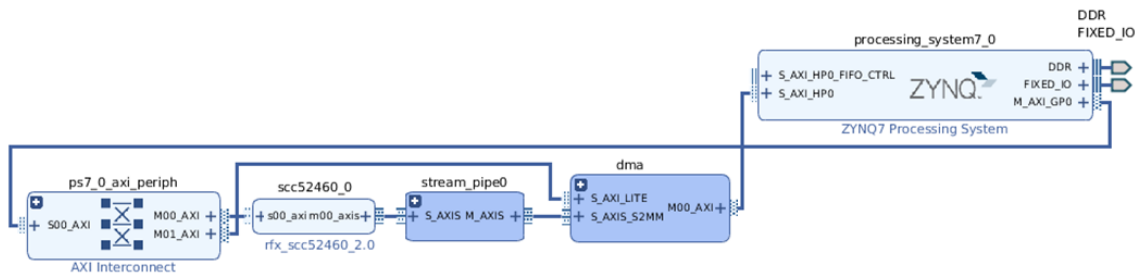
### 11.4. Structure of the AXI dataflow



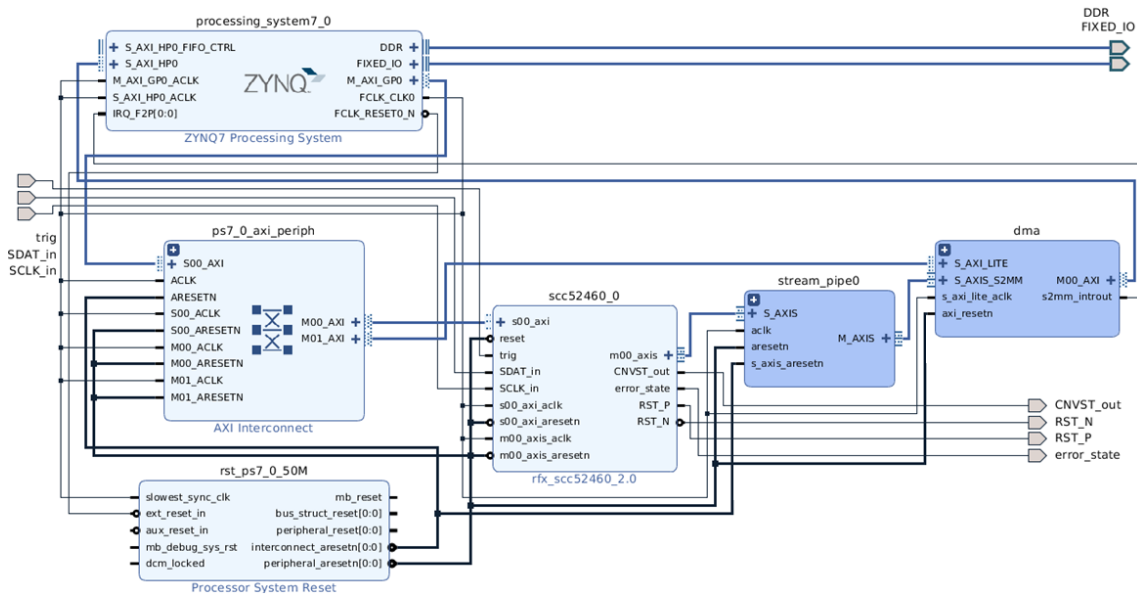
**Figure 11-4 block diagram of the interfacing principle with the hardware module**

The overall connection schema is presented in Figure 11-4 in which the analog frontend modules is shown connected to a AXI data chain that passes through a FIFO buffer and the DMA block toward the actual RAM target location. The DAQ here

represents all the previously described acquisition mechanism. The AXI stream data channel is also buffered using a AXIStream FIFO logic core to ensure the possible delay caused by the configuration time of the DMA block that the CPU performs at each transferred data chunk. A more detailed schema of this data pipe is shown in Figure 11-5 and Figure 11-6: here the Zynq internal connections between the FPGA and the processor logic is represented by the “processing\_system7\_0”, the DAQ module by the “scc52460\_0” and the AXI channel configuration by “stream\_pipe0”. Figure 11-6 also shows that the PS unit is here connected to the “dma” block with two independent busses: the GP0 port connected to an AXI-LITE slave interface of the dma IP and the HP0 connected to an AXIStream interface. The former is the access to configuration registers handling the transaction information such as the address of start/end of writing. The latter is the actual High-Performance data channel to the Programmable Logic to Memory interconnect and DDR memory interface.



**Figure 11-5 AXI-DMA logic chain and HDL implementation**



**Figure 11-6 ATCA MIMO ISOL MODULE implemented on RFX\_scc52460 logic IP with AXI interconnect to stream pipe and DMA**

In Figure 11-7 the “stream\_pipe0” block and the dma block schemas have been expanded to show the internal organization of the acquisition chain and the dma interconnection. The acquisition, as already mentioned, is based on the same channel configuration used for the FIFO AXI data movement in the previous section where the AXIStream FIFO buffer has been added. The dma block exploits the standard Xilinx AXI Direct Memory Access IP connected to the interconnect component here

represented by a Xilinx SmartConnect instance. The Xilinx dma component supports many kinds of connection and also the scattered memory acces through the so called Scatter Gather dma read/write. For the purpose of the present study the non trivial software implementation of this data transfer method has been skipped and a simple access to kernel contiguous allocated memory has been used.

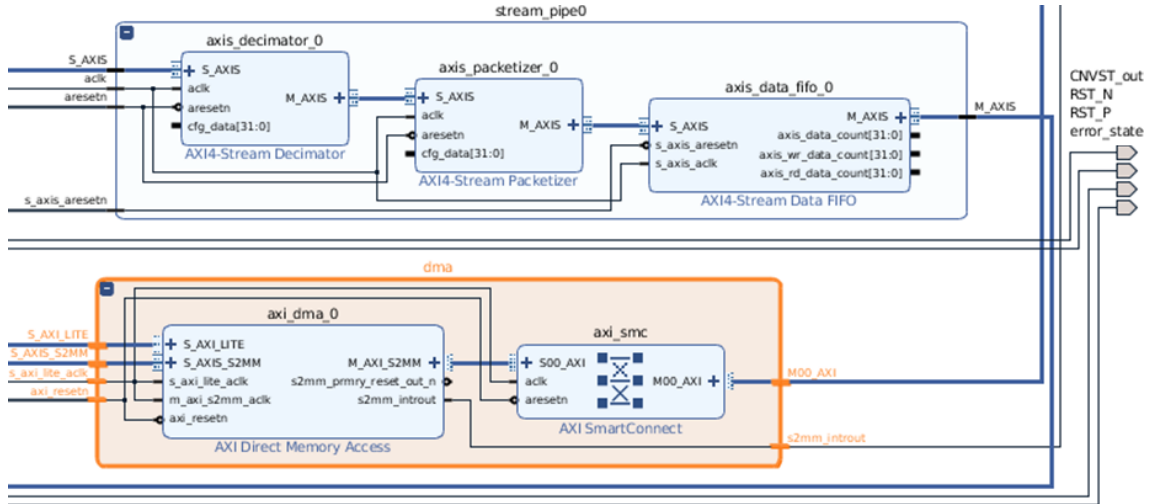


Figure 11-7 AXI-DMA and stream\_pipe IP internal HDL Vivado rappresentation

# Chapter 12. Final tests and results

## 12. 1. Python

The tool used for collect and analyze data is python 2.7.

Three modules are used for scientific data processing, data plot and math processing.

- Scipy
- Pyplot
- Numpy

To perform the tests, large data arrays are created with a size of one or two million of elements than saved in enumerated files.

s0001.py, s0002.py, etc

The entire file collection is later plotted.

The python program is resident in the Redpitaya and saved with analysis.sh

Is executed by the console with the command line:

```
. go_analisi.sh
```

This allows to execute a source code instead a compiled program. Since python is an interpreted language it is necessary to execute it from a bash shell

The go\_analysis.sh script contains the following sequence of actions:

```
cat scc52460_wrapper.bit > /dev/xdevcfg
./axwrite 0x43c0000 1 0
./axiwrite 0x43c1000 3 90 20 450
sleep 1
insmod scc52460.ko // where k is the kernel files extension
```

The first command is responsible to write the bitstream file into the special device devoted to the FPGA configuration. Once all the file has been written to that device file the FPGA is completely reprogrammed and the logic starts to operate.

The axiwrite command is a custom software we developed to easily access the memory mapped logic registers from the user space. Here the first argument is the hex code of the pointed memory address, the second parameter is the number of 32 bit registers that is going to be touched and the following arguments are the values to be written. The second line refers to the register 0x43c00000 that has been mapped to the logic AXI stream decimator and it sets the decimation to 0 that is no decimation enabled. The other axiwrite command points to address 0x43c10000 that refers to the ADC module IP interface and sets the parameters specific to the actual acquisition. The three reported numbers are: the store interval, the start of conversion time and the total period, all coued in number of clock periods form the input clock to the ADC module multiplied by 10E6.

At this point the module starts to acquire data from the ADC and to store in the FIFO IP. The with insmod that follows load the scc52460 kernel module in memory activating the FIFO interface with the kernel. In this way we can use a custom executable that

talks with the kernel driver by means of ioctl calls and mmap access to reset and read data from the FIFO.

In order to have a good statistical estimate of the frequency analysis it is necessary to estimate a number of samples ranging from 100000 to 500000.

There are two instances of axi\_write. The first activates the decimator, the second sets the system.

## 12. 2. Acquisitions test

Three types of tests are performed to determine the quality of the system before connect the probe.

1. Noise measurement.
2. Measure in amplitude.
3. Frequency estimation.

Two instances of the python program are executed for each test, one to plot the frequency diagram and one to obtain the spectrum diagram.

The P001.py file implements the reader for the data array to which the FFT plot is to apply.

We carry out the following steps in sequence:

1. Noise Measurement by applying a 50 $\Omega$  resistive terminator to the ADC input. 1000000 samples for each data set are captured.
2. Voltage measurement, a 10K $\Omega$  potentiometer applied to a +/- 10V dual voltage is connected in input. 100,000 samples are taken. The potentiometer is manually moved to a fixed voltage value before launching the acquisition.
3. Frequency measurements, of a sinusoid, generated by a laboratory instrument is converted for a second or 2 million samples. The amplitude provided is fixed to 10.00Vpp with frequencies ranging from 10Hz to 1MHz.
4. Acquisition test of the magnetic probe, two sessions are performed first by setting the decimation to 10 for one million samples, the second setting the decimation to 9 for two million samples.

The next instance of the HDL program, executed by axi\_write command was set with decimation parameter equal to 10, using the line command:

```
Axi_write 0x43c00000 1 10 //decimation parameter = 10
```

Noise acquisition test, with ohmic load of 50Ω termination. Each set of data corresponds to 100,000 samples.

Data is plotted twice to get two graphs, one for frequency and one for spectrum.

Voltage acquisition, First test with potentiometer 10K, connected to a +/- 10V dual supply voltage.

```
./scc52460 -test
./dev/scc562460 2 100000 P1/vx00n
```

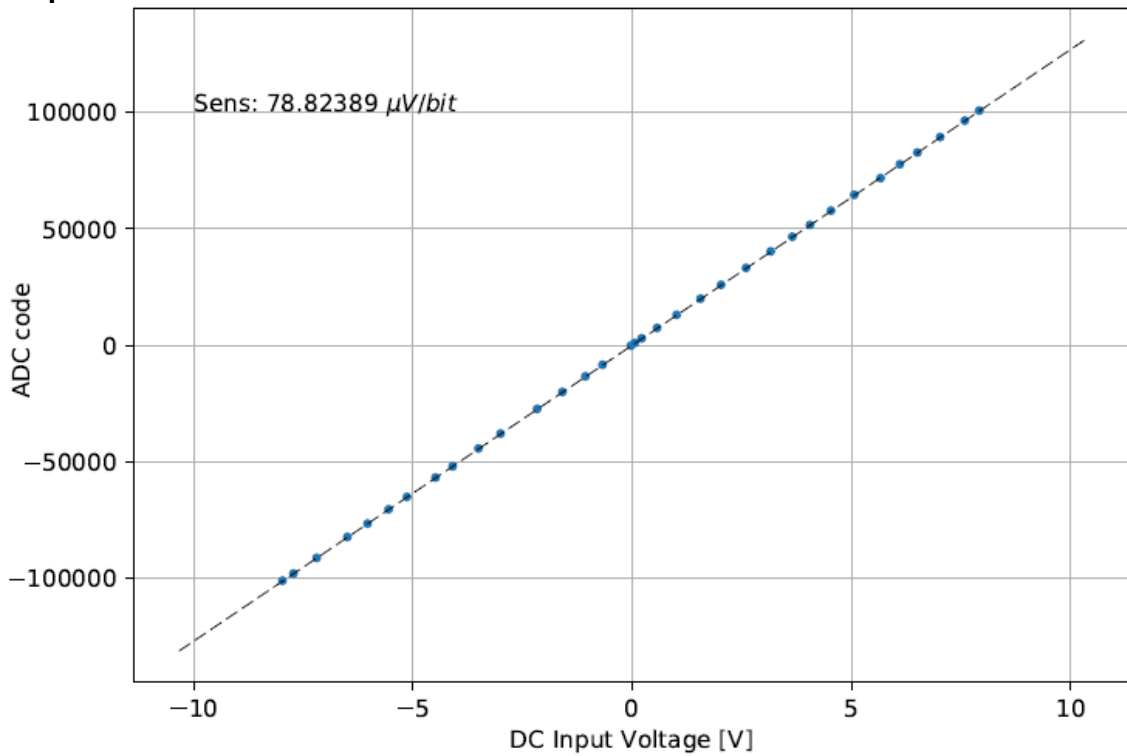
The term n is the file number where to save the 100,000 samples you captured.

V	Data file	V	Data file
-0.033	001	+7.020	019
+0.055	002	+7.580	020
+0.055 no tester load	003	+7.91	021
+0.055 switch off neon	004	-0.681	022
+0.212	005	-1.072	023
+0.564	006	-1.597	024
+1.009	007	-2.172	025
+1.555	008	-3.006	026
+2.023	009	-3.511	027
+2.593	010	-4.100	028
+3.155	011	-4.490	029
+3.647	012	-5.140	030
+4.050	013	-5.560	031
+4.530	014	-6.040	032
+5.060	015	-6.500	033
+5.660	016	-7.200	034
+6.100	017	-7.730	035
+6.500	018	-7.980	036

**Table 12-1 List of ADC module input voltage sensitivity test files**



**Amplitude test.**



**Figure 12-1 ADC module input voltage sensitivity**

Module Sensitivity and linearity input stage and adc AD7641 converter.  
this graph shows the sensitivity and linearity of all the components of the ATCA MIMI module circuit and not just of the ADC to the continuous.

**Frequency tests.**

Each line in the table is set of 2,000,000 Samples for Acquisition time of 1 Second.

The applied signal is sinusoidal  $V_{pp} = 10V$  or  $\pm 5V_p$

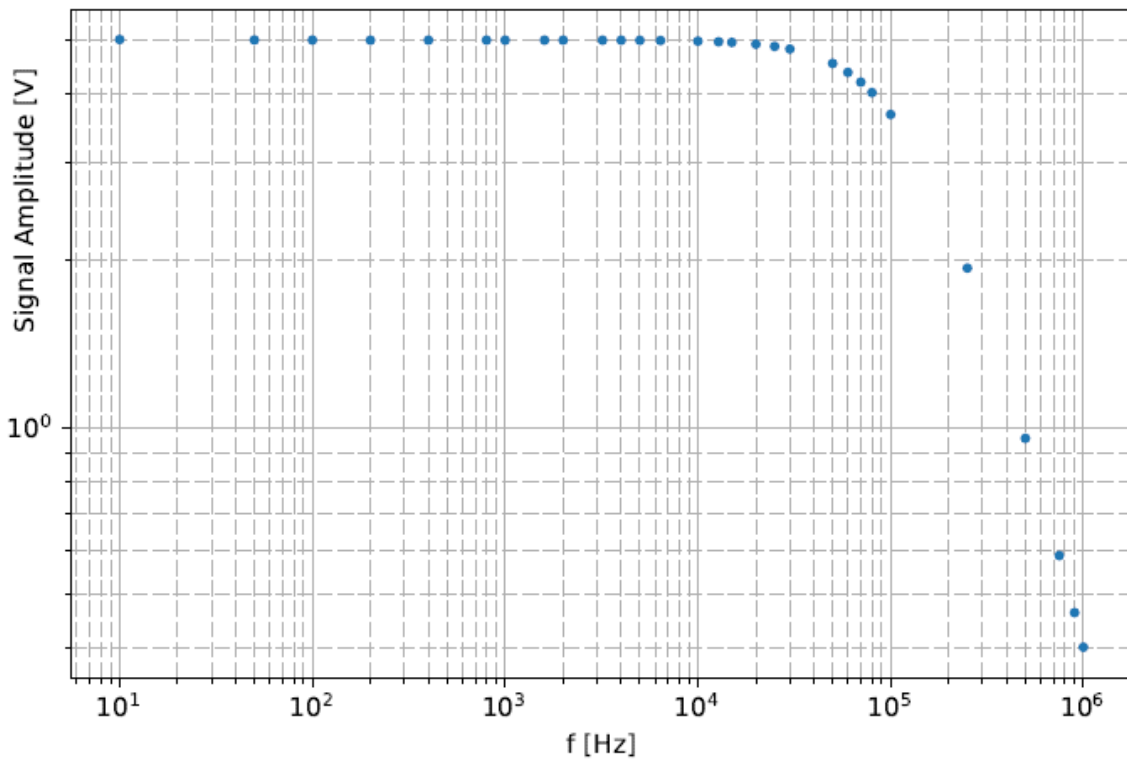


**Figure 12-2 functions generator HP33120A**

frequency	File #	frequency	File #
10Hz	001	15000Hz	016
50Hz	002	20000Hz	017
100Hz	003	25000Hz	018
200Hz	004	30000Hz	019
400Hz	005	50000Hz	020
800Hz	006	60000Hz	021
1000Hz	007	70000Hz	022
1600Hz	008	80000Hz	023
2000Hz	009	100000Hz	024
3200Hz	010	250000Hz	025
4000Hz	011	500000Hz	026
5000Hz	012	750000Hz	027
6400Hz	013	900000Hz	028
10000Hz	014	1MHz	029
12800Hz	015		

**Table 12-2 List of ADC module frequency response test files**

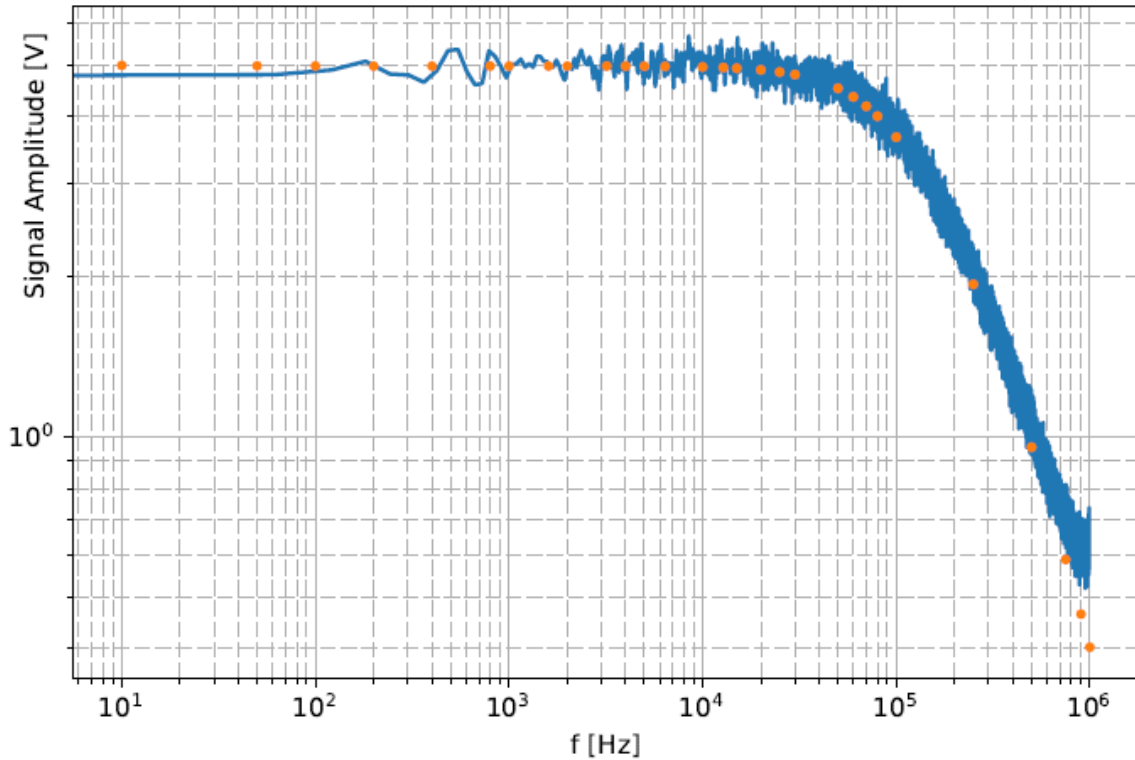
### 12. 3. Frequency response



**Figure 12-3 Frequency response of ADC module with sinusoidal input**

Amplitude in function of the frequency on the sampling signals shown in the previous table

In the graph is in evidence the cutoff at 100kHz, first order filter 1.



**Figure 12-4 frequency response of ADC module with white noise input**

Near the Nyquist frequency, on the right side of the graph, it deviates from the signal value by the effect of the Aliasing, it is therefore discosted higher.

The Graph show the frequency response obtained by fixed frequency sinusoidal signals (points - see table), and the frequency response obtained by FFT of an externally generated white noise signal using the device HP 33120A.

```
import numpy as np
import matplotlib.pyplot as plt

fdir = r'data/'

m = 7.88238920E-05 # ADC module input sens V/bit

fdata = { 1 : 10, 2 : 50,
          3 : 100, 4 : 200,
          5 : 400, 6 : 800,
          7 : 1000, 8 : 1600,
          9 : 2000, 10 : 3200,
          11 : 4000, 12 : 5000,
          13 : 6400, 14 : 10000,
          15 : 12800, 16 : 15000,
          17 : 20000, 18 : 25000,
          19 : 30000, 20 : 50000,
          21 : 60000, 22 : 70000,
          23 : 80000, 24 : 100000,
          25 : 250000, 26 : 500000,
          27 : 750000, 28 : 900000,
          29 : 1000000 }
```

```

freq = np.empty( len(fdata ) )
mis = np.empty( len( fdata ) )

# -----
# aggiunge lo spettro dal white noise del generatore

fdir = r'data/'
fname = fdir+'rumore003.out' ; fs = 125E6/63
yn = np.fromfile( fname, dtype=np.int32 )
yn[0] = yn[1]

ynz = (yn - yn.mean())

from scipy import signal
f, Pxx = signal.welch( ynz, fs, nperseg=32768 )

# -----

idata = 0

# fname = fdir+ r'fx%03d.out' % 14
# y = np.fromfile( fname, dtype=np.int32 )
# y[0] = y[1]

for k in fdata.keys() :
    fname = fdir+ r'fx%03d.out' % k
    y = np.fromfile( fname, dtype=np.int32 )
    y[0] = y[1]

    mis[idata] = y.std()*m*np.sqrt(2)
    freq[idata] = fdata[k]

    idata += 1

plt.ion()

fig = plt.figure( 'amp_freq_noise' )
fig.set_size_inches( 7.4, 4.7 )
fig.clf()
plt.plot( f, np.sqrt(Pxx)*0.95 )
plt.plot( freq, mis, '.' )

#ax=plt.gca()
#ax.set_yscale('log'); ax.set_xscale('log')
#ax.set_xlim( 100, 1E6 )

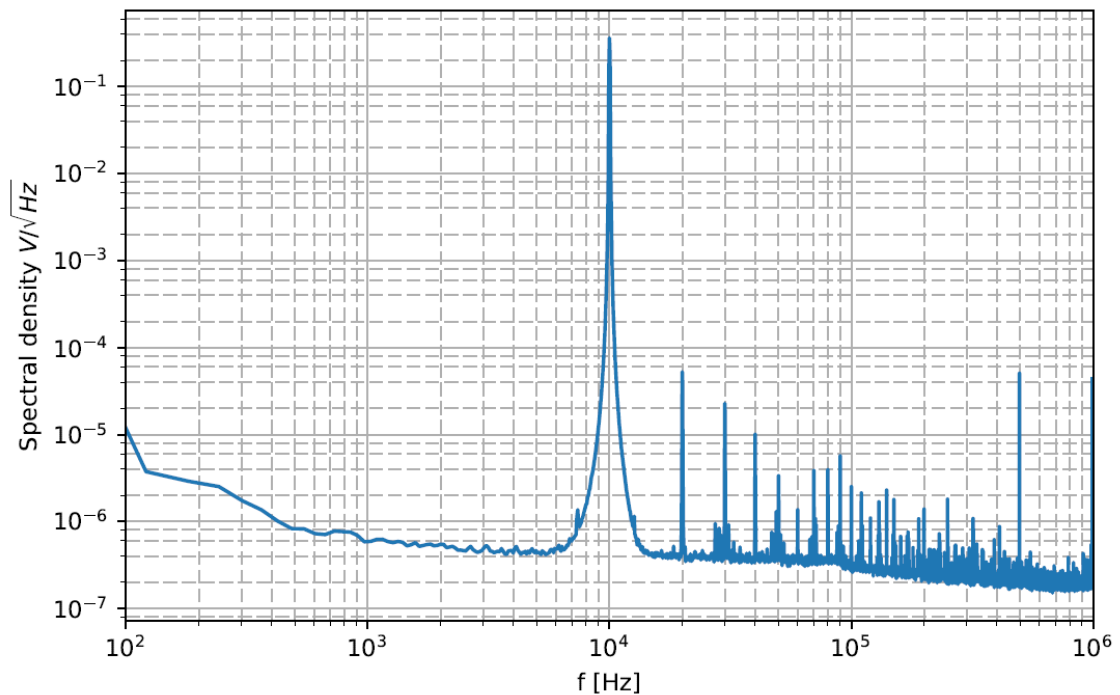
ax = plt.gca()
ax.set_yscale('log'); ax.set_xscale('log')

ax.grid(True, which='major', ls='-')
ax.grid(True, which='minor', ls='--')

ax.set_ylabel( r'Signal Amplitude [V]' )
ax.set_xlabel( 'f [Hz]' )

```

At one of the 10 kHz signals has been applied to the Fourier transform.



**Figure 12-5 Harmonic spectrum of ADC module at 10kHz sinusoidal input**

On the graph is show the spectrum of a signal at 10 kHz for S/N and distorsion

$\sim 10^6 \rightarrow 120$  dB from noise floor at 10 kHz.

We note the harmonics of the signal at  $k * 10$  kHz (20,30,40, ... kHz). The second harmonic results at  $10E-3$ , 60 dB are probably originated by the 14-bit function generator HP33120A.

A 500 kHz peak is then noted that is generated by the DC/DC converter installed on the scc52460 module.

Estimate power spectral density using Welch's method.

Welch's method computes an estimate of the power spectral density by dividing the data into overlapping segments, computing a modified periodogram for each segment and averaging the periodograms

```
import numpy as np
import matplotlib.pyplot as plt
```

```
plt.ion()
```

```
fdir = r'data/'
m = 7.88238920E-05 # ADC module input sens V/bit
```

```
fdata = { 1 : 10, 2 : 50,
          3 : 100, 4 : 200,
          5 : 400, 6 : 800,
          7 : 1000, 8 : 1600,
          9 : 2000, 10 : 3200,
          11 : 4000, 12 : 5000,
```

```

13: 6400, 14: 10000,
15: 12800, 16: 15000,
17: 20000, 18: 25000,
19: 30000, 20: 50000,
21: 60000, 22: 70000,
23: 80000, 24: 100000,
25: 250000, 26: 500000,
27: 750000, 28: 900000,
29: 1000000 }

freq = np.empty( len(fdata ) )
mis = np.empty( len( fdata ) )

# plot dello spettro del segnale a 10 kHz per rapporto S/N e distorsione.
# -----

fname = fdir+ r'fx%03d.out' % 14
y = np.fromfile( fname, dtype=np.int32 ); fs=125E6/63
y[0] = y[1]
yz = (y - y.mean())*m

from scipy import signal
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.welch.html
f, Pxx = signal.welch( yz, fs, nperseg=32768 )

fig = plt.figure( 'sig_noise' )
fig.set_size_inches( 7.4, 4.7 )
fig.clf()
plt.plot( f, np.sqrt(Pxx) )

ax = plt.gca()
ax.set_yscale('log'); ax.set_xscale('log')
ax.set_xlim( 100, 1E6 )

ax.grid(True, which='major', ls='-')
ax.grid(True, which='minor', ls='--')

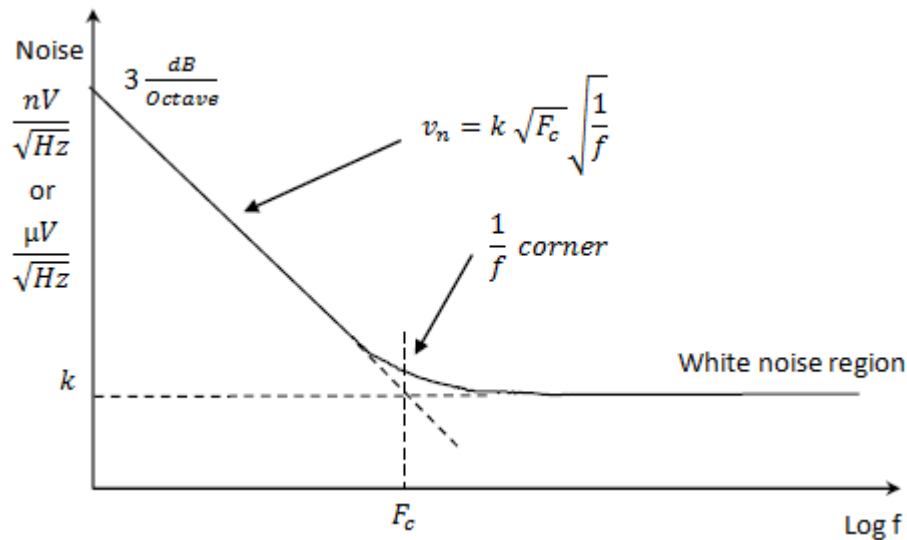
ax.set_ylabel( r'Spectral density  $\sqrt{V/\sqrt{Hz}}$ ' )
ax.set_xlabel( 'f [Hz]' )

```

## 12. 4. The noise spectral density

The ATCA MIMO ISOL modules under examination should be used in streaming applications for long periods, in the order of 10s in the RFX experiment, for 3600s in the future ITER experiment.

