Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2012

Daniel Filipe Simões Malafaia

Frontend em tempo real para Cognitive Radio inspirado na Cóclea Humana

Real Time Front-end for Cognitive Radio Inspired by the Human Cochlea

"Any radio with the capacity to jump around the spectrum optimising for power, range and required data rates, will, at the very least, require an extremely flexible RF front end."

— John Walko

Daniel Filipe Simões Malafaia

Frontend em tempo real para Cognitive Radio inspirado na Cóclea Humana

Real Time Front-end for Cognitive Radio Inspired by the Human Cochlea

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica de José Manuel Neto Vieira, Professor do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president

Professor Doutor Tomás António Mendes Oliveira e Silva
Professor Associado da Universidade de Aveiro (por delegação da Reitora da Universidade de Aveiro)

vogais / examiners committee

Professor Doutor Roberto Gómez Garcia
Professor Associado, Universidad de Alcalá

Professor Doutor José Manuel Neto Vieira
Professor Auxiliar, Universidade de Aveiro (orientador)

Professor Doutor Nuno Miguel Gonçalves Borges de Carvalho
Professor Associado com Agregação, Universidade de Aveiro (co-orientador)

agradecimentos / acknowledgements

O autor deseja agradecer ao orientador José Neto Vieira, que sempre prestou toda a ajuda necessária para o avanço e conclusão desta tese. Ao co-orientador Nuno Borges por todas as suas informações e disponibilidade prestada. Ao José Pedro Magalhães que não só criou alicerces para tornar esta tese possível, como trabalhou comigo para o progresso da mesma. Ao Teófilo Monteiro que também teve um papel importante neste projecto. Diogo Ribeiro, que me ajudou com os equipamentos do laboratório de RF. Ao Arnaldo Oliveira e Nelson Silva por toda a ajuda prestada na área da computação reconfigurável. Ao Hugo Mostardinha por todo o apoio no laboratório de RF. Aos meus amigos Pedro Costa e Bruno Pereira que me acompanharam desde o início do curso até à conclusão da tese. Aos meus pais e irmão que me têm acompanhado desde sempre, aos meus primos que espero que tenham a hipótese de também agradecer aos familiares quando acabarem a sua tese, e ao resto da minha família, tias, tio e aos meus avós que me criaram. A todos os outros que não couberam neste agradecimento mas que de alguma forma me ajudaram, o meu obrigado.

Palavras-chave

Bancos de Filtros Híbridos, Rádio Cognitivo, Rádio Definido por Software, FPGA, Engenharia Reversa, Medições em RF, Processamento de Sinal Digital em Tempo Real

Resumo

Nesta tese vamos discutir a implementação e desenvolvimento de um front-end inspirado na cóclea humana que é capaz de amostrar sinais RF com uma larga largura de banda e gama dinâmica. Este front-end usa um multiplexer de RF de 8 canais amostrado por uma placa com 8 ADCs a funcionar a 250MSPS. Uma placa de desenvolvimento com uma FPGA controla a ADC e implementa os filtros de síntese digitais e liga a um computador pessoal para transferir toda a informação e mudar os coeficientes dos filtros em tempo real.

Keywords Hybrid Filter Banks, Cognitive Radio, Software Defined Radio, FPGA, Reverse Engineering, RF Measurements, Real-time Digital Signal Processing

Abstract In this thesis it will be discussed the real time implementation and development of a front-end inspired by the Human Cochlea that is able to sample RF signals with a large bandwidth and dynamic range. This front-end uses an 8 channel RF multiplexer sampled by an 8 channel 250MSPS ADC board. A FPGA board controls the ADC, implements the digital synthesis filter bank and connects to a personal computer to transfer the data and to change the filters in real-time.

Contents

Nomenclature

ADC or A/D	Analog-to-Digital Converter
ASIC	Application Specic Integrated Circuit
DAC or D/A	Digital-to-Analog Converter
dB	Decibel
DDR	Double data rate
DSP	Digital Signal Processing
ENOB	Effective Number Of Bits
FFT	fast Fourier transform
FIFO	First In, First Out
FIR	Finite impulse response
FMC	FPGA Mezzanine Card
FPGA	Field-Programmable Gate Array
FSM	Finite-State Machine
GPIB	General Purpose Interface Bus
GPS	Global Positioning System
HFB	Hybrid Filter Bank
Hz	Hertz
IIR	Infinite impulse response
LUT	Lookup table
MAC	Media Access Control
MATLAB	MATrix LABoratory
PC	Personal Computer
PLL	Phase lock loop

PNA	general-Purpose Network Analysis
RF	Radio Frequency
SDR	Software Defined Radio
SINAD	Signal-to-Noise-and-Distortion
SMA	SubMiniature version A
SNR	Signal-to-Noise Ratio
SPI	Serial Peripheral Interface Bus
SPS	Samples Per Second
txt	Text file
UART	Universal asynchronous receiver/transmitter
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VNA	Vector network analyzer

Part I

Introduction

Chapter 1

Radio - State of the art

1.1 The radio technology

The radio technology is all the technology used for wireless communication based on transmission and reception of an electromagnetic signal, which may be broadcasted, for example the transmission of television signals and GPS, or point-to-point such as WiFi or Bluetooth.

In recent years there has been a rapid spread and evolution of this technology caused by a continuous demand for higher data transfer rates, reliability, service quality, autonomy and mobility of the equipment. With the evolution of photolithography it has been possible to develop radio equipment with low dimensions and reasonable autonomy.

However, due to the great diversity of communication standards and protocols that operate on different radio bands caused the integration in silicon to be complex, and often an individual integrated circuit is used, independently, for each desired communication standard. So, instead of a device having only a universal radio system, it's common to have a large number of front-ends, which compromises the autonomy, reliability and cost of the device.

The incorporation of a large number of radio front-ends raises serious problems to the manufacturers. The most complex being the interference between the radios within the device or with the exterior. To minimize this interference, it's common to build analog filters that are constructed so that the radio signal fulfils a given spectral mask. The development of these filters is however complex and expensive, especially if it is dimensioned for high frequencies of operation.

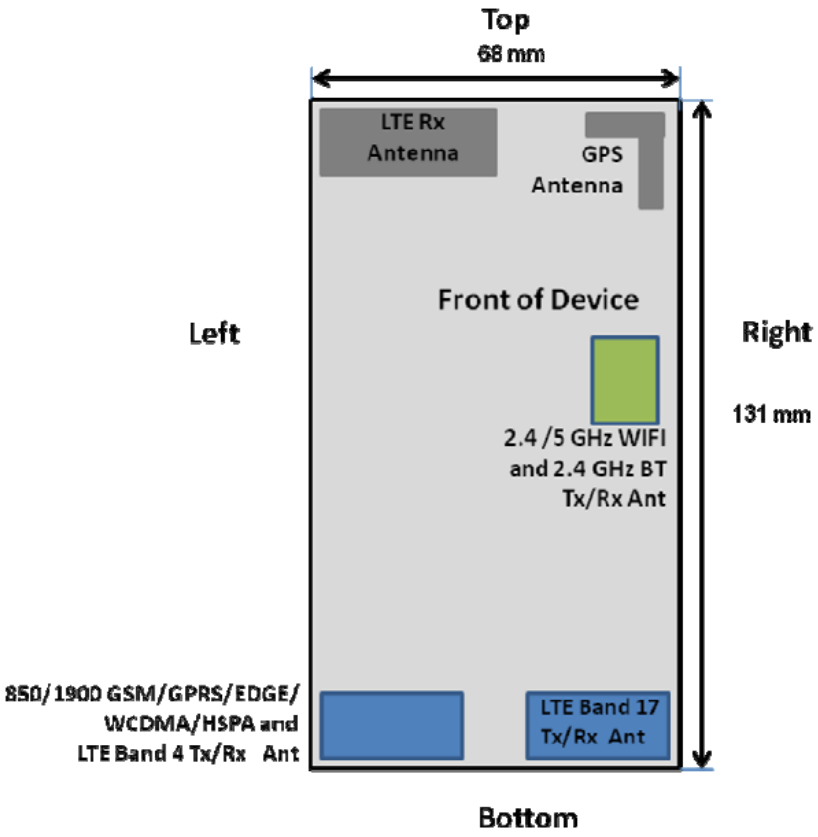


Figure 1.1: A common smartphone today with five different antennas and respective radios (Galaxy 2 LTE)[1]

Another problem is the non-linearity of various electronic components that are used in the radio. These components affect the original signal, and create once again the need to increase the complexity of the radio in order to offset all of these non-linearities.

Thus, the evolution of wireless communications technology will necessarily increase in the digital domain of the radio, as has been taking place in other areas of mobile technology. This evolution will allow to overcome problems such as the quality factor of the analog filters and the creation of phase or heat noise by electronic components used in the analog domain.

1.2 Software Defined Radio

The Software-Defined Radio was first introduced by Joseph Mitola as a concept of the future of the radio [2].

To Mitola, a radio would be able to modulate and demodulate any signal entirely in software, regardless of the standard, operating frequency or bandwidth of the signal. The transmitter had, in this vision, the ability to create any waveform, and translates it to any frequency to be later converted to the analog domain, by an DAC, and then amplified and sent. On the other hand the receiver should collect a certain bandwidth of the spectrum, amplify it and convert it to the digital domain, with an ADC. After that conversion the process of detection and demodulation of the signal would be carried by software.

If we assume that the resources needed to implement the Software-Defined Radio are able to work at a satisfactory bandwidth, then we can create a multi-band operation mobile radio with lower production cost, ability of adapting to new techologies and performance optimization[3]. Besides this, an SDR provides an universal front-end, i.e. the same front-end can be used for various standards and can operate in different frequency bands or with different protocols. Moreover, unlike traditional approaches, this radio enables simultaneous reception of several channels. Being another benefit of this type of radio the ability to add any new functionality with a simple software update.

Despite the enormous potential that this concept has, in reality there are some technological problems that affect this implementation:

- The antennas need to have a high bandwidth and low losses;

- The amplifiers must have wide linear dynamic range of operation and low power consumption;

- The A/D and D/A must have a high sample rate, high resolution and dynamic range, high bandwidth, low cost and low power consumption;

- The digital processing units (DSP) must be capable of operating at high speeds to process the high bandwidth signals in real time.

There are already in the market devices that are named as "Software-Defined Radios", such as the USRP[4] and WARP [5]. These systems are primarily directed to research and enable bandwidths of about 100MHz. In its receiver architecture there is a demodulator that translates the signal to an intermediate frequency and then the resulting signal is sampled using two ADCs with a sampling frequency of about 160MSamples per second and 16-bit resolution. The processing is performed with a combination of DSPs and a FPGA, thus providing the possibility of reprogramming and reconfigure the system.

1.3 Cognitive Radio

Traditional wireless networks work in a license-free spectrum, with is called the ISM bands, the ISM bands are parts of the radio spectrum used for personal, industrial, scientific and medical uses. As these bands became crowded with all the new uses, as for example WiFi and Bluetooth, the need for a new system is becoming increasingly important. The expected solution will be the Cognitive Radio.

Cognitive radio is a system based on a Dynamic spectrum access (DSA) from a secondary and unlicensed user. The DSA works by opportunistically accessing a licensed spectrum band owned by the so called "primary users", this access, nevertheless, is subject to some regulations. The usage of this method would be expected to alleviate the current spectrum usage.

Cognitive Radios work by sensing the licensed bands and detecting the occupation of those. If one of those spectrum bands are temporary free then secondary users will occupy it.

However the practical implementation of such devices is surrounded by several challenges both in software and hardware. First, there's a need of synchronization, when two communicating devices decide to change band, they must synchronize with each other to resume the communication. Therefore, there is a need to create protocols to allow a share of information about the available frequencies list in each location of the devices, bandwidth and type of modulation that is going to be used.

In the Hardware there's necessary to sense the entire viable spectrum and detect witch bands are being already used. If the band is not being currently used, the CR should be able

to change the frequency of the transmission to allocate it into the available band. It should also detect if any of the primary users want to start to use their rightful band and then stop the transmission.

1.4 Analog to Digital Converters

The A/D converter is a key component of a Software-Defined Radio and also one of the key elements that hider to implement this type of architecture.

One of the desirable properties of an ADC is the possibility to sample a large frequency spectrum. However, for large bandwidths of operation, in the order of GHz, with the present technology we start to have a low ENOB. The decreasing dynamic range for these bandwidths would create quantization errors that will limit the reception in this type of radio.

Thus the conversion between the analog domain and digital is technological limited by the ADC. It would be desirable that they had:

- A high number of quantization bits in order to guarantee a high dynamic range;
- A wide dynamic range so that signals with high power will not cause other information with lower values to be lost;
- A large sampling frequency to cover a large bandwidth;
- Reduced price and power consumption.

It's not expected that these needs are met in the coming years, so need to find alternatives.

Currently the best ADC in the commercial market is the ADC12D1800 [6] from Texas Instruments with 3.6GSPS and 12bit with an ENOB of 9.4bits and a SNR of 58.5dB, and typically consumes 4.4W.

1.5 Author publications

As the result of the work done by the author to this thesis the following publication in a conference was approved and it will be published in:

1. D. Malafaia, J. Magalhães, J. Vieira "Real Time Front-end for Cognitive Radio Inspired by the Human Cochlea" RWW, Texas, 2013

Part II

Bio-Inspired front-end

Chapter 2

Cochlear radio

2.1 Human Auditory System

Nature have found a way to create a sampling system that can sample a large bandwidth without high power signals effecting low power ones thus maintaining the dynamic range of the system[7].

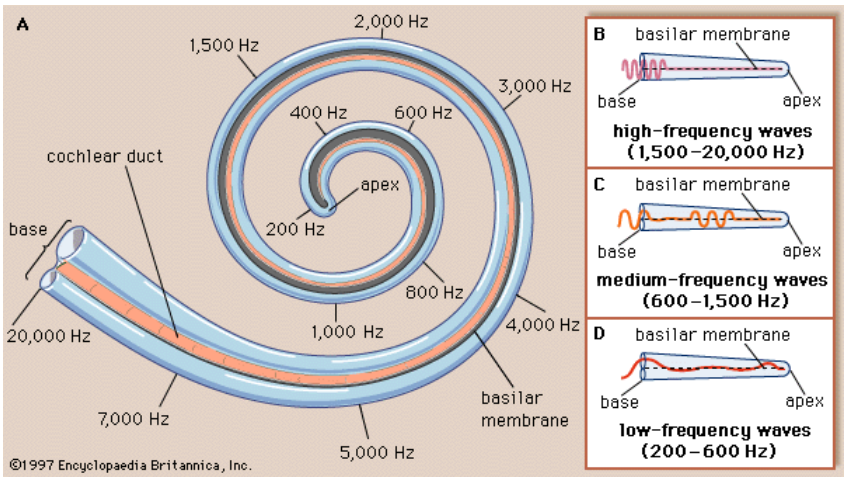


Figure 2.1: Analysis of sound frequency in the cochlea[8]

This was achieved through evolution of the auditory system of mammals. Humans can cope with a dynamic range of 120dB and several octaves of bandwidth.

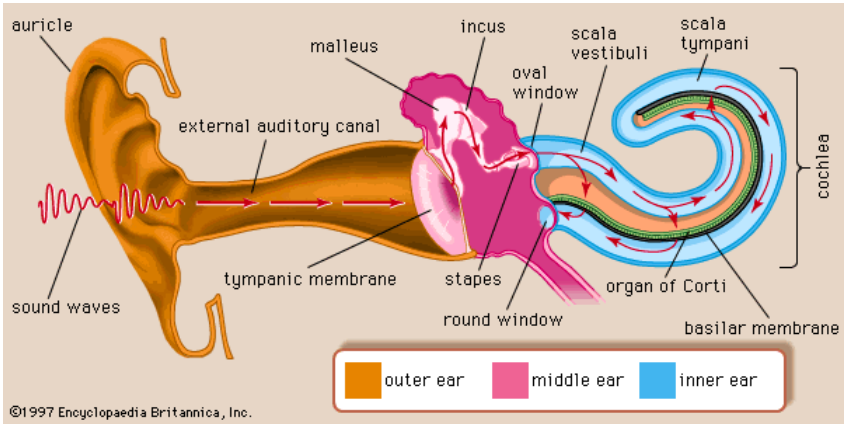


Figure 2.2: Human hearing mechanism[9]

As we can see at figure 2.2 it works as follows: the sound enters in the ear auricle and is guided by the external auditory canal to the tympanic membrane. Struck by the sound-waves, the tympanic membrane oscillates and make the ossicular chain (malleus, incus and stapes bones) reproduce and amplifies the oscillation into the inner ear and later enter in a curled tube located at the inner ear the cochlea. This is important because it provides the impedance matching between the outer ear and the inner ear. The cochlea then contains several membranous sections, which are filled with watery fluids. The ossicular chain vibrate the fluid, thus making nerve terminations, hair cells, in the basiliar membrane to sense these mechanical vibrations and convert them into electrical pulses. The hair cells convert the signal with a low sampling frequency and a low equivalent number of bits, so this is where the cochlea shows its fundamental importance by dividing the input signal in bands so that a single hair cell will have only a small band to convert, achieving then the incredible performance of the human hearing.

Later the electrical pulses are then sent via the cochlear nerve and travel then trough the nervous system reaching the primary auditory cortex in the brain where they are processed[10].

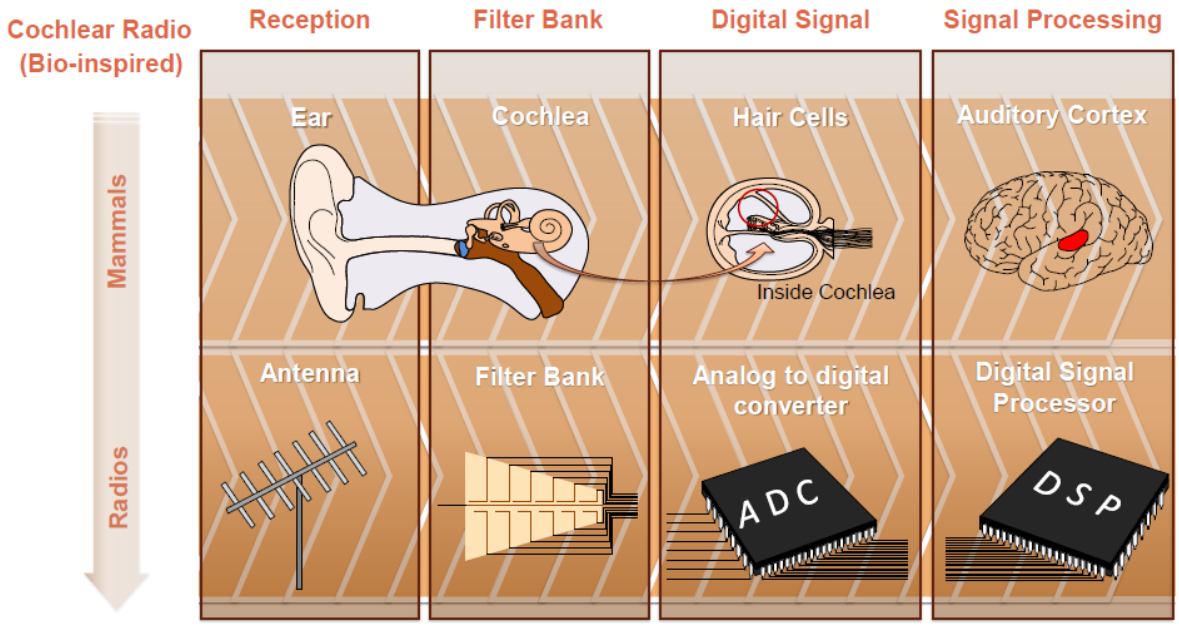


Figure 2.3: Analogy between the auditory system and RF[11]

Creating an analogy to RF, as show in figure 2.3 we can say that the ear auricle works as an antenna, the external auditory canal as a waveguide, the cochlea as an filter bank, the nerve terminations as mid-end ADCs, the nervous system as a data-bus and the primary auditory cortex is then a DSP.

2.2 Proposal for an SDR front-end based on hybrid filter banks

2.2.1 Front-End proposal

Using the mammals auditory system as a guideline we can create an alternative for the radio front-ends used nowadays and create a radio for Cognitive Radio that could achieve the requirements for the ideal Software Defined Radio by ensuring a high-bandwidth operation while maintain a high dynamic-range. The proposal is the following:

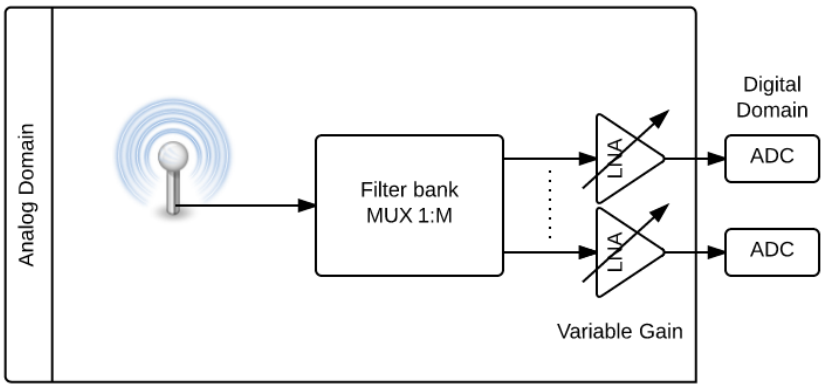


Figure 2.4: Front end proposal

The RF signal coming from the antenna is divided into M contiguous frequency bands, through a bank of analog high-frequency filters. These bands are passed to an intermediate frequency and are then converted to the digital domain with an ADC for each channel. As the spectrum of the input signal is divided into M signals with a fraction M of the bandwidth, then the sampling rate of the ADC becomes M times lower. Thus it's possible to obtain an even better negotiation with the conversion resolution, which can be increased by $\sqrt{M}$ bits compared to conventional receivers.

This approach has the following advantages:

- The smaller bandwidth of the M divided signals enables the use of an M time lower sampling frequency for each ADC;

- The need for a lower sampling rate of the ADC relaxes the requirements, allowing a better engagement with the resolution of the conversion, making it possible to improve the it in $\sqrt{M}$;

- A higher resolution conversion will lower quantization noise improving the SNR of the system;

- The fact that the bandwidth of the input signal for each ADC is lower will result in less power consumption;

- The reduced bandwidth of each channel also enables to relax important factors of the amplifier such as the non-linearity;

- Since we are converting small parts of the spectrum in each ADC we will have less problems with jamming signals, as we can see in figure 2.5.