The problem of the reverse ramp of the noise intensity with the frequency entails an output drift due to the necessary integration of the analog signal.



**Figure 12-6 General characteristic of Op. Amp voltage noise.**

While the spectral density is constant in the right side of the corner frequency at the left, it rise to the frequency decrease at 3 dB/octave.

The power spectral density in this region is inversely proportional to frequency, and therefore the voltage noise spectral density is inversely proportional to the square root of the frequency.

The general equation which describe the voltage noise spectral density in the  $\frac{1}{f}$  region is:

$$v_n = k \sqrt{F_c} * \sqrt{\frac{1}{f}}$$

Where  $k$  is the level of the “white” voltage noise level, and  $F_c$  is the  $\frac{1}{f}$  corner frequency.

The best analog devices, for standard purpose, show the corner frequency in the range 1Hz to 10Hz. Medium quality operational amplifiers designed with JFET technology can have an  $F_c$  at about 100Hz. above this threshold, for example 1kHz or 2kHz, it makes no sense to investigate in our application.

Due to the fact that the noise spectral density is the function of frequency in order to obtain the rms noise, the noise spectral density curve should be integrated into the used device frequency application band.



Indicating with  $F_L$  the lower limit frequency, in the interest band, with  $F_C$  upper limit, in the  $\frac{1}{f}$  region, the rms noise is:

$$v_{n,rms}(F_L, F_C) = v_{nw} \sqrt{F_C} \sqrt{\int_{F_L}^{F_C} \frac{1}{f} df} = v_{nw} \sqrt{F_C \ln \left[ \frac{F_C}{F_L} \right]}$$

Where the  $v_{nw}$  is the voltage noise spectral density in the white noise region.

At very low frequencies, in the  $\frac{1}{f}$  region  $F_C \gg (F_H - F_L)$  the expression for the rms noise is:

$$v_{n,rms}(F_H, F_L) \approx v_{nw} \sqrt{F_C \ln \left[ \frac{F_H}{F_L} \right]}$$

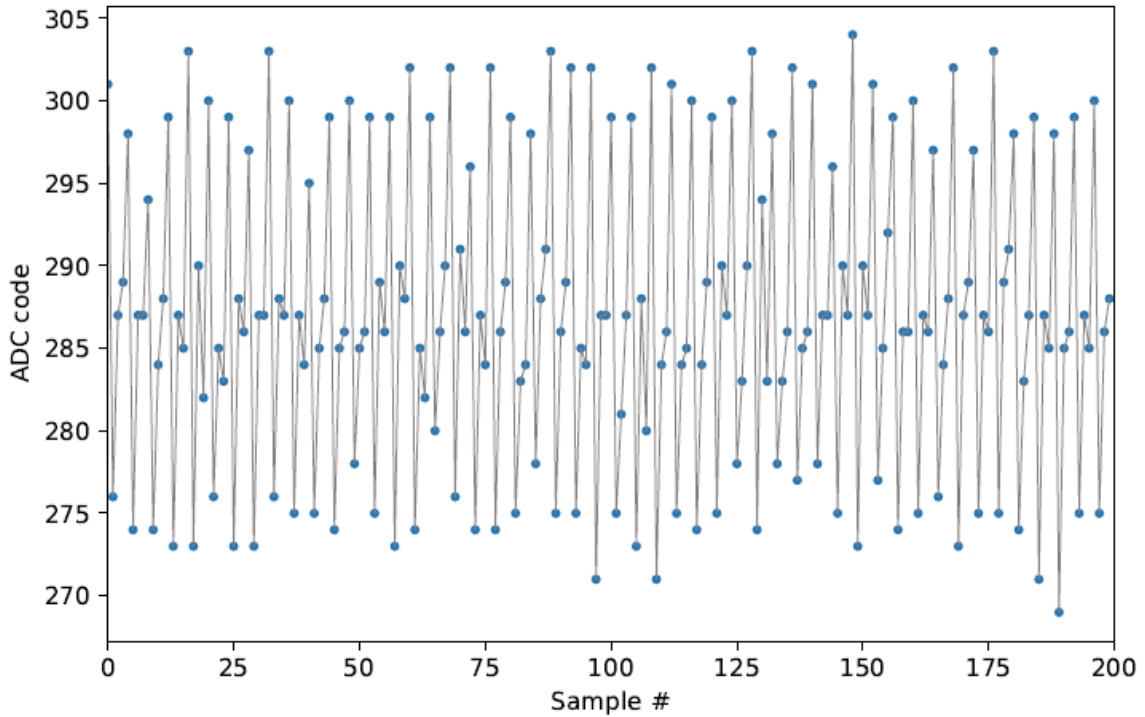
Note that there is no way of reducing this 1/f noise by filtering if operation extends to dc. Making  $F_H = 0.1$  Hz and  $F_L = 0.001$  Hz still yields an rms 1/f noise of about 18 nV rms, or 119 nV peak-to-peak. The point is that averaging results of a large number of measurements over a long period of time has practically no effect on the rms value of the 1/f noise.

This well known issue shows rising attention within the fusion community where high stability experiments ask for longer pulses duration. Accordingly many solutions have been tested, for both the analog and digital integration equipments, each one with advantages and disadvantages. The most promising approach, that will be adopted for ITER experiment, seems to be the use of a chopper stabilized op amp, to remove the low frequency noise [22]. Nevertheless this brings other problematics and further noise components that must be managed both in the analog stage and numerically in the acquired signals as described in [23]. We will restrict our study to the classical approach performing the detailed characterization of the standard integrator component and exposing the limits of the technology to show at which extent a simple DAQ system can be used to integrated signals.

## 12. 5. Measurement of intrinsic noise

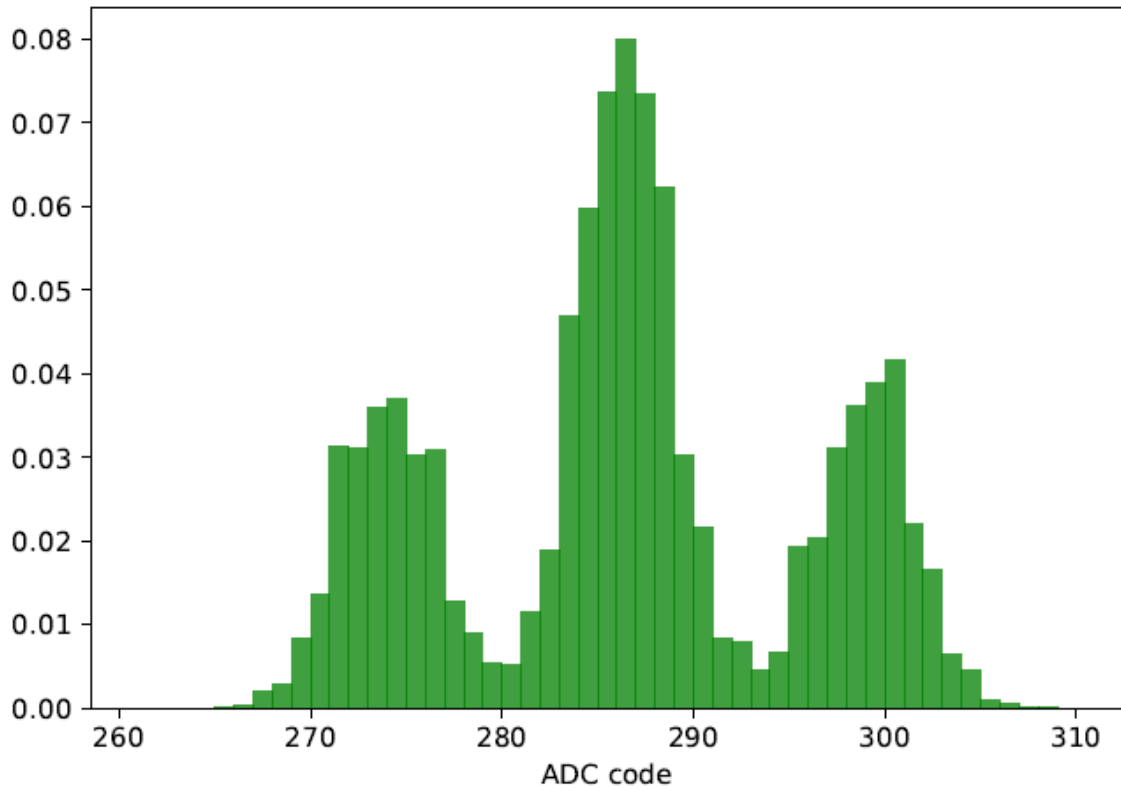
In this kind of measures in order to carry out numerical integration it is particularly important to have low input noise, it is intended to check if the input noise is compatible with the integration number.

All tests are performed with the input connected to a 50Ohm terminator.

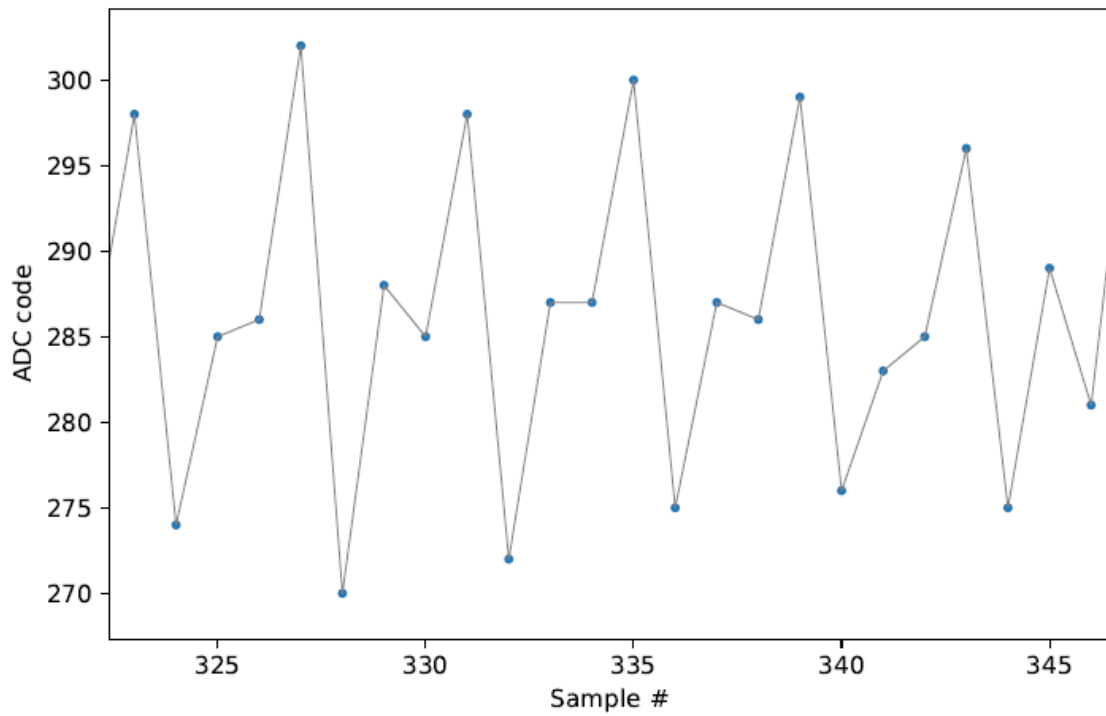


**Figure 12-7 Noise sampling of ADC module with 50 Ohm terminated input**

In this chart, the distribution of the intrinsic analog noise in the center band and the superimposed impulse noise, better seen in the next image, due to the DC/DC converter switches performed on every 4 samples.

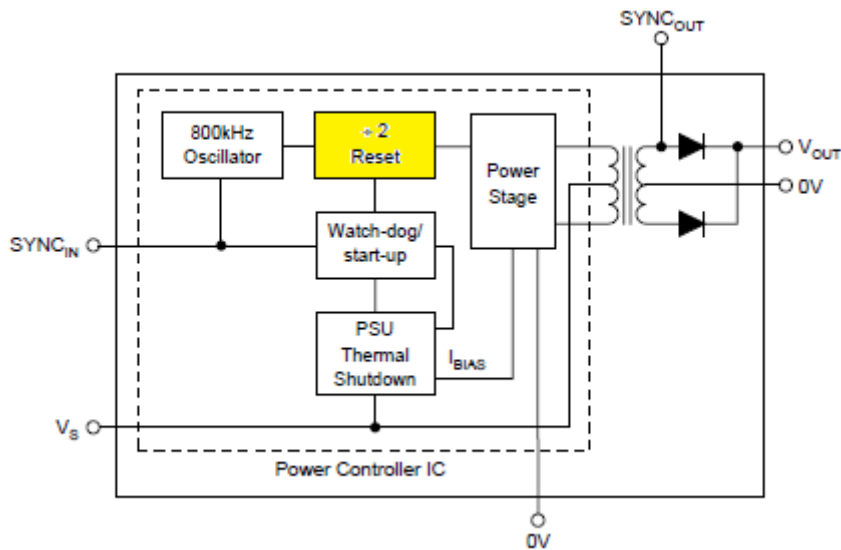


**Figure 12-8 Sampled Noise histogram of ADC module with 50 Ohm terminated input**



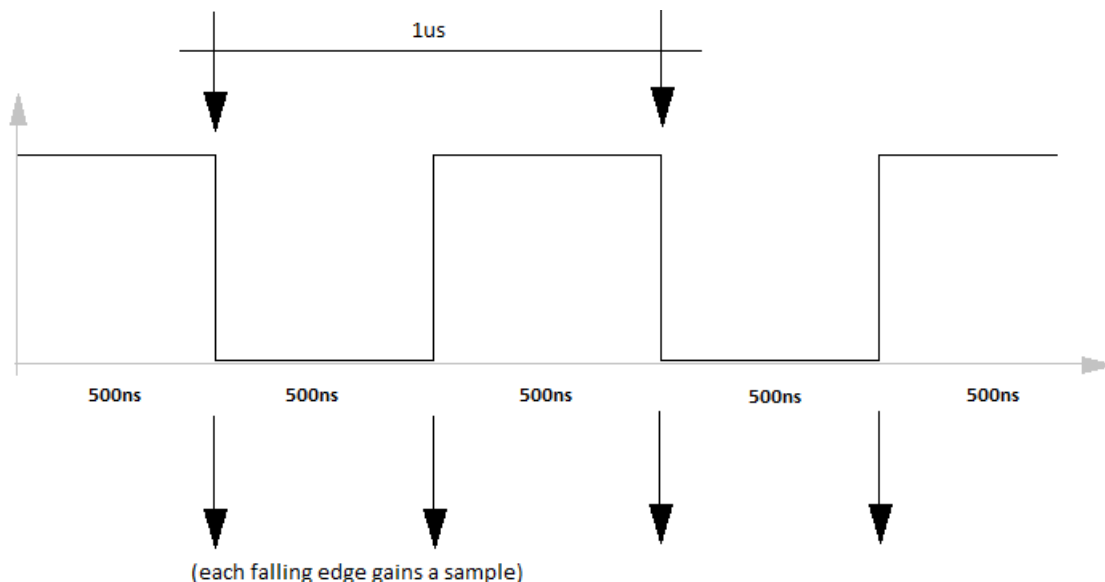
**Figure 12-9 Noise sampling of ADC module with 50 Ohm terminated input (Zoom)**

It is evident that every 4 points, the previous graph in (Figure 12-9 Noise sampling of ADC module with 50 Ohm terminated input (Zoom)) , the DC/DC converter turns on and off generating impulse noise.



**Figure 12-10 internal divider integrated on DC-DC converter**

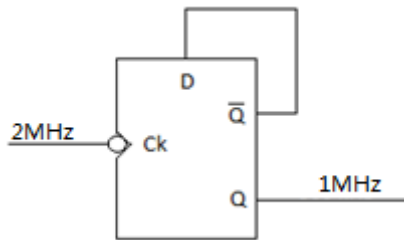
**Noise\_ist** Noise histogram, the presence of the two lateral spikes is probably introduced by the action of the dc / dc converter insert on the circuit context. For comparison is show the same histogram present in the ADC datasheet. The histogram includes 50,000 samples at a time of 25 milliseconds.



**Figure 12-11 Timing sequence from converter start to DC-DC synchronization**

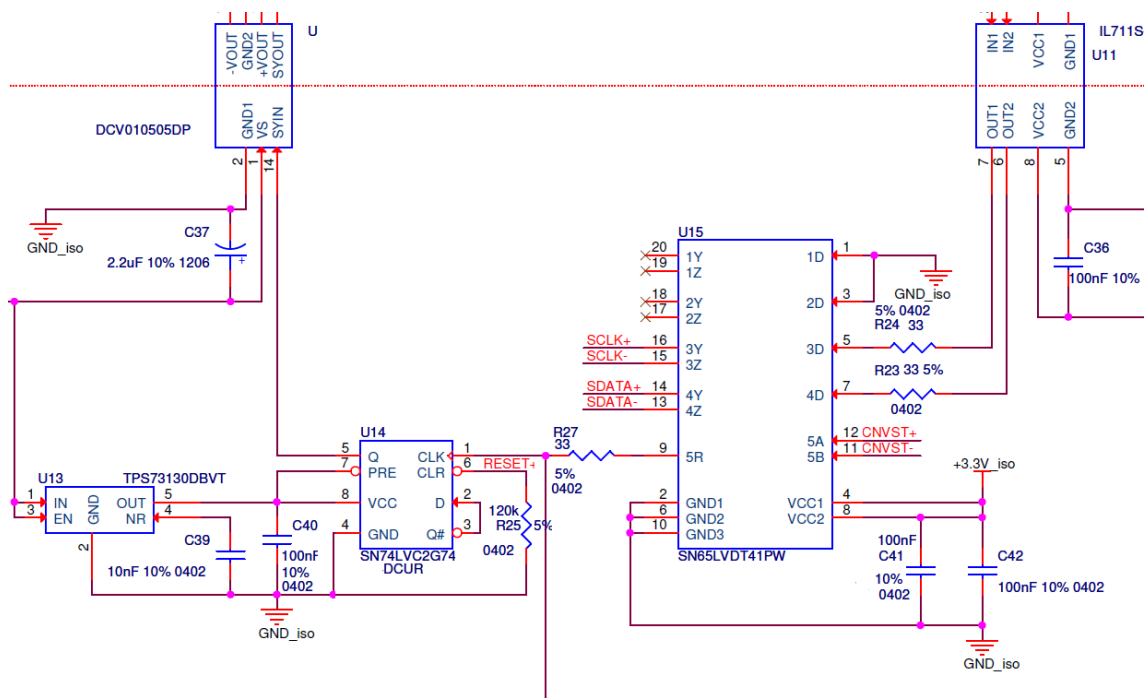
Each falling edge is a sample so every two clock in input correspond to one in the output of the FlipFlop on the wiring diagram.

For each falling edge is generate a switching of DC/DC converter that introduces noise to 500kHz.



**Figure 12-12 Flip Flop divider**

The SN74LVC2G74 is a Single Positive-Edge-Triggered D-Type Flip-Flop With Clear and Preset, inserted in the circuit of the ATCA-MIMO-ISOL module in order to synchronize the power of the analog conversion device and stabilize the speeds.



**Figure 12-13 The SN74LVC2G74 Flip flop in the diagram context.**

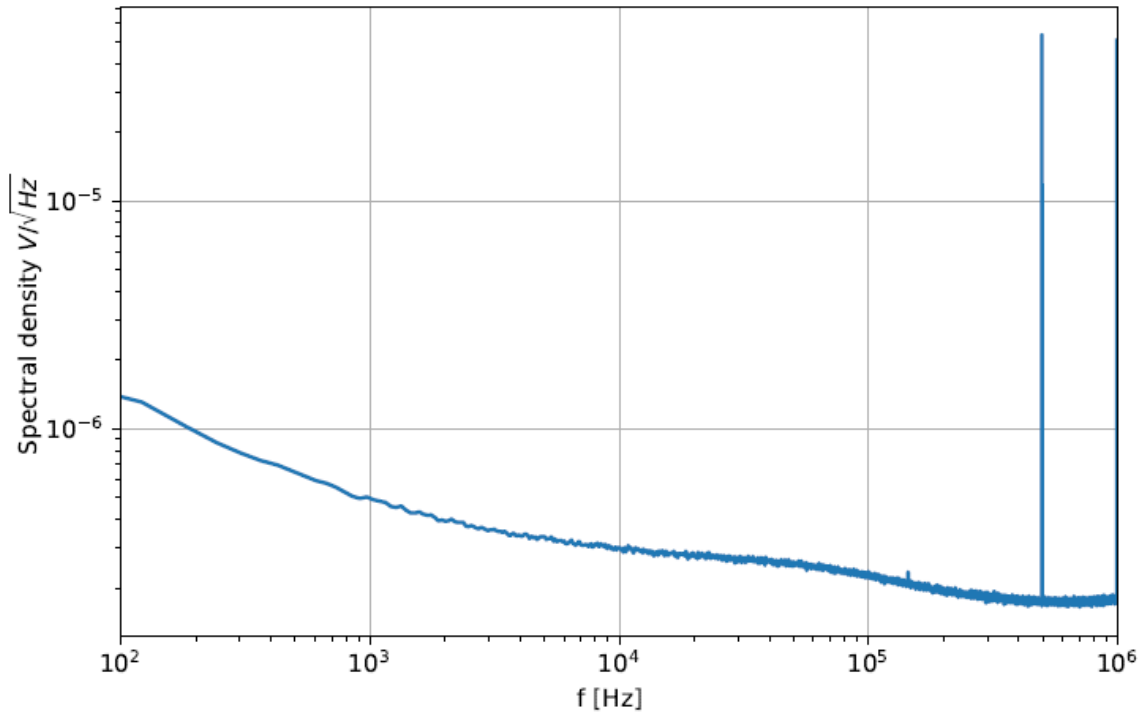
A low level at the preset (PRE) or clear (CLR) input sets or resets the outputs, regardless of the levels of the other inputs.

When PRE and CLR are inactive (high), data at the data (D) input meeting the setup time requirements is transferred to the outputs on the positive-going edge of the clock pulse.

Clock triggering occurs at a voltage level and is not related directly to the rise time of the clock pulse.

Following the hold-time interval, data at the D input can be changed without affecting the levels at the outputs.

Following graph are the noise resampled by placing the ADC reference to Pitaya's ground.

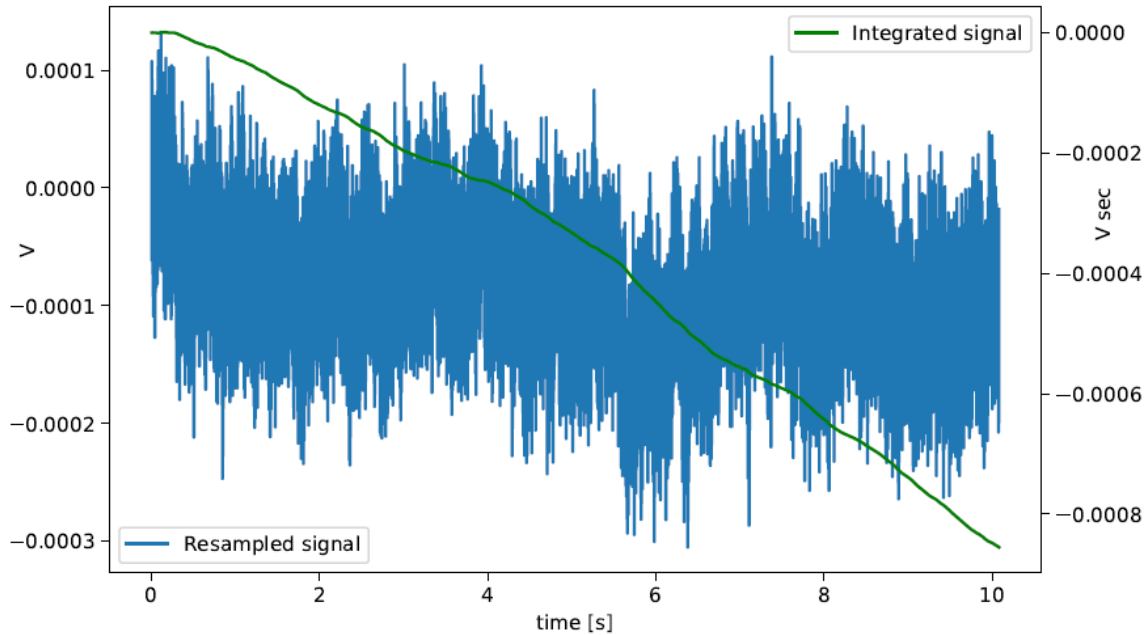


**Figure 12-14 Sampled Noise spectrum of ADC module**

**Noise\_spectr** sampled noise spectrum in absence of signal..

In the left part of the graph, there is the presence of low frequency noise that brings the line beyond  $10^{-6}$ .

## 12. 6. Signal numerical integration



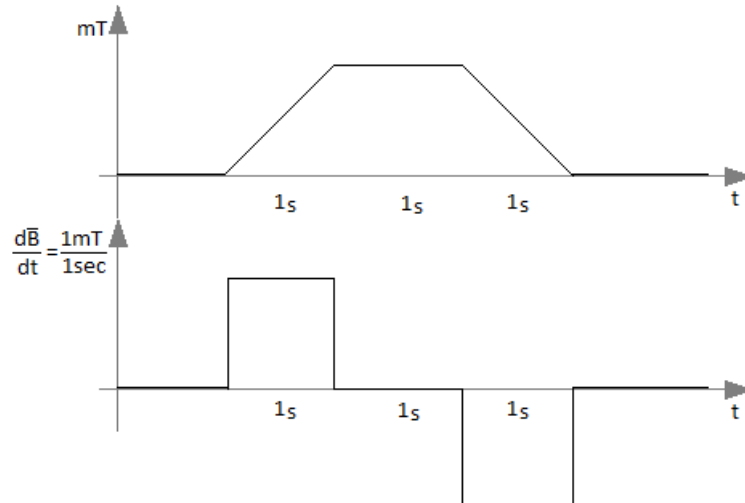
**Figure 12-15 Resampled input and time integrated signal**

The figure show a signal with a duration of 10 seconds, resampled at 10kSample/s from the original 2 Msample/s, where each point is obtained by averaging 200 points of the original signal. In this case the ADC input was connected to a 50Ω terminator.

The goal is to check if the circuit noise is compatible with the aim to acquire the RFX magnetic probes and integrate their signal numerically.

Given the probe typical area of 0,024m<sup>2</sup>, the minimum desired signal that has to be detected is due to a magnetic field variation as shown in Figure 12-16 Minum level reference magnetic signal. , In this baseline case we consider the voltage induced on the magnetic probe by a variation of 1 mT in 1 second:

$$\frac{1mT}{1sec} * 0,024 [m^2] = 24 * 10^{-6} Volt$$



**Figure 12-16 Minum level reference magnetic signal.**

On the other hand the maximum expected signal can be obtained by the wide field variation due to say a fast plasma termination event, which occurs in about 2 ms. So the maximum peak voltage expected is:

$$0.5 \text{ T} / 2 \text{ ms} \sim 6 \text{ Volt}$$

The required dynamic range will be the the ratio of these two exterem cases:

$$\text{max/min} = 6 \text{ V} / 24 \mu\text{V} = 250'000$$

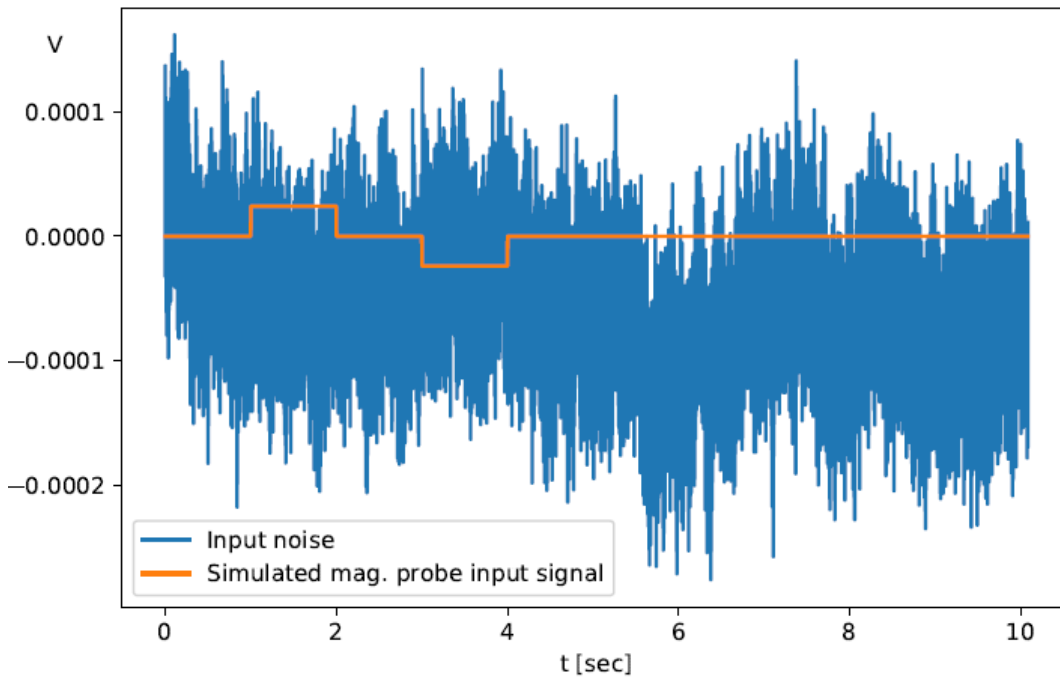
Thus the required number of bits of the ADC to manage this dynamic range is:

$$\log_2( 250'000 ) \sim 18$$

being  $2^{18} = 256\text{K}$  . So in principle the ADC module should be able to correctly detect these signals.



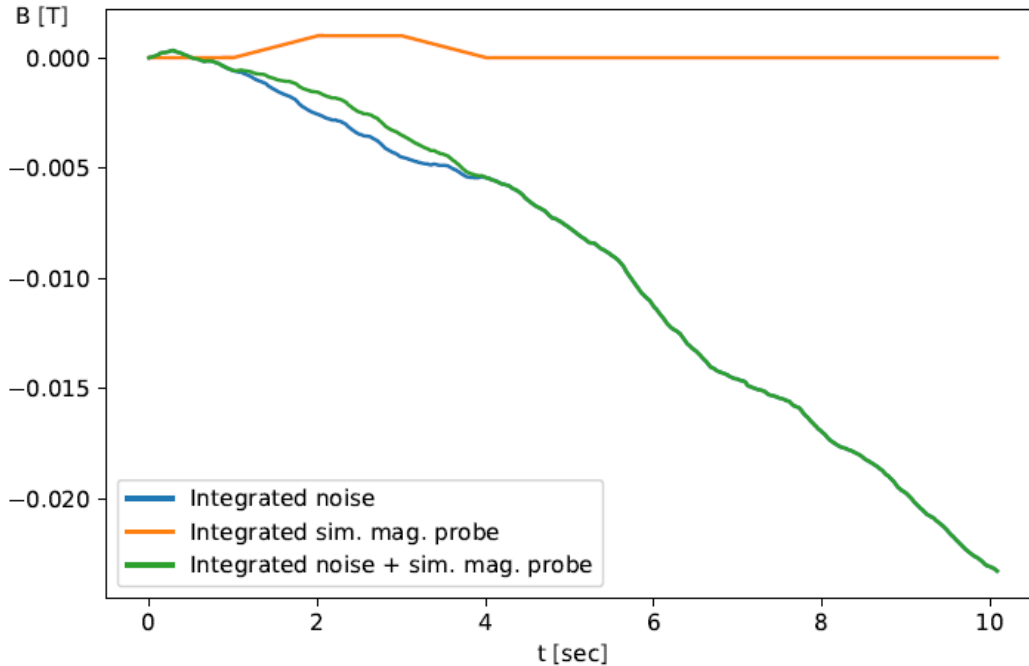
Now we check if it is really possible to measure and integrate the hypothetical signal of Figure 12-16 Minum level reference magnetic signal., by adding and comparing it with the noise measured from the existing ADC module (Figure 12-17).



**Figure 12-17 Resampled noise compared with the simulated minum level reference magnetic signal.**

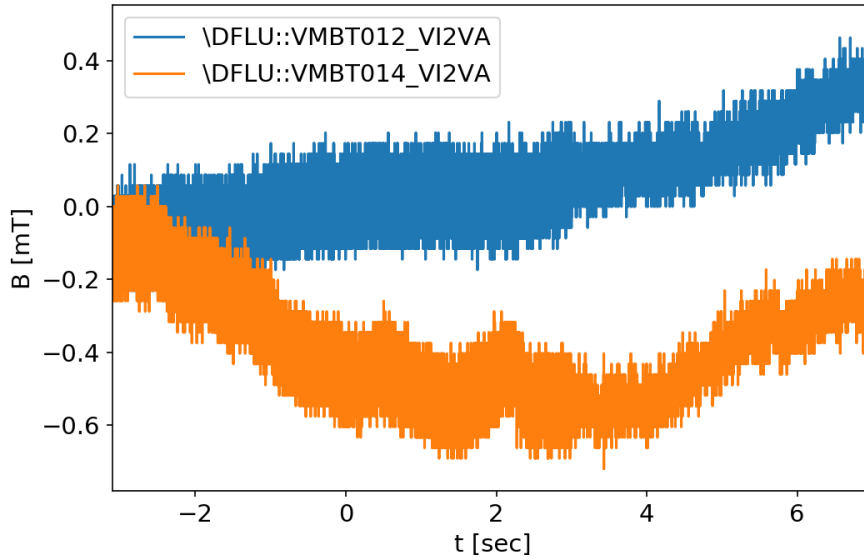
We can see that the signal level is more or less one third of the noise.

The resulting numerical integration of these signals is shown in Figure 12-18.



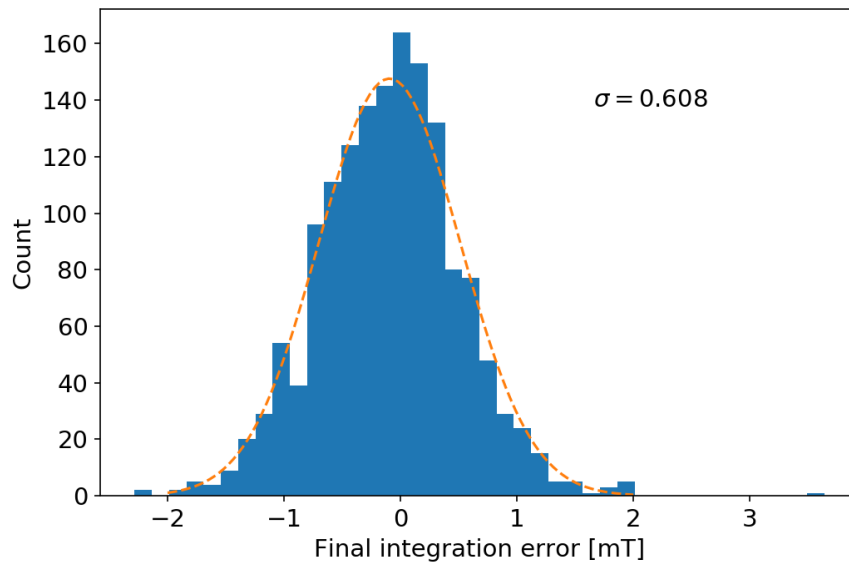
**Figure 12-18 Resampled integrated input compared with the simulated worst case.**

The orange line is the ideal integrated signal without noise. The green line indicates the integrated sum of ideal signal with the noise, which is cannot be practically distinguished by the integrated noise alone.



**Figure 12-19 Typical drift error of the RFX-mod analog signal integrator.**

By comparison the analog integration system currently in use on RFX-mod, produces an error of about 0.6 mT drift at 10 seconds (Figure 12-19 and Figure 12-20), which has to be compared with the 20mT obtained from the ADC module as is (Figure 12-18).



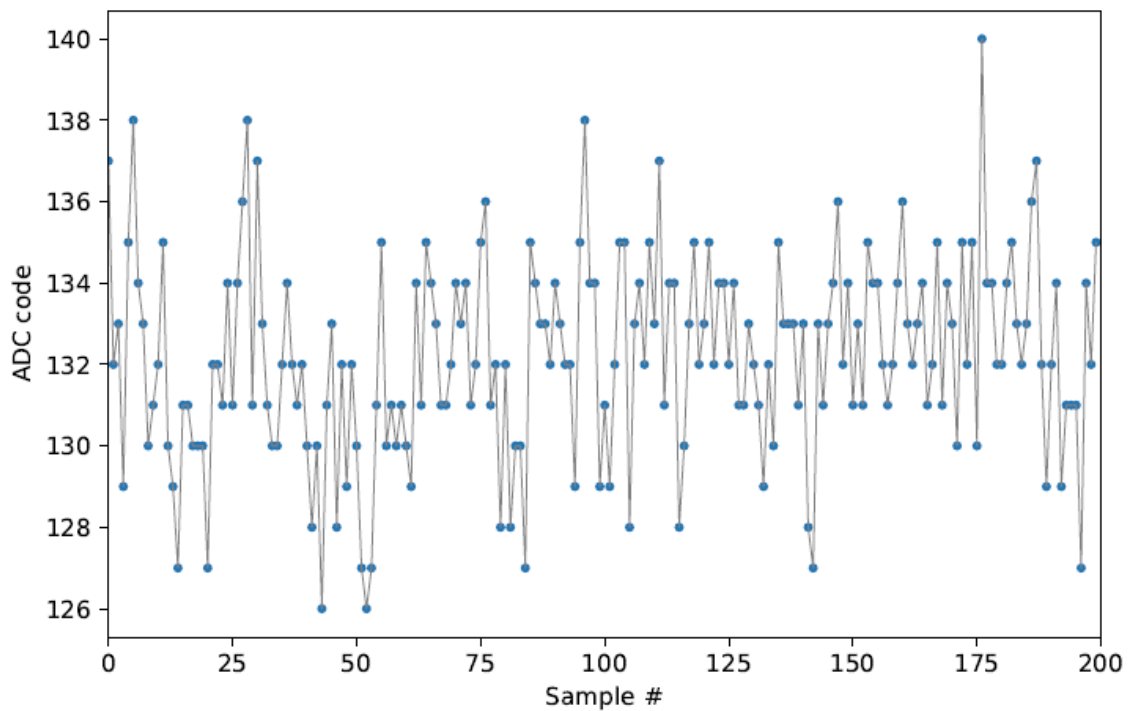
**Figure 12-20 Distribution of final integration error of the analog integrators of RFX-mod after 10 s.**

## 12. 7. Test without DC/DC converter

In order to understand the noise source of the ADC module with the reduce it, the module has been modified and other tests carried out

The first modification consisted in eliminating the DC / DC converter to remove possible low frequency noise due to the switching of the rectifier diodes present in the final section of the integrated circuit.

In order to test the only ADC integrated circuit with no spurious signals and interference due to the rest of the circuit, the DC / DC converter has been eliminated and fed directly from a well-stabilized continuous source (linear power supply).



**Figure 12-21 Noise sampling of ADC module with 50 Ohm terminated input, without DC-DC converter**

File **noise\_samp\_nodc** to compare with the previous one, note that the amplitude is halved and the frequency decreases.

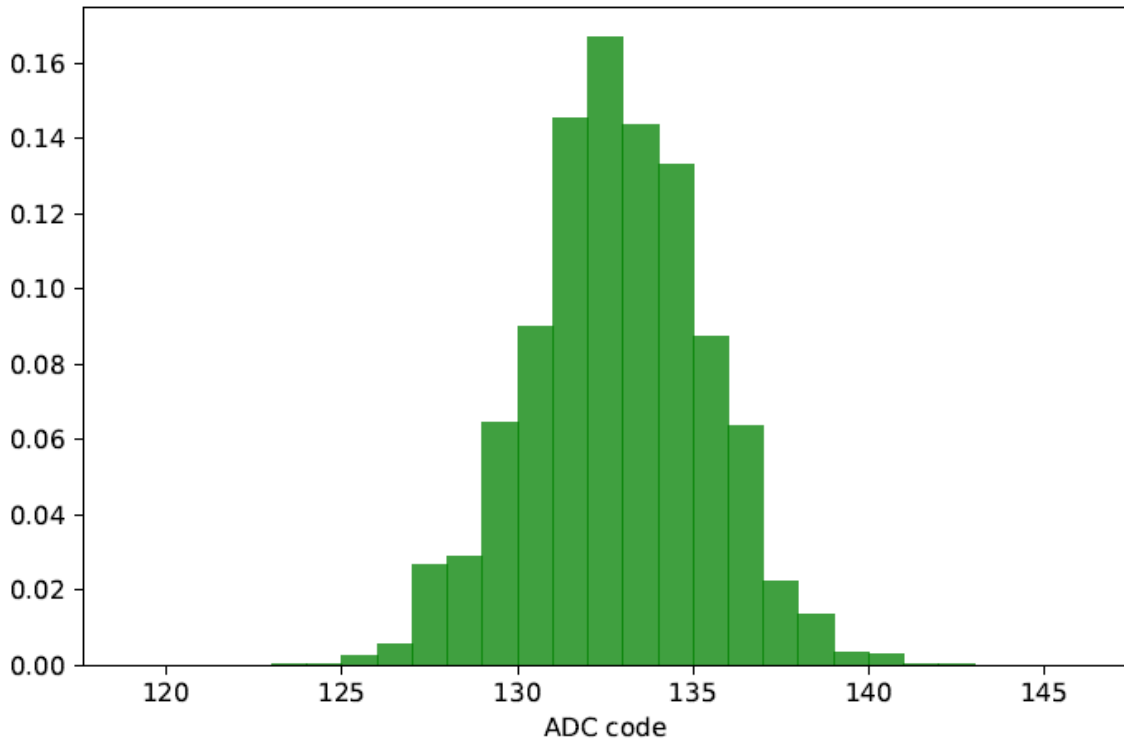
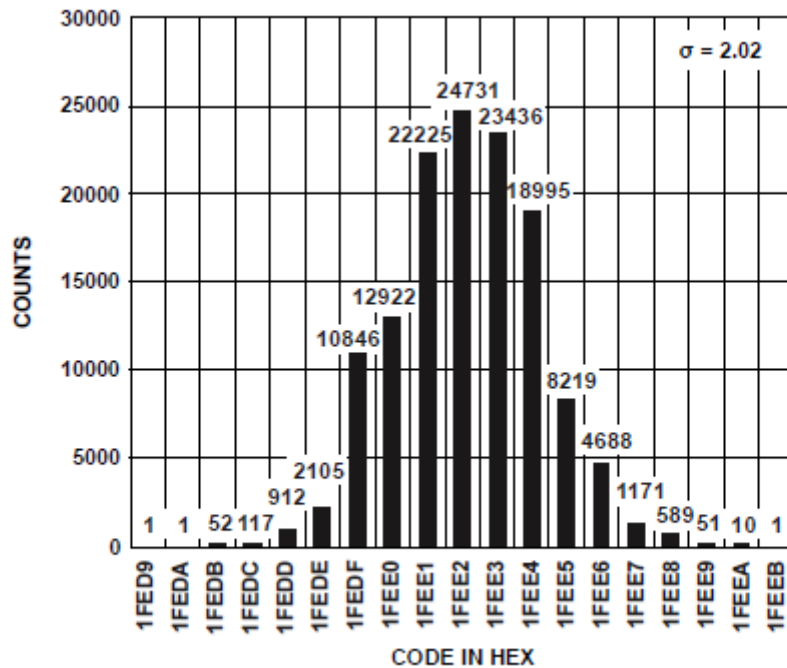


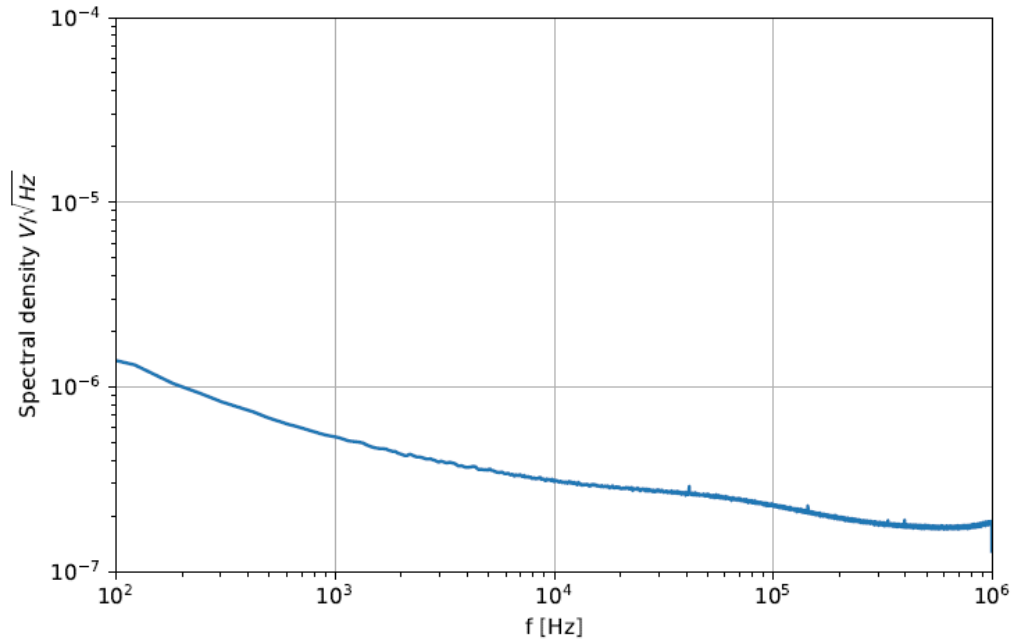
Figure 12-22 Sampled Noise histogram of ADC module with 50 Ohm terminated input without DC-DC converter



Histogram of 261,120 Conversions of a DC Input at the Code Center (Internal Reference)

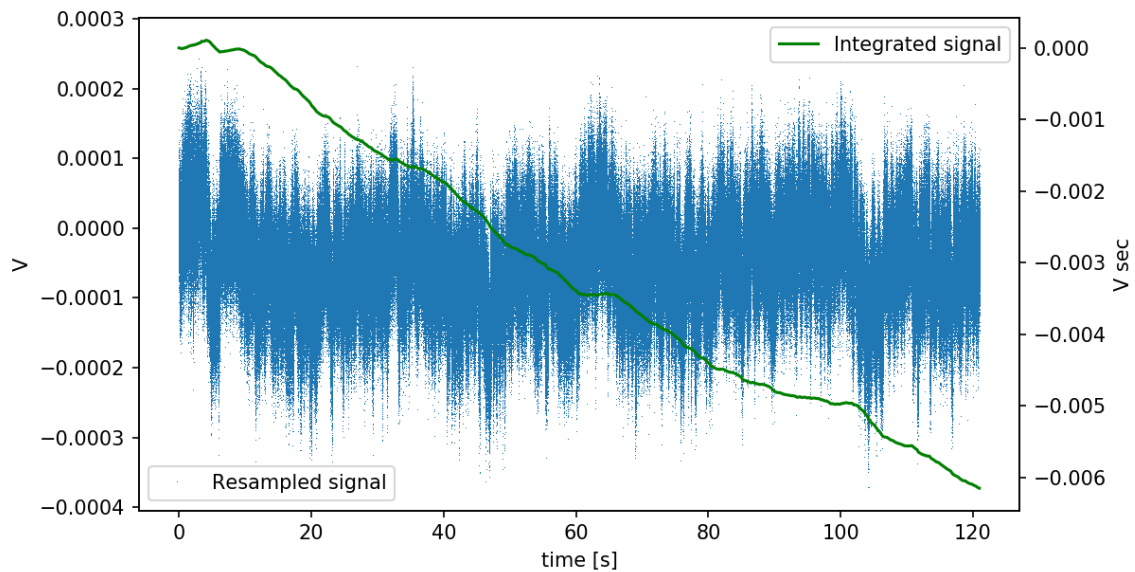
Figure 12-23 Histogram noise input from AD7641 datasheet

The histogram shows that the frequencies introduced by the DC / DC converter are removed.



**Figure 12-24 Sampled Noise spectrum of ADC module without DC-DC converter**

From the three graphs above you see clearly that the DC/DC converter noise has actually disappeared. The spikes are missing, the histogram is corrected compared to the data book's one, and in the spectrum there is no peak at 500kHz



**Figure 12-25 Resampled input direct and time integrated signal, without DC-DC converter**

After the Integration of this signal, inspite the elimination of the noise due to the DC / DC converter, no improvement is achieved on the signal integration capacity. Therefore, it is necessary to investigate, in the next paragraph, what happen at low frequency to estimate the reasons of the present noise in the bottom frequency band.

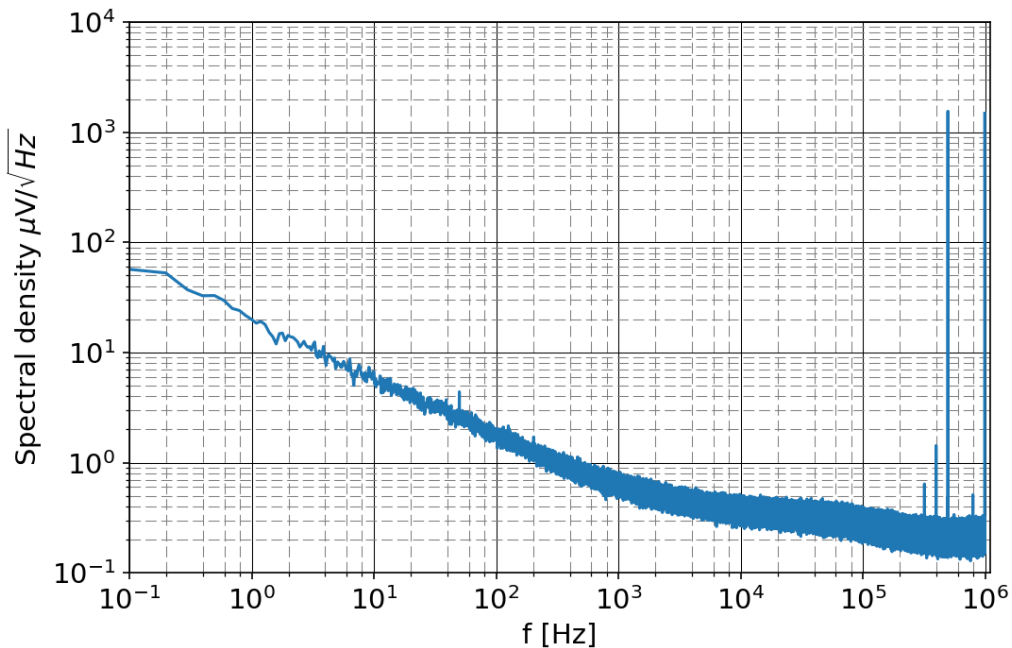
## 12. 8. Low frequency noise noise analysis

In order to improve the signal integration capability we are interested to observe the behavior of the noise of the circuit for long times which translates in the property of the frequency spectrum at low frequencies ( $< 1$  Hz) which are poorly or not documented at all, in the electrical specifications of the electronic components. So we carry out the some measurements in this range to verify the circuit characteristics.

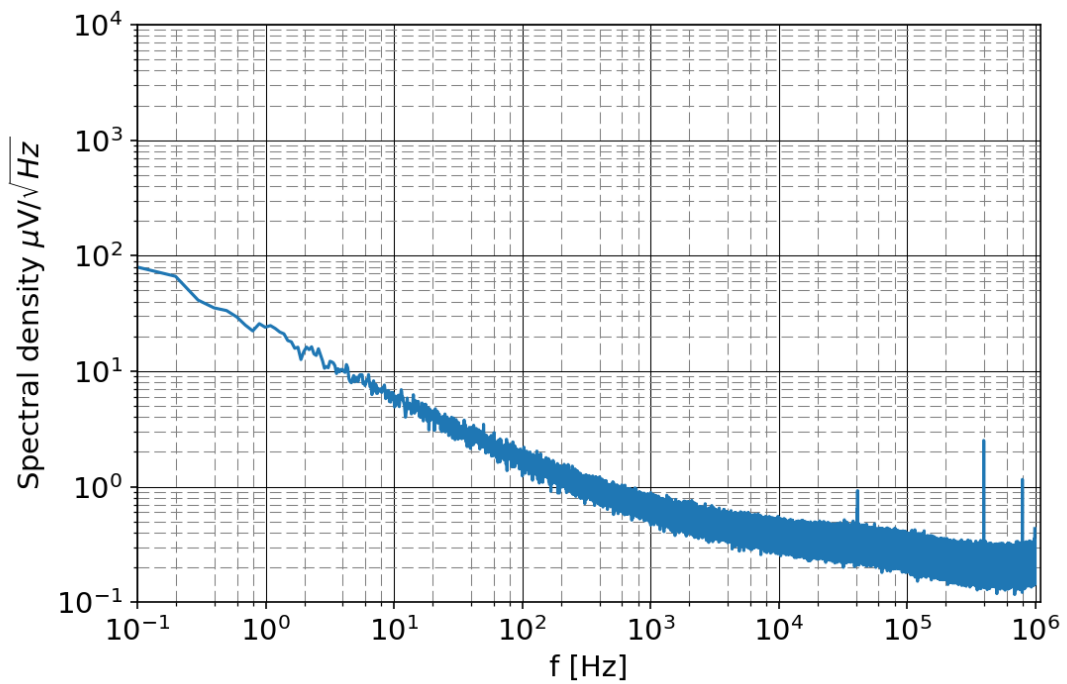
Sampling window of 40 sec has been taken to measure the low-frequency spectrum as can be see in the graphs below which span 7 decades, from 0.1 Hz to 1 MHz.

The measurments have been carried out on successive modifications made on the ADC module, aiming to identify the  $1/f$  noise sources.

The first case shown in Figure 12-26 refers to the original ADC module, which shows a the relatively high level of  $1/f$  noise. This noise causes the failure of the numerical integration as explained in the previous § 12. 6.



**Figure 12-26 Noise spectrum of the ADC module in its basic configuration, with 50Ω termination at input.**

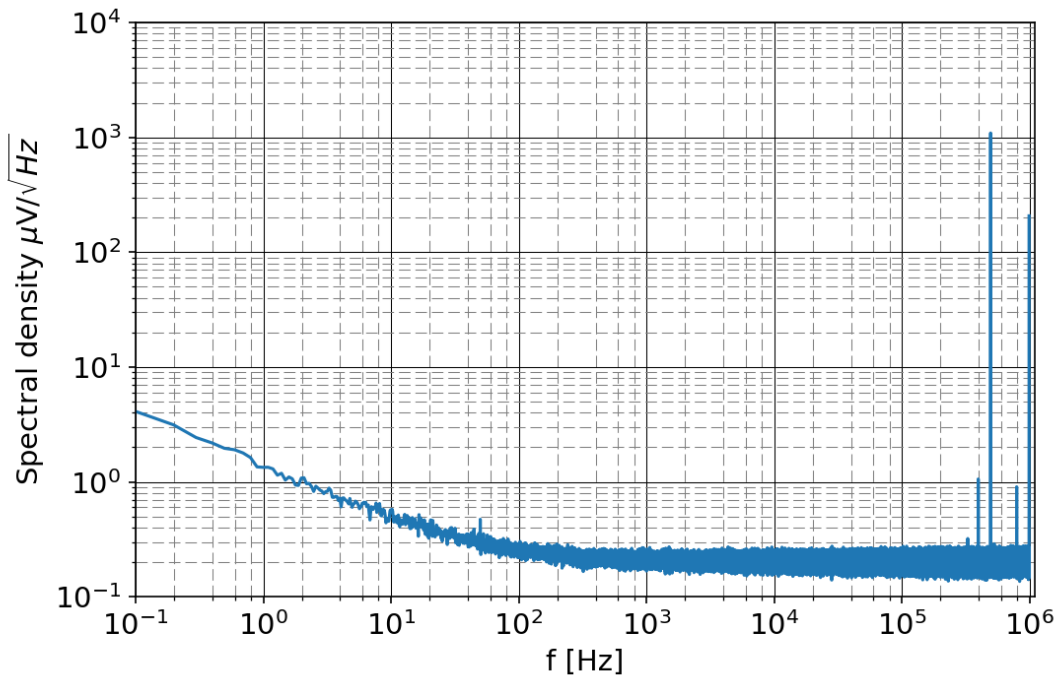


**Figure 12-27 Noise spectrum of the ADC module with linear power supply and 50Ω terminator**

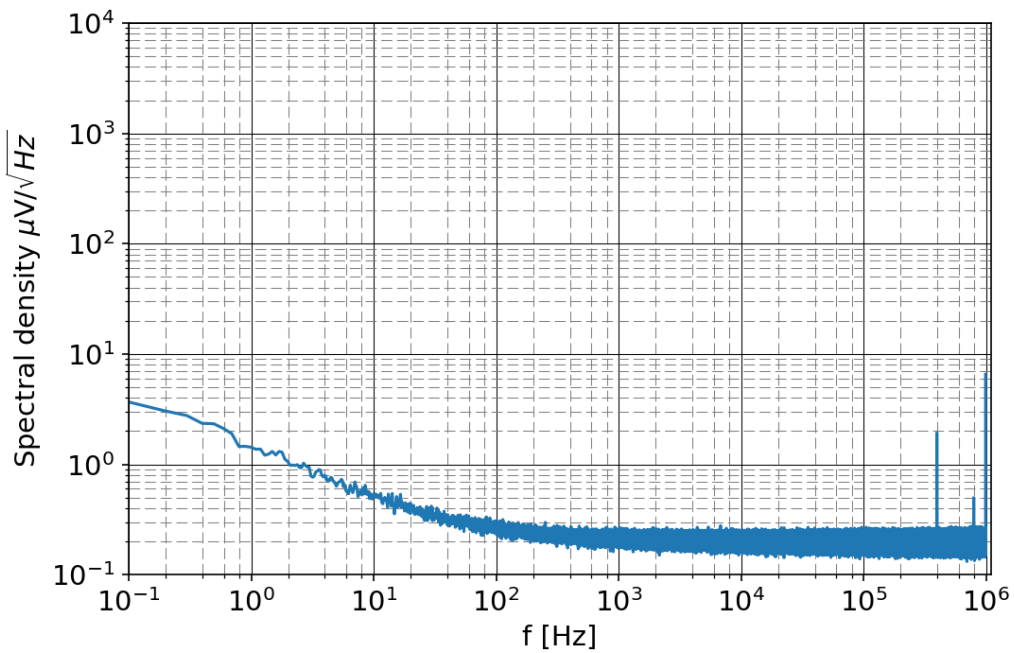
Figure 12-27 shows the noise spectrum of the module with the DC/DC converter removed and fed by a linear power supply, with 50 Ohm terminated input. While the peak frequencies at 500 kHz and 1 MHz are removed, the 1/f noise is still there. Thus the DC/DC converter does not appear to contribute to low frequency noise, so no improvement on the integration capability is expected, as seen in § 12. 7.