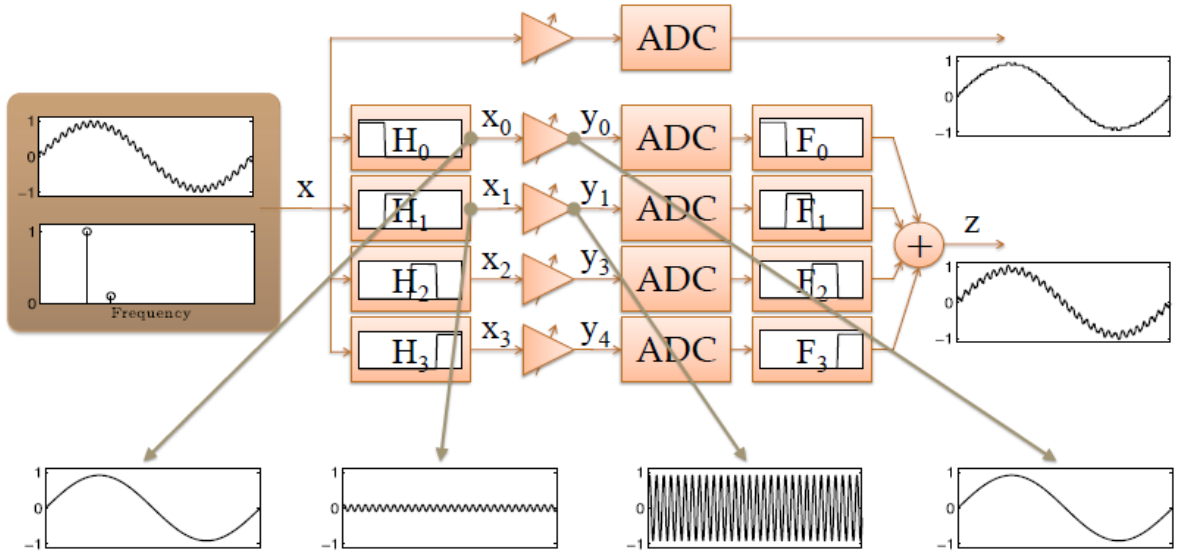


Figure 2.5: Blocking effect reduction using a Hybrid Filter Bank compared with just one ADC[11]

This method is also know by the name Hybrid Filter Bank, because we use analog filters to divide the signal and sample it and then the signal is reconstructed with digital filters, making it a analog/digital hybrid.

2.2.2 Hybrid Filter Banks

The hybrid filter banks are then one solution to avoid the limitations of the current A/D converters. These allow an improvement on the resolution of the conversion and its dynamic range. However, this approach creates a new problem, the need to ensure perfect reconstruction of the input signal. That is, it's intended that the signal is divided into several bands in analog filter banks and then can be reconstructed in the digital domain without any loss or distortion of information.

So, to archive a perfect reconstruction is necessary to very accurately know the characteristics of the analog filters that we are using in order to be able to generate the digital filters such that the transfer function of all the hybrid filter bank is only a delay. However, in practice the perfect reconstruction is impossible due to the nonideality of the elements in the analog filters so the practical objective is to obtain very approximate models of the reality in order to create a filter bank with almost perfect reconstruction.

2.3 Digital Signal Processing

The DSP unit is one of the most important components of any Software Defined Radio and especially to a cochlear radio because this component will be responsible for the reconstruction and processing of the signal received. The processor will receive the ADC data and reconstruct it to approximate the original signal and then use it for any function needed by the user. Another feature of any cognitive-defined radio is the reconfiguration, to allow the transfer of new software and to implement any standard in the receiver or any other new functions.

The processor can be a DSP, dedicated ASIC or an FPGA. A DSP can consume a lot of power and works typically with a serial processing. An ASIC is a very fast hardware that implements any function even parallel ones, and enables a cheap and low power solution for any radio. A FPGA can offer the same performance as an ASIC but usually consumes more power and is expensive, but is reconfigurable.

2.4 Reconfigurable Hardware

2.4.1 FPGA

Through the times many tools have been used to create programmable electronic logic SPLDs, PROMs and others.

One of the most recent tools for testing, prototype implementation and even as a final product is the FPGA (field programmable gate array). These circuits contain an array of a large amount of logic blocks that can be programmed to interconnect as desired by the user. Each of that block is usually constituted by LUTs, multiplexers, flip-flops (FF) and some other additional logic, and can perform simple tasks, from a simple NOT gate to a synchronous sequential state machine.

In the FPGA that was used in this thesis, a Virtex6[12], these blocks are called CLBs (configurable logic block), they are grouped in slices and they can be a common ordinary slice, as showed in figure 2.6, or a DSP slice, the DSP slice have another kinds of logic, a more specific logic for DSP operations.

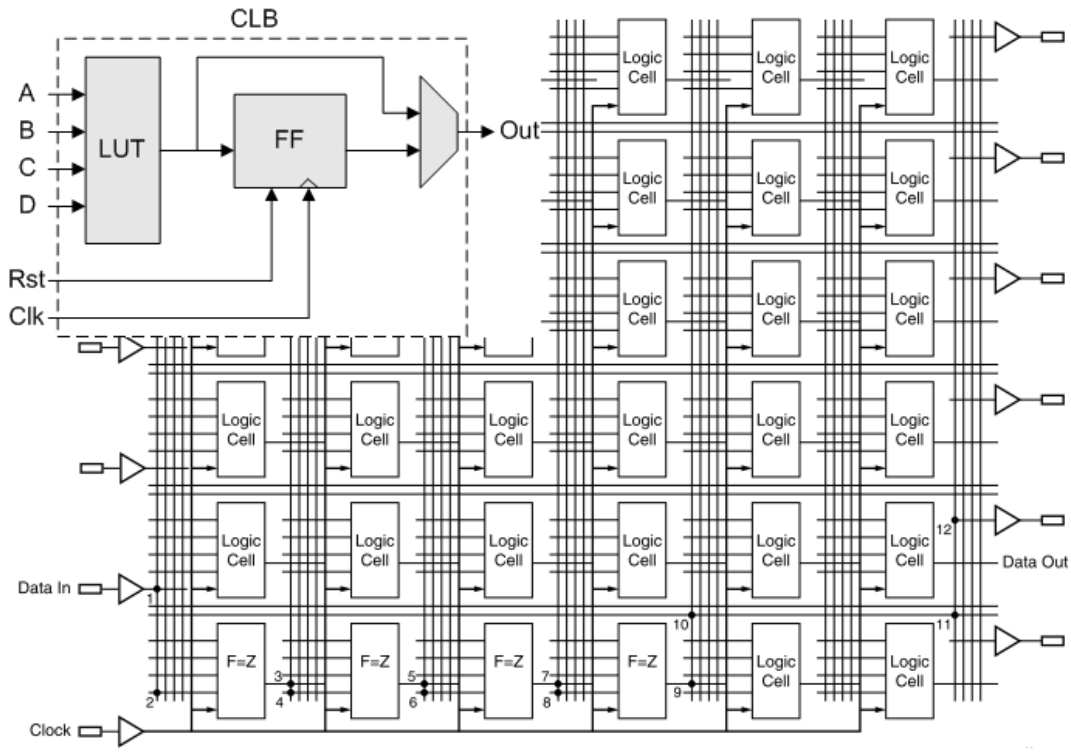


Figure 2.6: A slice is a array of Logic Cells, on the upper left we can see the common constitution of a logic cell in a FPGA[13]

In order to make the connections between CLBs flexible those are made with programmable interconnect fabric, leading to small frequencies of operation created by the large capacities from the multiple possible paths. But as we have access to a lot of available logic, we can create complex parallel processing than can allow a higher throughput than for example a pipeline processor.

2.4.2 VHDL

To take all the possibilities from a FPGA we require a way to "program" it to do any operations needed by the user, as in a processor a user does not program in machine code neither in a FPGA we connect directly each block, but a compiler is used for that. A circuit system is written by the user using hardware description languages (HDL), where the circuit is described as well as the inputs and outputs. HDLs were originally created for simulation and they are still used for such but it started to be used to synthesis real hardware as well. It is important to note that not every circuit designed as a behavioural simulation may work as a real circuit and the code can have instructions that may be implementable but not behave correctly in reality.

With this method of programming we allow parallelism by creating a large amount of independent process running at the same time, this is the true advantage of a FPGA over a hard-wired processor.

The industry mainly uses two types of HDLs, VHDL and Verilog. For this thesis VHDL was used to program the FPGA but some blocks ,not created by the author, are written in

Verilog. As we can interconnect very easily Verilog blocks to VHDL blocks this does not cause any trouble.

2.5 FML605 board based in Virtex6 XILINX FPGA

2.5.1 Integrated synthesis environment

The VHDL code needs to be translated to configuration data and then loaded into the FPGA. For this purpose we used the XILINX ISE 12.4/13.3[14] that will translate and synthesis the VHDL code into a bit-stream ready to transfer to the FPGA board memory, and then the XILINX IMPACT to load it on the FPGA.

When we write VHDL code in the ISE, it synthesis that code, generating the hardware and connections that implement the behaviour of the system created. Syntax errors are then detected at this point and must be corrected before moving anywhere further.

In addiction to the code written in VHDL, the user creates a constraint file. This constraint file has all timing requirements needed in the project and in/out connections of a VHDL port to a pin of the FPGA circuit package. The timing requirements permit to ensure that a certain signal or clock have a certain delay or can run at a certain frequency without any problems and with a certain tolerance.

The constraint file is used in the Implementation phase of the code, this step generates a netlist, from the previous generated hardware, that is mapped into the FPGA, allocating the resources in specific locations to permit the best routing and also providing the interconnections to that allocation. This creates a FPGA configuration file and a report of the used resources as for an estimation of the power consumption.

This configuration file is then programmed intro the FPGA and after conclusion the system it's ready to use.

2.5.2 ML605 board

For this thesis a ML605[15] evaluation board was used. This board contains a Virtex6 XC6VLX240T-1FFG1156 FPGA, and each slice of this FPGA is constituted by four six-input LUTs, multiplexers, four FlipFlop/Latches and four FlipFlops. The most important characteristics of this FPGA are the follow:

Logic cells	Slices	Distributed RAM (Kb)	DSP48E1	Block RAM blocks			Ethernet MACs
				18 Kb	36 Kb	Max (Kb)	
241.152	37.680	3.650	768	32	16	14.976	4

Table 2.1: Virtex6 XC6VLX240T FPGA key characteristics [16]

The evaluation board that we use have the following characteristics[17]:
- FPGA Virtex6 XC6VLX240T-1FFG1156;
- Configuration through USB2.0 or Flash memory;
- Power supplied through transformer;
- DDR3 (512MB) and Flash memories (34MB+16MB+8Mb);
- 200MHz differential oscillator and 16MHz socket clock;
- SMA sockets for external clock or user GPIO;

- LCD display, LEDs, Push-buttons and Switches;
- Communication through RS232, Ethernet, USB2.0, GTX and PCI Express.

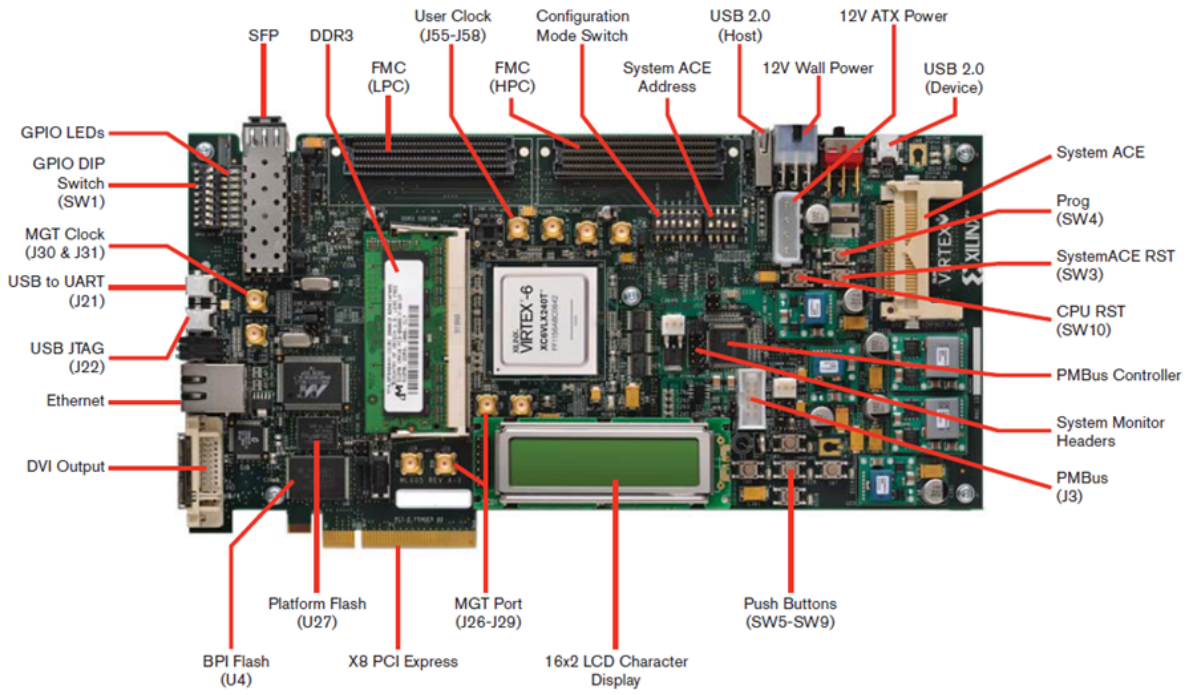


Figure 2.7: ML605 Dev. Board[18]

2.6 4DSP FMC108 8-ADC 250MSamples board

To perform the signal acquisition and convention to digital we used an ADC board from 4DSP[19]. 4DSP is a Nevada, USA based corporation that develop and manufacture commercial off the shelf data-conversion boards (ADCs and DACs) and respective FPGA Intellectual Property. Is also member of Xilinx Alliance Program, so the majority of the products are developed directly for Xilinx evaluation boards.

The specific board used is this thesis is a FMC108, a 4DSP board with 4ADCs, each of which has two channels, as each channel is being used individually it enables that 8 analog signals to be sampled at the same time. It also offers a connection to an external clock and another one for an external trigger.

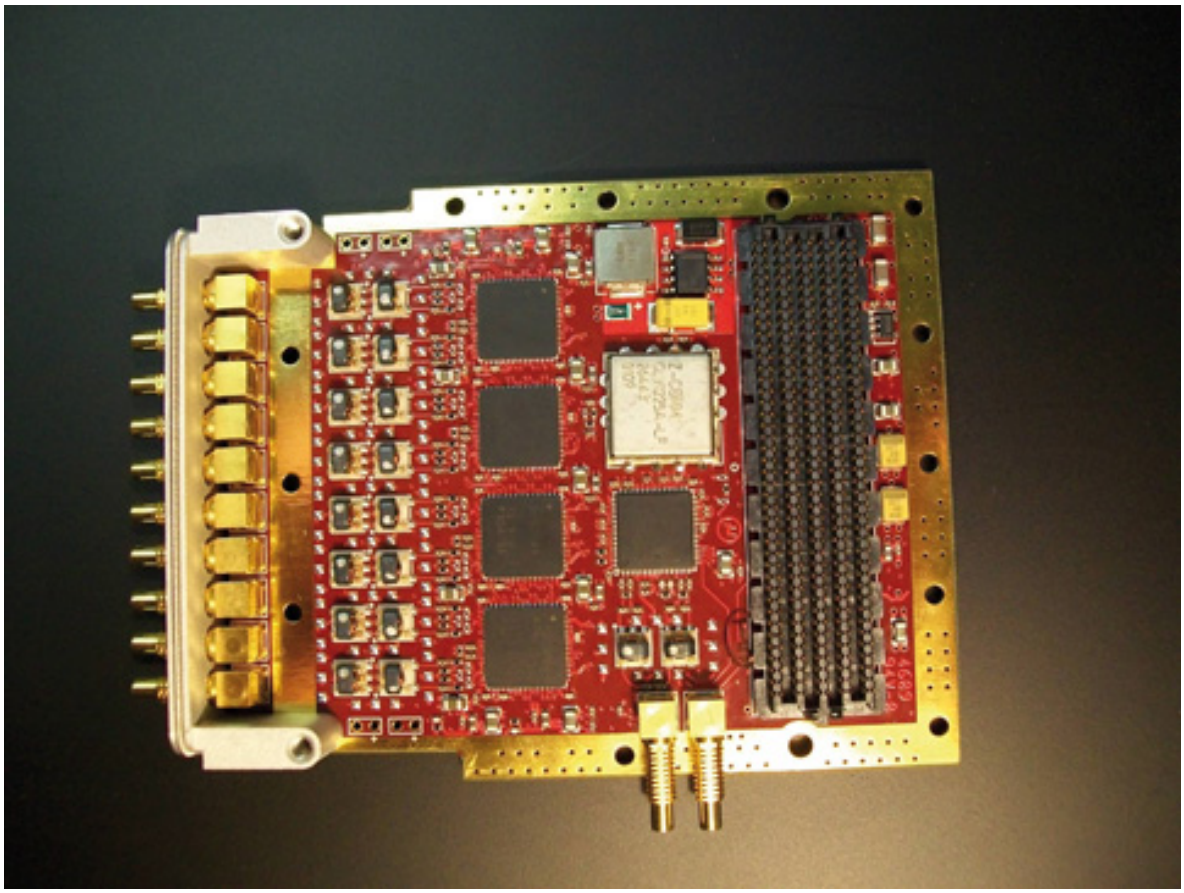


Figure 2.8: FMC108 board[20]

Each ADC channel has a resolution of 14-bit and a maximum sampling rate of 250Msps. The ADC clocks can be clocked by the internal clock source that offers some level of selectivity freedom of the clock frequency, but it can also be clocked externally by any frequency from 3 to 250MHz.

The ADCs used on the board are the TI's ADS62P49[21] dual channel 14-bit 250Msps ADCs with a DDR LVDS output. That also allows a programmable gain up to 6dB individually for each channel. The frequency of the analog input signals in this board must be between 3MHz to 650MHz as there is a AC coupling to get a better SNR from the ADCs, the input voltage can go as high as 2Vp-p (10dBm), and the input is tuned at 50ohm impedance.

A clock tree is used that offers some flexibility and high performance. A AD9510[22] PPL and clock distribution IC are the base of the clock tree and a VCO is connected to one of the clock inputs (the other clock input can be used by an external clock). An external reference 100MHz crystal is then connected to the VCO.

There's also on the board two ADT7411[23] devices which monitors the temperature of the ADCs and the clock input frequency for each ADC, each of these devices controls two ADC chips.

The input channels, clock and trigger are connected by a SSMC connectors in the front panel, these connectors are not a standard so SSMC to SMA cables were used to connect to other laboratory equipment.

As the temperatures of the board can reach 85^{o}C during use and to avoid damage a external fan was annexed to the FMC108 and is powered by the ML605, that had one extra fan power supply that was not at use. This alteration allows the FMC108 to run at no more than 50^{o}C.

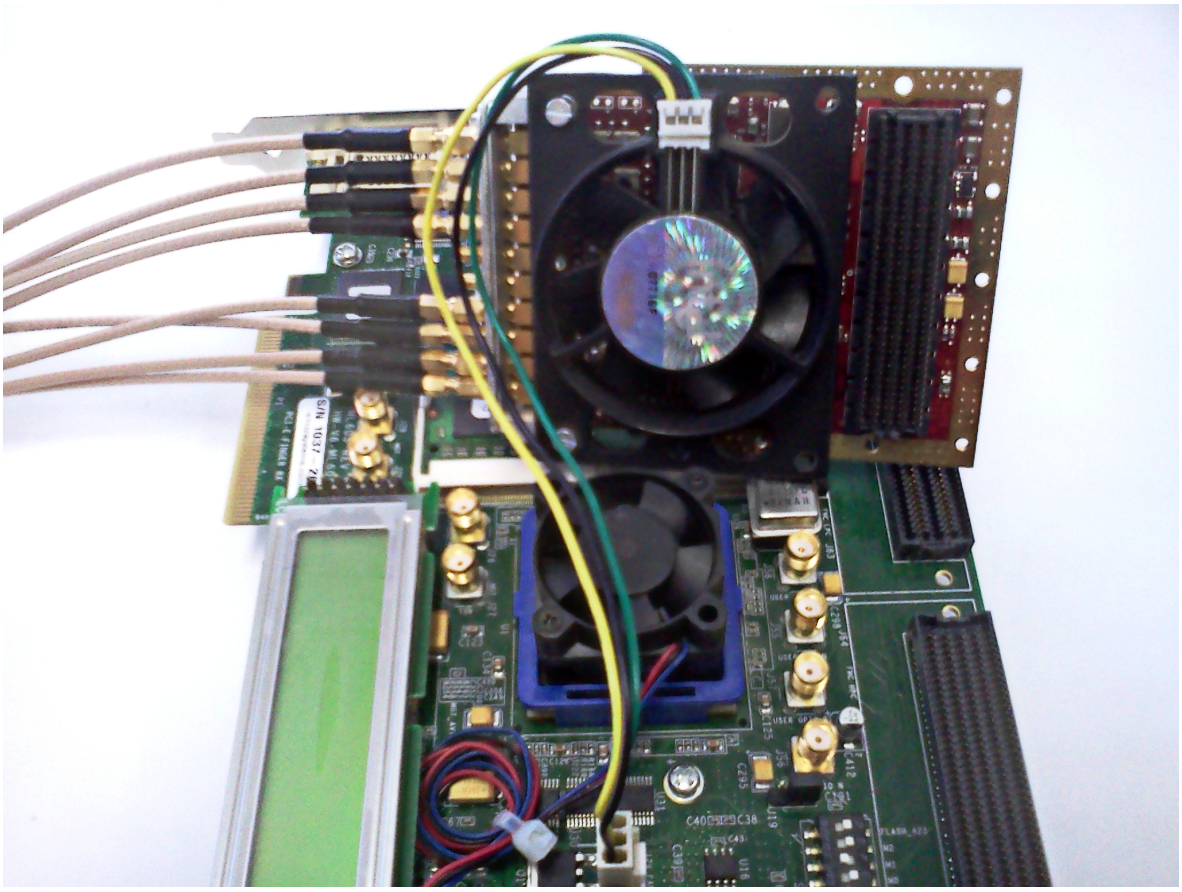


Figure 2.9: The external fan bolted to the FMC108

The board connects via FMC to the FPGA evaluation board. 4DSP provides with the board the respective firmware based in their high-level "Stellar architecture" written in VHDL that allows to sample each channel at a time.

2.7 The Cochlea analog filter bank

For this thesis a custom made filter bank was used, this filter bank was designed/built by José Pedro Magalhães[24]. It works by coupled filters, using discrete capacitors and coils into a RT/d5880 substrate with a $\varepsilon_r = 2.2$ and 1.58mm thick.

This filter bank have 8 channels adapted to 50Ω impedance, and projected to each filter to have a bandwidth of approximately 5MHz and be a filter of 3 order. So this will allow to the filter bank to be used as a multiplexer between 50 and 100MHz.