.

.



**Figure 12-28 Noise spectrum of the ADC module with analog input section bypassed and DC/DC converter power supply.**



**Figure 12-29 Noise spectrum of the ADC module with analog input section bypassed and linear power supply.**

Figure 12-28 and Figure 12-29 show the noise spectra after bypassing the analog input amplifier and connecting the input directly to the ADC ports; the former case with circuit fed by its DC/DC converter and the latter by the linear power supply. In both cases the  $1/f$  noise is drastically reduced.

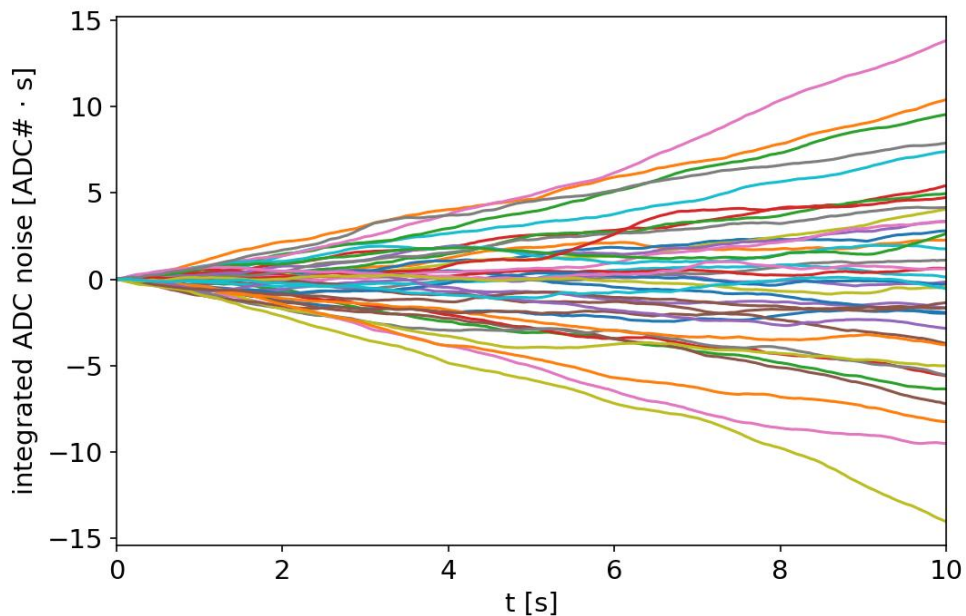


## 12.9. Noise integration and expected integrated signals on modified ADC module

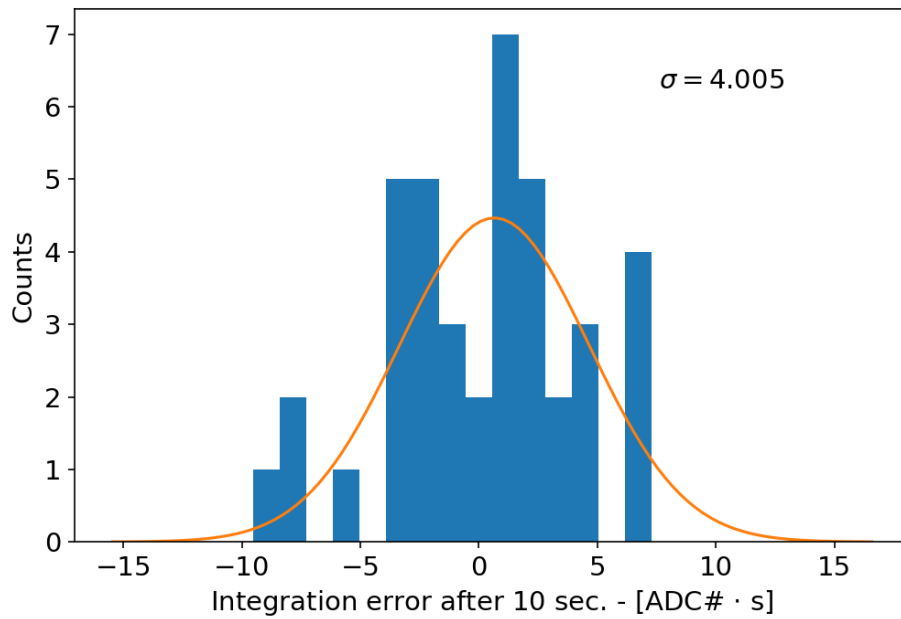
To quantify the effect of the  $1/f$  noise, the signals taken in two cases have been statistically analyzed: i) with the ADC module in its original configuration and ii) with the analog input section bypassed.

For each configuration several sample signals have been acquired to obtain a statistically significant result.

In Figure 12-30 the signal ensemble obtained by direct numerical integration of the sample signals are reported. The quality of the integration can be evaluated by estimating the typical error obtained at the end of the integration, given by the histogram of the final values and summarized by the resulting standard deviation.

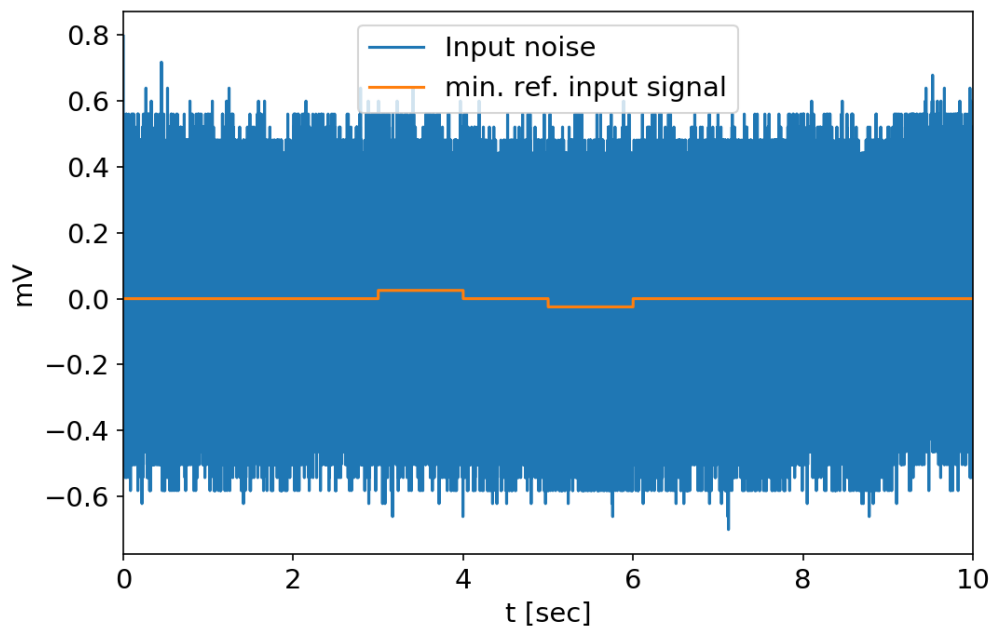


**Figure 12-30 Numerically integrated signals obtained from the ADC module in its original configuration**

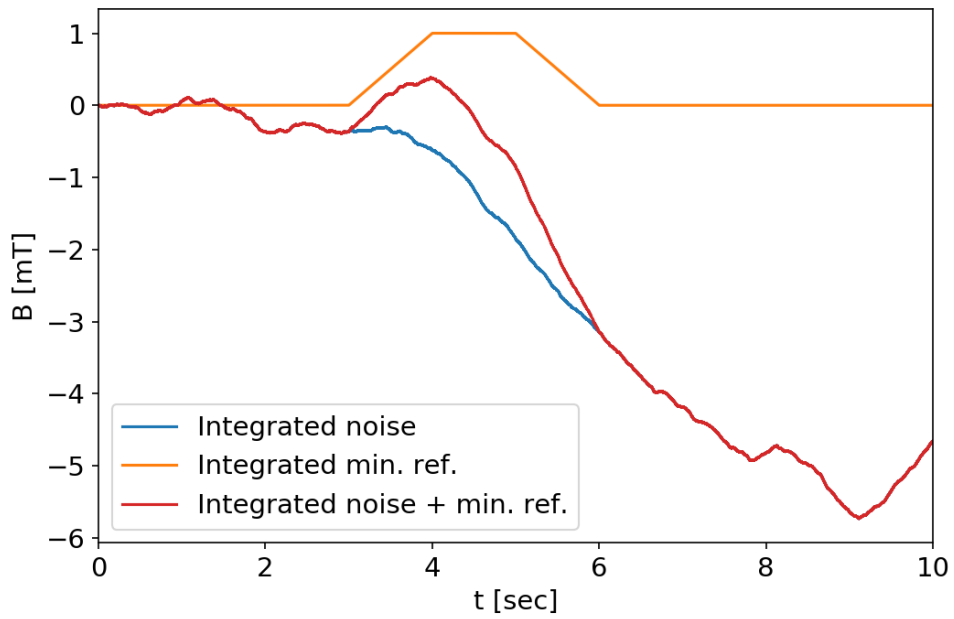


**Figure 12-31 Histogram of the final values after 10 s of numerical integration of the sample signals from original ADC module configuration**

The simulated low level input test signal is added to the one experimental signal. As in § 12. 6, the signal is hardly visible compared to the noise (Figure 12-32) and the numerical integration fails (Figure 12-33).

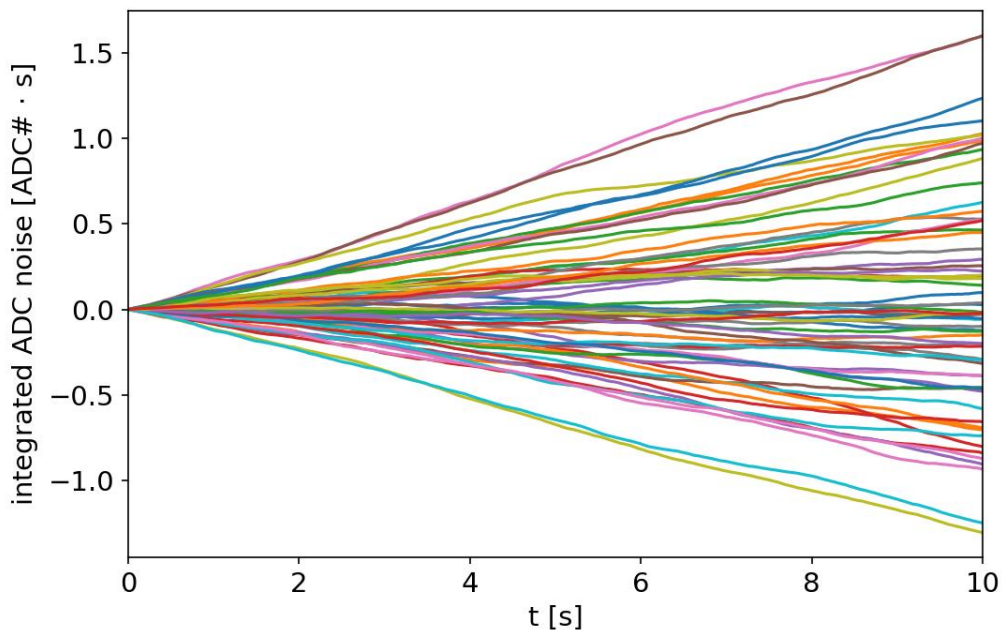


**Figure 12-32 Low level reference signal and experimental noise of original ADC module configuration**

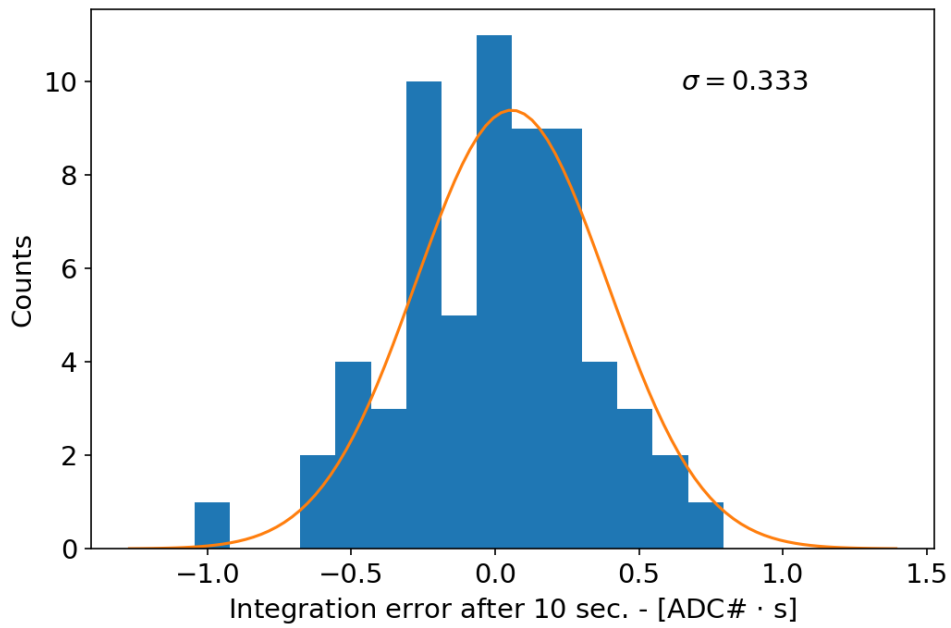


**Figure 12-33 Numerical integration of low level reference signal and experimental noise of original ADC module configuration.**

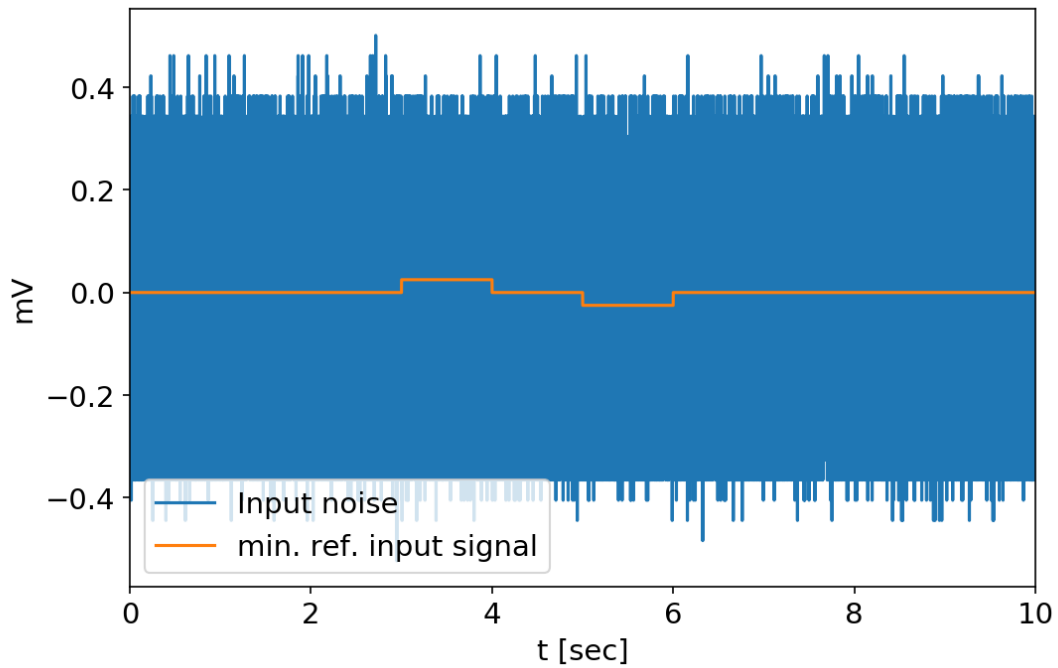
The same procedure has been applied to a set of sample signals obtained from the ADC module with the bypassed analog input section. The results are shown on the four following Figures.



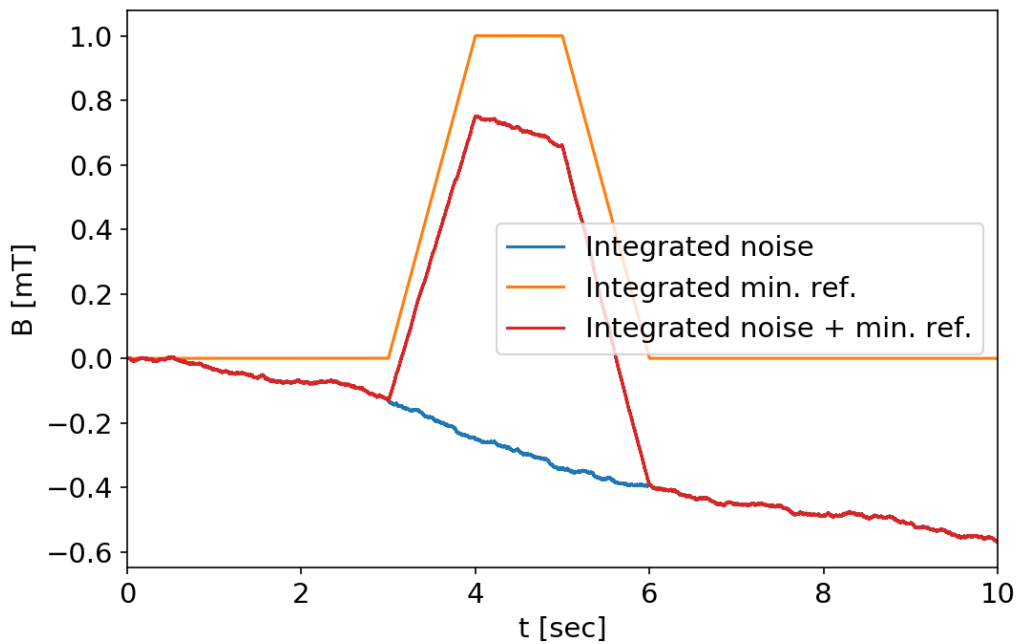
**Figure 12-34 Numerically integrated signals obtained from the ADC module in with the analog input section bypassed.**



**Figure 12-35 Histogram of the final values after 10 s of numerical integration of the sample signals from the ADC module with the analog input section bypassed.**

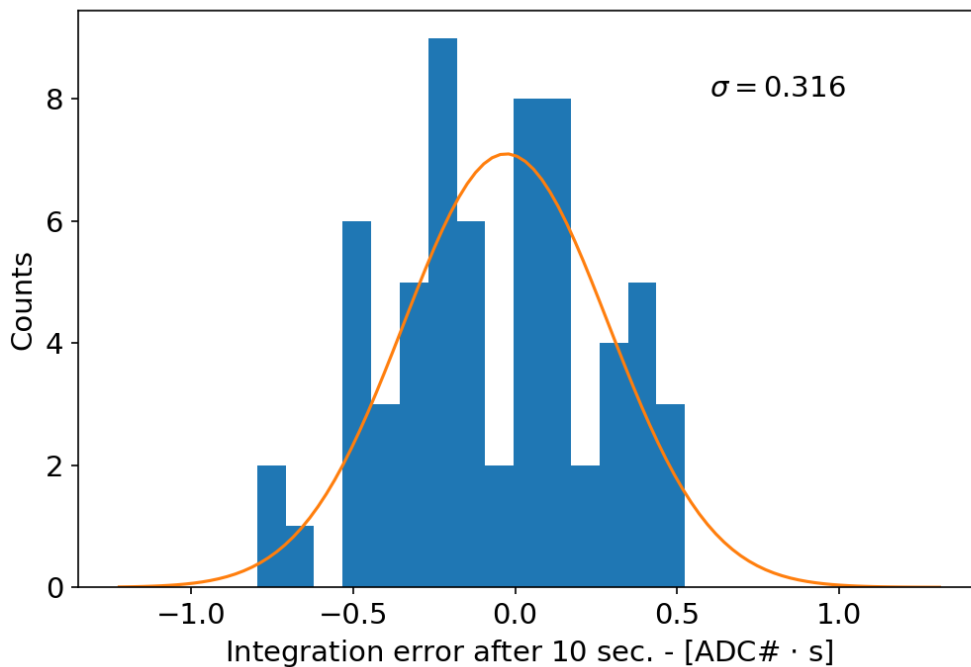


**Figure 12-36 Low level reference signal and experimental noise of the ADC module with the analog input section bypassed.**



**Figure 12-37 Numerical integration of low level reference signal and experimental noise of the ADC module with the analog input section bypassed.**

It is worth noticing that the significant reduction of the noise, allows a satisfactory recovery of the integrated signal, even if the simulated input is still low compared to the average noise (Figure 12-36)



**Figure 12-38 Histogram of the final values after 10 s of numerical integration of the sample signals from the ADC module with the analog input section bypassed and linear power supply.**

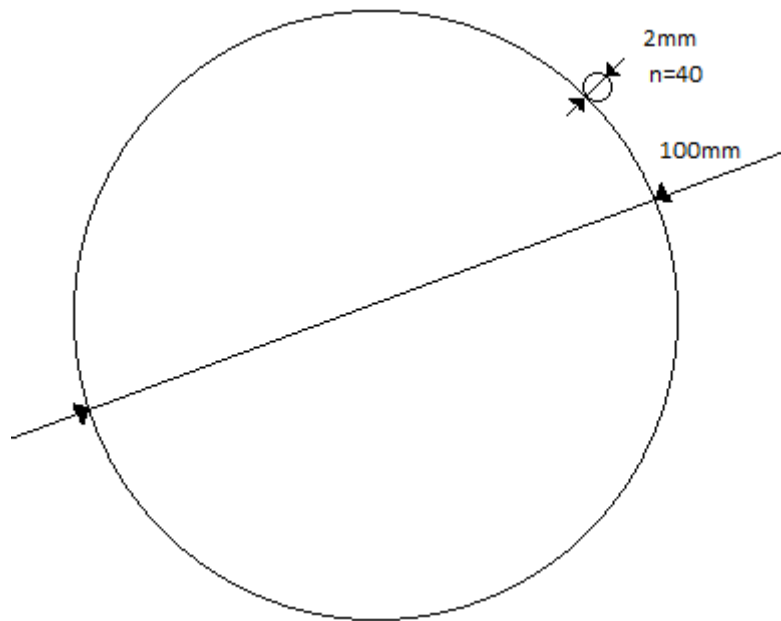
As a final result Figure 12-38 reports the histogram of final integration values for the ADC module with the bypassed analog input section *and the linear power supply*, which shows a negligible improvement.

The conclusion is that the ADC converter itself along with the DC/DC converter is capable of performing as required. This means that only the analog input section needs to be redesigned.

## 12. 10. Magnetic probe measurement test circuit

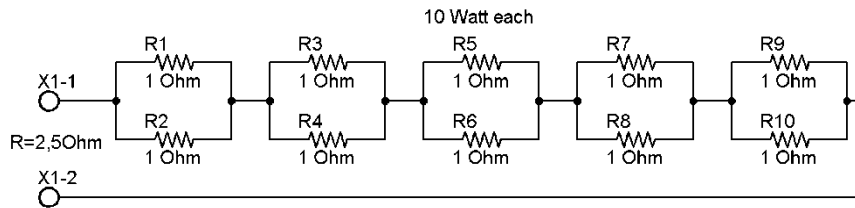
As a final test, the ADC module, with the bypassed input section has been connected to a magnetic probe, put inside a solenoid acting as a magnetic field source.

The test solenoid was made by wrapping 40 turns of copper conductor of 2mm diameter and the coil inductance is estimated to be about 0.1 mH.

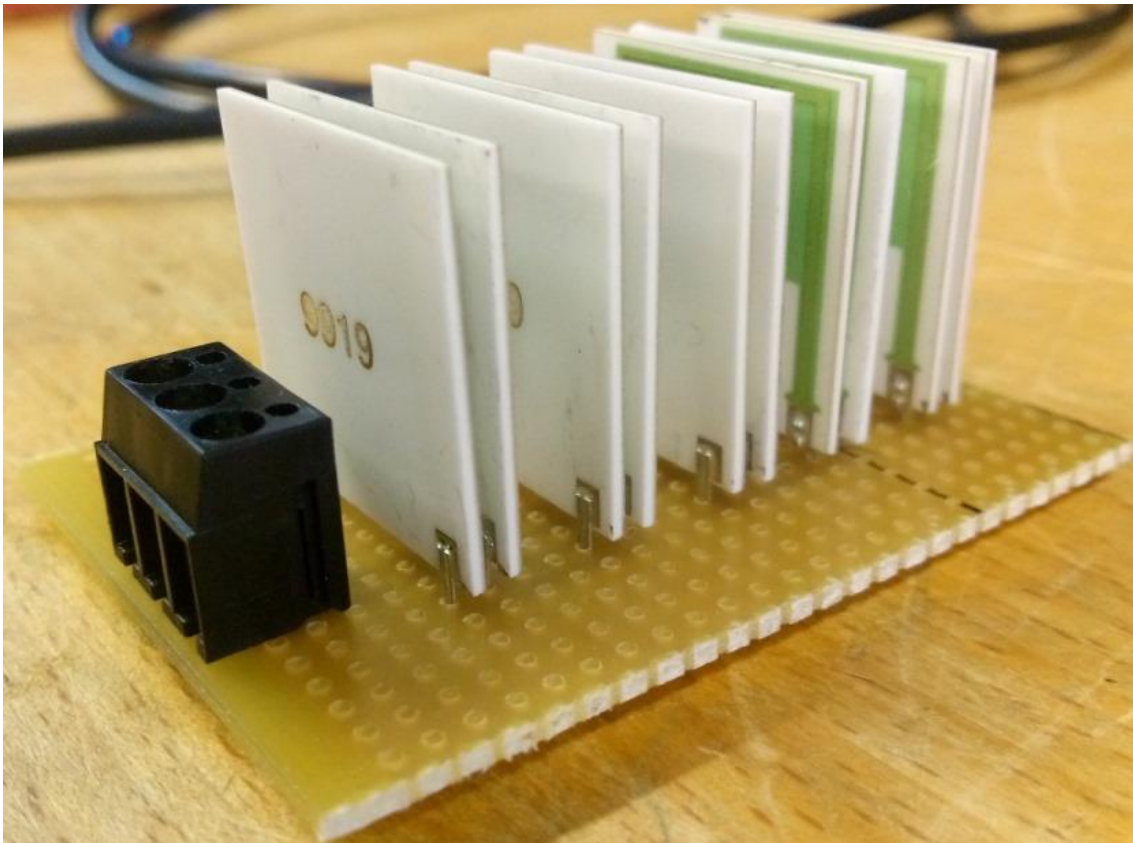


**Figure 12-39 Test coil sketch**

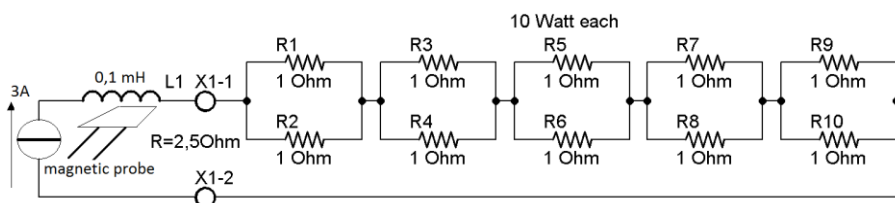
In order to obtain a correct impedance to the circuit context, a resistive load of  $2.5 \Omega$  is connected in series capable of dissipating a power of 100W, supplying in test by 4A it will dissipate 40W. The maximum current applicable to the total resistive load is 6.32A.



**Figure 12-40 Resistive load circuit**

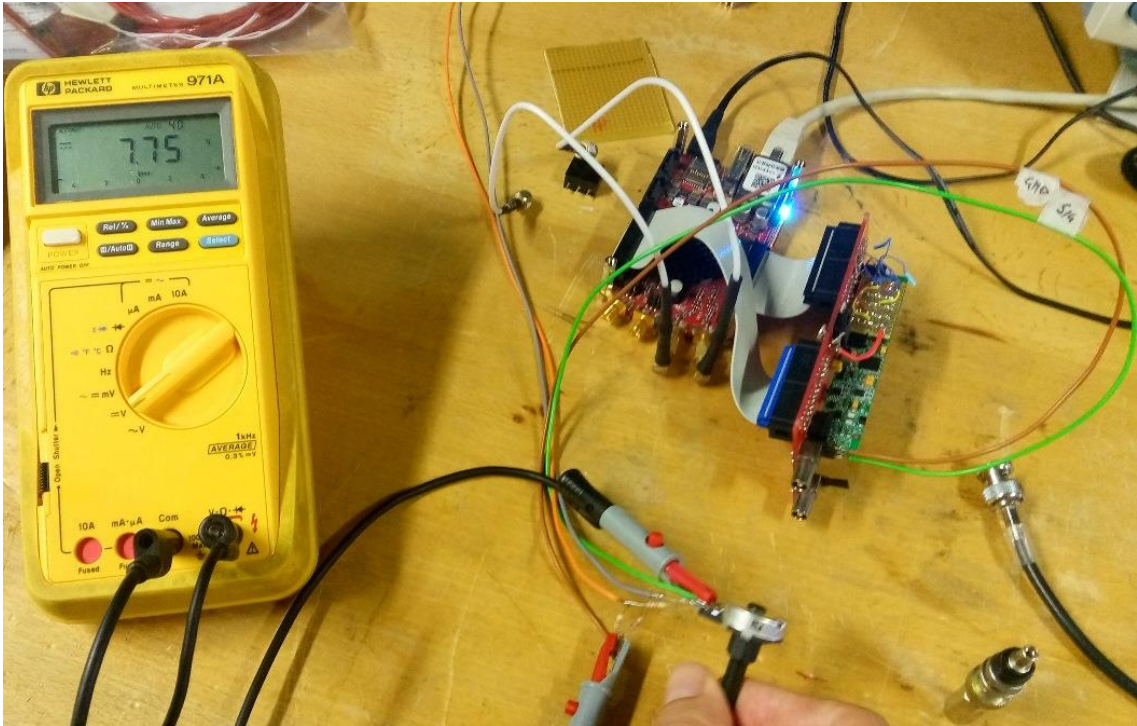


**Figure 12-41: 2.5 Ω resistive load.**

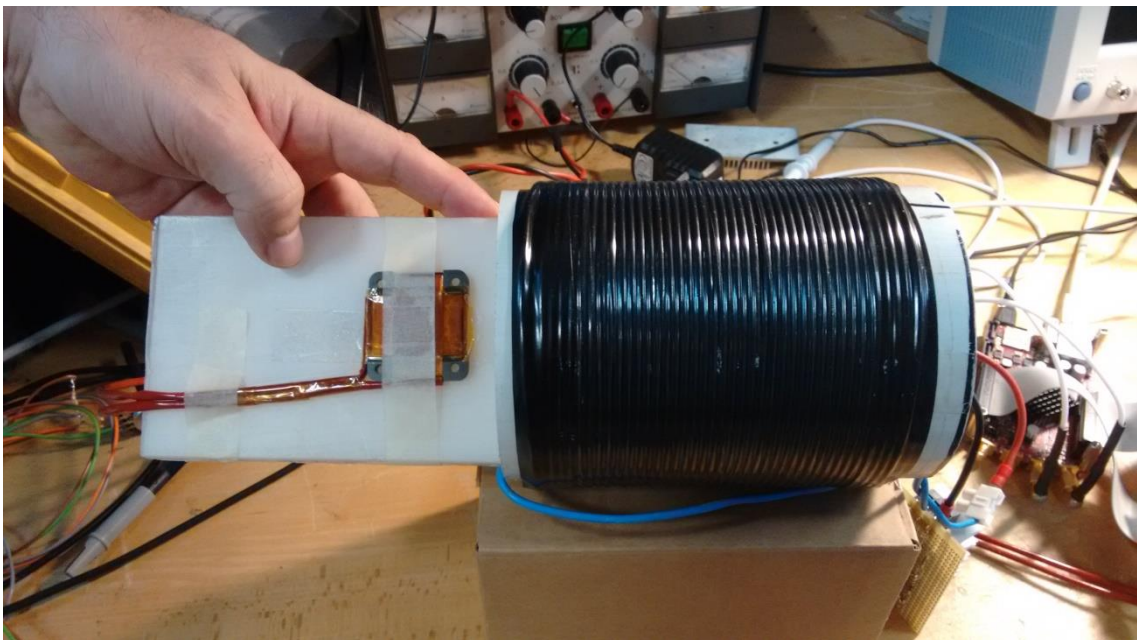


**Figure 12-42 Resistive load connect to test coil and probe**

The goal is to check if circuit noise is compatible with the acquire of the RFX probes. produced and tested by Laboratorio Elettrofisico s.r.l. Via G. Ferrari, 14-20014 Nerviano (MI) Italy in the specific case of sensor used on the toroidal device RFX in Padua. The probe has a typical area of  $0,024\text{m}^2$



**Figure 12-43 Voltage signal by a potentiometer**



**Figure 12-44 Test coil and magnetic probe**



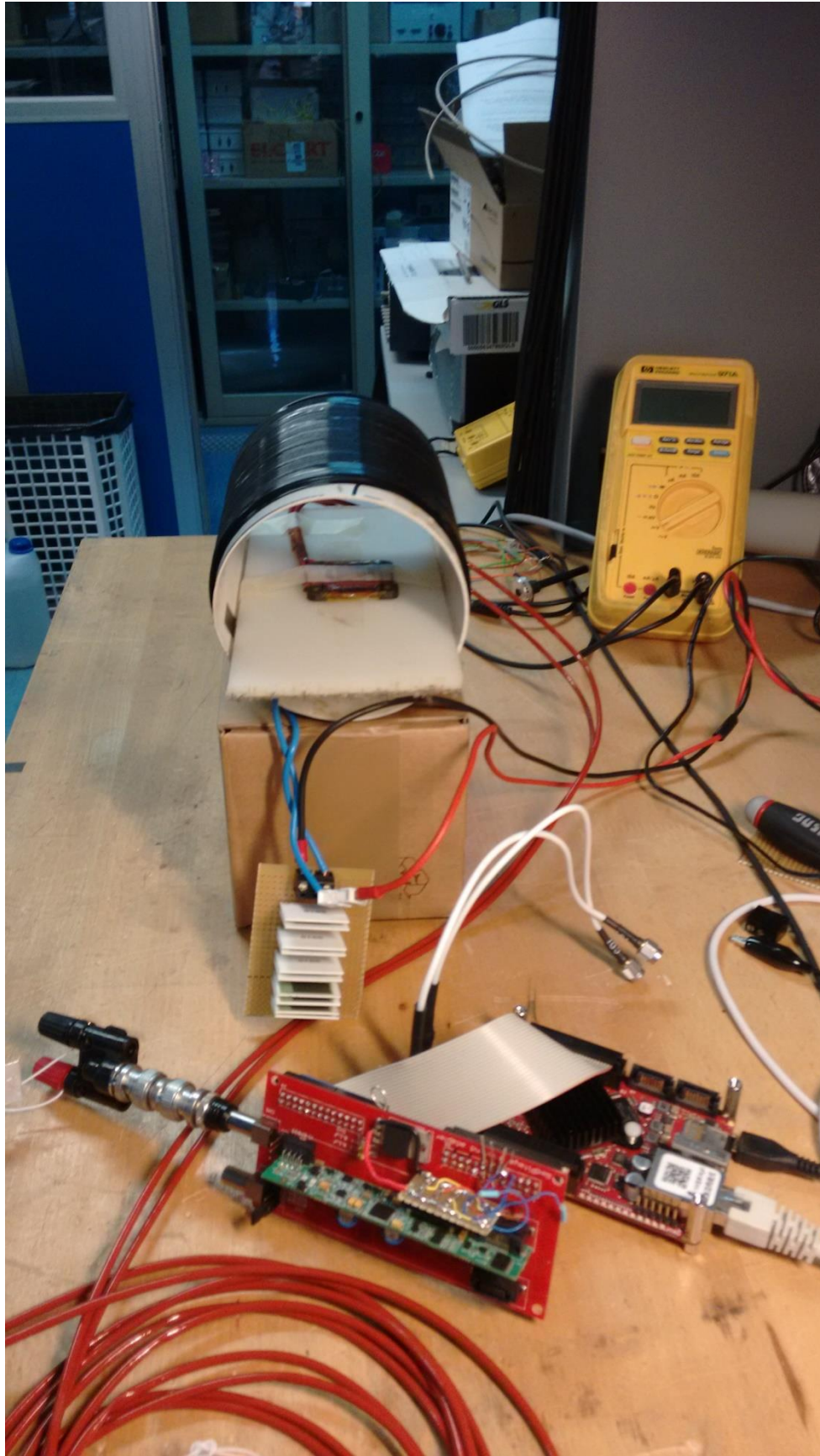
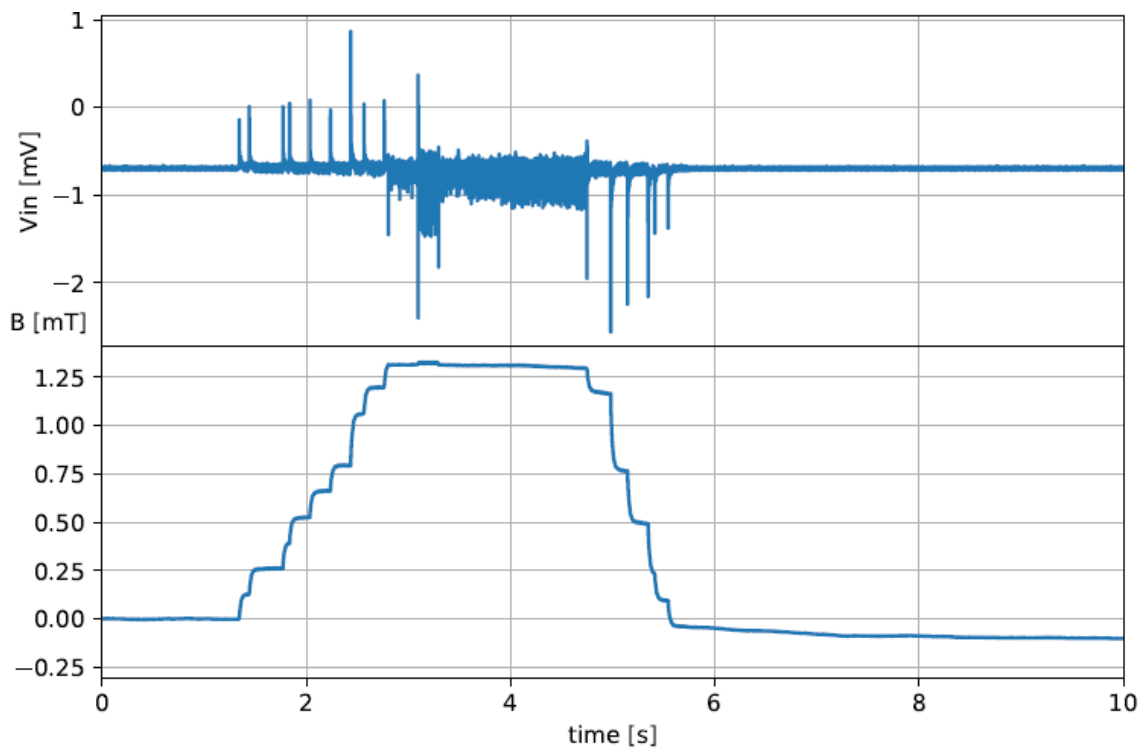


Figure 12-45 Acquisition system connected to coil and probe



**Figure 12-46 Direct acquired signal from the probe directly connected to the ADC input and its time integrated signal**

In this test obtained the signal are obtained bypassing the feedback operational amplifier and the reference at the ADC input. The signal is provided by the magnetic probe. The probe is inserted into the coil connected to a power supply manually regulated.

Once numerically integrated the signal of the magnetic field can be recovered.

# Chapter 13.

## Conclusion

The RFX-mod experiment located in Padova, Italy is now being further upgraded to RFX-mod2.

One of the major step planned consist in a substantial improvement of the magnetic measurements.

The new magnetic pick-up coil sensors will be moved inside the vacuum vessel, so their usable signal bandwidth is expected to significantly increase and their number increased aiming at substantial better spatial resolution.

These desirable characteristics are reflected in tough requirements for the acquisition system.

The extension of the existing analog integration system [10] would be costly, require several rack unit space and severely limit the useful bandwidth, thus a more compact and cost effective solutions are being investigated.

The ATCA MIMO ISOL system came out to be too noisy to be applied directly on RFX-mod but from the tests it can be deduced that in the circuit context the MIMO ISOL ADC modules the source of noise was detected in specific places of the analog front end, which could be replaced with components more suitable for noise than the actual ADC.

The choice of the ADC, in the analog modules under exams, is essentially correct as the integration of intrinsically generated noise from the ADC chip is tolerable and falls within the speed and noise parameters.

However, in this case as new generation components are available, it is likely to replace the ADC with a 20-bit model, which would further increase the dynamic range of the circuit.

The ATCA MIMO ISOL modules has been found insufficient for the following two causes.

- 1) The galvanically separated power supply with the used DC/DC converter introduces a noise component at the 500kHz frequency, visible in the noise spectral density diagram, of very significant amplitude.
- 2) However the use of DC/DC converter dose not appears to create problemes, provided it is synchronized with the acquisition system.
- 3) More significant than the previous one for the inability to use fro RFX-mod2 is the presence of noise in the lower part of the spectrum, so at low frequency, which produces a deriva on integrated signal, with origin in the analog frontend consisting essentially of the configuration of the operational amplifiers conceptually placed to introduce a signal attenuation and buffering. As shown in the trials, the problem is weakened by entering directly into ADC input.

It follows that the possible use of the ATCA MIMO ISOL modules in the experiments with the purpose of integrated magnetic probe measurements may result in distorted and therefore inadequate results.

So the new frontend design is necessary using a 20 bit ADC (e.g. Linear Technology LTC2378-20), which has a better noise characteristic and 2 bits more which can accommodate the higher expected signal level  
<<http://www.linear.com/product/LTC2378-20>> .

With these device the direct digital integration appears feasible, even if it require state of the art components However it requires a very careful design, using high quality precision component with low 1/f and low drift components. Even use the best available commercial components, care must be taken when choosing the dynamic range and the bandwidth attenuation.

# A. Appendix: Source codes

Code used for serialized the data acquire by ZYNQ FPGA as output provided by the ACTA-MIMI-ISOL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity AD7641_serial_slave is
  Generic (
    SERIAL_DATA_LEN : integer := 18
  );
  Port (
    store_tics      : in  integer;
    cnvst_hi_tics   : in  integer;
    cnvst_lo_tics   : in  integer;

    data_out        : out std_logic_vector (31 downto 0);
    error_out       : out std_logic;
    store_out       : out std_logic;

    clk             : in  std_logic;
    reset           : in  std_logic;
    SDAT_in         : in  std_logic;
    SCLK_in         : in  std_logic;
    CNVST_in        : in  std_logic;
    CNVST_out       : out std_logic;
    RST_P           : out std_logic;
    RST_N           : out std_logic
  );
end AD7641_serial_slave;

architecture Behavioral of AD7641_serial_slave is

  type state_type is (st_read_wait, st_read, st_idle);
  signal state, next_state : state_type := st_idle;

  signal cnvst_reset      : std_logic := '1';
  signal read_reset      : std_logic := '1';
  signal cnvst_gen       : std_logic := '0';
  signal cnvst           : std_logic := '0';
  signal rst_pos         : std_logic := '0';
  signal rst_neg         : std_logic := '0';

  constant HALF_DATA_LEN : integer := SERIAL_DATA_LEN/2;
  signal data_0, data_1   : std_logic_vector(HALF_DATA_LEN-1 downto 0) := (others => '0');
  signal data_count_0    : integer := 0;
  signal data_count_1    : integer := 0;
  signal data            : std_logic_vector(SERIAL_DATA_LEN-1 downto 0) := (others => '0');
  signal store           : std_logic := '0';
  signal SCLK_buf        : std_logic := '0';

begin

  cnvst <= CNVST_in or cnvst_gen;
  CNVST_out <= cnvst;
  SCLK_buf <= SCLK_in;
  error_out <= SCLK_buf;
  store_out <= store;
  --data_out(31 downto SERIAL_DATA_LEN) <= (others => '0');
  -- CNVST generator --
```

```

proc_gen_CNVST : process (clk, cnvst_reset)
  variable count : integer := 0;
  variable hit   : integer := 0;
  variable lot   : integer := 0;
begin
  hit := cnvst_hi_tics;
  lot := cnvst_lo_tics;
  if rising_edge(clk) then
    if cnvst_reset = '1' then
      cnvst_gen <= '0';
      count := 0;
    elsif (count < hit ) then
      cnvst_gen <= '1';
    elsif (count < lot) then
      cnvst_gen <= '0';
    else
      count := 0;
      cnvst_gen <= '0';
    end if;
    count := (count + 1);
  end if;
end process;

-- STORE --
proc_store : process (read_reset, clk)
  variable count : integer := 0;
  variable thc   : integer := 0;
begin
  thc := store_tics;
  if read_reset = '1' then
    count := 0;
    data_out(SERIAL_DATA_LEN-1 downto 0) <= (others => '0');
  elsif rising_edge(clk) then
    if cnvst = '1' then
      count := 0;
      store <= '0';
    elsif (count < thc ) then
      store <= '0';
      count := count+1;
    else
      store <= '1';
    end if;
  -- store
  if (count = thc-1) then
    data_out(SERIAL_DATA_LEN-1 downto 0) <= data;
    data_out(31 downto SERIAL_DATA_LEN) <= (others => data(SERIAL_DATA_LEN-1));
  end if;
end if; -- clk
end process;
-- main --
proc_main : process (clk, reset, cnvst)
begin
  if (reset = '1') or (cnvst_hi_tics = 0) then
    read_reset <= '1';
  else
    read_reset <= '0';
  end if;
end process;
--Reset process
proc_reset : process (clk, read_reset)
  variable pulse : integer := 0;
begin
  if read_reset = '1' then
    rst_n <= '0'; --ADC off
    rst_p <= '1'; --DC/DC off and FF disable (Q==0)
    cnvst_reset <= '1';
    pulse := 0;
  elsif rising_edge(clk) then
    rst_n <= '1'; --ADC on
    rst_p <= '0'; --DC/DC on and FF enable
    if (pulse < 10000 ) then

```

```

        pulse := pulse + 1;
        cnvst_reset <= '1';
    else
        cnvst_reset <= '0';
    end if;
end if;
end process;

-- READ process 0 --
proc_read_0 : process (SCLK_in, read_reset, store)
begin
    if read_reset = '1' then
        data_0 <= (others => '0');
    elsif store = '1' then
        data_count_0 <= 0;
    elsif rising_edge(SCLK_in) then
        data_0 <= data_0(HALF_DATA_LEN-2 downto 0) & SDAT_in;
        data_count_0 <= data_count_0 + 1;
    end if;
end process;

-- READ process 1 --
proc_read_1 : process (SCLK_in, read_reset, store)
begin
    if read_reset = '1' then
        data_1 <= (others => '0');
    elsif store = '1' then
        data_count_1 <= 0;
    elsif falling_edge(SCLK_in) then
        data_1 <= data_1(HALF_DATA_LEN-2 downto 0) & SDAT_in;
        data_count_1 <= data_count_1 + 1;
    end if;
end process;

gen_data : for i in 0 to HALF_DATA_LEN-1 generate
    data(i*2+1 downto i*2) <= data_0(i) & data_1(i);
end generate gen_data;

end Behavioral;

```

The IP module design is also hierarchically inserted into a AXI4 interface adapter wrapper that provides the IP with two busses: a AXI4\_LITE memory mapped buss to set configuration parameters within the adc module, and a AXI-STREAM bus that outputs the actual acquired data. Therefore a double connection is required both from the logic and the software driver points of view. This twofold bus connection is declared within the wrapper entity code presented in the following:

```

entity axi_SCC52460 is
    generic (
        SERIAL_DATA_LEN      : integer := 18;
        C_S00_AXI_DATA_WIDTH : integer := 32;
        C_S00_AXI_ADDR_WIDTH : integer := 4;
        C_M00_AXS_DATA_WIDTH : integer := 32
    );
    port (
        reset      : in  std_logic;
        SDAT_in    : in  std_logic;
        SCLK_in    : in  std_logic;
        CNVST_out  : out std_logic;
        error_state : out std_logic;
        RST_P      : out std_logic;
        RST_N      : out std_logic;
        -- Ports of Axi Slave Bus Interface S00_AXI
    );
end entity axi_SCC52460;

```

```

s00_axi_aclk      : in std_logic;
s00_axi_aresetn  : in std_logic;
s00_axi_awaddr   : in std_logic_vector(C_S00_AXI_ADDR_WIDTH-1 downto 0);
s00_axi_awprot   : in std_logic_vector(2 downto 0);
s00_axi_awvalid  : in std_logic;
s00_axi_awready  : out std_logic;
s00_axi_wdata    : in std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);
s00_axi_wstrb    : in std_logic_vector((C_S00_AXI_DATA_WIDTH/8)-1 downto 0);
s00_axi_wvalid   : in std_logic;
s00_axi_wready   : out std_logic;
s00_axi_bresp    : out std_logic_vector(1 downto 0);
s00_axi_bvalid   : out std_logic;
s00_axi_bready   : in std_logic;
s00_axi_araddr   : in std_logic_vector(C_S00_AXI_ADDR_WIDTH-1 downto 0);
s00_axi_arprot   : in std_logic_vector(2 downto 0);
s00_axi_arvalid  : in std_logic;
s00_axi_arready  : out std_logic;
s00_axi_rdata    : out std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);
s00_axi_rresp    : out std_logic_vector(1 downto 0);
s00_axi_rvalid   : out std_logic;
s00_axi_rready   : in std_logic;
-- Ports of Axi Stream master interface
m00_axis_aclk    : in std_logic;
m00_axis_aresetn : in std_logic;
m00_axis_tvalid  : out std_logic;
m00_axis_tdata   : out std_logic_vector(C_M00_AXS_DATA_WIDTH-1 downto 0)
);
end axi_SCC52460;

```

## HDL FIFO implementation

### HEADER: Scc52460\_2.h

```

#ifndef SCC52460_2_0_H
#define SCC52460_2_0_H

#include <linux/types.h>
#include <asm/ioctl.h>

#ifdef __cplusplus
extern "C" {
#endif

#define DEVICE_NAME "scc52460_2" /* Dev name as it appears in /proc/devices */
#define MODULE_NAME "scc52460_2"

#define RFX_SCC52460_IOCTL_BASE 'W'
#define RFX_SCC52460_RESOFFSET _IO(RFX_SCC52460_IOCTL_BASE, 0)
#define RFX_SCC52460_RESET _IO(RFX_SCC52460_IOCTL_BASE, 1)
#define RFX_SCC52460_CLEAR _IO(RFX_SCC52460_IOCTL_BASE, 2)
#define RFX_SCC52460_GETSR _IO(RFX_SCC52460_IOCTL_BASE, 3)

enum AxiStreamFifo_Register {
    ISR = 0x00, /*<< Interrupt Status Register (ISR)
    IER = 0x04, /*<< Interrupt Enable Register (IER)
    TDFR = 0x08, /*<< Transmit Data FIFO Reset (TDFR)
    TDFV = 0x0c, /*<< Transmit Data FIFO Vacancy (TDFV)
    TDFD = 0x10, /*<< Transmit Data FIFO 32-bit Wide Data Write Port
    TDFD4 = 0x1000, /*<< Transmit Data FIFO for AXI4 Data Write Port
    TLR = 0x14, /*<< Transmit Length Register (TLR)
    RDFR = 0x18, /*<< Receive Data FIFO reset (RDFR)
    RDFO = 0x1c, /*<< Receive Data FIFO Occupancy (RDFO)
    RDFD = 0x20, /*<< Receive Data FIFO 32-bit Wide Data Read Port (RDFD)
    RDFD4 = 0x1000, /*<< Receive Data FIFO for AXI4 Data Read Port (RDFD)
    RLR = 0x24, /*<< Receive Length Register (RLR)
    SRR = 0x28, /*<< AXI4-Stream Reset (SRR)

```



```

TDR = 0x2c, ///< Transmit Destination Register (TDR)
RDR = 0x30, ///< Receive Destination Register (RDR)
///< not supported yet .. ///<
TID = 0x34, ///< Transmit ID Register
TUSER = 0x38, ///< Transmit USER Register
RID = 0x3c, ///< Receive ID Register
RUSER = 0x40 ///< Receive USER Register
};

enum AxiStreamFifo_ISRenum {
ISR_RFPE = 1 << 19, ///< Receive FIFO Programmable Empty
ISR_RFPE = 1 << 20, ///< Receive FIFO Programmable Full
ISR_TFPE = 1 << 21, ///< Transmit FIFO Programmable Empty
ISR_TFPE = 1 << 22, ///< Transmit FIFO Programmable Full
ISR_RRC = 1 << 23, ///< Receive Reset Complete
ISR_TRC = 1 << 24, ///< Transmit Reset Complete
ISR_TSE = 1 << 25, ///< Transmit Size Error
ISR_RC = 1 << 26, ///< Receive Complete
ISR_TC = 1 << 27, ///< Transmit Complete
ISR_TPOE = 1 << 28, ///< Transmit Packet Overrun Error
ISR_RPUE = 1 << 29, ///< Receive Packet Underrun Error
ISR_RPORE = 1 << 30, ///< Receive Packet Overrun Read Error
ISR_RPURE = 1 << 31, ///< Receive Packet Underrun Read Error
};

#pragma pack(1)
struct rfx_scc52460_fifo {
int isr; ///< interrupt status register
int esr; ///< interrupt status register
int pad[0x40];
};

struct rfx_scc52460 {
struct rfx_scc52460 *next,*prev;
const char *name;
struct rfx_scc52460_fifo fifo;
int fd;
int prets;
int posts;
};

#ifdef __KERNEL__
// api functions here //

#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/types.h>

struct rfx_scc52460_fifo *scc52460_get_device(const char *dev_file ) {
static struct rfx_scc52460_fifo *dev = NULL;
int fd;
if(!dev) {
if(dev_file) fd = open(dev_file, O_RDWR | O_SYNC);
else fd = open("/dev/"DEVICE_NAME, O_RDWR | O_SYNC);
if(fd < 0) {
printf(" ERROR: failed to open device file\n");
return NULL;
}
dev = mmap(NULL, sizeof(struct rfx_scc52460_fifo), PROT_READ | PROT_WRITE, MAP_SHARED,fd,0);
}

if(!dev) {
printf(" ERROR: failed to mmap device memory\n");
return NULL;
}
return dev;
}

int scc52460_release_device() {
struct rfx_scc52460_fifo *dev = scc52460_get_device(0);
int status;
if(dev) {
status = munmap(dev, sizeof(struct rfx_scc52460_fifo));
if(status == 0) dev = NULL;
}
return status;
}

#endif // __KERNEL__

#ifdef __cplusplus
}
#endif

```

```
#endif // SCC52460_2_0_H
```

## SOURCE: Scc52460\_2.c

---

```
#include <linux/module.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/mm.h>
#include <linux/vmalloc.h>

#include <linux/io.h>
#include <linux/ptrace.h>

#include <asm/uaccess.h> // put_user
#include <asm/pgtable.h>

#include <linux/fs.h>

#include <linux/platform_device.h>

#include <linux/interrupt.h>

#include "scc52460_2.h"

#include <asm/io.h>
#include <linux/slab.h>

#define SUCCESS 0

static struct platform_device *s_pdev = 0;
static int s_device_open = 0;

// FOPS FWDDECL //
static int device_open(struct inode *, struct file *);
static int device_release(struct inode *, struct file *);
static ssize_t device_read(struct file *, char *, size_t, loff_t *);
static ssize_t device_write(struct file *, const char *, size_t, loff_t *);
static int device_mmap(struct file *filp, struct vm_area_struct *vma);
static loff_t memory_lseek(struct file *file, loff_t offset, int orig);
static long device_ioctl(struct file *file, unsigned int cmd, unsigned long arg);

static struct file_operations fops = {
    .read = device_read,
    .write = device_write,
    .open = device_open,
    .release = device_release,
    .mmap = device_mmap,
    .llseek = memory_lseek,
    .unlocked_ioctl = device_ioctl,
//    .poll = device_poll,
};

struct scc52460_dev {
    struct platform_device *pdev;
    struct cdev *cdev;
    int busy;
    void * iomap;
};
```

```

    void * iomap1;
    struct semaphore sem;      /* mutual exclusion semaphore */
};

////////////////////////////////////
// FIFO IO //////////////////////////////////////
////////////////////////////////////

void Write(void *addr, enum AxiStreamFifo_Register op, u32 data ) {
    *(u32 *)(addr+op) = data;
}

u32 Read(void *addr, enum AxiStreamFifo_Register op ) {
    return *(u32 *)(addr+op);
}