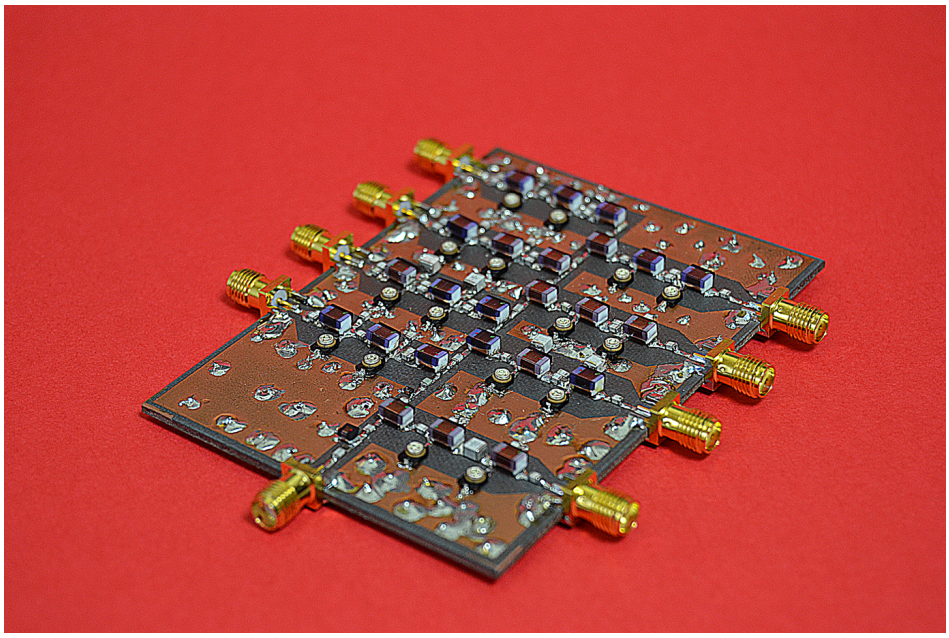


Figure 2.10: The RF multiplexer with 8 channels for frequencies between 50 and 100MHz

This filter have the following transmission coefficients:

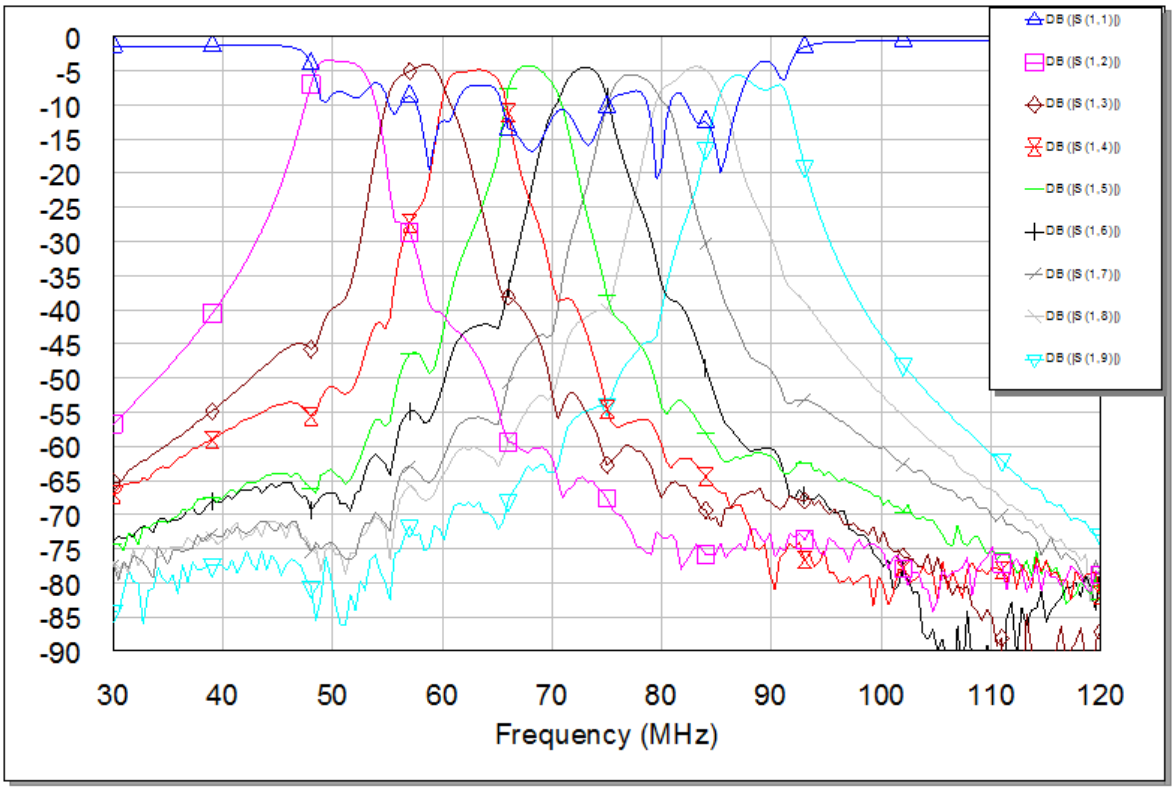


Figure 2.11: Transmission coefficients of the filter bank used

2.8 The behaviour of a reconstruction filter bank with subsampling

All the simulation process for this thesis was done using MATLAB[25] that is an IDE with its own high-level programming language.

For simplicity, we will study the case where $M = 2$ or when the hybrid filter bank has only 2 channels.

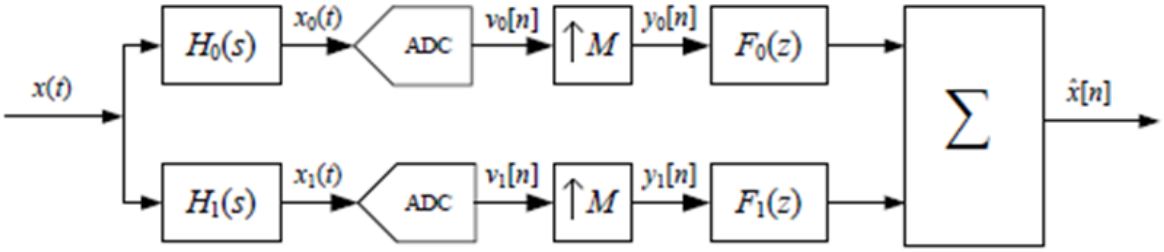


Figure 2.12: Hybrid Filter Bank

In this case the analog signal is filtered by the analysis filter bank in two frequency bands which are then passed to the digital domain by a sub-sampling ADC. The signal is then interpolated by a factor equal to the number of bands and then filtered by the synthesis filter bank. After this, the 2 signals are then summed giving rise to the reconstructed signal.

The previous system is equivalent to the following all digital filter bank:

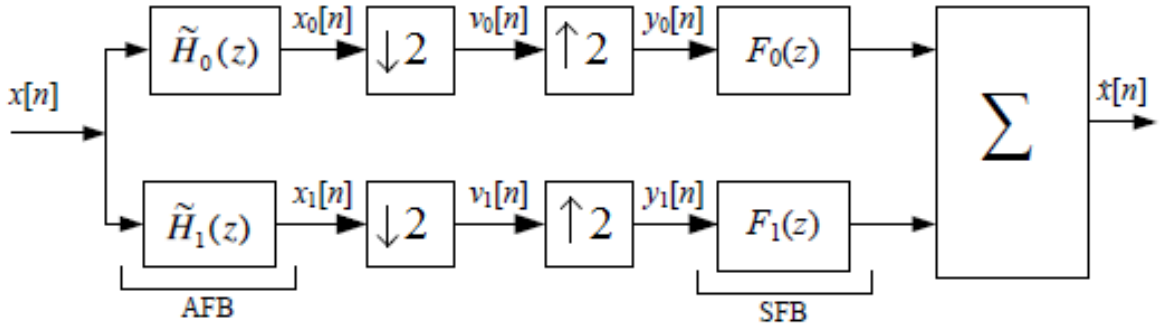


Figure 2.13: HFB digital equivalent

Let's now see the output equation of the entire system, after passing through the decimation we get the signals:

$$\begin{cases} V_0(z) = \frac{1}{2T_s}[X(z^{\frac{1}{2}})\tilde{H}_0(z^{\frac{1}{2}}) + X(-z^{\frac{1}{2}})\tilde{H}_0(-z^{\frac{1}{2}})] \\ V_1(z) = \frac{1}{2T_s}[X(z^{\frac{1}{2}})\tilde{H}_1(z^{\frac{1}{2}}) + X(-z^{\frac{1}{2}})\tilde{H}_1(-z^{\frac{1}{2}})] \end{cases} \tag{2.1}$$

so, the reconstructed signal will then be:

$$\hat{X}(z) = V_0(z^2)F_0(z) + V_1(z^2)F_1(z) \tag{2.2}$$

Replacing$V_1$and V_2 we have:

$$\hat{X}(z) = \frac{1}{2T_s}[\tilde{H}_0(z)F_0(z)+\tilde{H}_1(z)F_1(z)]X(z)+\frac{1}{2T_s}[\tilde{H}_0(-z)F_0(z)+\tilde{H}_1(-z)F_1(z)]X(-z) \tag{2.3}$$

define from now onT_0 and T_1as:

$$T_0(z) = \frac{1}{2T_s}[\tilde{H}_0(z)F_0(z) + \tilde{H}_1(z)F_1(z)] \tag{2.4}$$

$$T_1(z) = \frac{1}{2T_s}[\tilde{H}_0(-z)F_0(z) + \tilde{H}_1(-z)F_1(z)] \tag{2.5}$$

Where T_0 is the distortion, which allows to achieve perfect reconstruction if only it is a delay or a constant. And T_1 is called aliasing and must be 0 in order to have a linear frequency response, this is caused by the decimation of the signal.

The discrete time Fourier transform of the output signal is then:

$$\hat{X}(e^{jw}) = X(e^{jw})T_0(e^{jw}) + X(e^{j(w-\pi)})T_1(e^{jw}) \tag{2.6}$$

in matrix form it's:

$$\begin{bmatrix} e^{-jw_q d} \\ 0 \end{bmatrix} = \frac{1}{2T_s}\begin{bmatrix} \widetilde{H}_0(e^{jw_q}) & \widetilde{H}_1(e^{jw_q}) \\ \widetilde{H}_0(e^{j(w_q-\pi)}) & \widetilde{H}_1(e^{j(w_q-\pi)}) \end{bmatrix}\begin{bmatrix} F_0(e^{jw_q}) \\ F_1(e^{jw_q}) \end{bmatrix} \tag{2.7}$$

As we want to know the synthesis filters we then have,

$$F(e^{jw_q}) = 2T_s H^{-1}(e^{jw_q})T(e^{jw_q}) \tag{2.8}$$

In witch w_q is the arbitrary set of frequencies with q = [0 ... Q-1].

Assuming that we want the synthesis filters to be FIR filters which have the advantage of being always stable, although it loses part of the impulse response of the filter compared to the IIR filter implementation. So in order to obtain the response of synthesis filters in the time domain we have to solve,

$$\begin{cases} F_1 = W^H f_1 \\ F_2 = W^H f_2 \end{cases} \tag{2.9}$$

In witch $(.)^H$ is the conjugate transpose and $W_q^H = [W^0 W^{-q} ... W^{-q(L-1)}]$ with $W = e^{j\frac{2\pi}{Q}}$ and L the number of coefficients of the filters. Generally, we use a Q greater than or equal to L resulting in an indeterminate system where we can find a minimum L_2 solution.

In order to f we have:

$$\begin{cases} f_1 = W^+ F_1 \\ f_2 = W^+ F_2 \end{cases} \tag{2.10}$$

in witch $W^+$is the pseudo-inverse of the Fourier transformation matrix W , in other words W^+ is the product of the inverse matrix W with the identity matrix[26].

2.9 Interchannel phase delay effect on HFBs

2.9.1 Effects of phase mismatches between the measured and the real system

When projecting a Hybrid Filter Bank we need to design the Synthesis Filter Bank, which is part of the digital system, in function of the response of the analog counterpart system, the Analysis Filter Bank. During the designing process we assume that the analog system is invariant in time and, thus, the resulting relationships between the analog and digital parts are always true. However, if little deviations occur in the analog part of a HFB, the Digital domain system previously designed may not be suitable to offer the initial reconstruction figures. Such deviations may occur due to detrition on the connectors, use of cables with different transmission properties or different lengths, misplaced connectors, etc. In theory, the digital system could be updated and optimized in order to guarantee that the initial reconstruction conditions are rediscovered. However, such an adaptive system is complex to design owing to the increase of the physical equipment and to the highly complex digital procedures required for that purpose. Other solution is to predict the aforementioned deviations and check if the reconstruction deterioration is acceptable or inadmissible. Thus, we want to characterize the performance of an HFB, when the analog part is deliberately subjected to interchannel phase mismatches.

Next equations relates, on an analog transmission line, the phase delay (θ) with the length of the line (l) and the frequency at use (f_o).

$$l = \frac{\theta}{f_o}\frac{v_p}{360} \tag{2.11}$$

$$\Longrightarrow \theta = l\frac{360 f_o}{v_p} \tag{2.12}$$

where v_p is the velocity of propagation in a substrate with a given dielectric constant ϵ_r and magnetic constant μ_r:

$$v_p = \frac{C}{\sqrt{\epsilon_r \mu_r}} \tag{2.13}$$

Now, for example, at $f_o = 100MHz$, a phase delay of 1 degree ($\theta = 1^o$) is equivalent to the inclusion of a line with $l = 5.6mm$. In a practical case, the mismatches in the length of the cables or in the connectors are unlikely to exceed that value, unless for human negligence as a misplaced connector or the use of male-male/female-female type connectors. In any case, we should check if our system is sensitive to phase errors of one degree or two degrees.

Before proceeding to the simulations, maybe it is convenient to explain how the overall procedure was done.

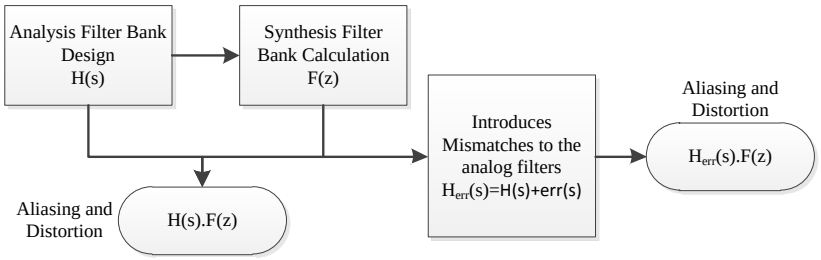


Figure 2.14: Main procedure to test

Figure 2.14 shows a diagram of the procedure, and it can be seen that a HFB is originally designed and, thereafter, some phase errors are placed on the analog part and the aliasing and distortion is recalculated. We know that the output of a Hybrid Filter bank may be given by

$$\hat{Y}(e^{j\omega}) = \sum_{p=-\infty}^{\infty} \widetilde{X}\left(e^{j\left(\omega-\frac{2\pi}{M}p\right)}T_p\left(e^{j\omega}\right)\right) \tag{2.14}$$

where

$$T_p(e^{j\omega}) = \frac{1}{MT_s}\sum_{m=0}^{M-1} \widetilde{H}_m\left(e^{j\left(\omega\frac{2\pi}{M}p\right)}\right) F_m\left(e^{j\omega}\right)$$

represents both the aliasing and distortion over the reconstructed signal and, thus, is the value we want to measure for the different conditions that the analog filters will be subjected to.

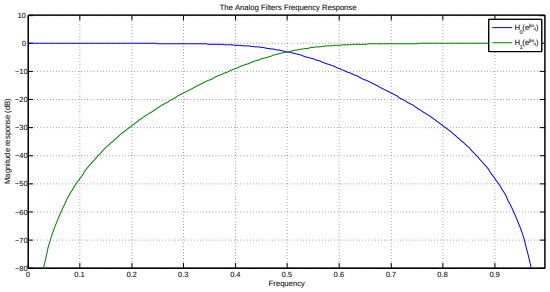
At last, the phase errors are placed right down on the H matrix that characterizes the analog part of the system and the one that is used to calculate aliasing and distortion. Equation (2.15) shows how the phase errors are placed into each filter that constitutes the H matrix, where θ_m is the phase delay error to add to the original filter response H_m.

$$\ddot{H}_m(s) = \|H_m(s)\|\, e^{j\{\angle H_m(s)+\theta_m(s)\}},\ m = 0,\ ...,\ M-1 \tag{2.15}$$

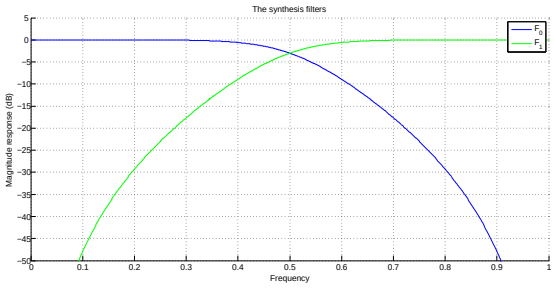
If we assume that the phase error $\theta_m(s)$ is linear with frequency, i.e., has a constant group delay, then we may modulate $\theta_m(s)$ with the inclusion of a line section, with a non-negligible length l, at the end of a given analog channel. Thus, to add the phase error $\theta_m(s)$ to the original filter response, we just have to calculate equation (2.12) for the range of frequencies in consideration and for a given length l. Notice that we assume that all lines are ideal, i.e., are correctly matched and are lossless.

2.9.2 The HFBs used for this test

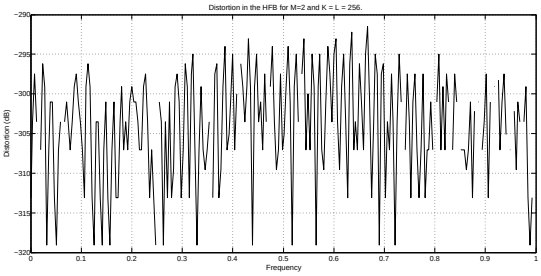
To test the interchannel phase delay effects, two hybrid filter banks will be created. In the first case, a simple $M = 2$ case is constructed, where the analog filters were designed through a butterworth technique.



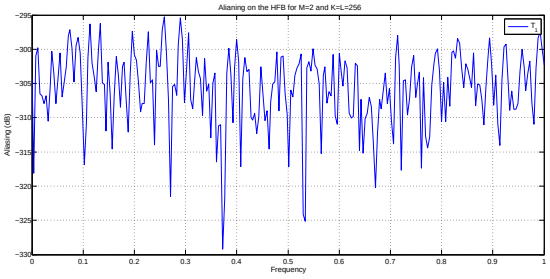
(a) The analog filters magnitude responses



(b) The digital filters magnitude responses



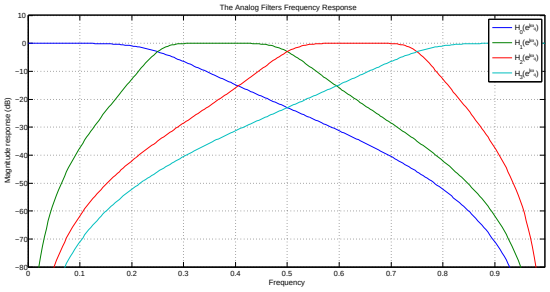
(c) The distortion in the HFB



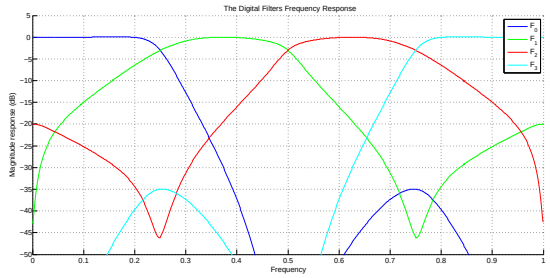
(d) The aliasing in the HFB

Figure 2.15: The original hybrid filter bank for M=2

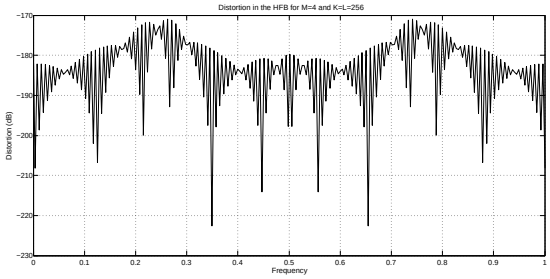
Figure 2.15 shows the analysis and synthesis filters responses, as well as the aliasing and distortion results, where it can be seen that reconstruction is possible with $-300dB$ errors and beyond.



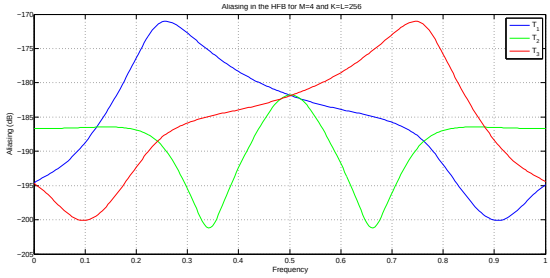
(a) The analog filters magnitude responses



(b) The digital filters magnitude responses



(c) The distortion in the HFB



(d) The aliasing in the HFB

Figure 2.16: The original hybrid filter bank for M=2

The $M = 4$ case, also with butterworth filters, is presented in figure 2.16 where it shows that reconstruction error figures placed around $-180dB$ which is also good.

2.9.3 Absolute Phase Delay

In this part, we want to examine the behavior of the previously designed filter banks, when equal length lines are employed before the A/D converters and without a digital counterpart update.

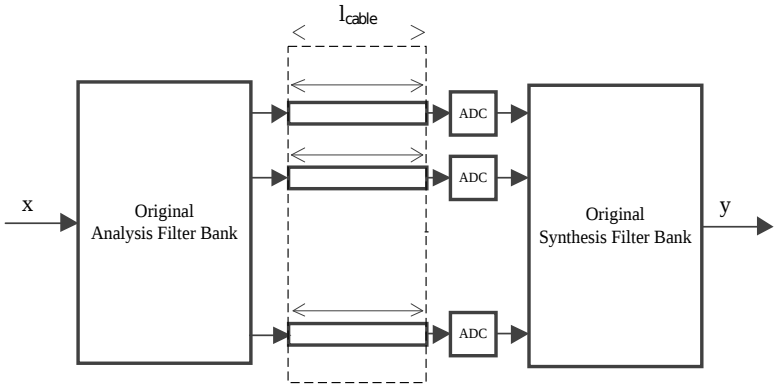
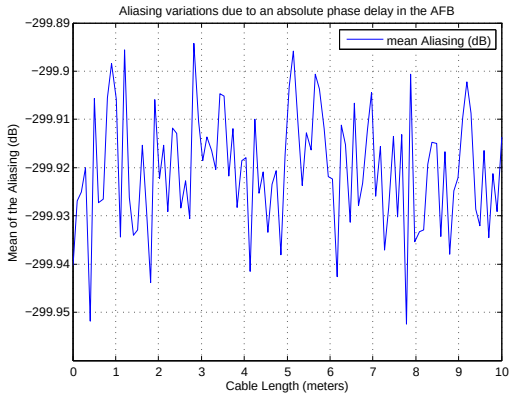
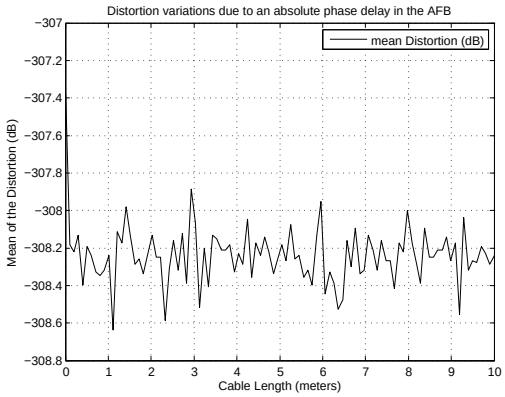


Figure 2.17: The system to be tested

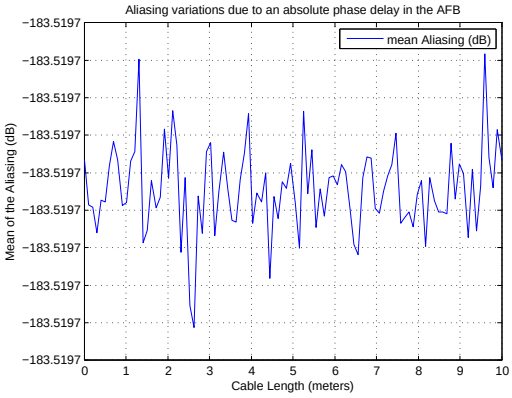
Figure 2.17 shows a diagram of the system to be tested and it can be seen that the procedure is equivalent to the insertion of individual lines, with the same length l_{cable}, at the end of each channel filter. In theory, if ideal transmission lines are assumed, then the extra lines is will only delay all channels in phase in a equal manner and, therefore, won't impair the initial reconstruction figures, only causing an extra delay to the reconstructed signal. Thus, all lines could be replaced by a singular line, with the same length, placed at the input port.



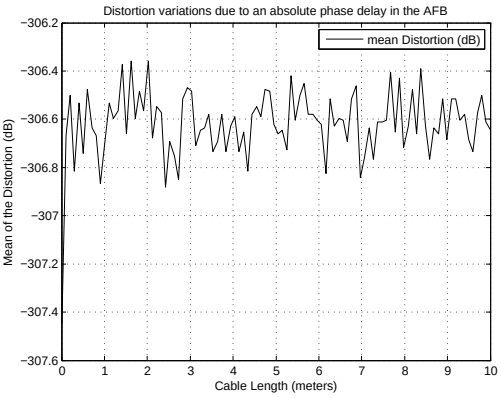
(a) Mean aliasing for $M = 2$ and for a wide range of cable length l



(b) Mean distortion for $M = 2$ and for a wide range of cable length l



(c) Mean aliasing for $M = 4$ and for a wide range of cable length l



(d) Mean distortion for $M = 4$ and for a wide range of cable length l

Figure 2.18: The aliasing and distortion mean errors for a sweep in θ value from 0^o to 180^o

Figure 2.18 shows the simulation results where it is plotted the mean of distortion and aliasing, for each l_{cable} considered and for both $M = 2$ or $M = 4$ cases, and considering that the cables are lossless. From the results, it can be seen that equal length lossless lines do not affect the reconstruction levels, as the interchannel relationships remain unalterable. The only effect of such lines is the increase of the delay to the reconstructed signal y. In short, equal lines at the end of each channel do not affect the HFBs performance.

2.9.4 Relative Phase Delay

2.9.4.1 Test 1

In this part, we'll examine the behavior of the filter banks designed in Section 2.9.2 when an individual line with length l is placed at the end of one channel. Lets start by assuming that the HFB is designed to operate from 50MHz to 75Mhz. Therefor, the schematic from figure 2.19 is employed. In short, with this simulation, we are assuming that one of the channels suffers from variation in its phase response which is not covered by the digital part of the

system. By knowing this, we want to measure the associated reconstruction error.

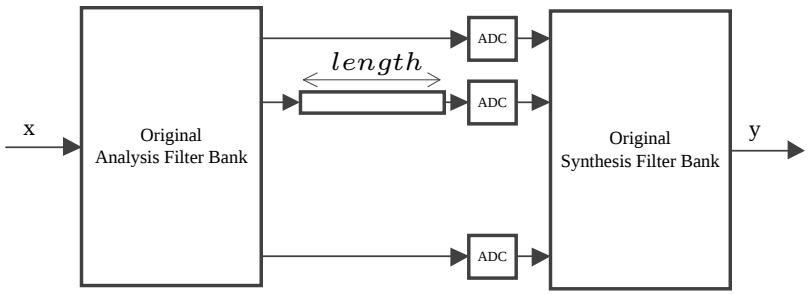
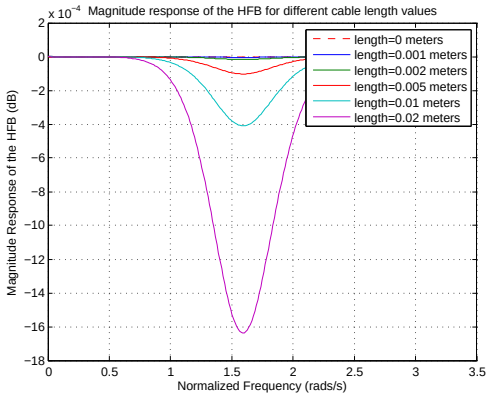
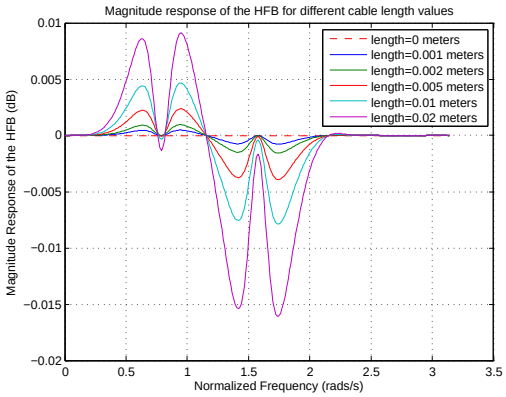


Figure 2.19: System schematic to employ - only one channel has it's phase delayed

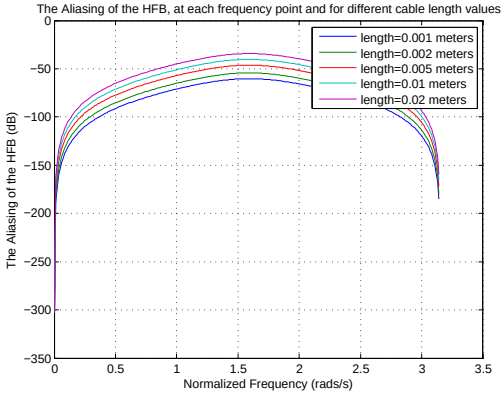
Figure 2.20 *(a)* and *(b)* shows the overall system magnitude responses, either for $M = 2$ and $M = 4$, when a cable with length l is placed in the second channel. It can be seen that for $l = 0$ meters, the magnitude response of the HFB is perfect because it creates zero distortion for all the considered bandwidth. However, as l increases, it can be seen that distortions start to appear at some frequencies. Those frequencies are, in fact, the cross frequencies between the delayed filter and it's adjacent ones, whereas the digital part is unable to produce a convenient reconstruction own to the interchannel relationships that have changed.



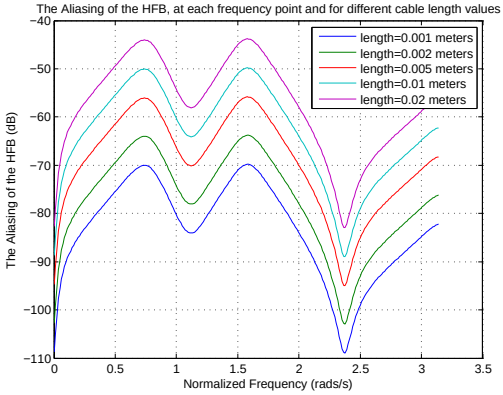
(a) Magnitude Response for $M = 2$



(b) Magnitude Response for $M = 4$



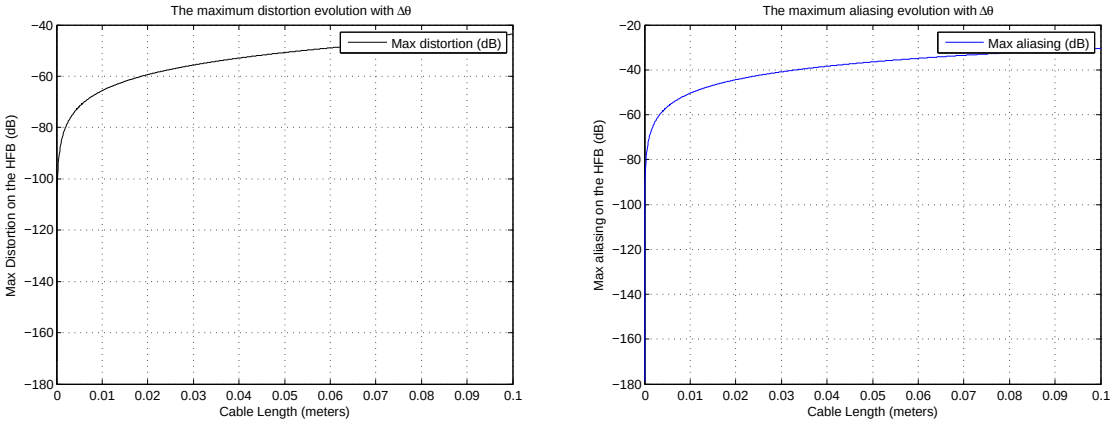
(c) Aliasing terms for $M = 2$



(d) Aliasing terms for $M = 4$

Figure 2.20: The magnitude response of the HFB and the Aliasing terms for a variety of l

The same effect will occur in terms of aliasing, figure 2.20 *(c)* and *(d)*. Therefore, we can see that a slight channel delay will disturb the HFB from correctly reconstructing an input signal, thus causing degradation of reconstruction figures. To conclude this case of study, figure 2.21 shows the **maximum values** of aliasing and distortion for each cable length value l, on a $M = 4$ system, and the aforementioned conclusions can be observed once again.



(a) Distortion (b) Aliasing

Figure 2.21: Maximum value of distortion and aliasing for different $\Delta\theta$ values

Notice that the error starts to saturate at some point. This is because the main focus of error is only at frequencies nearby the crosspoints between adjacent filters, while in the other frequencies, the signal is still correctly reconstructed.

2.9.4.2 Test 2

Lets now test another configuration. Again, lets assume that the HFB is designed to operate from 50MHz to 75Mhz.

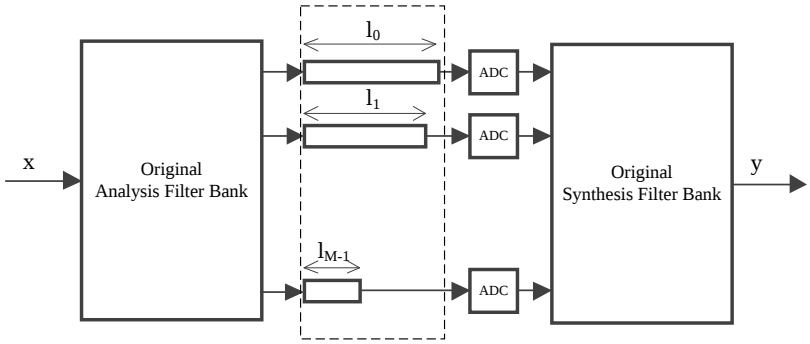
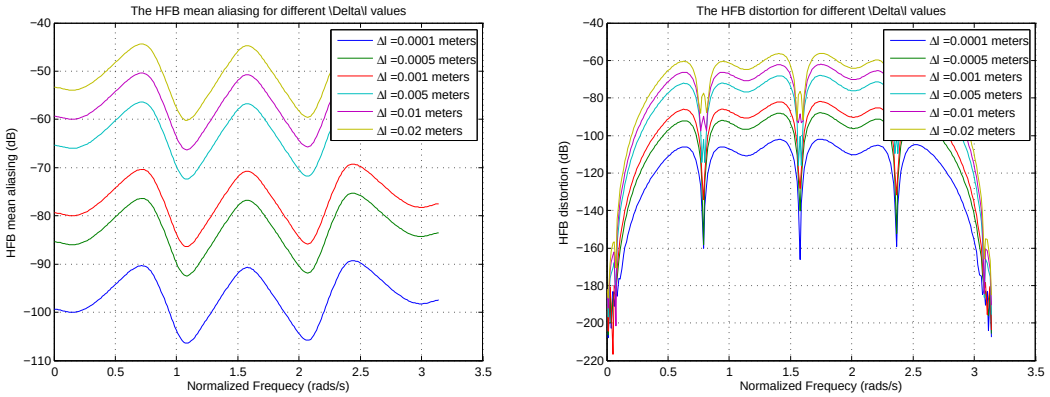


Figure 2.22: Test procedure - All different lengths

Also, assume that the schematic from figure 2.22 is applied, i.e., lets consider that we use M cables, with different lengths l_m between each other, and that the length of each cable is given by:

$$l_m = l_{m-1} + \Delta l,\ m = 1..M-1 \tag{2.16}$$

where l_m is the length of the cable in channel m and Δl is the length difference between adjacent channels. Assume also that all cables are ideal and lossless.



(a) Distortion $M = 2$ (b) Aliasing $M = 4$

Figure 2.23: The distortion and aliasing in the HFB for different values of l_m.

Having this, the following results from figure 2.23 were obtained, whereas the distortion and aliasing were calculated for different values of Δl. It can be seen that for small differences as $\Delta l = 0.1mm$, no major concerns have to be taken as the reconstruction errors are placed under $-90dB$ which is still very good. On other hand, for Δl above $1cm$, the error of reconstruction might be higher than $-50dB$, and therefore jeopardizing the advantages of the Hybrid Filter Bank.

2.9.5 Conclusions of the effects of phase mismatches

It may be helpful to describe the measurement errors in terms of degrees rather then meters. We saw that an measurement error of 1cm in the 50MHz-75MHz range is enough to degrade the system performance. However, in a system operating from 1.4GHz to 1.7GHz, for example, the same $1cm$ would caused a completely different error. Thus, to generalize beyond the frequency usage, we need to establish the measurement error in terms of degrees of interchannel delay. With that measurement, we may point a maximum interchannel delay, in degrees, that will apply to every HFB system, for any frequency. For that purpose, consider the following equation, which is an extension of eq.2.11:

$$\Delta\theta = |\theta_2 - \theta_1| = \frac{360f}{v_p}(l_2 - l_1) = \frac{360f}{v_p}\Delta l \tag{2.17}$$

where $\Delta\theta$ represents the interchannel delay between two adjacent channels, whose present l_2 and l_1 as their respective measurement errors in meters. Now with $\Delta\theta$ it is possible to have an generic measurement for the interchannel phase delay error, i.e., it is guaranteed that an error of $\Delta\theta = 1^o$ has the same effect either at low frequencies or at high frequencies. However, the measurement error Δl will be different. Table 1 shows the measurement error Δl needed to obtain an interchannel delay of one degree ($\Delta\theta = 1^o$) either in the Cochlea case or in the Star Junction case.

	High Frequency Multiplexer	Cochlea Multiplexer
Operating frequency	$1.3GHz - 1.7GHz$	$50MHz - 100MHz$
ϵ_r	9.8	2.2
$\Delta l_{@\Delta\theta=1^o}$	$\approx 0.2mm$	$> 1cm$

Table 2.2: The measurent error Δl needed to achieve $\Delta\theta = 1$

It the cochlea case, when $l_1 = 0$ (meaning no measurement errors in the channel 1), a one degree of interchannel phase delay ($\Delta\theta = 1^o$) is obtained if channel 2 has a measurement error over $1cm$ of length (which is almost the length of a male-male/female-female connector). As seen from figures 2.20 and 2.23, a 1cm line will cause the reconstruction error to rise up from $-300dB$ to $-40dB$ in the $M = 2$ and from $-180dB$ to $-50dB$ in the $M = 4$ case. In practice, such error measurement seems quite unreasonable, unless female-female or male-male connectors are used and not included into the digital system.

On the other hand, if we consider the Star-Junction Multiplexer case, it can be seen that the prone to error is much higher. For example, considering that $l_1 = 0$ (no errors in the measurement of channel 1), a 1 degree of interchannel phase delay will occur if the channel 2 is measured with 0.2mm of error. We know, from the cochlea case, that a $\Delta\theta = 1^o$ error means the rising of aliasing and distortion up to approximately $-50dB$. Thus, the prone of error in the 50-100MHz analog filter bank case may be inappropriate to produce real-time measurements and reconstructions as it is very easy to commit measurement errors that will highly degrade the system performance.

On the other hand, in relation to the Cochlea multiplexer, special care as to be taken in order to guarantee measurement errors inferior to 1cm. If the use of male-male or female-female connectors is unavoidable, then those connectors responses must also be measured and taken into account when measuring the overall analog system performance.

Part III

Accurate Measurements in RF

Chapter 3

Accurate Analog Filter Bank Measurements

3.1 Matlab code development for interaction with a GPIB Oscilloscope and Signal Generator

The majority of the measurement equipment used in RF has a GPIB (IEEE-488)[27] interface, which is a short-range protocol created on the late 1960s by Hewlett-Packard for use in laboratory equipment and it became the standard. It uses a parallel bus with individual control lines.

All the equipment that we used had a GPIB connector to interact with them. For use with our personal computer we used a GPIB adapter to Ethernet or to USB. There are 2 ways to start the serial connection from Matlab, the VISA[28] and the MATLAB Toolkit for R&S[29], in both toolkits we then define the IP from the equipment and associate it in Matlab, the equipments use the Agilent protocols[30] that also need to be installed on the computer. A basic command to know if everything is connected is to query for a '*IDN?' GPIB command, if everything is working the equipment should respond with its name. After confirming this we are ready to send any command and receive any response.

3.2 Code development to interact with the oscilloscope and the signal generator to create a frequency sweep and analyzing the resulting data in a sampled transfer function

The first step to create a frequency sweep and measuring the results is to connect Matlab to the SMU200A generator and to the DPO72004B oscilloscope and query them for a '*IDN?' to see if everything is working as expected. After this we turn off the RF of the generator to change the configurations. We choose then the output power, and the initial frequency.

From the oscilloscope we choose to acquire in sample mode and at real time. We then choose the sampling rate (in our case we used 250MHz), and the record length that is the amount of samples that we are going to acquire. Now this is a tricky selection because only some frequencies are selectable from the range of the oscilloscope, so after the selection a query should be made to know if the sample rate was successfully changed to what we had previously defined. Then, for each of 4 channels of the oscilloscope used, we do the following: turn the

channel on, choose DC coupling, 50Ω termination, 0 position and offset, and the specific scale in volts. Now the scale is very important because different scales can have different behaviour, so every channel should have the same scale, this scale can also be changed in the middle of the measurements but as we are going to see later this will add additional errors to the measurement.

After this initial setup of the oscilloscope we should choose the AUTO trigger, this allow for the trigger to be a software trigger sent by the PC user. Then we define the waveform data transfer definitions: "DAT:ENCdg SRIBinary" to define the output data format as an integer and to transfer first the least significant byte, then specify the number of bytes per data point using, in our case we choose 2 bytes so the command is "WFMOutpre:BYT_Nr 2", the we specify the number of points of the waveform that we want to transfer using "DATA:START 1" and "DATA:STOP 'x"' where 'x' is the number of samples that we want to sample.

With everything ready we activate the output of the generator and now it's time to start the sweep.

We run this process 'n' times where n is the number of frequencies of the sweep:

1 - We define the 'n' frequency of the generator;

2 - Wait at least 2ms for the generator to stabilize the output;

3 - We start sampling with the oscilloscope with 'ACQUIRE:STATE RUN' that will sample a single sequence, each sequence contains 25000 samples;

4 - Then for each of the 4 channels we do the following: select the data source to the channel then we run the "CURV?" command and the waveform of that channel is transferred;

5 - We then use that values and run a FFT with 25000 points and save the maximal point and its frequency for each channel and save it. Running the FFT with 25000 points at the sampling frequency of 250MHz allows a frequency resolution of 1 Hz.

We now have the transfer function of the frequency interval selected for each channel.

3.3 Characterization of the Oscilloscope

We now want to legitimate with a laboratory experiment that is possible to measure S parameters of a filter bank with a signal generator and an ADC (in this experiment we will use a oscilloscope).

With this lab experience we want to obtain the S parameters with a Network Analyser of a filter bank with 2 channels in a particular range of frequency. This measurement is made using all the system apparatus including the filter bank, a directional coupler and the cables used to connect the components. After the measurement of the S parameters with the Network Analyser the frequency response of the filter bank will be measured using a signal generator and an oscilloscope. This experiment will be made using as input, in the filter bank, a sinusoid with a determined frequency and observing the output of the filter bank, this way we will be able to measure the variation of amplitude and phase that the system provokes in the signal. This procedure will be repeated for every frequency that was analyzed in the Network Analyzer.

As we are working with low frequencies (50-100MHz) and with 50Ω components, we are expecting that the reflections of the circuit are negligible (S11, S12 and S22), if that is true then we shall have similar results from the two sources of measurement.

So this experiment is divided in two. The first one is the measurement with the Network Analyser and the second one is the measurement of the response in frequency with the

oscilloscope

3.3.1 Used material

In the first part with measurement of the S parameters with the Network Analyser the material used is:

- Filter bank with 8 channels (50MHz-100MHz)
- ZX30-9-4-S+ (Directional coupler)[31]
- Agilent E8361C (Network Analyser)[32]
- VNA calibration kit
- 6x 50Ω male SMA Loads and 2x female
- 2x CBL-1.5FT-SMSM+ (Coaxial cable of 50Ω with 1.5ft length)[33]
- 2x CBL-3FT-SMSM+ (Coaxial cable of 50Ω with 3ft length)[34]

The second experiment will be the measurement of the frequency response with the oscilloscope and the material used is:

- SMU200A Signal Generator[35]
- Oscilloscope DPO72004B (20GHz) with Ethernet communication[36]
- PC with Matlab to interact between the generator and the oscilloscope
- Filter bank with 8 channels (50MHz-100MHz)
- ZX30-9-4-S+ (Directional coupler)
- 6x 50Ω Load male SMA
- 3x CBL-1.5FT-SMSM+ (Coaxial cable of 50Ω with 1.5ft length)
- 2x CBL-3FT-SMSM+ (Coaxial cable of 50Ω with 3ft length)

3.3.2 Experimental assembly

The experimental assembly of the first experience is the following:

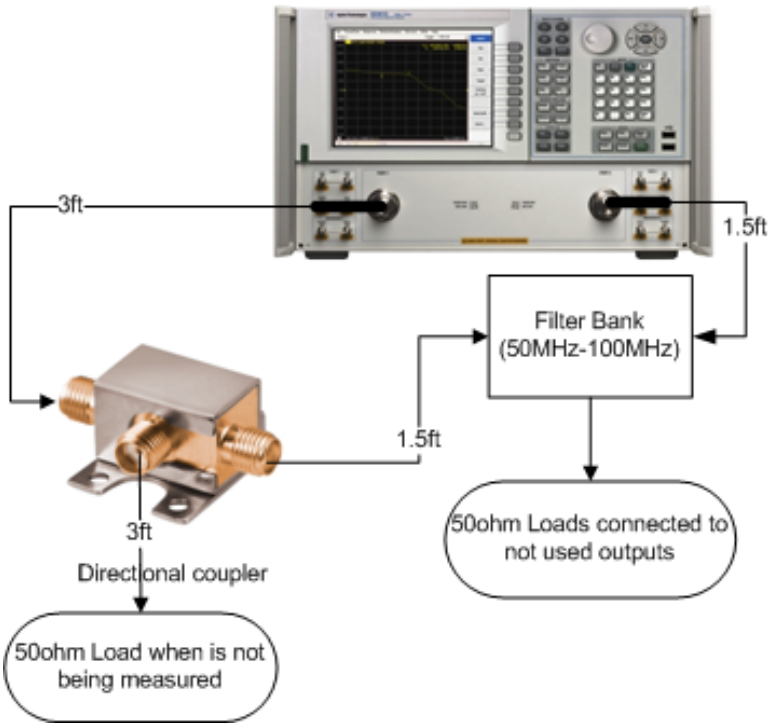


Figure 3.1: 1st experimental assembly

After the calibration, one of the channels of the Vector Analyzer is connected to the 3ft cable that connects to the directional coupler, and the other channel is used to measure the response of the 1st and the 2nd channel of the filter bank and the 2nd channel of the directional coupler, every other outputs in every measurement must have a 50Ω load.