////////////////////////////////////
// FOPS //////////////////////////////////////
////////////////////////////////////

// OPEN //
static int device_open(struct inode *inode, struct file *file)
{
    if(!file->private_data) {
        struct scc52460_dev *dev = file->private_data = kmalloc(sizeof(struct
scc52460_dev),GFP_KERNEL);
        // dev->cdev = container_of(inode->i_cdev, struct scc52460_dev, cdev);
        dev->pdev = s_pdev;
        struct resource *r_mem = platform_get_resource(s_pdev, IORESOURCE_MEM, 0);
        u32 off = r_mem->start & ~PAGE_MASK;
        dev->iomap = devm_ioremap(&s_pdev->dev,r_mem->start+off,0xffff);
        r_mem = platform_get_resource(s_pdev, IORESOURCE_MEM, 1);
        off = r_mem->start & ~PAGE_MASK;
        dev->iomap1 = devm_ioremap(&s_pdev->dev,r_mem->start+off,0xffff);
        dev->busy=0;
    }
    struct scc52460_dev *dev = file->private_data;
    if(!dev) return -EFAULT;
    else if (dev->busy) return -EBUSY;
    else dev->busy++;
    return capable(CAP_SYS_RAWIO) ? 0 : -EPERM;
}

// CLOSE //
static int device_release(struct inode *inode, struct file *file)
{
    struct scc52460_dev *dev = file->private_data;
    if(!dev) return -EFAULT;
    if(--dev->busy == 0)
    {
        devm_iounmap(&dev->pdev->dev,dev->iomap);
        devm_iounmap(&dev->pdev->dev,dev->iomap1);
        kfree(dev);
    }
    return 0;
}

```

```

// READ //
static ssize_t device_read(struct file *filp, char *buffer, size_t length,
                           loff_t *offset)
{
    struct scc52460_dev *scc_dev = (struct scc52460_dev *)filp->private_data;
    void* dev = scc_dev->iomap;
    void* dev1 = scc_dev->iomap1;
    u32 *b32 = (u32*)buffer;

    u32 i, occ = Read(dev, RDFS);
    for(i=0; i < min(length/sizeof(u32), occ); ++i) {
        put_user(Read(dev1, RDFS), b32++);
    }
    return i*sizeof(u32);
}

// WRITE //
static ssize_t device_write(struct file *filp, const char *buff, size_t len,
                            loff_t *off)
{
    printk (<1>Sorry, this operation isn't supported.\n");
    return -EINVAL;
}

// MMIO //

static struct vm_operations_struct vm_ops;
static void mmap_close_cb(struct vm_area_struct *vma) {
    ; // do nothing
}

static const struct vm_operations_struct mmap_mem_ops = {
    .close = mmap_close_cb,
#ifdef CONFIG_HAVE_IOREMAP_PROT
    .access = generic_access_phys,
#endif
};

pgprot_t phys_mem_access_prot(struct file *file, unsigned long pfn,
                              unsigned long size, pgprot_t vma_prot)
{
    if (!pfn_valid(pfn))
        return pgprot_noncached(vma_prot);
    else if (file->f_flags & O_SYNC)
        return pgprot_writecombine(vma_prot);
    return vma_prot;
}

static int device_mmap(struct file *filp, struct vm_area_struct *vma)
{
    int status = 0;
    struct resource *r_mem = platform_get_resource(s_pdev, IORESOURCE_MEM, 0);

    unsigned long off = vma->vm_pgoff << PAGE_SHIFT;
    unsigned long physical = r_mem->start + off;
    size_t vsize = vma->vm_end - vma->vm_start;
    size_t psize = (r_mem->end - r_mem->start) - off;
    unsigned long pageFrameNo = __phys_to_pfn(physical);

    // register cb //
    vma->vm_ops = &mmap_mem_ops;
}

```

```

printk(KERN_DEBUG "<%s> file: mmap()\n", DEVICE_NAME);

printk(KERN_DEBUG "<%s> file: set physical = %x, size = %d\n",
        MODULE_NAME, physical, psize);

printk(KERN_DEBUG "<%s> file: destination = %x, size = %d\n",
        MODULE_NAME, vma->vm_start, vsize);

vma->vm_page_prot = phys_mem_access_prot(filp, vma->vm_pgoff,
        vsize,
        vma->vm_page_prot);

if (vsize > psize)
    return -EINVAL; /* spans too high */

status = remap_pfn_range(vma, vma->vm_start,
        pageFrameNo, vsize, vma->vm_page_prot);
if (status) {
    printk(KERN_ERR
        "<%s> Error: in calling remap_pfn_range: returned %d\n",
        MODULE_NAME, status);
    return -EAGAIN;
}
return status;
}

// LSEEK //
static loff_t memory_lseek(struct file *file, loff_t offset, int orig)
{
    loff_t ret;

    mutex_lock(&file_inode(file)->i_mutex);
    switch (orig) {
    case SEEK_CUR:
        offset += file->f_pos;
    case SEEK_SET:
        /* to avoid userland mistaking f_pos=-9 as -EBADF=-9 */
        if (IS_ERR_VALUE((unsigned long long)offset)) {
            ret = -EOVERFLOW;
            break;
        }
        file->f_pos = offset;
        ret = file->f_pos;
        force_successful_syscall_return();
        break;
    default:
        ret = -EINVAL;
    }
    mutex_unlock(&file_inode(file)->i_mutex);
    return ret;
}

// IOCTL //
static long device_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
    int status = 0;
    struct scc52460_dev *scc_dev = file->private_data;
    void* dev = scc_dev->iomap;
    // void* dev1 = scc_dev->iomap1;

    switch (cmd) {
    case RFX_SCC52460_RESOFFSET:

```

```

        printk(KERN_DEBUG "<%=s> ioctl: RFX_SCC52460_RESOFFSET\n", MODULE_NAME);
        if (copy_to_user((u32 *) arg, &dev, sizeof(u32)))
            return -EFAULT;
        break;

    case RFX_SCC52460_RESET:
        printk(KERN_DEBUG "<%=s> ioctl: RFX_SCC52460_RESET\n", MODULE_NAME);
        Write(dev, ISR, 0xFFFFFFFF);
        Write(dev, RDFR, 0xa5);
        return 0;
        break;

    case RFX_SCC52460_CLEAR:
        printk(KERN_DEBUG "<%=s> ioctl: RFX_SCC52460_CLEAR\n", MODULE_NAME);
        Write(dev, RDFR, 0xa5);
        return 0;
        break;

    case RFX_SCC52460_GETSR:
        printk(KERN_DEBUG "<%=s> ioctl: RFX_SCC52460_GETSR\n", MODULE_NAME);
        Write(dev, ISR, 0xFFFFFFFF);
        return 0;
        break;

    default:
        return -EAGAIN;
        break;
}
return status;
}

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// PROBE //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

```

static int id_major;
static struct class *scc52460_class;

static int rfx_scc52460_probe(struct platform_device *pdev)
{
    struct resource *r_mem;
    struct device *dev = &pdev->dev;

    s_pdev = pdev;
    printk("PLATFORM DEVICE PROBE...\n");

    // CHAR DEV //
    printk("registering char dev %s ...\n", pdev->name);
    id_major = register_chrdev(0, DEVICE_NAME, &fops);
    if (id_major < 0) {
        printk ("Registering the character device failed with %d\n", id_major);
        return id_major;
    }
    printk(KERN_NOTICE "mknod /dev/%s c %d 0\n", DEVICE_NAME, id_major);

    scc52460_class = class_create(THIS_MODULE, DEVICE_NAME);
    if (IS_ERR(scc52460_class))
        return PTR_ERR(scc52460_class);
}

```

```

// scc52460_class->devnode = scc52460_devnode;
device_create(scc52460_class, NULL, MKDEV(id_major, 0),
             NULL, DEVICE_NAME);

/* Get iospace for the device */
r_mem = platform_get_resource(pdev, IORESOURCE_MEM, 0);
if (!r_mem)
{
    dev_err(dev, "Can't find device base address\n");
    return 1;
}

// int irq = platform_get_irq(pdev,0);
// request_irq(irq,)

printk(KERN_DEBUG"mem start: %x\n",r_mem->start);
printk(KERN_DEBUG"mem end: %x\n",r_mem->end);
printk(KERN_DEBUG"mem offset: %x\n",r_mem->start & ~PAGE_MASK);
// printk(KERN_DEBUG"irq: %x\n",irq);

return 0;
}

static int rfx_scc52460_remove(struct platform_device *pdev)
{
    printk("PLATFORM DEVICE REMOVE...\n");
    if(scc52460_class) {
        device_destroy(scc52460_class,MKDEV(id_major, 0));
        class_destroy(scc52460_class);
    }
    unregister_chrdev(id_major, DEVICE_NAME);
    return 0;
}

static const struct of_device_id rfx_scc52460_of_ids[] = {
    { .compatible = "xlnx,axi-fifo-mm-s-4.1",},
    {}
};

static struct platform_driver rfx_scc52460_driver = {
    .driver = {
        .name = MODULE_NAME,
        .owner = THIS_MODULE,
        .of_match_table = rfx_scc52460_of_ids,
    },
    .probe = rfx_scc52460_probe,
    .remove = rfx_scc52460_remove,
};

static int __init rfx_scc52460_init(void)
{
    printk(KERN_INFO "inizializing AXI module ...\n");
    return platform_driver_register(&rfx_scc52460_driver);
}

static void __exit rfx_scc52460_exit(void)
{
    printk(KERN_INFO "exiting AXI module ...\n");
    platform_driver_unregister(&rfx_scc52460_driver);
}

module_init(rfx_scc52460_init);
module_exit(rfx_scc52460_exit);
MODULE_LICENSE("GPL");

```

## Kernel driver code

```
/* AXI DMA for scc52460
 *
 * This example demonstrates the use of dma transfer in two ways:
 *
 * 1) A single send/receive short circuit loop PS->PL->PS where the overall
 *    memory involved are allocated and handled within the kernel.
 *
 * 2) Send and receive a mmaped memory, the user can do the transfer directly.
 *    See: test_example3.c
 */

#include <linux/module.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <asm/errno.h>

#include <linux/slab.h> // kmalloc
#include <xen/page.h>

#include <asm/uaccess.h> // put_user
#include <asm/pgtable.h>
#include <linux/mm.h>

#include <linux/dmaengine.h> // dma api
#include <linux/dma/xilinx_dma.h> // axi dma driver

#include <linux/dma-mapping.h>
#include <linux/platform_device.h>

#include <linux/delay.h>

#include "axidma_example3.h"

// FOPS FWDDECL //
static int device_open(struct inode *, struct file *);
static int device_release(struct inode *, struct file *);
static ssize_t device_read(struct file *, char *, size_t, loff_t *);
static ssize_t device_write(struct file *, const char *, size_t, loff_t *);
static int device_mmap(struct file *filp, struct vm_area_struct *vma);
static long device_ioctl(struct file *file, unsigned int cmd, unsigned long arg);

static int xilinx_axidmatest_probe(struct platform_device *pdev);
static int xilinx_axidmatest_remove(struct platform_device *pdev);

////////////////////////////////////
// GLOBALS //////////////////////////////////////
////////////////////////////////////

static int s_major; // Major number assigned to our device driver */
static int s_device_open = 0; // Is device open? Used to prevent multiple
                             // access to the device */

#define SUCCESS 0
#define WAIT 1
```



```

#define NO_WAIT 0

// FOPS //
static struct file_operations fops = {
    .read = device_read,
    .write = device_write,
    .open = device_open,
    .release = device_release,
    .mmap = device_mmap,
    .unlocked_ioctl = device_ioctl,
};

// DEVICE ID //
static const struct of_device_id rfx_axidmatest_of_ids[] = {
    { .compatible = "xlnx,axi-dma-test-1.00.a", },
    {}
};

// PLATFORM //
static struct platform_driver rfx_axidmatest_driver = {
    .driver = {
        .name = MODULE_NAME,
        .owner = THIS_MODULE,
        .of_match_table = rfx_axidmatest_of_ids,
    },
    .probe = xilinx_axidmatest_probe,
    .remove = xilinx_axidmatest_remove,
};

// DMA //
static struct dma_chan *tx_chan;
static struct dma_chan *rx_chan;

static struct completion tx_cmp;
static struct completion rx_cmp;
static dma_cookie_t tx_cookie;
static dma_cookie_t rx_cookie;

static char *src_dma_buffer;
static char *dst_dma_buffer;
static dma_addr_t tx_dma_handle;
static dma_addr_t rx_dma_handle;
static unsigned long dma_buffer_length;

////////////////////////////////////
// TEST_TRANSFER //////////////////////////////////////
////////////////////////////////////

/* Handle a callback and indicate the DMA transfer is complete to another
 * thread of control
 */
static void axidma_test_transfer_sync_callback(void *completion)
{
    complete(completion);
}

/* Prepare a DMA buffer to be used in a DMA transaction, submit it to the DMA engine
 * to queued and return a cookie that can be used to track that status of the

```

```

* transaction
*/
static dma_cookie_t axidma_test_transfer_prep_buffer(struct dma_chan *chan, dma_addr_t buf,
size_t len,
enum dma_transfer_direction dir, struct completion
*cmp)
{
enum dma_ctrl_flags flags = /*DMA_CTRL_ACK */ DMA_PREP_INTERRUPT;
struct dma_async_tx_descriptor *chan_desc;
dma_cookie_t cookie;

chan_desc = dmaengine_prep_slave_single(chan, buf, len, dir, flags);
if (!chan_desc) {
printk(KERN_ERR "dmaengine_prep_slave_single error\n");
cookie = -EBUSY;
} else {
chan_desc->callback = axidma_test_transfer_sync_callback;
chan_desc->callback_param = cmp;
cookie = dmaengine_submit(chan_desc);
}
return cookie;
}

/* Start a DMA transfer that was previously submitted to the DMA engine and then
* wait for it complete, timeout or have an error
*/
static void axidma_start_test_transfer(struct dma_chan *chan, struct completion *cmp,
dma_cookie_t cookie, int wait)
{
unsigned long timeout = msecs_to_jiffies(10000);
enum dma_status status;

init_completion(cmp);
dma_async_issue_pending(chan);

if (wait) {
timeout = wait_for_completion_timeout(cmp, timeout);
status = dma_async_is_tx_complete(chan, cookie, NULL, NULL);
if (timeout == 0) {
printk(KERN_ERR "DMA timed out\n");
} else if (status != DMA_COMPLETE) {
printk(KERN_ERR "DMA returned completion callback status of: %s\n",
status == DMA_ERROR ? "error" : "in progress");
}
}
}

static int axidma_test_transfer(unsigned int dma_length)
{
int i, status = 0;

char *src_buffer = dma_alloc_coherent(tx_chan->device-
>dev, dma_length, &tx_dma_handle, GFP_KERNEL);
char *dst_buffer = dma_alloc_coherent(rx_chan->device-
>dev, dma_length, &rx_dma_handle, GFP_KERNEL);

if (!src_buffer || !dst_buffer) {
printk(KERN_ERR "Allocating DMA memory failed\n");
return -EIO;
}

// fill source //
for (i = 0; i < dma_length; i++)
src_buffer[i] = i;

```

```

    // Mapping and unmapping is not actually needed as the alloc coherent is
    // already providing a dma_addr_t mem handler
    // tx_dma_handle = dma_map_single(tx_chan->device->dev, src_buffer, dma_length,
DMA_TO_DEVICE);
    // rx_dma_handle = dma_map_single(rx_chan->device->dev, dst_buffer, dma_length,
DMA_FROM_DEVICE);

    tx_cookie = axidma_test_transfer_prep_buffer(tx_chan, tx_dma_handle, dma_length,
DMA_MEM_TO_DEV, &tx_cmp);
    rx_cookie = axidma_test_transfer_prep_buffer(rx_chan, rx_dma_handle, dma_length,
DMA_DEV_TO_MEM, &rx_cmp);

    if (dma_submit_error(rx_cookie) || dma_submit_error(tx_cookie)) {
        printk(KERN_ERR "xdma_prep_buffer error\n");
        return -EIO;
    }

    printk(KERN_INFO "Starting DMA transfers\n");

    axidma_start_test_transfer(rx_chan, &rx_cmp, rx_cookie, NO_WAIT);
    axidma_start_test_transfer(tx_chan, &tx_cmp, tx_cookie, WAIT);

    // see above..
    // dma_unmap_single(rx_chan->device->dev, rx_dma_handle, dma_length, DMA_FROM_DEVICE);

    // dma_unmap_single(tx_chan->device->dev, tx_dma_handle, dma_length, DMA_TO_DEVICE);

    /* Verify the data in the destination buffer matches the source buffer */
    for (i = 0; i < dma_length; i++) {
        if (dst_buffer[i] != src_buffer[i]) {
            printk(KERN_INFO "DMA transfer failure");
            status = -EIO;
            break;
        }
    }

    printk(KERN_INFO "DMA bytes sent: %d\n", dma_length);
    dma_free_coherent(tx_chan->device->dev, dma_length, src_buffer, tx_dma_handle);
    dma_free_coherent(rx_chan->device->dev, dma_length, dst_buffer, rx_dma_handle);

    return status;
}

////////////////////////////////////
// IOCTLS //////////////////////////////////////
////////////////////////////////////

static int device_ioctl_testtrasfer(unsigned int dma_length)
{
    // TEST DMA FUNCTIONALITY //
    if(!dma_length) dma_length = BUFFER_SIZE;
    return axidma_test_transfer(dma_length);
}

static int device_ioctl_mmap_testtransfer(void ) {

    // char *src = src_dma_buffer;
    // char *dst = dst_dma_buffer;
    // unsigned long len = dma_buffer_length;

    tx_cookie = axidma_test_transfer_prep_buffer(tx_chan, tx_dma_handle, dma_buffer_length,
DMA_MEM_TO_DEV, &tx_cmp);

```

```

        rx_cookie = axidma_test_transfer_prep_buffer(rx_chan, rx_dma_handle, dma_buffer_length,
                                                    DMA_DEV_TO_MEM, &rx_cmp);
    if (dma_submit_error(rx_cookie) || dma_submit_error(tx_cookie)) {
        printk(KERN_ERR "xdma_prep_buffer error\n");
        return -EIO;
    }

    axidma_start_test_transfer(rx_chan, &rx_cmp, rx_cookie, NO_WAIT);
    axidma_start_test_transfer(tx_chan, &tx_cmp, tx_cookie, WAIT);
    return 0;
}

```

```

////////////////////////////////////
// FOPS //////////////////////////////////////
////////////////////////////////////

```

```

// OPEN //
static int device_open(struct inode *inode, struct file *file)
{
    if (s_device_open) return -EBUSY;
    s_device_open++;

    return SUCCESS;
}

```

```

// CLOSE //
static int device_release(struct inode *inode, struct file *file)
{
    s_device_open--;
    return 0;
}

```

```

// READ //
static ssize_t device_read(struct file *filp, char *buffer, size_t length,
                           loff_t *offset)
{
    int bytes_read = 0;
    const char *msg = "use mmap to access device memory\n";
    const char *msg_Ptr = msg;
    while (length && *msg_Ptr) {
        put_user>(*msg_Ptr++, buffer++);
        length--; bytes_read++;
    }
    return bytes_read;
}

```

```

// WRITE //
static ssize_t device_write(struct file *filp, const char *buff, size_t len,
                            loff_t *off)
{
    printk("<1>Sorry, this operation isn't supported.\n");
    return -EINVAL;
}

```

```

static struct vm_operations_struct vm_ops;
static void mmap_close_cb(struct vm_area_struct *vma) {
    // release dma buffer //
    if(src_dma_buffer) dma_free_coherent(tx_chan->device-
>dev,dma_buffer_length,src_dma_buffer,tx_dma_handle);
    if(dst_dma_buffer) dma_free_coherent(rx_chan->device-
>dev,dma_buffer_length,dst_dma_buffer,rx_dma_handle);
}

// MMAP //
static int device_mmap(struct file *filp, struct vm_area_struct *vma)
{
    int result;
    unsigned long requested_size;

    requested_size = vma->vm_end - vma->vm_start;
    dma_buffer_length = requested_size / 2;

    // ALLOCATE BUFFERS //
    src_dma_buffer = dma_alloc_coherent(tx_chan->device-
>dev,dma_buffer_length,&tx_dma_handle,GFP_KERNEL);
    dst_dma_buffer = dma_alloc_coherent(rx_chan->device-
>dev,dma_buffer_length,&rx_dma_handle,GFP_KERNEL);
    if (!src_dma_buffer || !dst_dma_buffer) {
        printk(KERN_ERR "Allocating DMA memory failed\n");
        return -EIO;
    }

    // register free cb //
    vm_ops.close = mmap_close_cb;
    vma->vm_ops = &vm_ops;

    printk(KERN_DEBUG "<%s> file: mmap()\n", DEVICE_NAME);
    printk(KERN_DEBUG "<%s> file: memory size reserved: %d, mmap size requested: %lu\n",
        MODULE_NAME, BUFFER_SIZE, requested_size);

    if (requested_size > BUFFER_SIZE) {
        printk(KERN_ERR "<%s> Error: %d reserved != %lu requested\n",
            MODULE_NAME, BUFFER_SIZE, requested_size);
        return -EAGAIN;
    }

    // remap pages source //
    vma->vm_page_prot = pgprot_noncached(vma->vm_page_prot);
    result = remap_pfn_range(vma, vma->vm_start,
        virt_to_pfn(src_dma_buffer),
        dma_buffer_length, vma->vm_page_prot);

    if (result) {
        printk(KERN_ERR
            "<%s> Error: in calling remap_pfn_range: returned %d\n",
            MODULE_NAME, result);
        return -EAGAIN;
    }

    // remap pages destiation //
    result = remap_pfn_range(vma, vma->vm_start + dma_buffer_length,
        virt_to_pfn(dst_dma_buffer),
        dma_buffer_length, vma->vm_page_prot);

    if (result) {
        printk(KERN_ERR
            "<%s> Error: in calling remap_pfn_range: returned %d\n",
            MODULE_NAME, result);
        return -EAGAIN;
    }

    return 0;
}

```

```
}
```

```
static long device_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
    u32 devices;
    int status = 0;

    switch (cmd) {
        case XDMA_GET_NUM_DEVICES:
            printk(KERN_DEBUG "<%=s> ioctl: XDMA_GET_NUM_DEVICES\n", MODULE_NAME);
            devices = 1;
            if (copy_to_user((u32 *) arg, &devices, sizeof(u32)))
                return -EFAULT;
            break;
        case XDMA_TEST_TRASFER:
            printk(KERN_DEBUG "<%=s> ioctl: XDMA_TEST_TRASFER\n", MODULE_NAME);
            status = device_ioctl_testtrasfer(0);
            break;
        case XDMA_TEST_MMAPTRASFER:
            // printk(KERN_DEBUG "<%=s> ioctl: XDMA_TEST_TRASFER\n", MODULE_NAME);
            status = device_ioctl_mmap_testtransfer();
            break;

        default:
            return -EAGAIN;
            break;
    }
    return status;
}
```

```
////////////////////////////////////
// PLATFORM //////////////////////////////////////
////////////////////////////////////
```

```
static int xilinx_axidmatest_probe(struct platform_device *pdev)
{
    int err;
    // struct dma_slave_config tx_conf;
    // struct dma_slave_config rx_conf;

    printk("probing %s ...\n", pdev->name);

    // CHAR DEV //
    printk("registering char dev %s ...\n", DEVICE_NAME);
    s_major = register_chrdev(0, DEVICE_NAME, &fops);
    if (s_major < 0) {
        printk ("Registering the character device failed with %d\n", s_major);
        return s_major;
    }
    printk(KERN_NOTICE "mknod /dev/xdma c %d 0\n", s_major);

    /* Allocate DMA slave channels */
    tx_chan = dma_request_slave_channel(&pdev->dev, "dma0");
    if (IS_ERR(tx_chan)) {
        pr_err("xilinx_dmatest: No Tx channel\n");
        dma_release_channel(tx_chan);
        return -EFAULT;
    }

    rx_chan = dma_request_slave_channel(&pdev->dev, "dma1");
```

```

    if (IS_ERR(rx_chan)) {
        err = PTR_ERR(rx_chan);
        pr_err("xilinx_dmatest: No Rx channel\n");
        dma_release_channel(tx_chan);
        dma_release_channel(rx_chan);
        return -EFAULT;
    }

    return 0;
}

static int xilinx_axidmatest_remove(struct platform_device *pdev)
{
    dma_release_channel(rx_chan);
    dma_release_channel(tx_chan);

    printk("unregistering char dev ...\n");
    unregister_chrdev(s_major, DEVICE_NAME);

    return 0;
}

static int __init axidma_init(void)
{
    printk(KERN_INFO "initializing module %s\n", rfx_axidmatest_driver.driver.name);
    return platform_driver_register(&rfx_axidmatest_driver);
}

static void __exit axidma_exit(void)
{
    printk(KERN_INFO "exiting module %s\n", rfx_axidmatest_driver.driver.name);
    platform_driver_unregister(&rfx_axidmatest_driver);
}

module_init(axidma_init);
module_exit(axidma_exit);
MODULE_LICENSE("GPL");

```





## B. Appendix: Software tools for PL section.

### The design on Vivado

This thesis has been developed using the Vivado IDE software distributed by Xilinx.

The version is "Xilinx\_Vivado\_SDK\_2015.4\_1118\_2\_Win64" which also provides powerful simulation and debugging tools as well as tutorials and informational materials.

The launch of the setup icon has as of today looks like this.



**Figure B-1 Xilinx Vivado setup package.**

The development platform look like as an integrated portal where the six most useful tools are directly accessible as visible in Figure B-2.



**Figure B-2 Vivado Welcome Page.**

## **Create a new Verilog project Step by Step.**

Let's see an example to lead the creation of a new project compatible with the RedPitaya hardware based on **xc7z010clg400-1** chip

It show how to implement a simple arithmetic logic units inside the FPGA (PL) and furnish it with the necessary interfaces in order to control the ARM (PS section).

In essence we should customize a new IP block.

### ***Basic Getting Started:***

*Vivado a project, be it developed in VHDL or Verilog, it is organized into blocks called IP (intellectual property) of which the first entered will be the Top Module.*

*Each IP is defined and developed through an instance for this reason Vivado environment, when are necessary to create a custom IP, will see the open environment twice.*

*In the example there will be a first Vivado activation refers to the top module, which is the central chip xc7z010clg400-1 installed in RedPitaya platform, and a second activation for custom IP here call RFX-ALU.*

*It should be recalled that for each IP there are two types of display and possibility of customization. Each one can be manipulated using visual editor or Verilog script.*

*Using the block editor, accessible from the "Open Design" in the flow Navigator, you can set the signals, buses, synchronized controls, buffers, while scripts are made*

*internal functions, such as the calculation  $F = A + B$  that we are going present in this example.*

*Using the block editor will realize the interconnections between IP blocks that make up the project.*

*The interconnections between the buses, with the addition of standard interfaces based on AXI, is managed by an automatic tool of Vivado launchable from link, in block design, indicated with "Run Block Automation".*

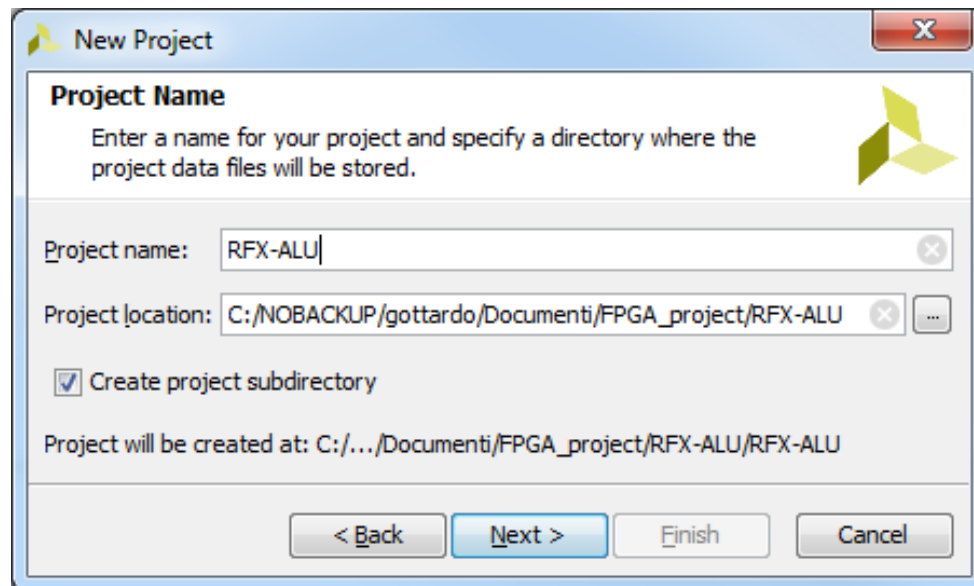
Let's see in ten steps how to make a verilog project on Vivado environment.

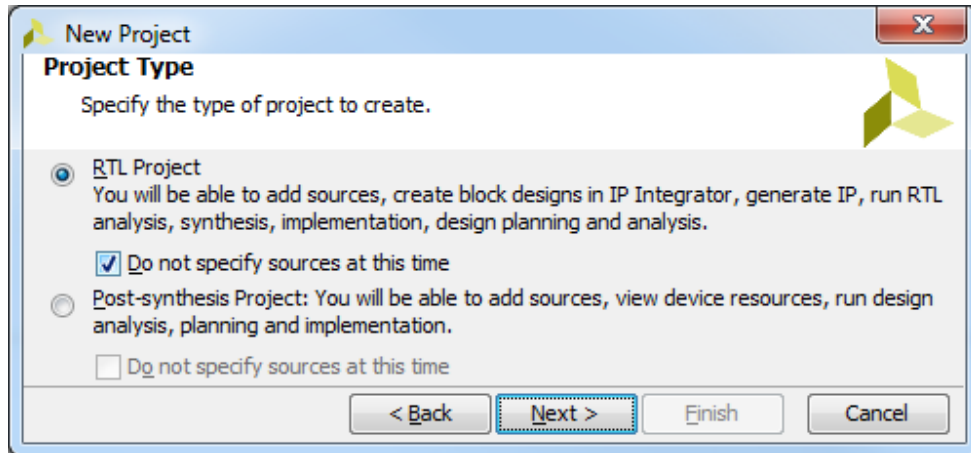
## Step 1 - Hardware definition.

Let's create a new RTL project based on Vivado xc7z010clg400-1.

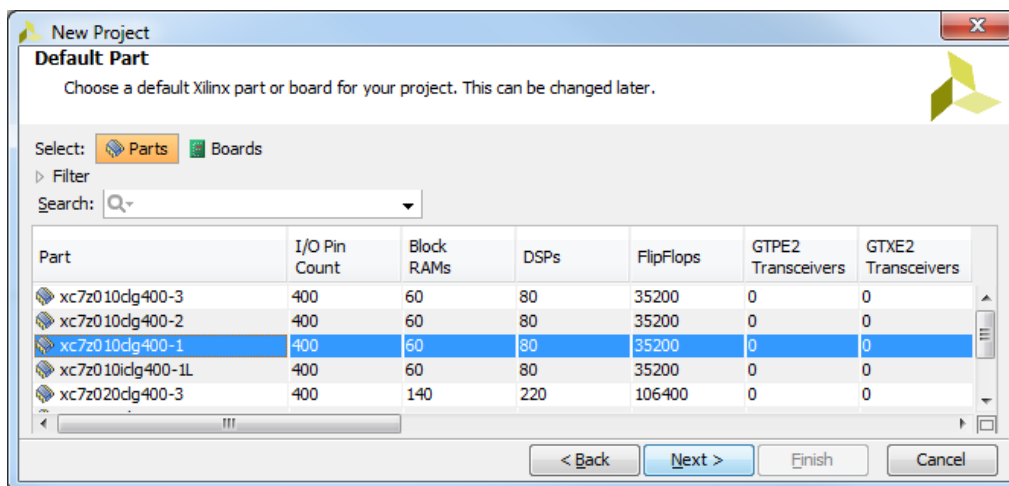


Let's set the name for the new project and the location with the appropriate subfolder. These will automatically be created.

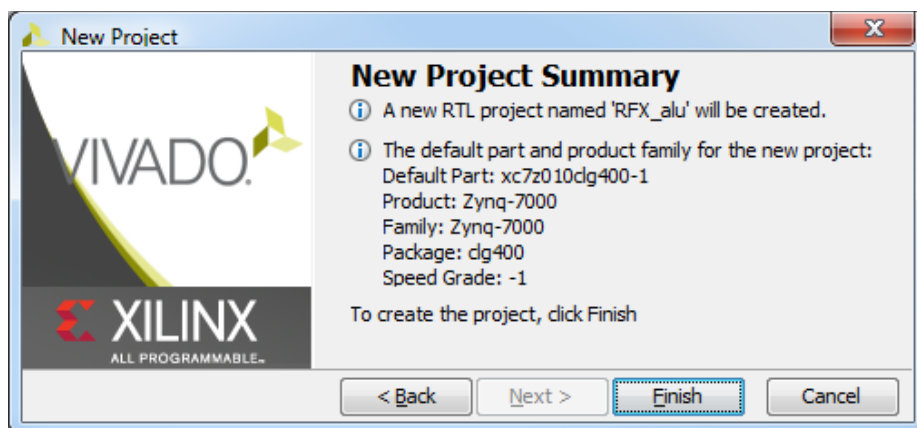




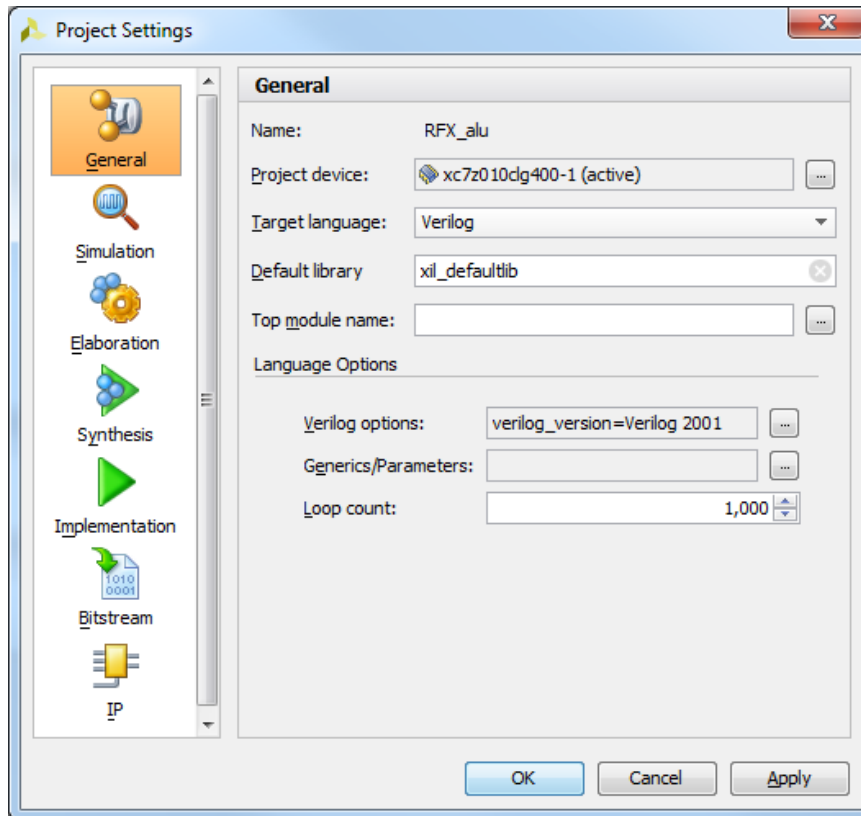
Among some passages set Verilog syntax option as "verilog Verilog version = 2001".



If what is presented is not equal to the image click on the triangle "Filter" and then seek for item xc7z010clg400-1 list, or the chip assembled in RedPitaya. This will display a table of contents.



Set on menu "tool-> Project Settings", the Verilog HDL (Hardware Description Language) in order to have compatibility with RedPitaya platform, then select "Target language -> Verilog".



**Warning:** There are two entries extending the Target menu languages:

- VHDL
- Verilog

The aim of this chapter is to create an FPGA project compatible with RedPitaya platform for this reason we will select **Verilog**.

The setting will be maintained even for the more internal modules.

## Step 2 – Adding the source to be packaged in IP block design.

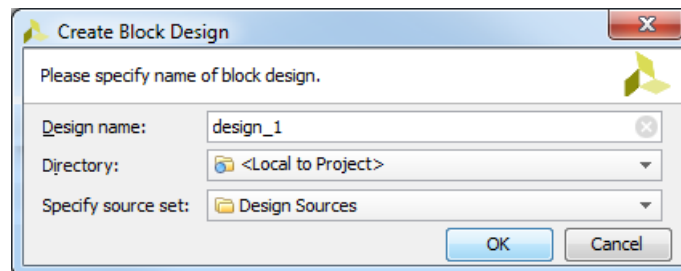
The logical blocks can be inserted in Vivado in two ways.

1. From verilog input scripts.
2. From preconfigured ip (intellectual property) packages. These are available in selected repository.

Let's see, according to the second method, how to create a macro that automatically interconnect with other packages in Vivado IP Integrator board using the tool "Block Design".


On the left-side panel "Flow Navigator" and we put the focus on IP Integrator. At this time should show activates only the possibility of creating a new design with the "Create Block Design."

Acting on "Create Block Design" appears the follow window :

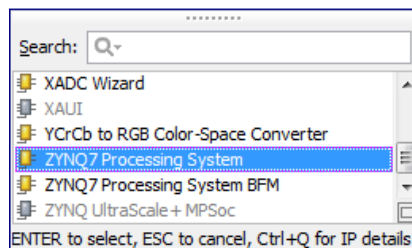


Acting on "OK" it create a new instance of the Block design named Design 1 where to add the new Zynq instance that the central block of the design.

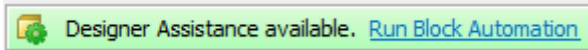
At the center of the editor window the message appears:

This design is empty. Press the  button to add IP.

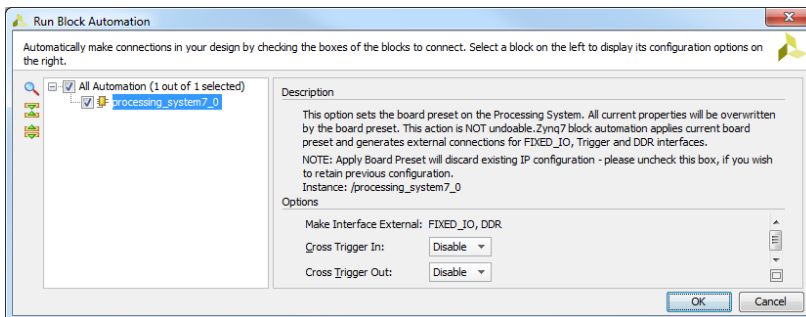
Click add IP, which is the small icon shown above and looking for the target "**ZYNQ7 Processing System**" available in IP packets.



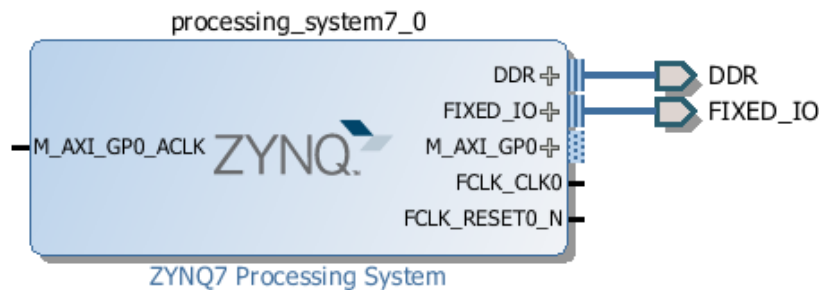
The new IP block for the processor will appear on the page lonely and lacking any connection. Observing the green commands bar it notice the presence of a link, which acts "Run Block automation".



Acting on Block Run Automation will be added to the IP of the first connections to the DDR and FIXED\_IO ports, you are asked if you need to connect the trigger signals, but we could also do it at a later time.



The IP block will appear equipped of the first connections as shown below.



It is essential to create the **wrapper** before you enter the next step.

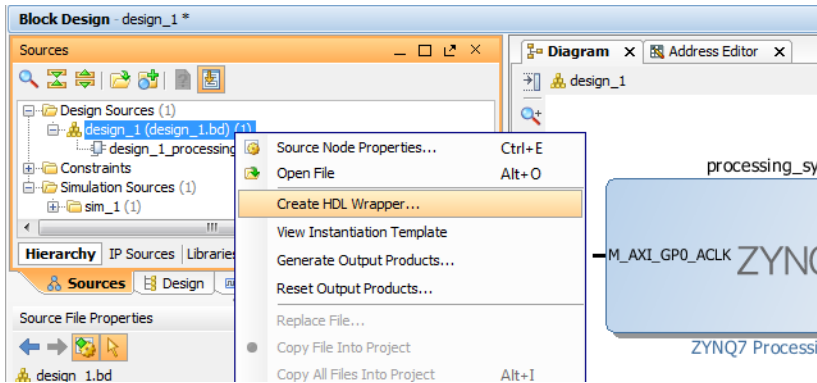
This allows to unravel in the form of HDL script code the graphic representation shown above.

The creation of the wrapper can follow certain steps.

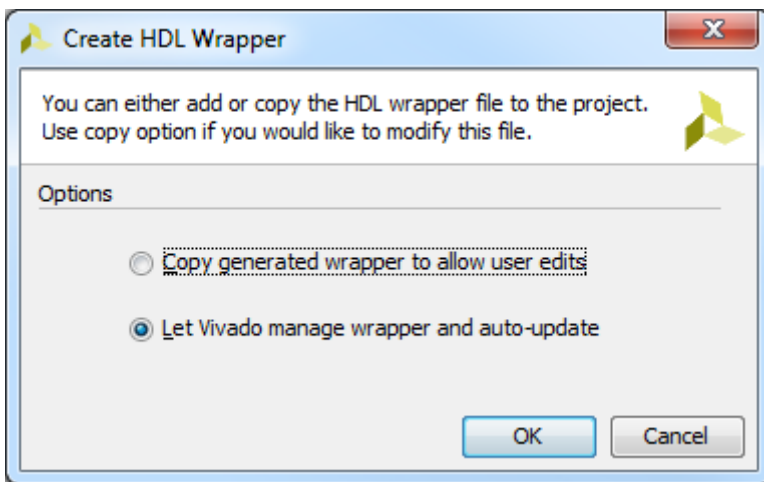
Right-click on the block to those who want to run the Wrapp.

The item appears on the center panel "Design Source 'shown as a folder that once opened shows us" design\_1 ", as shown on next figure.





launching the wrapper.

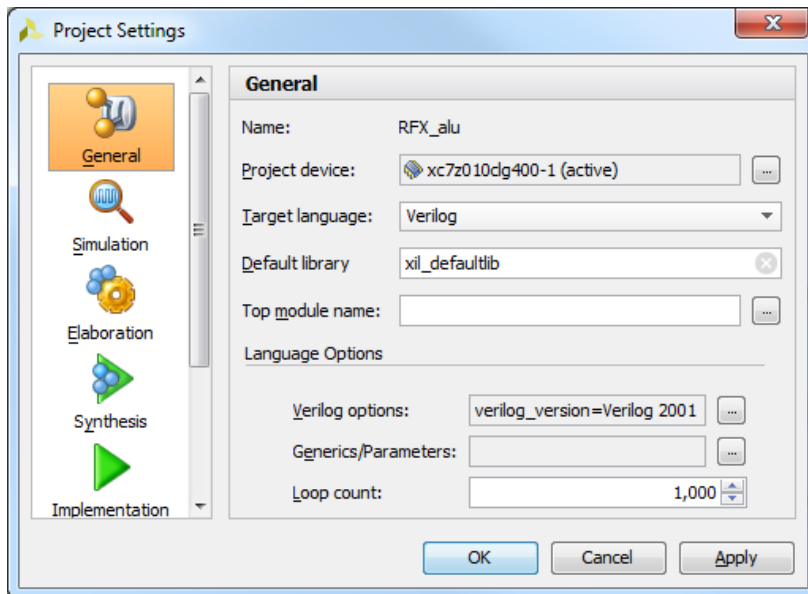


**Caution:** By running the wrapper at this time you may be notified that the `M_AXI_GP0_ACLK` the error signal is not connected yet. The situation will be resolved later.

### Step 3 – Creating a new Verilog IP

The preliminary action will select as Target Language Verilog, (it would be appropriate that this selection was made from the first step, but we could also do now).

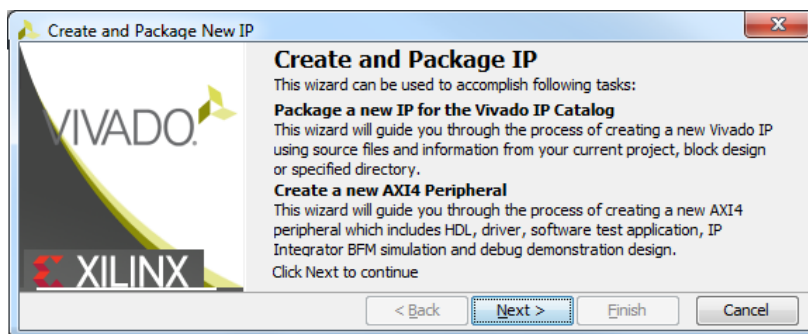
In the Flow Navigator act on Project Settings and set Verilog.



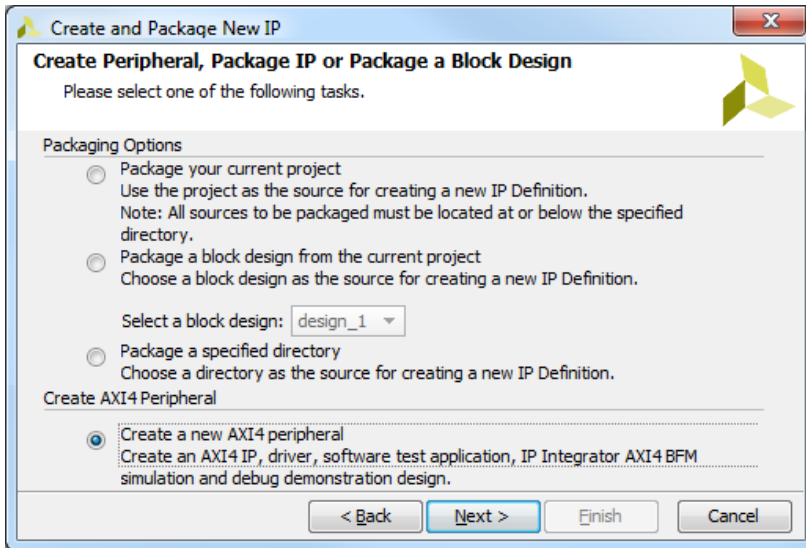
Vivado is able to create a custom package which integrates the functions Verilog that we are interested to add and use.

The new package will be a particular Verilog subproject.

In the main menu: **Tools->Create and Package IP.**



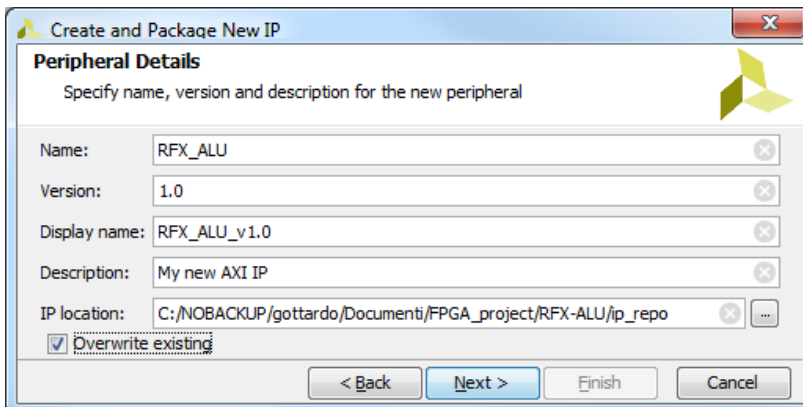
It's is possible to create multiple types of IP as shown below.



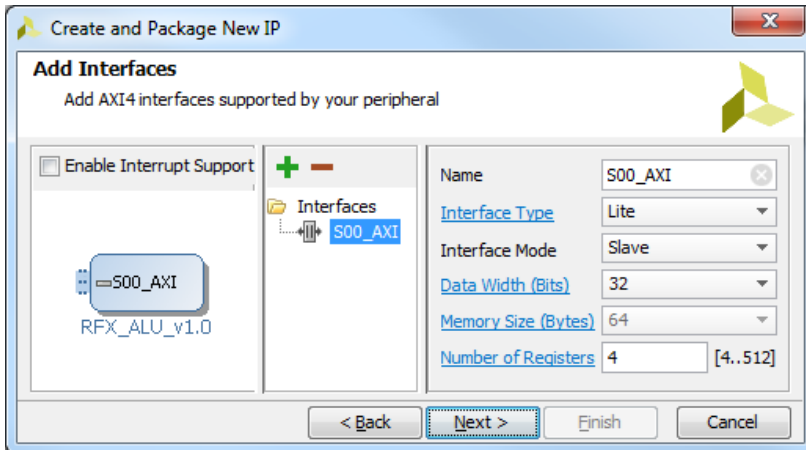
Add new IP as AXI 4 device. It is the responsibility of the Vivado tool add drivers and the necessary internal connections.

Click on Next.

The system ask for the new device name, type, location and descriptive information.

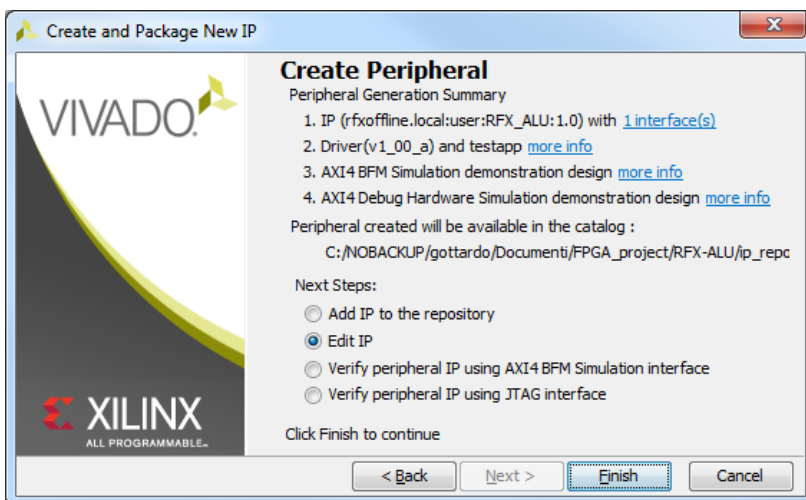


Pay attention at check box "Overwrite existing". These will destroying all previous tests.

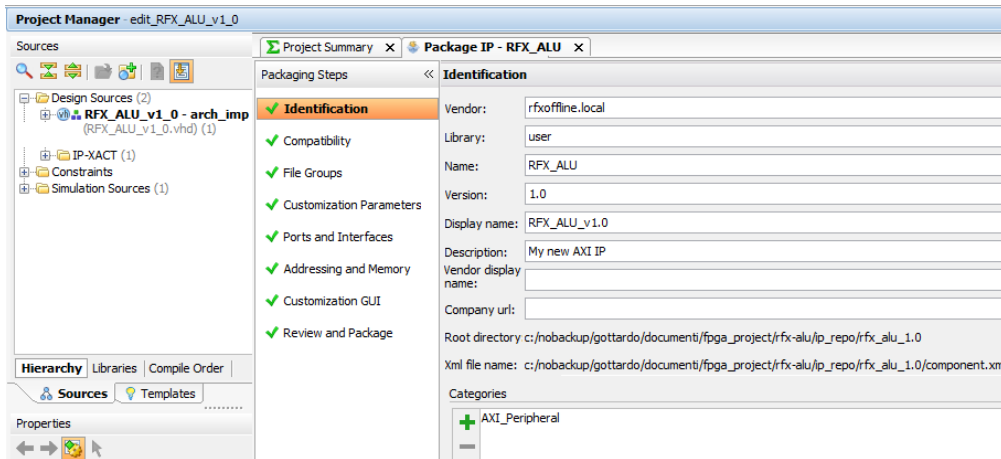


Continue with "Next" and we will see a window that summarizes the characteristics of the new IP block and allows to store it in a specific folder.

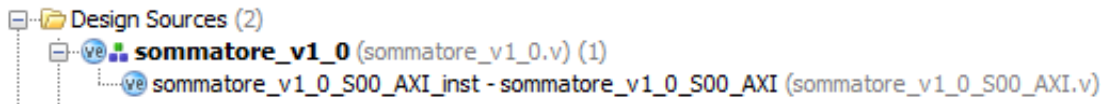
The archive may be made to folders created by the system, called repository, or in a customized folder that is suggested to call "src", or that will contain the source files. The "src" folder must be created manually.



The action opens a new execution of Vivado for the new IP. The previous stays open in the background.



On the **Design Sources** must appear the following situation:

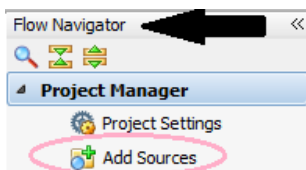


Carefully check that the round icon is written inside (ve) abbreviation for Verilog and not Vh.

## Step 4 – Editing IP and sources add

In the fourth step will proceed by clicking on the Add Sources of Project Manager in "Flow Navigator panel".

Must remember that now are active the second instance of Vivado which opened in creating the new custom IP.



This involves the activation of the new window:

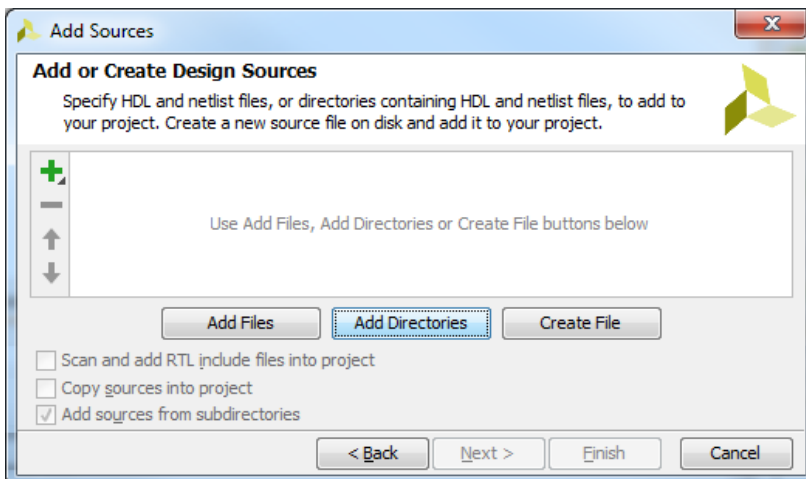


Create a new Verilog files in order to realize the addition to the design. We must be alert to the need to create a new subfolder, compared to the IP of the main project location, and then within the project, in which to place the source files.

*From some tests carried out it seems that placing new sources "to local project" involves a non-operation.*

The custom IP project is automatically added to a folder called "repo\_ip" contained in the Path of the active project, here we will find the rfx\_alu project, We will have to create a new folder named "src"

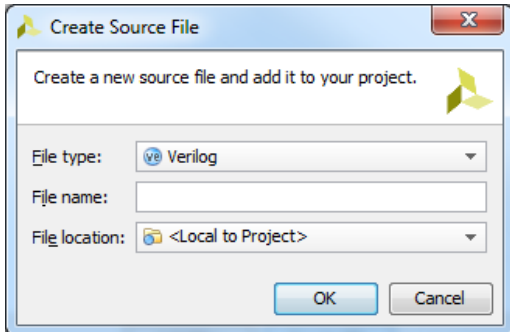
Here it is set to the correct path for the new script.



Click on "Add Directories" and add a comfortable directory "SRC" within the current project as shown below.

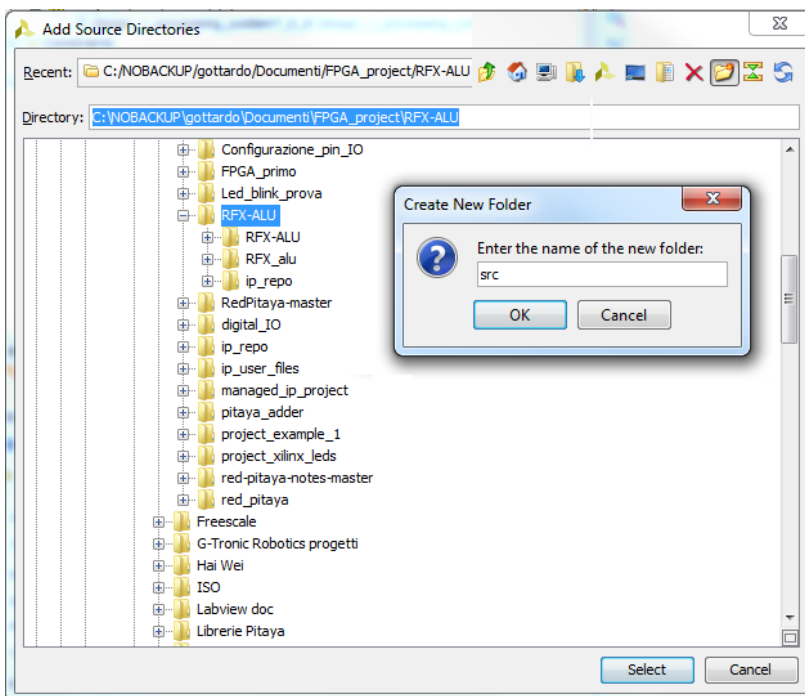
In recent versions of Vivado "Add Directories" may not be present then click on "Create File."

Will appear a window as shown below:

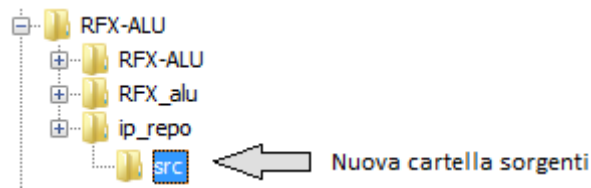


Click on File locations <Project Local>, then the browser will show the folders shown in the next image. Click the folder icon next to the window closing command, in order to create a new folder in which to save the files.

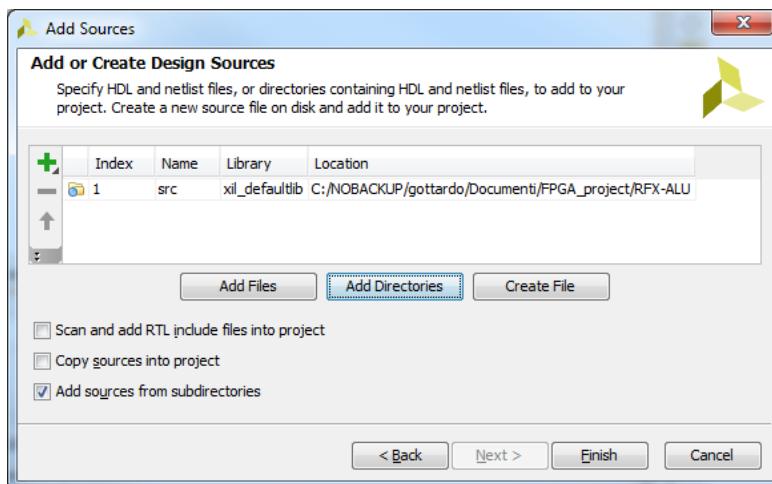
On the action "new folder" the "Create new folder window appears", called "src" that will position on the "ip\_repo".