The experimental assembly of the second experience is the following:

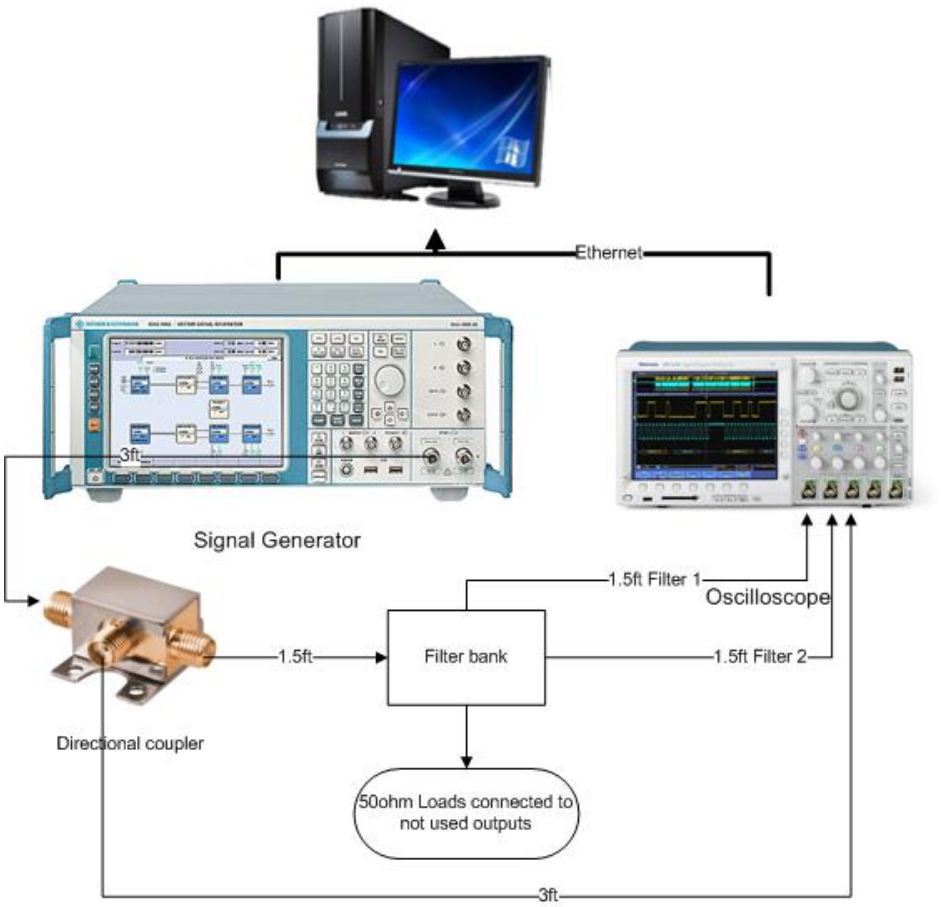


Figure 3.2: 2nd experimental assembly (between each connection it appears in feets the length of the cables)

In the filter bank we only use the 2 channels of the lowest frequencies, so, for other channels not to interfere with the result by the fact that they are not adapted to 50Ω then 50Ω loads must me connected to the output.

The computer must have installed the "Agilent IO Libraries Suite", that haves the libraries necessary to communicate between Matlab and the equipments.

3.3.3 Procedure to the first experiment

Initially the PNA should be calibrated, the connections of the PNA should be of the same type, sma female, to fit directly on the cables that we are using. After this we should measure the S-parameters of the two filters and the second output of the directional coupler between 50MHz and 65MHz, and sampled at 1501 points spaced frequency of 10kHz (50Mhz, 50.01Mhz, ... , 65MHz).

The data has to be saved with the extension "S2P" (Touchstone format) and loaded on a computer with Matlab for future analysis .

3.3.4 Procedure to the second experiment

Since the filters were designed to have very attenuated reflections is expected that the frequency response in both amplitude and phase are approximately the S21 parameter of the system.

To validate this assumption the following Matlab code was made for testing:

1 - Set the signal generator frequency to 50MHz and generate a sinusoid;

2 - Set the oscilloscope to work at 250MSamples with 50000 samples per measurement;

3 - Acquiring the three channels of the oscilloscope and run a FFT to know the peak value of the FFT that is not the first one, thus to avoid DC input and the FFT image;

4 - Increase the generator frequency by 10kHz;

5 - Repeat steps 2,3 and 4 until we reach the 65MHz.

3.3.5 Result analysis

Let now try to validate the assumption that the response obtained by experiment 2 should be approximately the same as the first experiment. To this to happen the values for the 3rd channel (coupler output) must be multiplied by the inverse of the response obtained from the coupler in the PNA, by doing this we can obtain with some accuracy the signal that was originally injected into the system for each frequency. Then the values obtained from the 1st and 2nd channel of the filter bank by the oscilloscope are compared to the recovered signal injected into the system to obtain the phase shift and amplitude that the system caused. So to obtain the response in phase and amplitude of the filter bank we made as follows:

1 - The values acquired from the oscilloscope from the coupler output are divided by the S21 parameter from the coupler obtained from the Network analyzer for each of the 1501 frequencies. We now have the reconstructed original signal for each frequency;

2 - The phase response of each channel is measured by subtracting the angle of the oscilloscope output of the 1rst channel and the 2nd by the reconstructed original signal for each of the 1501 frequencies;

3 - The magnitude response of each channel is measured by dividing the of the oscilloscope output of the 1rst channel and the 2nd by the reconstructed original signal for each of the 1501 frequencies.

3.3.6 Results

The results show that the filter bank is non-linear because for different signal powers the response is different, probably the largest supplier of this non-linearity are the inductors that are used in the circuit.

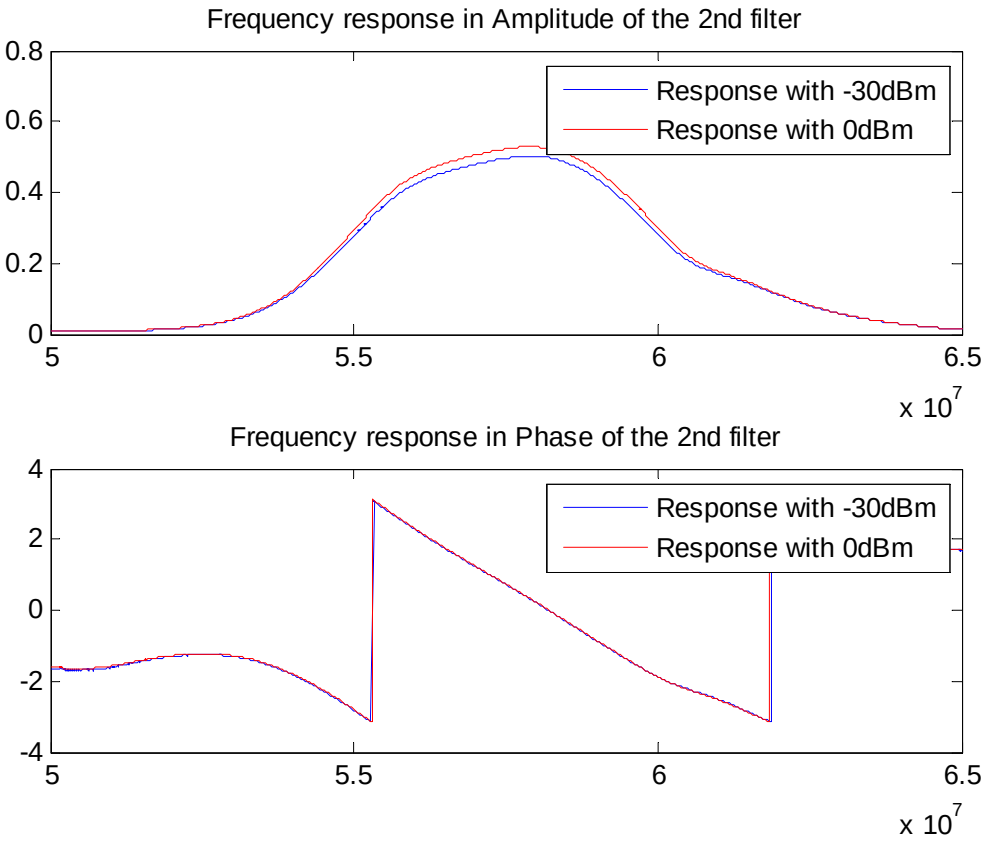


Figure 3.3: Response change at different input powers

To compare between the data that we got from the Network Analyzer with the one from the Oscilloscope we analyze the error with the equation 3.1 and plotted the figures 3.4 and 3.5.

$$Error = \frac{MeasurementOsc - MeasurementVNA}{MeasurementVNA} * 100\% \tag{3.1}$$

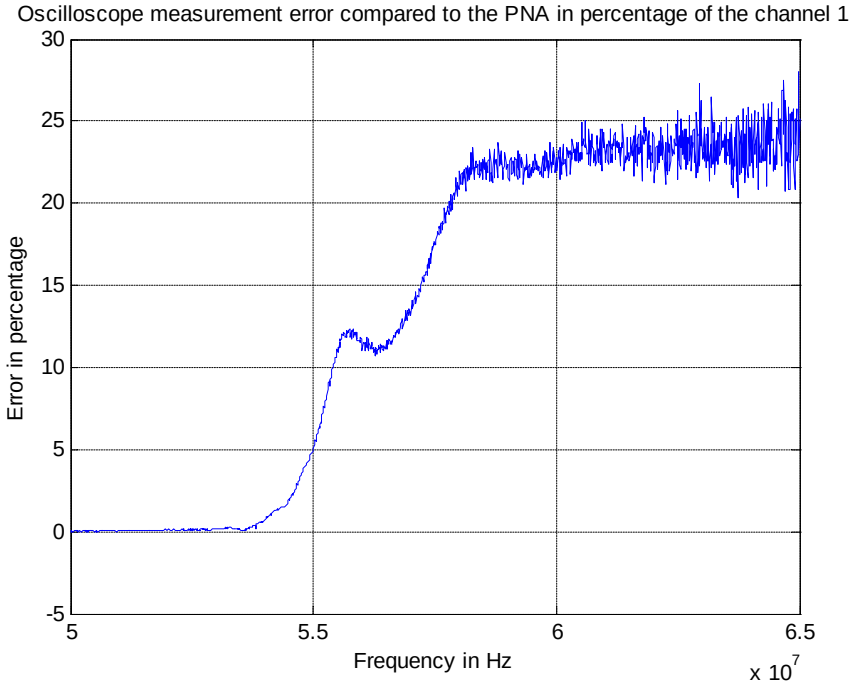


Figure 3.4: Channel 1 - Comparing the oscilloscope with the PNA results

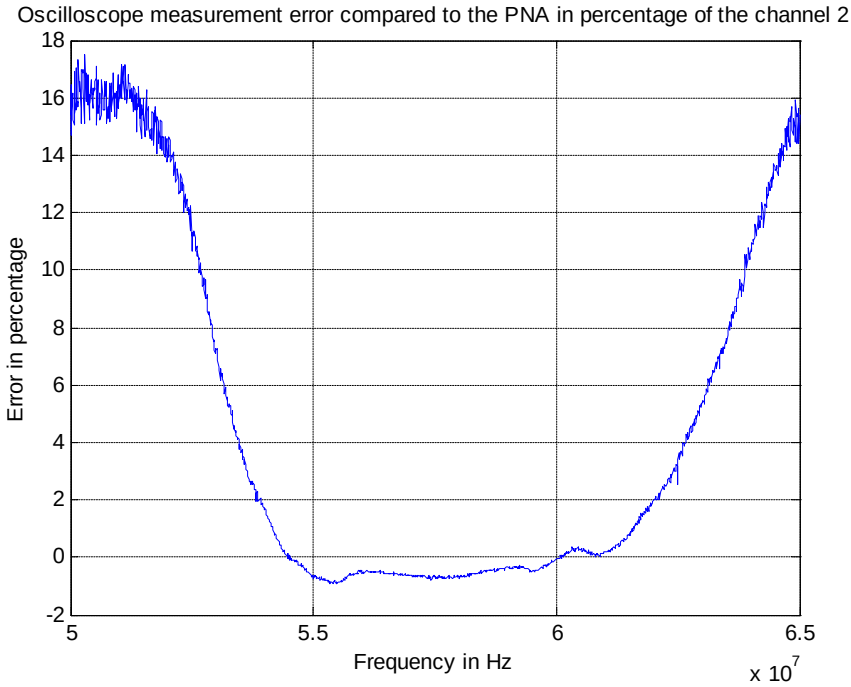


Figure 3.5: Channel 2 - Comparing the oscilloscope with the PNA results

So what we can see from the graphs is that, outside of the passing band of the filters, where the attenuation is very pronounced, we have a large error in measurement, this is due

to the poor resolution of the oscilloscope that acquires at 8bit (with a even lower ENOB). But inside the band where the attenuation of the filters is low we got good results.

We also try to write an auto-scale script, that if the signal during the frequency sweep would be too low or too high, the scale would be changed, the results show that we got non-linearity from the auto-scaling.

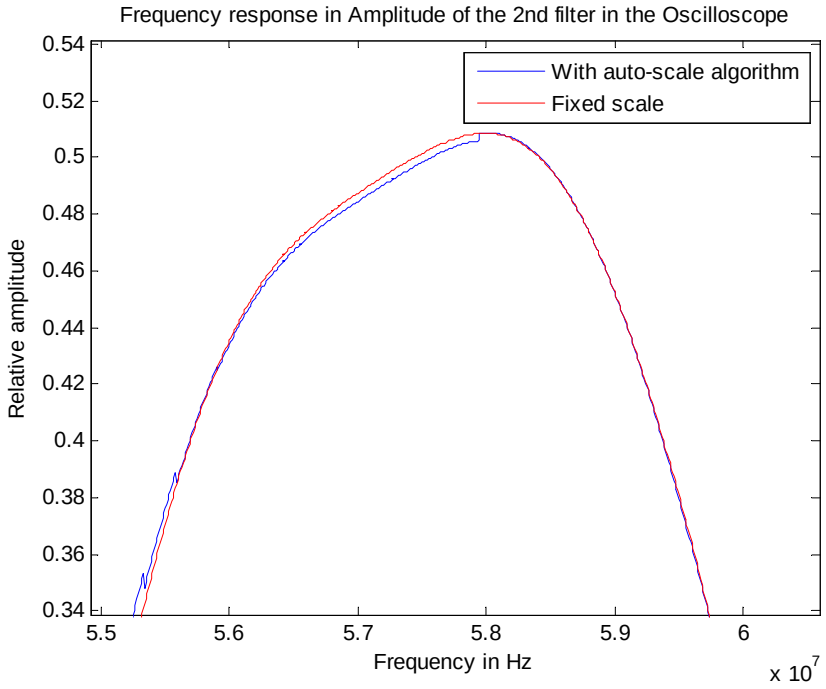


Figure 3.6: Autoscale for the channel 2

Anyway the conclusion is that the remaining S-parameters besides the S21 seem to be negligible, and the error we have during the pass band is what would be expected from a low resolution ADC reading from a very attenuated signal.

3.4 Characterization of the cables

In our continuous study to characterize the elements that will be used in this thesis, we will now study the RF cables. We intend to study the influence of the use of a coaxial cable in the measurements. It would be expected that the effect were approximately a pure delay, to confirm this statement measurements will be made in a Network Analyzer in order to study the results.

3.4.1 Transmission lines

An analog transmission line causes a certain delay of phase which depends on the following equation:

$$\theta = l\frac{360 f_0}{vp} \tag{3.2}$$

in witch θ is the phase delay,l the length of the line,f_0a the signal frequency and vp the propagation velocity in the line.

We will evaluate a coaxial cable with triple shielding from mini-circuits, the cable in question is a CBL-3FT-NMNM+, a 3Feet (aprox. 0.9144 meters) cable, with a male sma connection in both terminations. This cable will be evaluated in frequency from 50MHz to 100MHz that will be the interval of frequency that we are going to use it in laboratory. The manufacturer talks about a velocity factor (velocity factor in relation to the speed of light) that should be around 0,7.

3.4.2 Experiment

The material used was:

- Agilent E8361C (Network Analyzer)
- 1x CBL-3FT-SMSM+

The Network Analyzer must be configured with 0dBm of power, with the initial frequency in 50MHz and finishing at 100MHz, with a sampling of 5001 points and used an average of 16 values to reduce the noise. Then, after the configuration, the Network Analyzer is calibrated and then the cable in question is measured.

Using the equation 3.2 we expect to see that we expect to see in theory a phase shift to the frequency 50MHz of,

$$\theta = 0.9144 * \frac{360 * 50 * 10^6}{0.7 * 299792458} = 78.43^{\text{o}}$$

The S parameters results were the following,

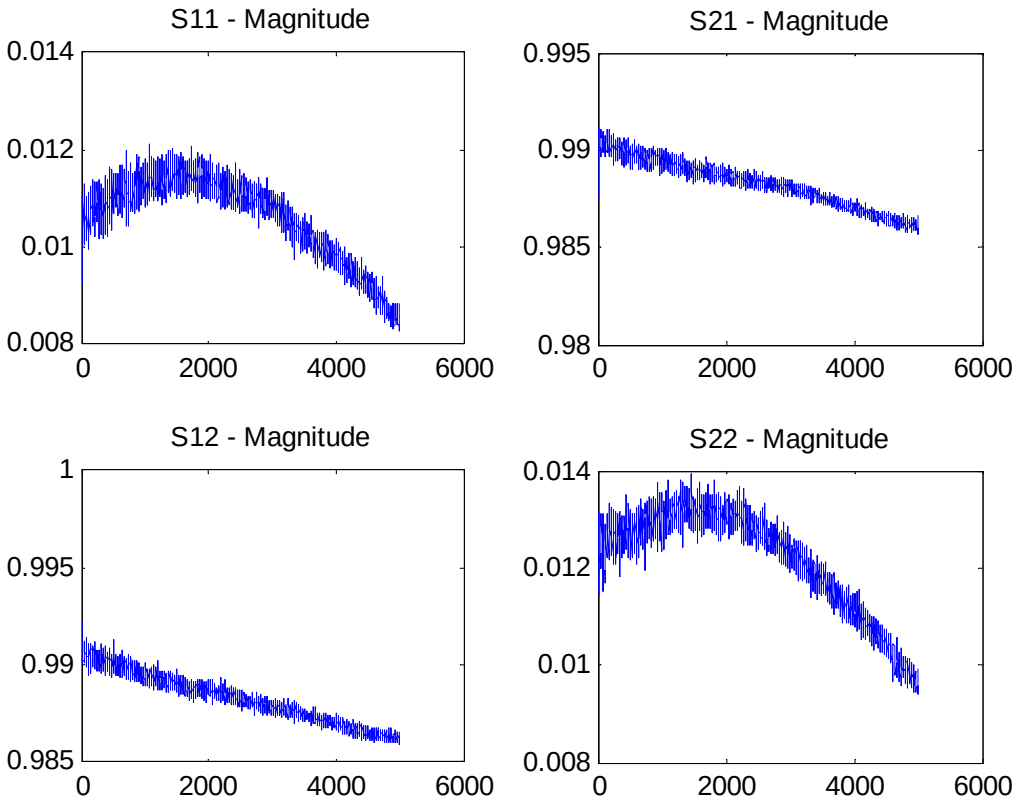


Figure 3.7: S parameters of the cable in magnitude

It is clear that the parameters S12 and S21 range from 0.98 to 0.99 so the cable passes almost all the power in both directions. The reflections are less than 0.014.

And in phase,

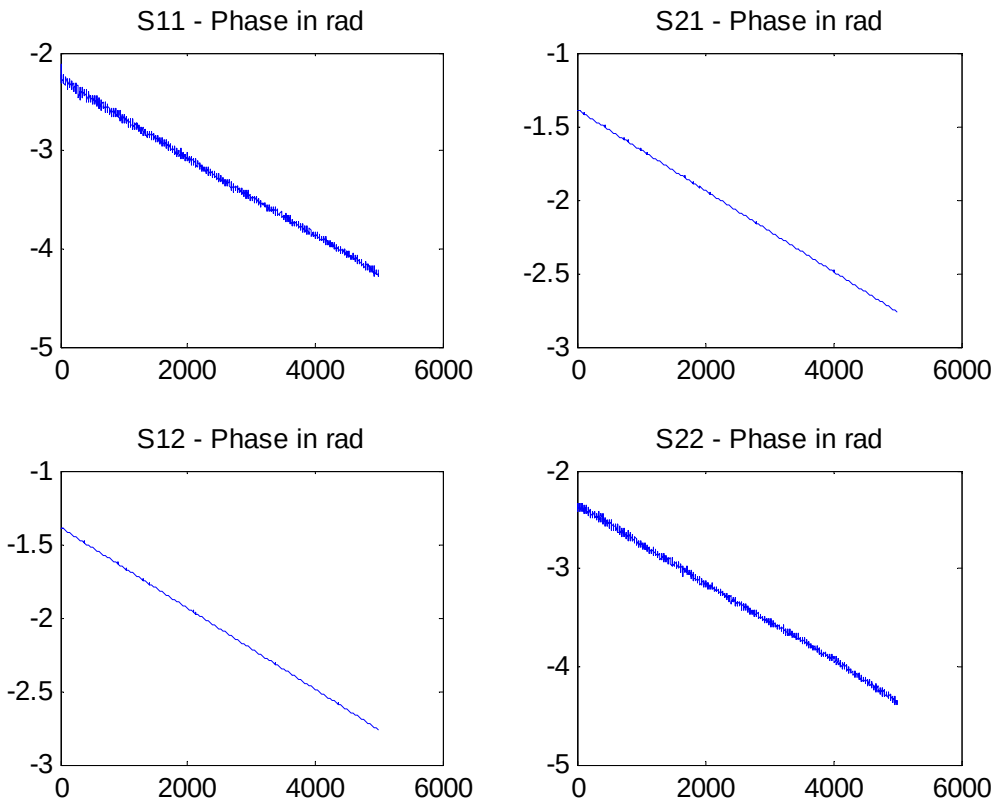


Figure 3.8: S parameters of the cable in phase

Also lets take a look at the derivative of the S21,

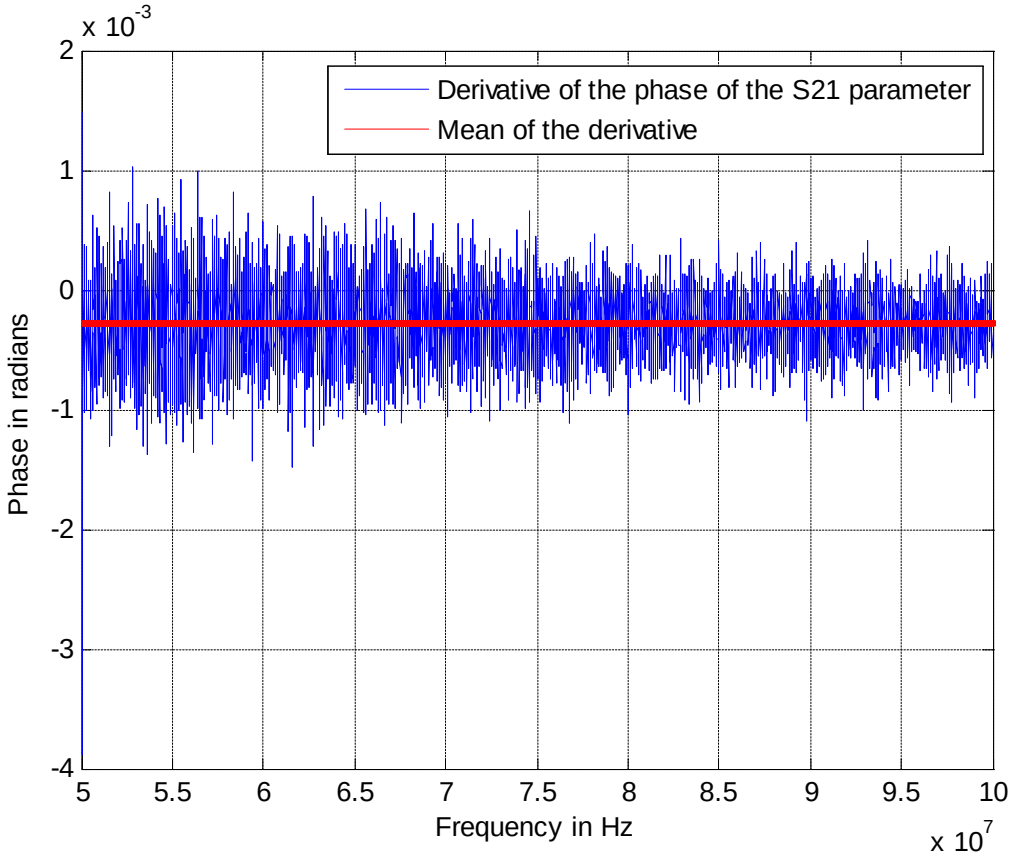


Figure 3.9: Derivative of the S21 phase

The results of the derivative of the S21 phase show a mean of $-0.276 * 10^{-3} rad/Hz$ and a variance of $97.167 * 10^{-9} rad/Hz$.

And the phase of the S21 at 50 MHz: -79.1156º, very close of what would be expected.

The theoretical calculated phase is slightly lower than that obtained, this is due to the lack of accurate knowledge of the velocity factor of the cable, but it's sufficient to confirm the values obtained. This cable phase is nearly linear, and some noise is created during the measurement by the equipment even with the use of various measurements to obtain a mean.

The cable can then be considered an ideal delay without a significant error.

Part IV

Realtime Cochlear Front-End

Chapter 4

Hardware development and characterization

4.1 Testing and evaluation of the ML605 development board with a Virtex-6 FPGA

4.1.1 Performance and Resource Utilization Benchmark for realtime implementation of FIR filters

As a base requirement in order to have a feasible hardware implementation, the filter bank sampling rate was chosen to be up to 100 MSPS. The number of filters per channel was set to 8 and the number of channels should be less or equal to 8. Moreover, each filter must have between 32 up to 64 reconfigurable coefficients.

Concerning the development tools, the Xilinx Integrated Software Environment (ISE) v13.2 and the Xilinx CORE Generator were chosen in this design and a Virtex6 LX240T was chosen as the target FPGA.

For this test implementation, each of the filters is connected to the exterior ports trough register banks. Although this approach requires extra hardware resources, its does also reduce the critical path and therefore enables an increase in throughput.

An FPGA-based implementation containing 64 filters was made, 8 filters for each channel. In this implementation, each filter was configured to produce a throughput of 80 MSPS, using a 400 MHz clock reference signal. As can be seen in table 4.1, the implementation using 32 coefficients per filter and 17 bits of precision is the only one that meets the specified operating frequency of 400 MHz.

# Coeffs. per filter	Output precision (bits)	FFs	LUTs	Slices	DSP48E1s	Fmax (MHz)
40	17	18,624 (6%)	8,648 (5%)	5,103 (13%)	576 (75%)	227,7
40	Full (23)	19,008 (6%)	8,781 (5%)	4,712 (12%)	576 (75%)	280,5
32	17	17,153 (5%)	8,149 (5%)	4,438 (11%)	512 (66%)	**401,6**
32	Full (23)	17,536 (5%)	8,300 (5%)	4,579 (12%)	512 (66%)	308,5

Table 4.1: Implementation results of CS2 when configured to implement 64 filters for a throughput of 80 MSPS.

A new implementation was testes in table 4.2 with the same 64 filters using the 400 MHz reference clock. In this implementation, each filter was also configured to produce the same throughput of 80 MSPS but using fewer coefficients than the previous experiment. This way, less hardware resources are required (specially the DSP48E1 slices), which in turn alleviates the timing enclosure. The obtained results show that all implementation variations for this experiment are feasible.

# Coeffs. per filter	Output precision (bits)	FFs	LUTs	Slices	DSP48E1s	Fmax (MHz)
26	17	15,360 (5%)	7,492 (4%)	3,966 (10%)	448 (58%)	**400,8**
26	Full (23)	15,744 (5%)	7,170 (4%)	3,961 (10%)	448 (58%)	**402,7**
30	Full (23)	15,936 (5%)	7,824 (5%)	3,722 (9%)	448 (58%)	**404,5**
35	17	17,216 (5%)	8,077 (5%)	4,491 (11%)	512 (66%)	398,2

Table 4.2: Implementation when configured to implement 64 filters for a throughput of 80 MSPS.

The table 4.3 summarizes the main implementation results when is implemented with 64 filters using a 400 MHz reference clock. The main difference from the previous experiments is the lower sampling rate of 50 MSPS and the higher order of the filters. Observing the results it is possible to conclude that all variations excluding the second one that has 64 coefficients per filter and uses full precision are feasible.

# Coeffs. per filter	Output precision (bits)	FFs	LUTs	Slices	DSP48E1s	Fmax (MHz)
64	17	18,752 (6%)	8,738 (5%)	4,666 (12%)	576 (75%)	**401,0**
64	Full (23)	19,136 (6%)	8,742 (5%)	4,805 (12%)	576 (75%)	297,0
48	17	15,522 (5%)	7,198 (4%)	3,875 (10%)	448 (58%)	**404,9**
48	Full (23)	15,936 (5%)	7,277 (5%)	3,802 (10%)	448 (58%)	**402,7**

Table 4.3: Implementation results of CS2 when configured to implement 64 filters for a throughput of 50 MSPS.

The obtained results allow concluding that high order filters, that is, filters with more than 35 coefficients are hard to implement on the target FPGA and are only possible for lower sampling rates. Moreover, if the desired filter bank implementation requires more than 64 filters, this will result in a design that is very hard to implement and is only feasible if the filters have lower orders.

Regarding timing enclosure, the surrounding logic will have a negative impact on it, which means that a full design containing not only the filter banks but also the necessary logic to implement the entire system will have worse figures of merit.

The obtained results also indicate that dense designs occupying more than 60% of the available DSP48E1 slices are only feasible for low sampling rates up to 50 MSPS. At last, the obtained results show that a sampling rate of 80 MSPS when the filters are using a clock signal of 400 MHz seems an interesting implementation relation for both case studies.

4.1.2 Creating firmware to interact with a laptop via serial port for a future change in the coefficients of FIR filters in real-time

To interface with the FIR filters and be able to change their coefficients, we used an open core RS-232 interface, as the serial connector from the ML605 emulates, in the operative system, a RS-232 port MATLAB can still use the serial commands to interact with the board.

The VHDL RS-232 open-source core have a simple interface, all we need to give is the data_in and data_in_ready, data_out and data_out_ready and a clock, this clock is used to create the baud-rate that can defined in VHDL.

To test the reconfigurable filters, we simulate the system by creating an interface to send values to a FIFO via the serial communication, and to be used to change the coefficients.

After the VHDL code finished and tested it was then used has a base to the filters implementation in the FMC108 system firmware.

4.2 4DSP solution overview

4DSP provides a reference solution of software and firmware for the ML605 development board. The design flow of this reference solution is based on 4DSP Stellar IP flow, this method of design organizes the firmware in blocks that 4DSP calls:

- Star: for a functional block with a specific task;

- Wormhole: for a connection between two stars. A wormhole consists of one or more signals in either one or both directions;

- Constellation: a collection of stars that forms the top level of the firmware.

The firmware was written in VHDL and Verilog, the majority being in VHDL. The code is written with some level of complexity so a previously study of the firmware and reverse engineering was needed for the modification of the code.

There is also an executable that runs on the personal computer and communicates with the firmware trough the Ethernet port, the source code of this executable is provided and is written in C++ language. C++ is a multi-paradigm and general-purpose programming language, is one of the most used programming languages, and is a derivation of C with the most important enhancement being the added classes, so the lexical of C is still applicable.

4.3 Initial tests with the firmware

To start the test we open the source code in VHDL and generate the bit-stream with the XILINX ISE version 12.3 but the generation gives us an error, some of the clock signals have a higher settling time than the constraints allow for this design as show in table 4.4.

Clock	Requirement	Best Result Archived
Ram Path1	2ns	2.293ns
Ram Path2	2ns	2.456ns
Ram Path3	2ns	2.355ns

Table 4.4: Setup time of the clocks that were not archived

The results are worst than the requirement. Does this make the firmware completely useless? Well first we need to know how the results are generated by the ISE.

The constraint validation is made with some considerations that varies from FPGA to FPGA and come with an associate incertitude. It's also calculated to work in the worst case possible, so even though they were not achieved and as 4DSP says that the firmware works well, we will ignore this alert given by the tool but we will work with some precautions because we are working in the limit. One of the precautions is to have the FPGA working at the lowest temperature possible.

After the .bit file is generated, we connect the FMC108 to the ML605, turn on the power and connected the JTAG-USB cable into the PC and also the Ethernet cable. The FPGA is programmed via iMPACT with the .bit file and a test signal was connected to 1 of the channels of the ADC board. Running the executable in the PC, 8 .txt files were created, each one to each ADC, the samples are written in ASCII and separated by a *newline* character.

The test was then made to each channel individually and the conclusion taken was that the system was working correctly, the maximum number of samples for each ADC is 32768, as the output FIFO in the hardware is 64Kb, each sample occupies 16bit and each channel had a different DC offset when not in use.

4.4 Creation of a protocol to communicate between Matlab and the C++ program receiving the ADC data

The executable application, written in C++, that communicates with the FPGA board has been changed to communicate trough a file with Matlab. This file is a shared memory

space between Matlab and the application that reads the ADC values.

We want to change the application in such a way that when we run it, it communicates to the FPGA and configures everything to start sampling, but will only trigger the ADCs when Matlab writes into the share memory to create such trigger, then the data is read by the ADCs, writen in a different file for each ADC channel and when the burst is over the application writes into the shared space warning Matlab that the measurement is over. Matlab reads, detect the change and stores the information from the ADCs. Whenever we want to measure again, all we need is to write into that file again to start the sampling.

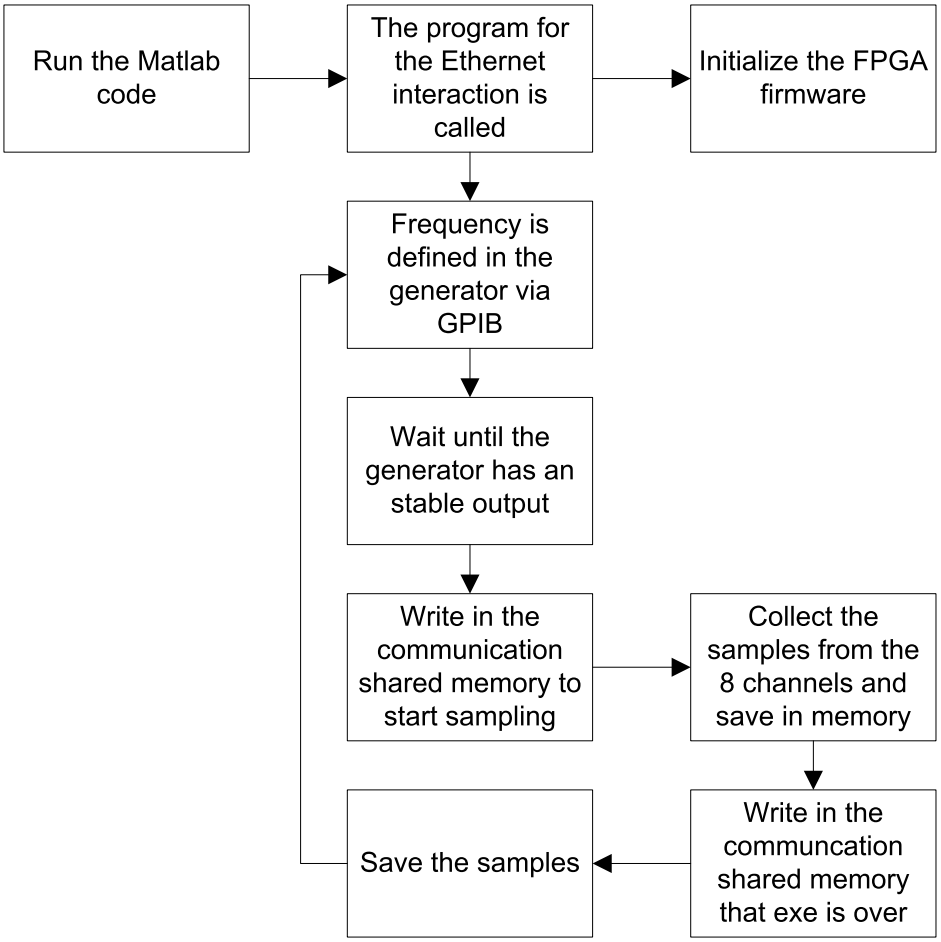


Figure 4.1: Flowchart of the communication between Matlab and the application that connects to the FPGA

This changes allow us now to create a automatic frequency sweep. For that we run a Matlab code that first create a shared memory that can be accessed by the program that will interface directly via Ethernet with our FPGA. This shared memory is called "txrx.txt". This txt file is going to have always a ASCII char in it, that char is used as semaphore between the two programs, when initialized by Matlab the '9' ASCII char is written, and then the executable that will make the interface with the FPGA is called from the same path that the Matlab workspace. The C++ code initiates the FPGA and starts reading in polling the txt file waiting for an order to proceed with the sampling. Matlab then, when everything is ready

for a ADC sampling, erases the contents from the communication file and writes a '1'. The C++ program reads the '1', and starts the sampling for the 8 ADCs, when all is ready it then writes a '0' on the commutation file and creates 8 files entitled 'adcX.txt' where X is the ADC channel from the FMC108 where that data was sampled. The Matlab code interprets the '0', reads the ADC files, save those in memory and when it's ready for another sampling then it writes a '1' again and the same happens. If there's no need for more samplings is written a '2' in the communication file, the C++ program closes and Matlab erases all the files that were created.

Char written in the communication file	Meaning
0	ADCs sampled with success and the ADC files are ready to be read
1	Everything is ready to start sampling
2	The measurements are over
9	Matlab is running

Table 4.5: Communication protocol

4.5 Review and amendment of the code C + + and VHDL to support simultaneous sampling of the 8 ADCs

4.5.1 VHDL

The original VHDL code is structured with the 4DSP Stellar IP. This structure splits the firmware into blocks, called *Stars*. This *Stars* communicate with each other using *Worm Holes*. A group of *Stars* and *Worm Holes* is called a *Constellation*. So 4DSP using their Stellar IP constructed the firmware for the FMC108 in the following way:

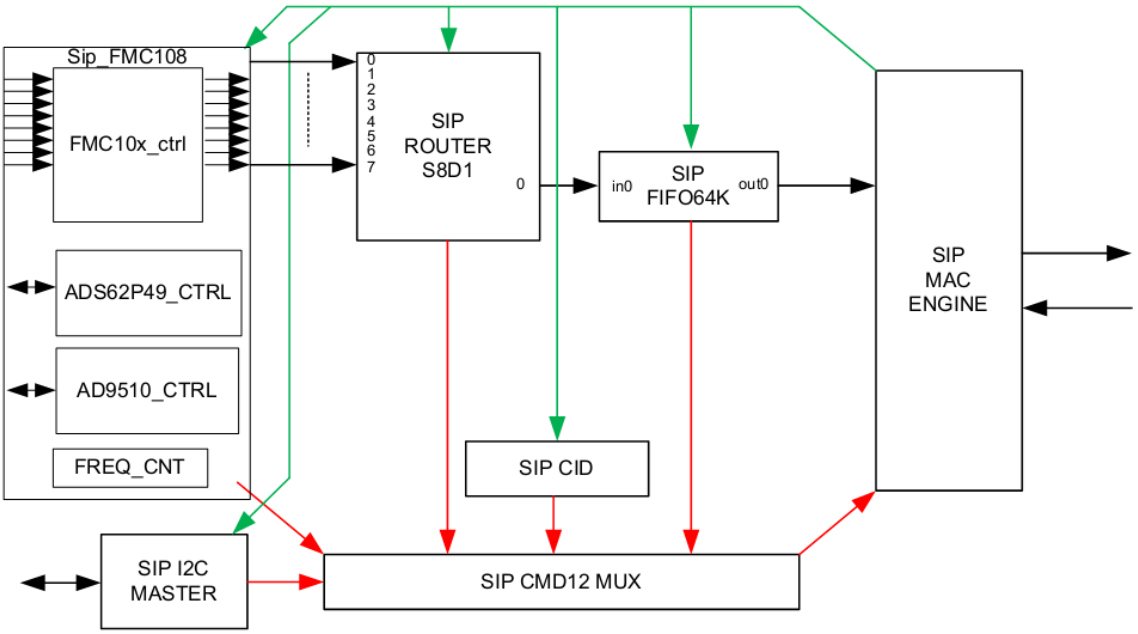


Figure 4.2: Original firmware[37]

The SIP CID is a block that hold the information about the constellation. In practice it allows to know what version of the firmware we are using on the board, and the memory mapping of the system. The Stellar IP puts every memory of the chips used (the ADC chips, the clock tree and the measurement chips) and of the blocks memory that receive commands, mapped into a FPGA memory that is used as a buffer to read/write operations.

The MAC ENGINE is a block that distributes every register read/write command coming trough the Ethernet Mac, it also transfers/receives all the data between the host trough the Ethernet MAC. In this block the FPGA clock is generated and distributed.

The FIFO64k is a 64 Kilo bytes FIFO that buffer that data that is going through the Ethernet.

The 8-to-1 Router is a block that is user defined to choose the ADC port that we want to receive from.

The I2C Master acts as master on the I2C which is connected to the voltage and temperature monitoring circuit (2x ADT7411).

The CMD12 MUX merges all the command output from all the blocks and connects them to a single command output to the MAC engine.

The AD9510_CTRL block communicates with the clocktree AD9510 chip.

The FREQ_CNT uses the ADT7411 data to calculate the frequency of the clocks used in each ADC.

The ADS62P49_CTRL block controls the ADCs chips. And finally, the ADC chips output digital bus and clocks, connect directly to the FMC10x_ctrl.

The ADC samples at 14bits, then after passing trough the Sip_FMC108 it becomes a 16bit value with the 2 least significant bits being 0, then is changed from big-endian to little-endian, 4 samples of each channel reach a FIFO and are then sent in frames of 64bits per channel.

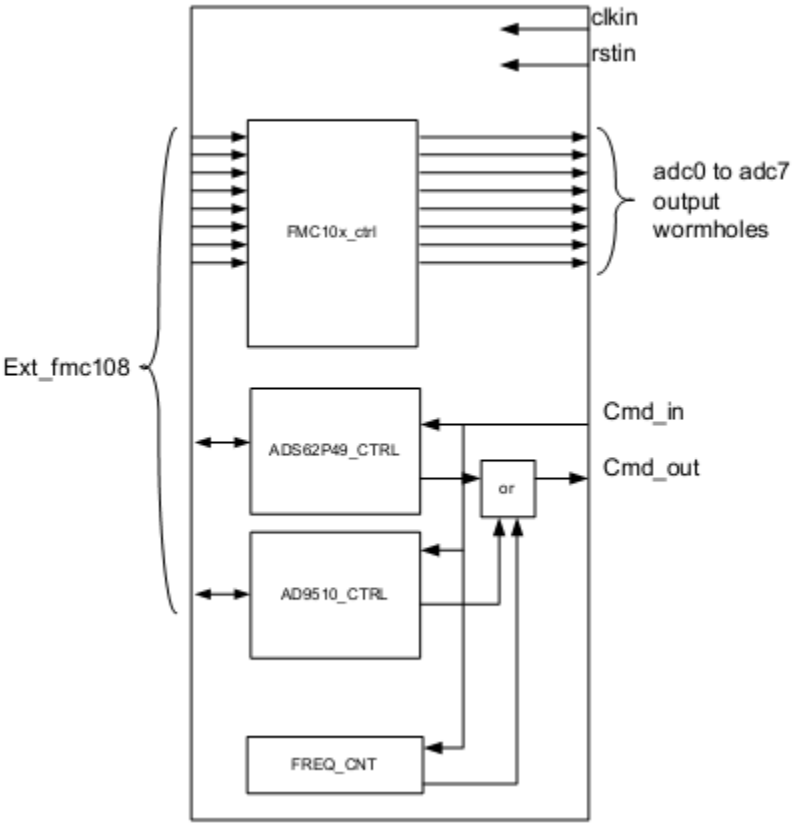


Figura 4.3: The firmware block that communicates with the ADCs directly[38]

Let's study in detail the hardware that receives the data from the ADC's and having a special attention to the clocks domains.