The aspect of the project will become this:



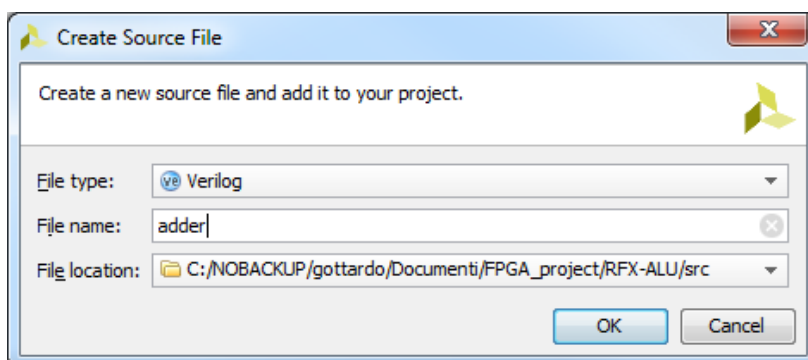
On confirmation the following window appears that asks us the final confirmation for the new files and path.



From the window above we can create a new file by clicking "Create File" of Verilog type.

So: Create a Verilog type script by giving the name "adder", and save it in the folder "ip\_repo / rfx\_alu1.0 / src"

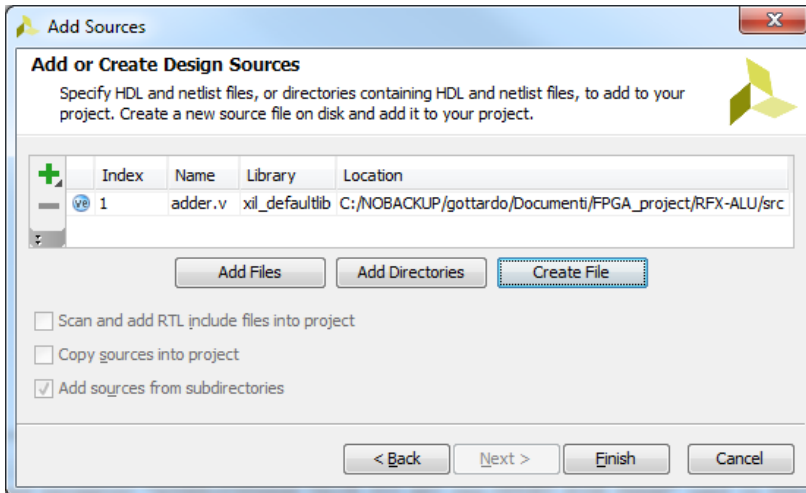
The provided settings and paths are in the image.



Confirming we will see the new Verilog script, distinguishable by the small round icon that contains "ve", as assigned and in the request path and file name.

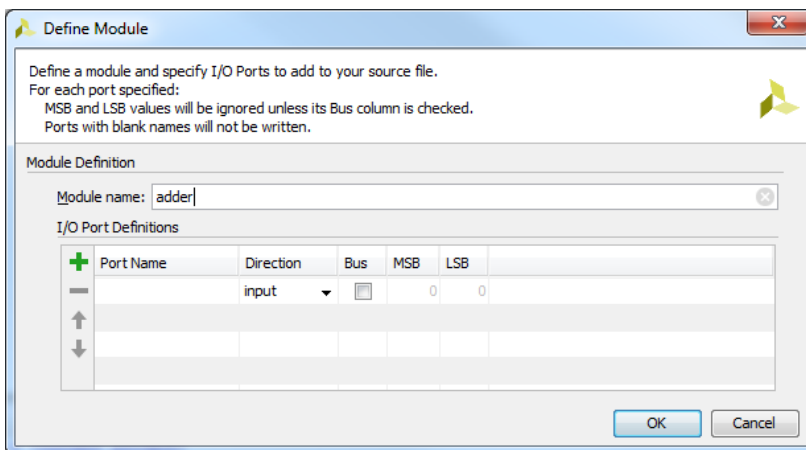
We will notice that in front the item "index" and "name", the icon (Ve) which confirms the initial choice of Verilog syntax.





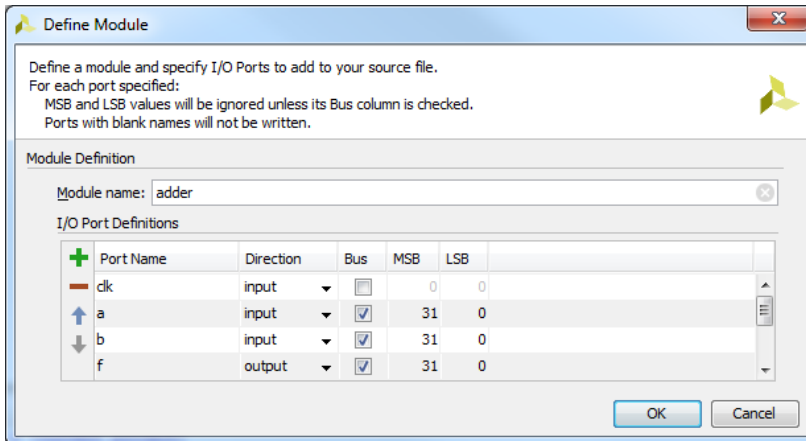
Let's confirm with "Finish". This will open a window that allows us to define the input and output ports of the new module.

It's essential to assign the correct extension in bit so that it can perform calculations correctly.

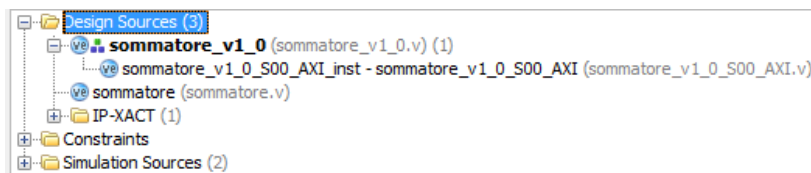


Now create the gates for the new module assigning the correct bit extensions, in these case the clock is 1 bit, while the input and output ports will be 32-bit.

Below we see the example:



The previous steps involves the addition of a new source module named `adder.v` allocated in **Sources** that will contain the logic module written in Verilog in which the GPIO and pins are preset with the name and extension as showed in the previous image.

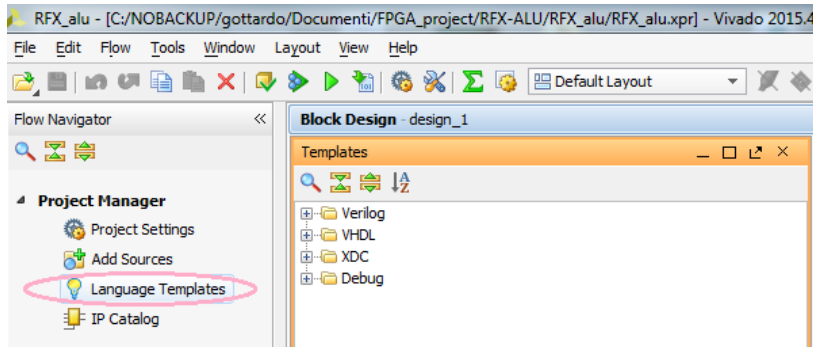


**Note:** in the image above is shown `sommatore.v` instead `adder.v`

Now is necessary manually act on the sources files to inserting the mathematical operation that you want to implement in Verilog by using Vivado templates.

## Step 5 – Module insertion using the template

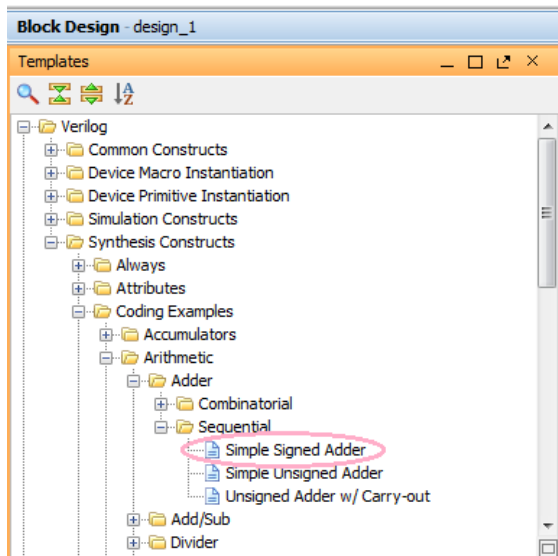
Click on Tab templates close to sources, in the Flow Navigator panel, as in the image.



Extend "Verilog" in which we will find a very specific and normally long path.

in this case the template will be in:

**Verilog ->Synthesis Constructs- >Coding Examples->Arithmetic- >Adder->Sequential- >Simple Signed Adder**



Appear a code preview like shown in the window below the , formed by a few lines of code, see figure.

```

1
2 parameter ADDER_WIDTH = <adder_bit_width>;
3
4 reg signed [ADDER_WIDTH-1:0] <sum> = {ADDER_WIDTH{1'b0}};
5
6 always @(posedge <CLK>)
7     <sum> <= <a_input> + <b_input>;
8
9

```

Will perform the code copy and paste of the adder.v script to complete the form in the indicated manner:

```

module adder(
    input clk,
    input [ADDER_WIDTH1:0] a,
    input [ADDER_WIDTH1:0] b,
    output [ADDER_WIDTH1:0] f
);
parameter ADDER_WIDTH = 32;
reg signed [ADDER_WIDTH1:0] f = {ADDER_WIDTH{1'b0}};

always @(posedge clk);
    f <= a + b;
endmodule

```

**Note:** The clipboard copy of the verilog code is done by clicking on the first icon on the left, if it is not enabled, we could proceed on manual copy in classic way. Returning to the "Sources" tab reappears the tree in which extending the "Design Sources" branch we find the adder.v form.

Performing the double click will have access to Verilog editor, in the right pane, where to insert the code.

In the next image we can observe the presence of two icons whit bulb shape, these allow the access to template Verilog code. These file can be integrated on code.

```
C:/NOBACKUP/gottardo/Documents/FPGA_project/RFX-ALU/src/adder.v
1 `timescale 1ns / 1ps
2 ///////////////////////////////////////////////////////////////////
3 // Company: RFX (reversed Field experiment) cnr Padova
4 // Engineer: Marco Gottardo
5 //
6 // Create Date: 04/12/2016 16:33:48 PM
7 // Design Name: RFX-ALU
8 // Module Name: adder
9 // Project Name: RFX_alu
10 // Target Devices:xc7z010clg400-1 on RedPitaya
11 // Tool Versions: Vivado 2015.4
12 // Description: Simple custom ALU on FPGA
13 ///////////////////////////////////////////////////////////////////
14
15 module adder(
16     input clk,
17     input [31:0] a,
18     input [31:0] b,
19     output [31:0] f
20 );
21
22     parameter ADDER_WIDTH = 32;
23     reg signed [ADDER_WIDTH-1:0] f = {ADDER_WIDTH{1'b0}};
24     always @(posedge clk)
25         f <= a + b;
26 endmodule
```

On this code it could also enter all other operations possible with typically ALU.  
For the moment it just perform the sum.

**Warning:** compared to the previous image, which shows the preview, items in angular brackets specifying those from specific instance.  
Remember to save the file by acting icon visible spot beside the path and name file.

## Step 6 – AXI wrapper

The wrapper is a software tool included in Vivado running the routing of the logical network, the form of graphics blocks of new IP or IP library in physical electrical connections to be made on the silicon surface of the FPGA.

The AXI wrapper allows to run a connection wizard for AXI interface associating the various clock and signals.

Open the file *rfx\_alu\_v1\_0\_S00\_AXI.v* and add the instance (ie the call or activation) on new adder module, append at the end of the file.

Vivado identifies automatically the editor area in which to insert the new code creating the start and end point where include the source code.

```
// Add user logic here
//User logic ends
```

*Example of code insertion:*

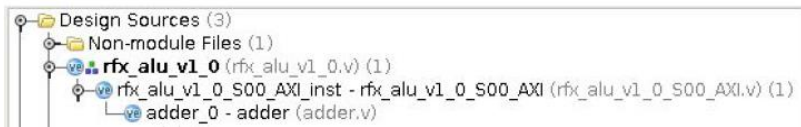
```
// Add user logic here
wire [31:0] operation_out;
adder#(32) adder_0(S_AXI_ACLK,slv_reg1,slv_reg2,operation_out);
// User logic ends
```

The "adder\_0" instance in alu AXI module will perform the operation at each cycle of the AXI interface clock (which generally does not coincide with the central processor clock cycles).

In the f register of the instance of new adder connected to pin operation\_out (32-bit) will read the result which will be copied on the memory mapped by AXI interface.

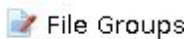
```
// Implement memory mapped register select and read logic generation
// Slave register read enable is asserted when valid address is available
// and the slave is ready to accept the read address.
assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;
always @(*)
begin
// Address decoding for reading registers
case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
2'h0 : reg_data_out <= slv_reg0;
2'h1 : reg_data_out <= slv_reg1;
2'h 2 : reg_data_out <= slv_reg2;
2'h3 : reg_data_out <= operation_out;
default : reg_data_out <= 0;
endcase
end
```

At the file save an message say that the adder\_0 instance was added in the interface S00\_AXI.



Adder implementation is over and we can go back to the IP package tool to finish the customization of the new IP.

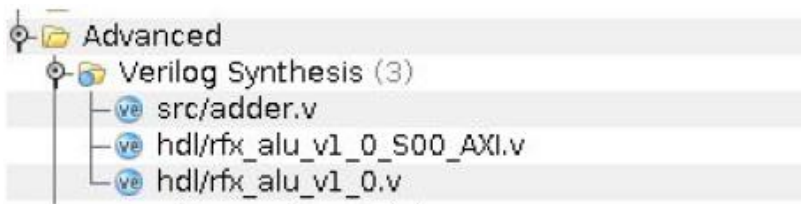
Click on "File group"



And so on the link:



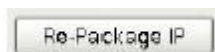
Then verify the correctness of the path relative to all the sources, as in the figure.



Finish the IP clicking "Reviews and Package":



Then execute the IP Re-Package:



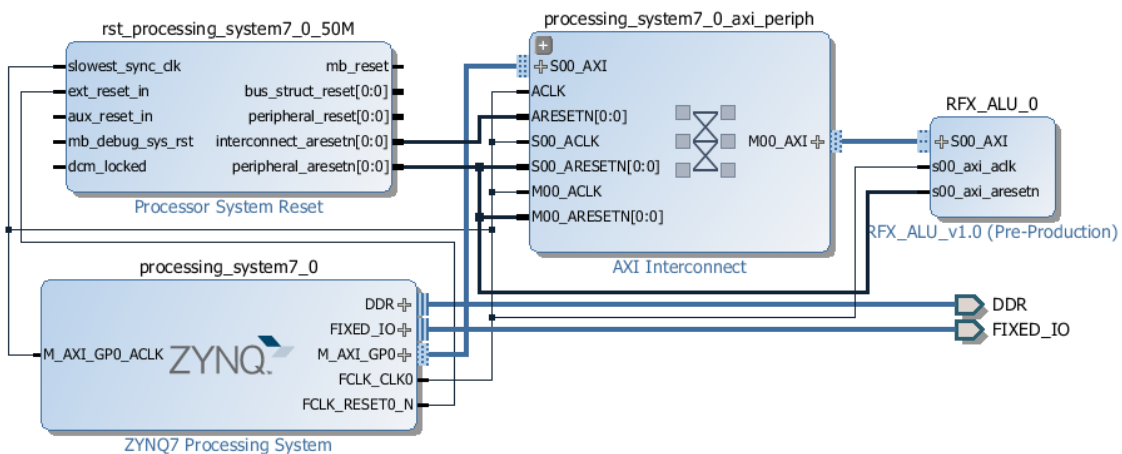
## Step 7 – Add a new custom IP to the design

Previous operations have set a new ip\_repo in the catalog so we can proceed with "add IP" and select our rfx\_alu.



Act on "Run Connection Automation" to obtain what is shown in FIG.

The tool automatically adds all minimally necessary for IP AXI interface to interconnect the SP to the AXI bus.



On the addresses editor of the final design, displayed above, is shown the allocated memory mapping by Vivado for the new device.

Cell	Slave Interface	Base Name	Offset Address	Ran...	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [ 1G ])					
rfx_alu_0	S00_AXI	S00_AXI_reg	0x43C0_0000	64K	0x43C0_FFFF

The assigned address is 0x43C00000.



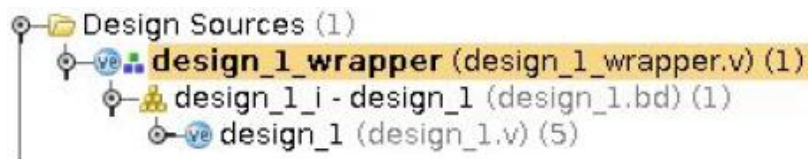
## Step 8 – TOP module creation and start synthesis

Vivado need for a top-level module as the root of the implemented logic and will want it in Verilog or VHDL version script.

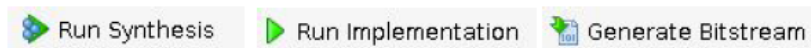
It will also be able to create a wrapper for design in VHDL block.

On design 1 perform a right click on Create HDL wrapper and create a manipulated wrapper.

The source code will look like this:



System is now ready to generate the bitstream file.



The generated bitstream file will be available in the path:

*project\_alu/project\_alu.runs/impl\_1/design\_1\_wrapper.bit*

## Step 9 – Manually creating a test application

In order to run the new application to RedPitaya board, without having to recompile the entire system image, it is necessary that the new application meets the specifications of the existing "ABI" precompiled libraries, as well as executions in RedPitaya.

It's necessary to use the Linaro toolchain.

```
TOOLCHAIN= "http://releases.linaro.org/14.11/components/toolchain/binaries/arm-linux-gnueabi/gcc-linaro-4.9-2014.11-x86_64_arm-linux-gnueabi.tar.xz"
curl -O $TOOLCHAIN
sudo mkdir -p /opt/linaro
sudo chown $USER:$USER /opt/linaro
tar -xpf *linaro*.tar.xz -C/opt/linaro
```

Proceed with the compilation of the test application using/dev/mem

In order to map it to the correct address.

Here's the code:

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <stdio.h>
#include <math.h>
#define RP_OK 0;
#define RP_EMMD 1;
#define RP_EUMD 1;
#define RP_ECMD 1;
#define RP_EOMD 1;
static int fd = NULL ;
int cmn_Init(){
    if (!fd) {
        if ((fd = open( "/dev/mem" , O_RDWR | O_SYNC )) == 1)
            { return RP_EOMD ; }
    }
    return RP_OK ;
}

int cmn_Release( ){
    if (fd) {
        if (close(fd) < 0) {
```

```

    return RP_ECMD ;
}
}
return RP_OK ;
}

int cmn_Map( size_t size, size_t offset, void ** mapped){
    if (fd == 1){
        return RP_ECMD ;
    }

    *mapped = mmap( NULL , size, PROT_READ | PROT_WRITE , MAP_SHARED , fd, offset);
    if (mapped == ( void *) 1){
        return RP_ECMD ;
    }
    return RP_OK ;
}

int main( int argc, char **argv) {
    printf( "Adder test \n" );
    if (argc<2) {
        printf( "usage: %s a b\n" ,argv[0]);
        return 1;
    }
    int *addr;
    cmn_Init();
    cmn_Map(16, 0x43c00000,( void **)&addr);
    *(addr+1) = atoi(argv[1]);
    *(addr+2) = atoi(argv[2]);
    printf( " result=%d \n" ,*(addr+3));
    cmn_Release();
    return 0;
}

```

Proceed by completing the code using the Linaro gcc compiler.

The console commands are:

```
/opt/linaro/gcc-linaro-4.9-2014.11x86_64_arm-linux-gnueabi/bin/arm-linux-gnueabi-gcc main.c -o adder
```

## Step 10 – Deploy on RedPitaya and test program execution

The term "Deploy" mean downloading the application within the embedded device in which we want to execute the developed design, in this case, on the RedPitaya board.

The procedure will have a strong resemblance to the one used to download the Linaro applications in ARM side, but in this case reconfigure the FPGA section, in a surprisingly short time.

To copy the generated .bits code within the RedPitaya perform from linux console:

```
scp project_alu.runs/impl_1/design_1_wrapper.bit root@<ip address>:/tmp
```

Substitute in place of the item <ip address> the TCP-IP address assigned to the card by your LAN by DHCP or by manually in case of fixed address. In the case of tests in card inserted in the local network of the CNR of Padova dell'istituto ionized gas, used to develop these thesis, the address is 192.168.62.13

For deploy the file created and compiled in the Red Pitaya:

```
scp adder root@<ip address>:/tmp
```

To program the FPGA use the device **xdevcfg** as shown below.

The created design expects the input parameters in order to execute the print sum to the screen:

```
Ssh root@<ip address>
```

```
redpitaya> cd /tmp
```

```
redpitaya> cat design_1_wrapper.bit /dev/xdevcfg
```

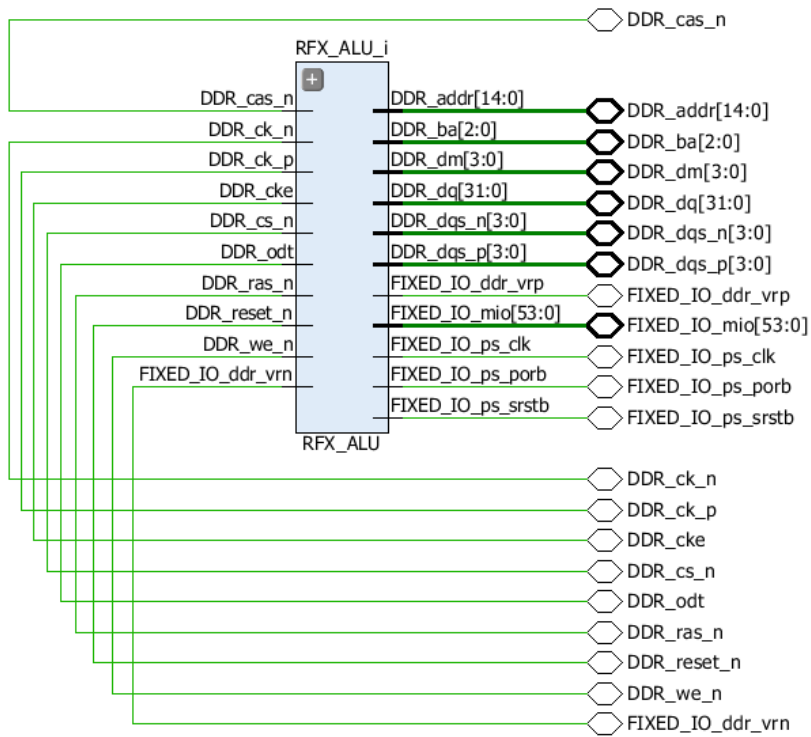
```
redpitaya> ./adder 1 2
```

```
Adder test
```

```
result=3
```

These 10 steps are a real and complete example of programming of the FPGA section of the Zynq chip 7xxx RedPitaya the family, but we could also use it to develop our Board based on this design.

Aspect of the RTL analysis of RFX\_ALU IP module.





# C. Appendix: FPGA to ARM interfacing

## The AXI write tool

Communication between the PS and PL is possible via the AXI write tool that allows the configuration of the 4 registers that parameterize the applications implemented in the FPGA. The allocation point in the program memory can also be set and allows execution of the application.

```
//AXI lite Write (Redpitaya side)

#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>

#define RP_OK 0;
#define RP_EMMD 1;
#define RP_EUMD 1;
#define RP_ECMD 1;
#define RP_EOMD 1;

static int fd = 0;

int Init()
{
    if (!fd) {
        if((fd = open("/dev/mem", O_RDWR | O_SYNC)) == -1) { return RP_EOMD; }
    }
    return RP_OK;
}

int Release()
{
    if (fd) {
        if(close(fd) < 0) {
            return RP_ECMD;
        }
    }
    return RP_OK;
}

void * Map(size_t size, size_t offset)
{
    static void* mapaddr = NULL;
    if(fd == -1) { return NULL; }
    if(!mapaddr)
        mapaddr = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, offset);
    if(!mapaddr) { return NULL; }
    return mapaddr;
}

int main(int argc, char **argv) {
    printf("Axi write test \n");

    if(argc<3) {
        printf("usage: %s address size [d1 d2 ... ]\n",argv[0]);
        return 1;
    }
}
```

```

}

size_t map_addr = strtol(argv[1], NULL, 16);
size_t map_size = atoi(argv[2]);
int i;
printf("writing on address %d with size %d\n", map_addr, map_size);
// int data[20];
// for(i=3; i<argc; ++i) {
//   data[i-3] = atoi(argv[i]);
// }

Init();
int *addr = Map(map_size, map_addr);

for(i=0; i<argc-3; ++i) {
    *(addr+i) = atoi(argv[i+3]);
}
for(i=0; i<argc-3; ++i) {
    printf("data[%d] = %d\n", i, *(addr+i));
}

Release();
return 0;
}

```



# References

- [1] <http://www.gtronic.it/test/index.php/fpga-high-speed-isolated-adc-redpitaya-shield/>
- [2] [http://www.gtronic.it/test/index.php/fpga-high-speed-isolated-adc-redpitaya-shield/frontend\\_pitaya\\_resistors\\_interface\\_and\\_gerber](http://www.gtronic.it/test/index.php/fpga-high-speed-isolated-adc-redpitaya-shield/frontend_pitaya_resistors_interface_and_gerber)
- [3] [frontend\\_pitaya\\_resistors\\_interface\\_and\\_gerber](http://www.gtronic.it/test/index.php/fpga-high-speed-isolated-adc-redpitaya-shield/frontend_pitaya_resistors_interface_and_gerber)
- [4] <http://www.gtronic.it/test/index.php/fpga-high-speed-isolated-adc-redpitaya-shield/>
- [5] [https://www.xilinx.com/support/documentation/ip\\_documentation/axi\\_fifo\\_mm\\_s/v4\\_1/pg080-axi-fifo-mm-s.pdf](https://www.xilinx.com/support/documentation/ip_documentation/axi_fifo_mm_s/v4_1/pg080-axi-fifo-mm-s.pdf)
- [6] Sonato P. et al., Fusion Eng. Des. **66–68**, 161-168, (2003)
- [7] P. Piovesan et al., Phys. Plasm. **20**, 056112 (2013)
- [8] Peruzzo, S., et al. "Design concepts of machine upgrades for the RFX-mod experiment." Fus. Eng. and Design (2017).
- [9] Marchiori, G., et al., Fusion Eng Des. **123**, 892-896, (2017).
- [10] N. Pomaro, F. Basso, Fusion Eng Des., **74**, 721-726 (2005)  
<https://doi.org/10.1016/j.fusengdes.2005.06.242>.
- [11] Gottardo M. First step on FPGA Xilinx. Introduzione alla progettazione dei sistemi SoC. ISBN: 9781326806064, (2016)
- [12] T. G. M. Kleinpenning and A. H. de Kuijper, "Relation between variance and sample duration of 1/f noise signals" , Eindhoven University of Tecnology, Eindhoven, The Netherlands.
- [13] Scoville J.T. et al, Nucl. Fusion **31** 875 (1991)
- [14] J. Wesson, "Tokamaks", Third edition, Clarendon Press Oxford, 2004
- [15] B. Carvalho et al., Fusion Eng. Des. **85**, 298-302 (2010).
- [16] B. Carvalho et al, JET Report EFDA–JET–CP(09)04/10 (2010).
- [17] G. Manduchi, et al. Fusion Eng. Des. **87**, 1907-1911, (2012).
- [18] M. Zuin, et al., Plasma Physics and Controlled Fusion **51**, 035012, (2009)
- [19] P. Innocente et al., Nuclear Fusion **54**, 122001 (2014).
- [20] <http://golem.fjfi.cvut.cz/wiki/Diagnostics/ParticleFlux/TripleProbe/Campaigns/1012MarcoM/files/TLP%20GOLEM.pdf>
- [21] [https://en.wikipedia.org/wiki/Langmuir\\_probe#Triple\\_probe](https://en.wikipedia.org/wiki/Langmuir_probe#Triple_probe)
- [22] [A.J.N. Batista et al., Fusion Eng. Des. \*\*123\*\*, 1025-1028, \(2017\)](http://www.gtronic.it/test/index.php/fpga-high-speed-isolated-adc-redpitaya-shield/frontend_pitaya_resistors_interface_and_gerber)
- [23] [Andreas Werner, Review of Scientific Instruments \*\*77\*\*, 10E307 \(2006\)](http://www.gtronic.it/test/index.php/fpga-high-speed-isolated-adc-redpitaya-shield/)



# Acknowledgements

I gratefully acknowledge the staff of Consorzio RFX, in particular Dr. Roberto Cavazzana and ing. Andrea Rigoni.

*Marco Gottardo*

Padova, 06/10/2017