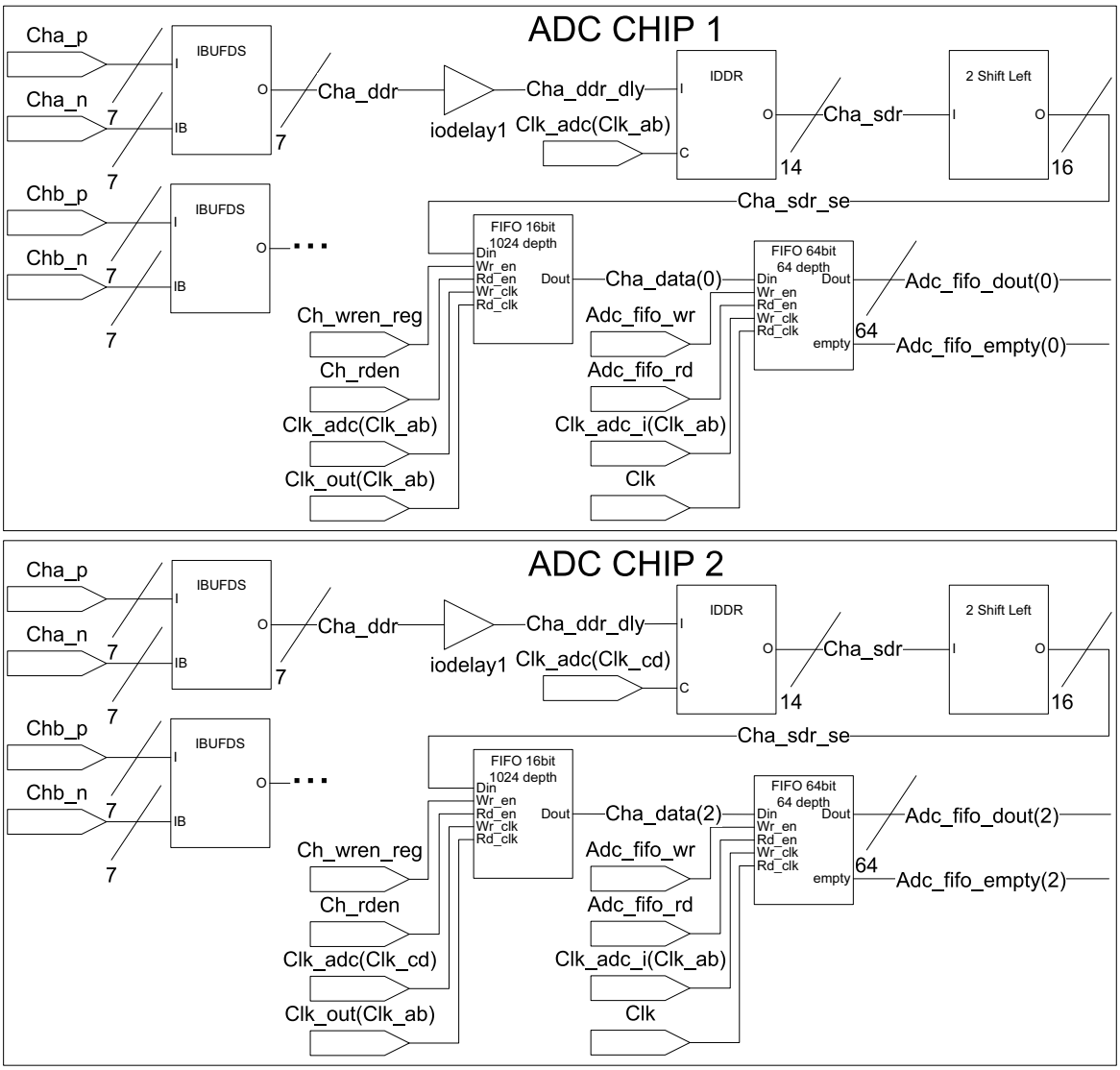


Figure 4.4: The original hardware of the ADCs input driver

So as we can see in figure 4.4 the data buses that comes from the ADCs work at Low-voltage differential signaling at 2.5V, so for the 1 ADC chip we got 2 channels each channel with a differential bus, we got the channel A with cha_p and cha_n and the channel B with chb_p and chb_n, the same applies to the channel B. Both channels use the same clock as they are converted in the same ADC chip. Is identical to the other 4 ADC chips.

So each channel got a differential bus, in total 14 wires, and connects directly to the FPGA pins, in the FPGA that data passes trough the IBUFDS that converts the differential signal into a single-ended signal. That signal then passes trough the iodelay1, this io resource from virtex6 permits creating programmable delays as small as 75ps (for the used clock in this project), this makes possible to correct many hardware problems, for example non-sync of the clock of the ADC that reaches the FPGA and the data bus. This data stream operates in DDR (Double data rate), so the data is transferred in both the rising and falling edges of the clock signal, in this case 7 bits of data is transmitted in the rising edge and the other 7 bits

of the data sampling (in total 14bit resolution) are sent in the falling edge, we use the IDDR resource from the Virtex-6 to convert it to a regular data rate, where 14bits are available. This conversion is made with the clock of each ADC chip, so the first two channels use the clk_ab, the next two use the clk_cd, the next two the clk_ef and the last two channels use the clk_gh.

Then in a designer choose each sample is shifted 2 bits to the left and the 2 extra bits are padded to 0.

The cha_srd_se signal is then driven to a FIFO, this FIFO changes the clock domain, before the FIFO each 2 channels have their own clock, after the FIFO every channel shares the clk_ab first ADC clock. The write enable of this FIFO is at '0' at the reset and changes to '1' after 32 clock falling edges, this is a measure to avoid any transient state from the clock we will later explain why this is not enough, and why this solution does not work in all situations. The read enable of this FIFO is '0' after reset and '1' after all empty signals of all the 16bit FIFOs became '0', so reads become active when the 16fifos of all channels have been written this avoid a faulty read as each FIFO is written at different times.

This data is then transferred to the next FIFO this is a 16bit IN and 64bit OUT FIFO, the rd_clk of this 64bit FIFO is the system clock, the clocks is derived from the FPGA board that is a 200MHz oscillator with 50ppm, so this FIFO changes the clock domain and the data became completely independent from the ADC board and is ready to be sent by the Ethernet. The write clock of this FIFO is without much surprise the clk_ab as it was the read clock from the previously FIFO. The wr_en signal is only '1' when the software sends a trigger command, this is a bad solution because we still have one previous FIFO being written and this will create problems in the future and was fixed by us later. The same applies to the rd_en signal, is only '1' after the software trigger.

After this FIFO we got an interesting Router, this router can be software defined at any time to select one of the outputs of the eight 64bit FIFOs and send it to a bigger FIFO that is then read by the Ethernet block to send the data to the Ethernet port.

We discovered that the reference design permits only 1 burst from the 8 ADCs with no consistency over time, one channel at a time, without any phase relationship. We contact 4DSP and inquired if it was possible to trigger the 8 ADCs at the same time and their response was the follows:

"Sure this is possible but not supported by the standard reference design unfortunately. Quite a little firmware/software modifications are to be expected. Typically the **firmware buffering**, **data routing** in the firmware and **synchronization** in the firmware needs to be changed and then the **software should be modified to reflect firmware changes**. The standard reference design offloads snapshots without any temporal coherence, one channel after the other, without any phase relation." [39].

So two firmware solutions were explored to enable simultaneous sampling and triggering of a fixed number of samples per channel:

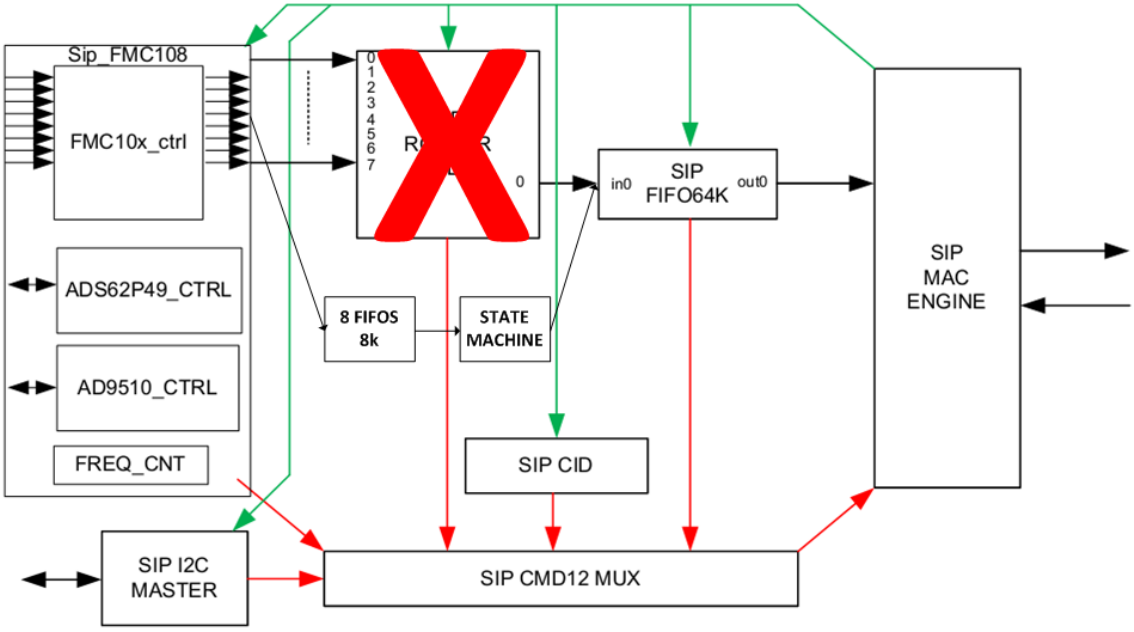


Figure 4.5: 1rst proposed amendment

1 - The first solution was more hardware based and it was to remove the Router 8-1, as seen at figure 4.5, and instead have 8 FIFOs of 8Kb, consequently, it would bear with bursts of maximum size of 4096 samples, they then link to a state machine whose goal is filling the FIFO 64K, first with the values of the 1rst ADC, then with the 2nd ADC, and so on until the 8th ADC. Thus, few modifications of the software will be needed, just a ReadBlock() from where we get bursts from know sizes from the 8ADCs. The 8 FIFOs are trivial FIFOs, Xilinx IP Cores may be used, where the signal READ from the FIFO will connect directly to "chX_dval". The state machine, expects to fill the FIFO64K when the 1rst FIFO8k is full, then wait for the 2nd and dump it and so forth.

2 - The second solution was simpler on the hardware side and more tricky on the software. The solution was to again remove the Router 8-1 and to connect the output of every channel directly to a modified FIFO 64K that is now a FIFO512K as seen at figure 4.6. The FIFO is altered to be 8 times bigger than the previous size to still allow the same sampling size for each channel and the write enable is a 'and' of the writing_enables from the 8 ADCs. The input bus is now a 8x16bit and the output is still 16bit.

At the software side, the C++ code should be modified to receive the samples interleaved for each channel and not first all the samples from the 1rst channel then all from the 2nd and so on.

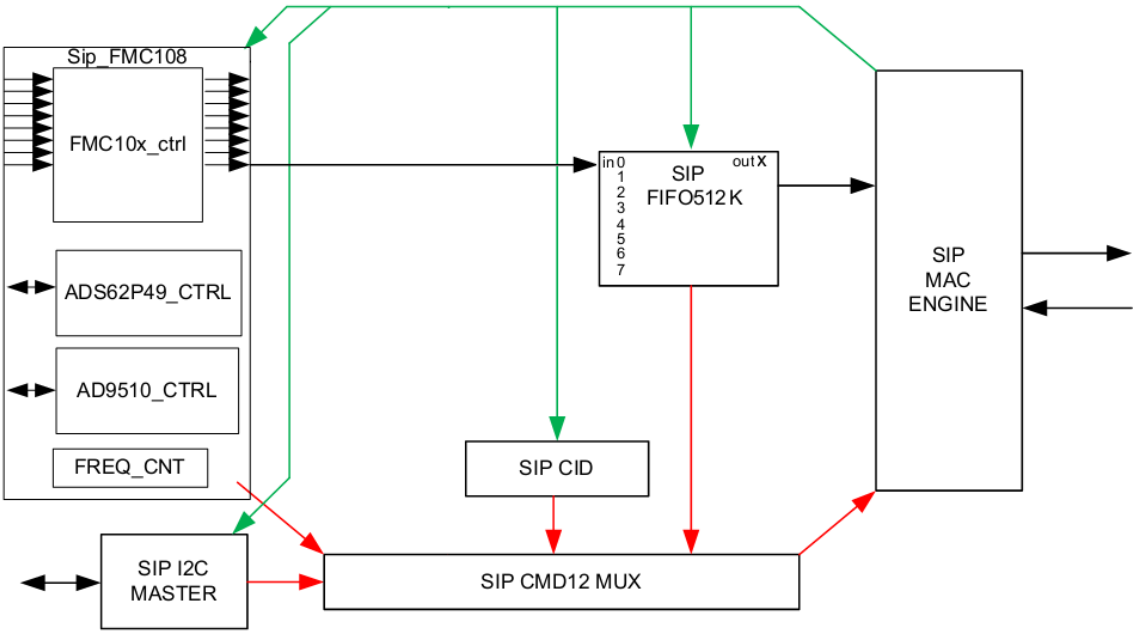


Figure 4.6: 2nd proposed amendment

As any alteration in the VHDL code takes a long time to generate a bit-stream (usually 15min) the 2nd solution was chosen because the debugging was more software oriented, with faster compiling and better tools of debugging that only Visual Studio offers.

So after implementation the data is sent to the computer the following way:

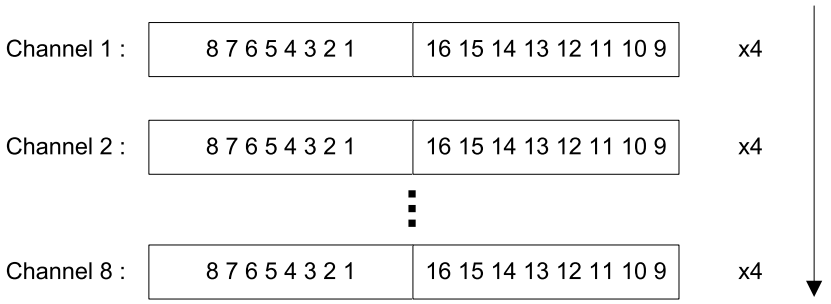


Figure 4.7: How the data is sent to the PC, the numbers represent the bit position of 1 sample

4.5.2 C++

So lets focus now into the C++ program written by 4DSP. We will first study it and then project changes to enable it to work with our altered firmware.

The original C++ program does the following:

1 - The Ethernet communication driver is initialized and a reset command is sent to the FPGA to reset the firmware and all the chips in the FMC108 board;

2 - A request is made to the FPGA to know about the memory mapping of the various chips and the *starts* identification;

3 - A command is sent to read by SPI the 2 ADT7411, that allows to know the temperature and state of the supply voltages in the circuit;

4 - Thermal limits are defined for a thermal auto-shut if those limits are reached;

5 - The clocktree is initialized, the values of the clock division are defined, the PPL is defined to work at 1250MHz, and the 4 clock outputs that will be used are enabled, also we choose if the clock that feeds the clocktree is internal or external;

6 - The ADCs are configured to work at high speed, with independent control and with a output at complement 2 (parameters like offset correction and gain are not defined);

7 - It's defined the values of each data bus delay. The 8 ADC data-buses are independent and have different lengths when they reach the FPGA, so there's a need to compensate those delays. In this case a unique delay system of Virtex-6, the IODELAYE1[40], is implemented for each channel, the max delay is of 32 and is started at 16. We can send increments from software to a maximum of 32, above this it comes back to 0. For 250MHz clock the delays set by 4DSP are of (10, 10, 10, 10, 20, 25, 20, 25). We test other values, and changing them make some of the data input came wrong. As each unit applies a 75ps of delay then in ns the delays are (1.95, 1.95, 1.95, 1.95, 0.3, 0.675, 0.3, 0.675);

8 - We communicate with the *star* responsible for reading the frequency of each clock;

9 - The number of burst and it's size is configured;

10 - Enable the channel that will be read;

11 - Arm the ADCs;

12 - Software Trigger;

13 - Ethernet FIFO is read and allocated into the PC memory;

14 - The data is written in 8 different files, one for each channel.

The program runs in called in a command line and have 2 inputs, the first one is to chose if the clock is internal or external(0- internal, 1- external), and other is used to select the Ethernet port that is used for communication.

Some alterations were needed to enable the sampling of the 8 channels at the same time. Before the step 10 we make a soft sync of the clock tree, this then guarantee that the output clocks are all in phase. In step 10, we enable the 8 channels and not only 1. Step 11 not only now arms all the ADC but also resets the input ADC FIFOs, this will avoid the problems that we were having with sample delays between channels, and it will be explained in detail later in this thesis.

Now, there used to be a function to write the data into the files (step 14), this function was a call to the 4DSP "ethapi.lib", it was erased, and a new one was written to allow it to work with the firmware changes implemented. It works the following way:

1 - Each sample comes coded in 2 bytes, the first byte that comes is the least significant byte, and the second is the more significant byte. So a sample is the sum of the first byte with the second byte shifted by 8;

2 - The samples come as discussed before, 4 in 4, so the first 4 samples (8 bytes) are from the 1rst channel, the next 4 samples are from the 2nd channel and so on;

3 - The function is run until the memory length used for this objective is full.

This completes the modifications on the C++ file.

4.6 ENOB of the ADCs

We need to test and characterize the ADCs that we got at our disposal in a way that will help us to evaluate the performance of the system. The ADC chip manufacturer details that the number of bits of the ADC that we are using in our project (ADS62P49) details that the resolution of the ADC is 14bit. But the number of bits is never the exact resolution that we can get from the ADC, this is due to noise in each sampling that can be due to a variety of factors, it could be due to thermal noise, clock jitter, sub-ranging errors and others. To know the real performance of an ADC a normally used parameter for characterize it is the ENOB (Effective Number Of Bits). The manufacturer once again tell us that the ENOB at 170MHz is 11bits. But this information is not really useful, we only have 1 frequency that was tested, and in the real life we don't have the ADC board that was used on the test, we got a 4DSP board that uses the same ADC but different components, layout and clock source. So it's important to characterize the ENOB and how to measure it.

4.6.1 ENOB

The ENOB may be considered as the number of bits of a perfect ADC whose quantization noise error would be equal to the total error from all the sources in the ADC under test. The ENOB is calculated by using the relationship for the theoretical SNR of an ideal N-bit ADC: SNR = 6.02N + 1.76 dB. The equation is solved for N, and the value of SNR is substituted for SINAD:

$$ENOB = \frac{(SINAD - 1.76dB)}{6.02}$$

If our signal does not achieve the full rage of the ADC we can then change the equation to:

$$ENOB = \frac{(SINAD_{MEASURED} - 1.76dB + 20log(\frac{FullScaleAmplitude}{InputAmplitude}))}{6.02}$$

SINAD is the Signal-to-Noise-and-Distortion and is the relationship between the signal and amplitude and all others spectral components including noise and harmonics but not the DC component. This parameter is mostly important in audio, and is measured in a certain frequency by generating a tone and measuring the samples to get the wanted signal + noise + distortion. Then a notch filter is used to remove just the exact frequency of the tone generated and the result is measured again, in this case we got only the noise + distortion. The SINAD is then the relationship between both:

$$SINAD = \frac{Psignal + Pnoise + Pdistortion}{Pnoise + Pdistortion}$$

So study an adaptation of this algorithm to our setup we first wrote a simulation code that try to emulate the method used in audio. The steps for it to run are the following:

1 - We generate a sinusoid at 60MHz and we sampled it at the sampling frequency of our ADCs, 250MSPS;

2 - Then we quantize the signal at N bits to simulate the quantization error created by the ADCs, as there is no other error in the simulation this will represent the effective number of bits (ENOB), so we choose 11bit as the manufacturer states in the ADC datasheet;

3 - We then run a FFT of the data to get the spectrum of the signal and we get rid of the first and last bin that have the DC component;

4 - We find the maximum bin number to know where the generated sinusoid is at the FFT;

5 - We determine the power spectrum of the data by squaring the spectrum;

6 - We define a span in bins for the input sinusoid, the majority of the sinusoid power should be contained in that span, we are using 200bins for a 25000 samples FFT;

7 - We estimate the noise+distortion floor power by making a mean of the power spectrum without the generated sinusoid span;

8 - In a duplicate of the power spectrum we replace the sinusoid span with the noise floor power;

9 - The signal power is then the sum of all the bins from the generated sinusoid span without the noise+distortion mean;

10 - The power of the noise+distortion is the sum of all the bins from the spectral power of the duplicate we modified;

11 - The SINAD is then calculated by $SINAD = 10 * log10(\frac{Ps}{Pnd})$;

12 - The ENOB is $ENOB = \frac{SINAD - 1.76 + 20log10(\frac{2^{N-1}}{abs(max(data))})}{6.02}$.

The result is the following:

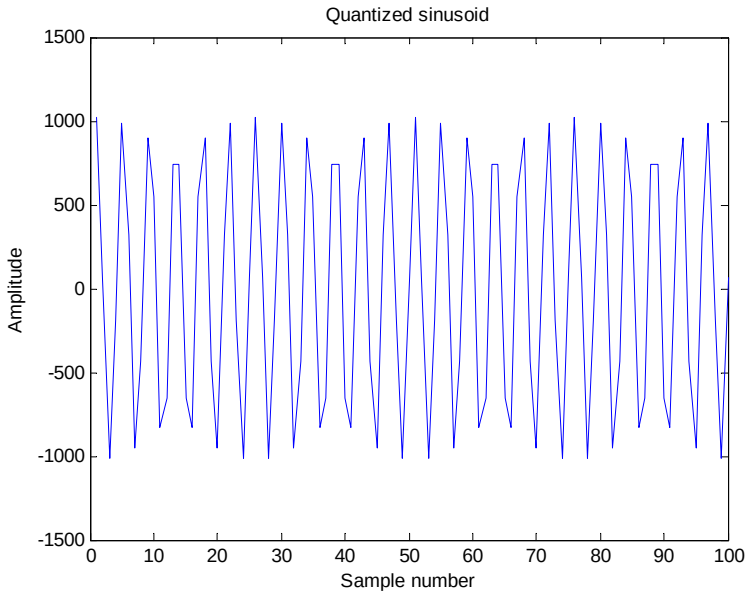


Figure 4.8: Quantized sinusoid

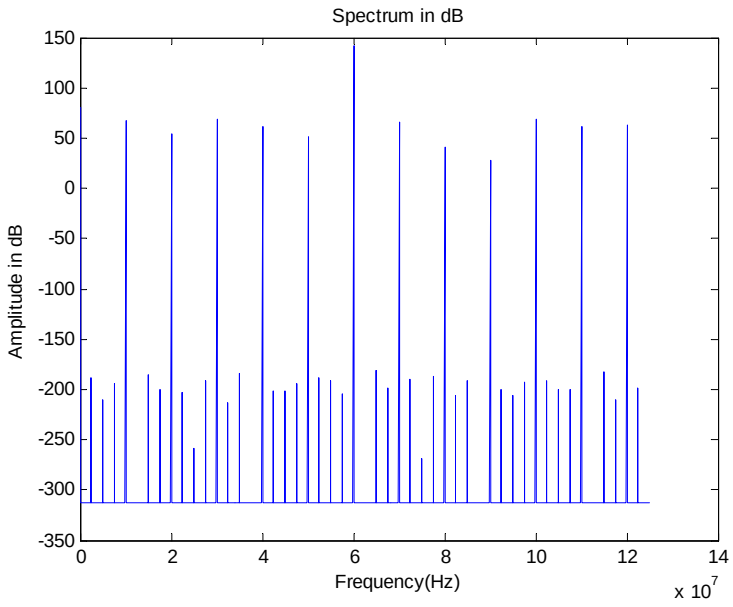


Figure 4.9: Signal spectrum in dB

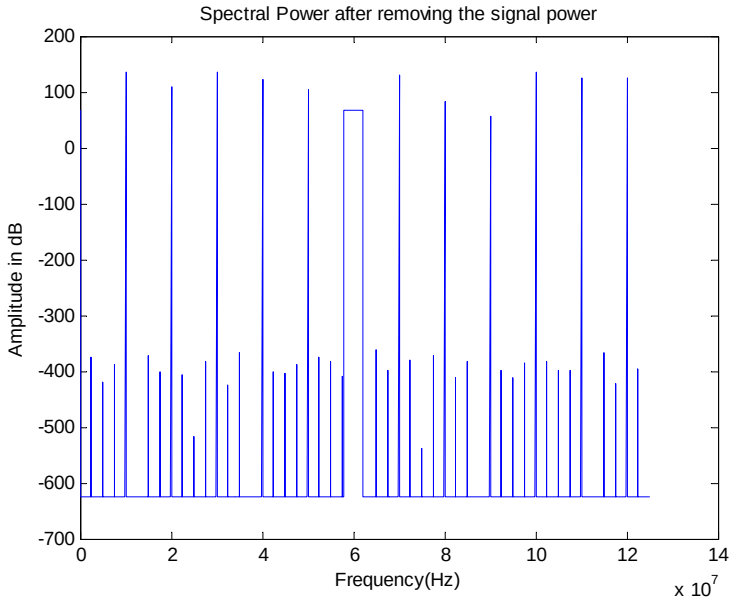


Figure 4.10: Spectral power after removing the signal power

The result is that we got a practical ENOB of 10.918bits when we should get 11bits. So it's a quite nice approximation of the ADC ENOB.

We later use the same algorithm in a practical test in a frequency sweep with the real system and the results with the first ADC are the following (the other ADCs have similar results):

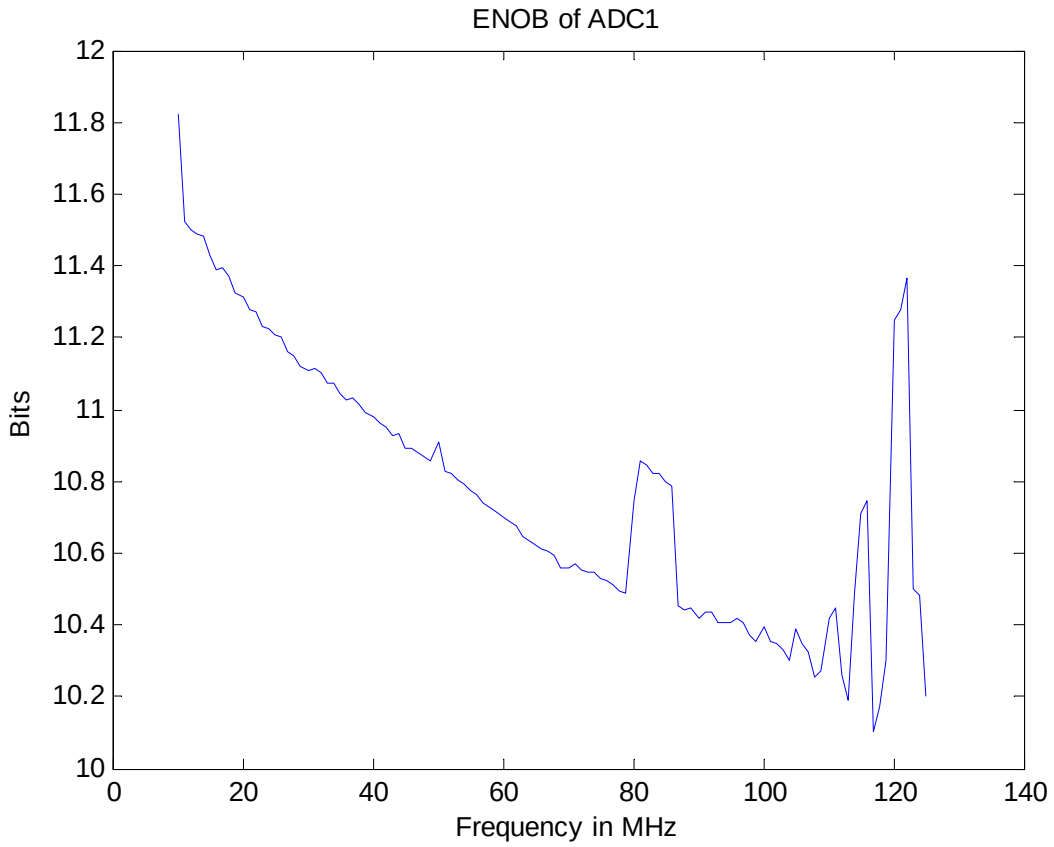


Figure 4.11: Real system ENOB of the first ADC

There's a lot of noise at the measurement but we can still take conclusions with it. So lets then compare with the ENOB chart published by 4DSP for the FMC108 board.

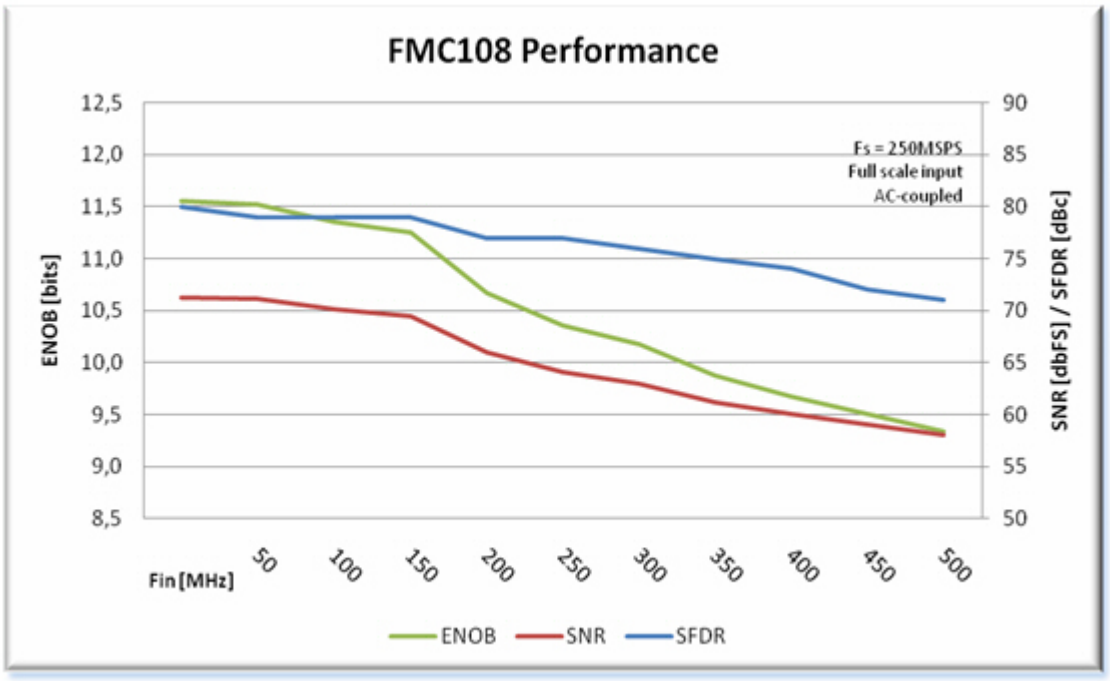


Figure 4.12: Publish ENOB, SNR and SFDR by 4DSP[20]

The results from 4DSP show a ENOB of 11.5 bits for 10MHz our results show a 11.6bits for the same frequency, a very similar result. At 125MHz the 4DSP results show a ENOB of 11.3bits and our results show something that oscillates from 10.1 to 11.4 bits. In between the practical results show a lower ENOB that the one from the manufacturer.

4.7 Measurements for analysis of the phase relationships of the 8 channels in order to support real-time DSP

4.7.1 Solve the problems to implement the least possible inter-channel phase

The first problem we have to deal with was that the phase relationship of some channels was near 100 samples more than the rest of the channels. After some review of the datasheets something appear to have a solution to the problem the clock tree AD9510 had a soft sync signal, that put all clocks in the same state and then let them oscillate again. After changing the C++ code to write by SPI to the AD9510, this seemed to solve the problem, and now we had only 10 samples of delay in some channels. But it still were not enough.

As the board does not offer any test pins for a analysis of the clocks or any other ADC buses with a oscilloscope or a logic analyzer, we were limited to use only the Chipscope tool. By debugging with ChipScope we find out that when we called the C++ program that interacts with the FPGA and calls the 4DSP Dynamic-Link Library (DLL) the first command that it would send was to make a firmware reset as well as a reset for all the chips in FMC108.

After knowing this a larger analysis was needed. We wanted to check what really happen to the ADCs clocks after a reset, so we generate a clock at the FPGA with the sole purpose to analyse the ADC clocks, as these clocks work at 250MHz and by the Nyquist theorem we

needed a sampling clock of at least two times that frequency, we decided to use a 500MHz because this FPGA is not usually used for such high frequencies and we decided not to risk more than 500MHz, and just repeat the tests to confirm that the sampling is not made at the clock rising or falling. This block then uses the FPGA board oscillator that works at 200MHz and using a PLL generates a 500MHz clock with the following characteristics:

Duty Cycle(%)	Pk-to-Pk Jitter (ps)	Phase Error (ps)
50	82.193	89.971

Table 4.6: Characteristics of the 500MHz clock used to sample the ADC clocks

The Chipscope is then built with the generated clock as the sampling clock, and for the trigger the reset signal is put back down to 0, because the clock tree chip will only reset at the transition of 1 to 0 at the reset signal. The results are the following:

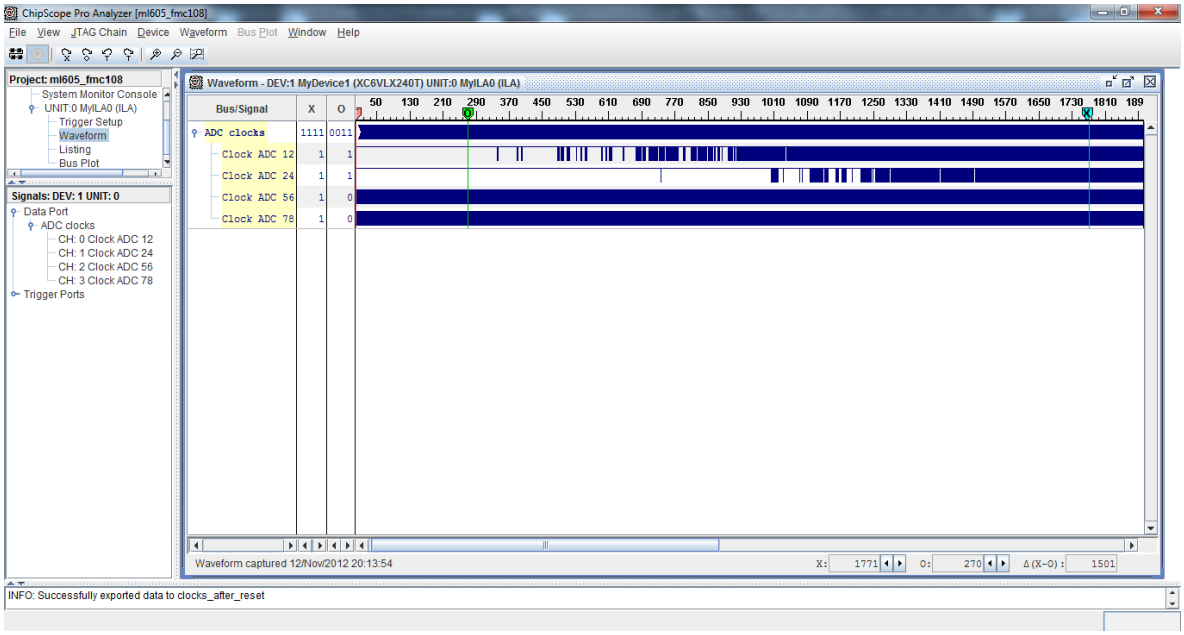


Figure 4.13: ChipScope GUI showing the 4 different clocks used in the ADCs, blue rectangles appear when the clock is oscilating

So the problem looks clean now, the AD9510 is reset and a transient state of the output clocks appeared, not all the clocks start at the same time and some and they start in a non-stable way. Let's take a closer look at the results:

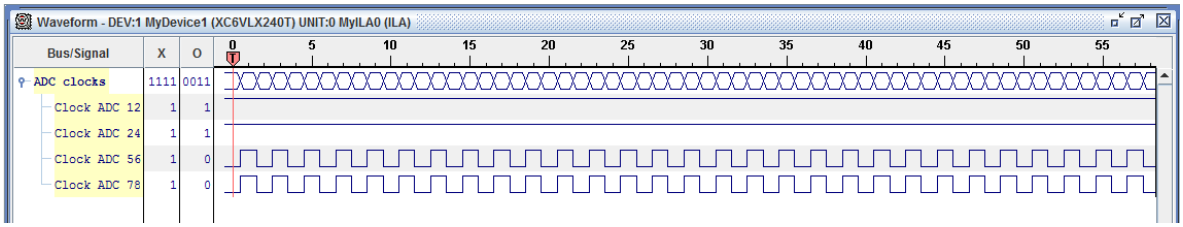


Figure 4.14: Just after the reset the clocks of the last 2 ADCs look fine but the other 2 are not oscillating

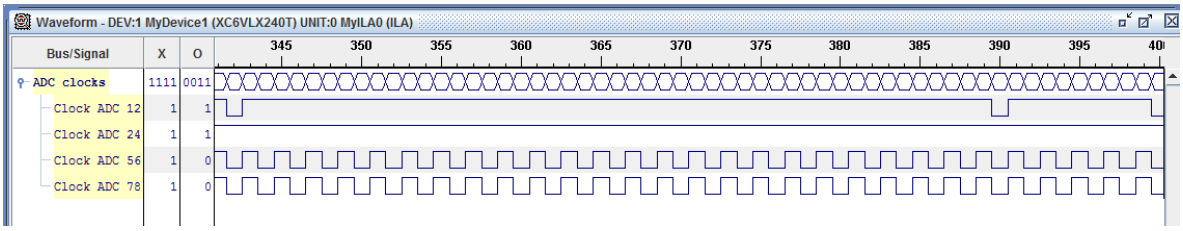


Figure 4.15: Then the clock of the first ADC starts oscillating but its not stable

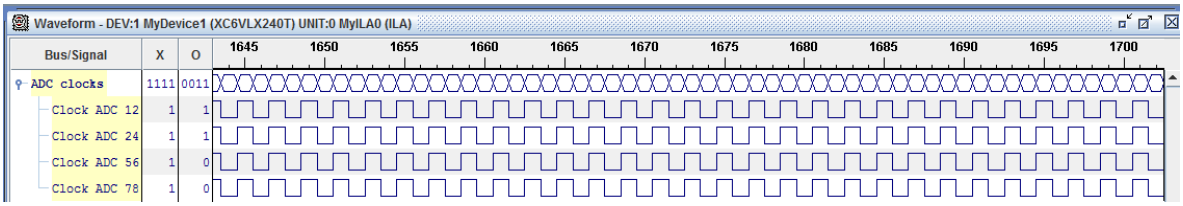


Figure 4.16: All the clocks at some point start to behave normally

Lets gather then the data in a graph to understand better the issue:

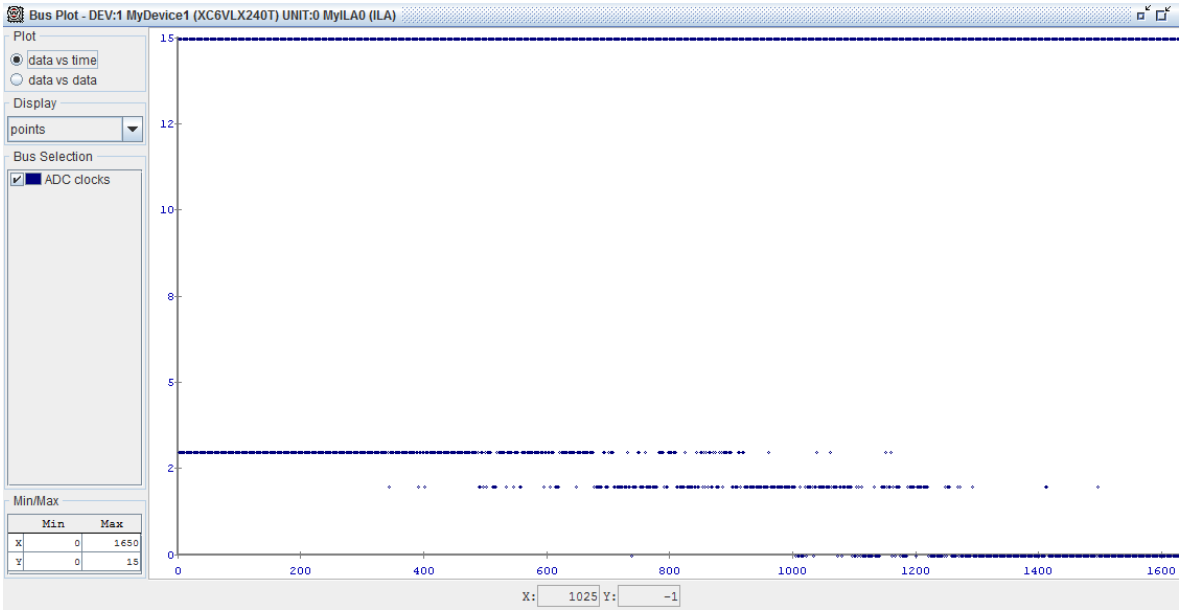


Figure 4.17: Graph of the clocks sampled

As we can see by the graph only after 1500 samples the clocks are in sync. As each sample is take at a frequency of 250MHz it takes 300ns for the clocks to stabilize.

So how this affect our system? As the first line of input FIFOs we have the write_enable activated after 32 ADC clocks and the write clock of each FIFO is the clock of the ADC feeding that FIFO, some of the FIFOs are then being written before the others, this created the delay. This is due then to the fact of the AD9510 does not start all the clocks at the same time. This information is not present on the datasheet of the clocktree.

The solution found was when the C++ program was called it would make the reset, boot all the system and then when we "arm" the ADC a reset signal was sent only to the input FIFOs. At that time the clocks are already stable. This modification solved the problem.

There was still a delay in chX and Y, so we introduce a sample delay in those 2 channels. This sample delay was made to be easily changed in the VHDL code, as for different frequencies of ADC clock this delay may be different, this is caused by different paths in the board for each ADC clock input, at different frequencies we may have delays in different channels, this delay is never superior to 1 sample, because if the clock phase relationship between channels is above 360^{o} the clock repeats itself, so it was like nothing had changed.

4.8 Measurements of the relative phases for the frequency range to be used

After all the problems solved, we want to evaluate the relative phase of all the FMC108 ADCs using the first one has a reference. For this purpose we will measure a generated frequency sweep from 50MHz to 80MHz and splitted with a power splitter to all ADC channels. The power splitter is measured before with a Vector Analyzer to remove it influence in the measurements. The measurement of the S parameters is made with 301 points.

After we have the splitter measured we use it to divide the signal to all ADCs and generate

a sweep again with 301 points between 50MHz and 80MHz.

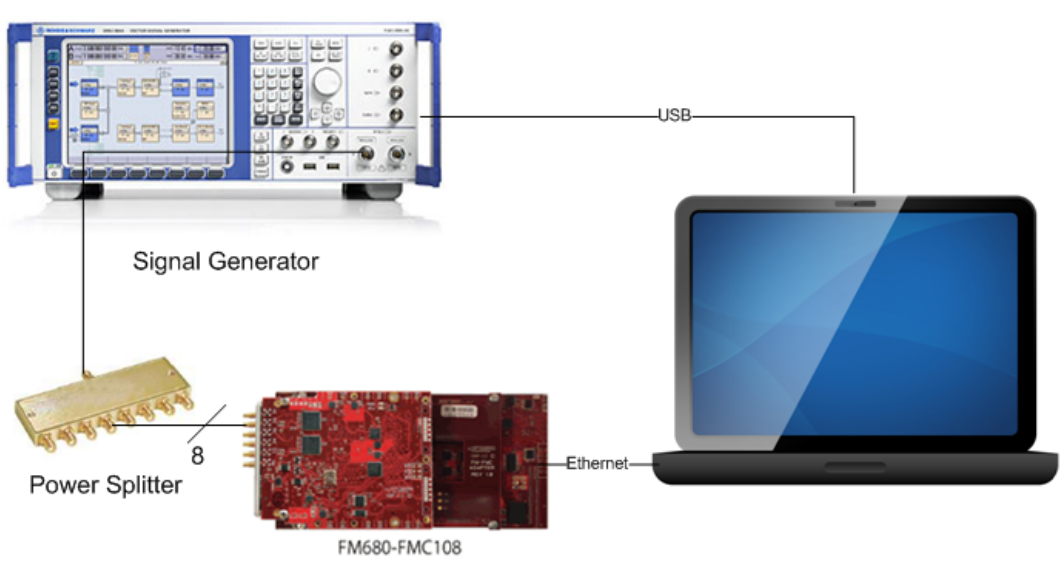


Figure 4.18: Experimental Set Up

4.8.1 Material used

The material used for this report is:
- Agilent E8361C (Network Analyser)
- VNA calibration kit
- SMU200A signal generator
- GPIB/USB adapter
- ZFRSC-183-S+ power divider[41]
- ML605 FPGA and FMC108 ADC bank
- CBL-3FT-SMSM+ (Coaxial cable of 50Ω with 3ft length)
- 8xSCA30316-18 (Coaxial cable of 50Ω SMA male to SSMA male 1.5ft length)[42]

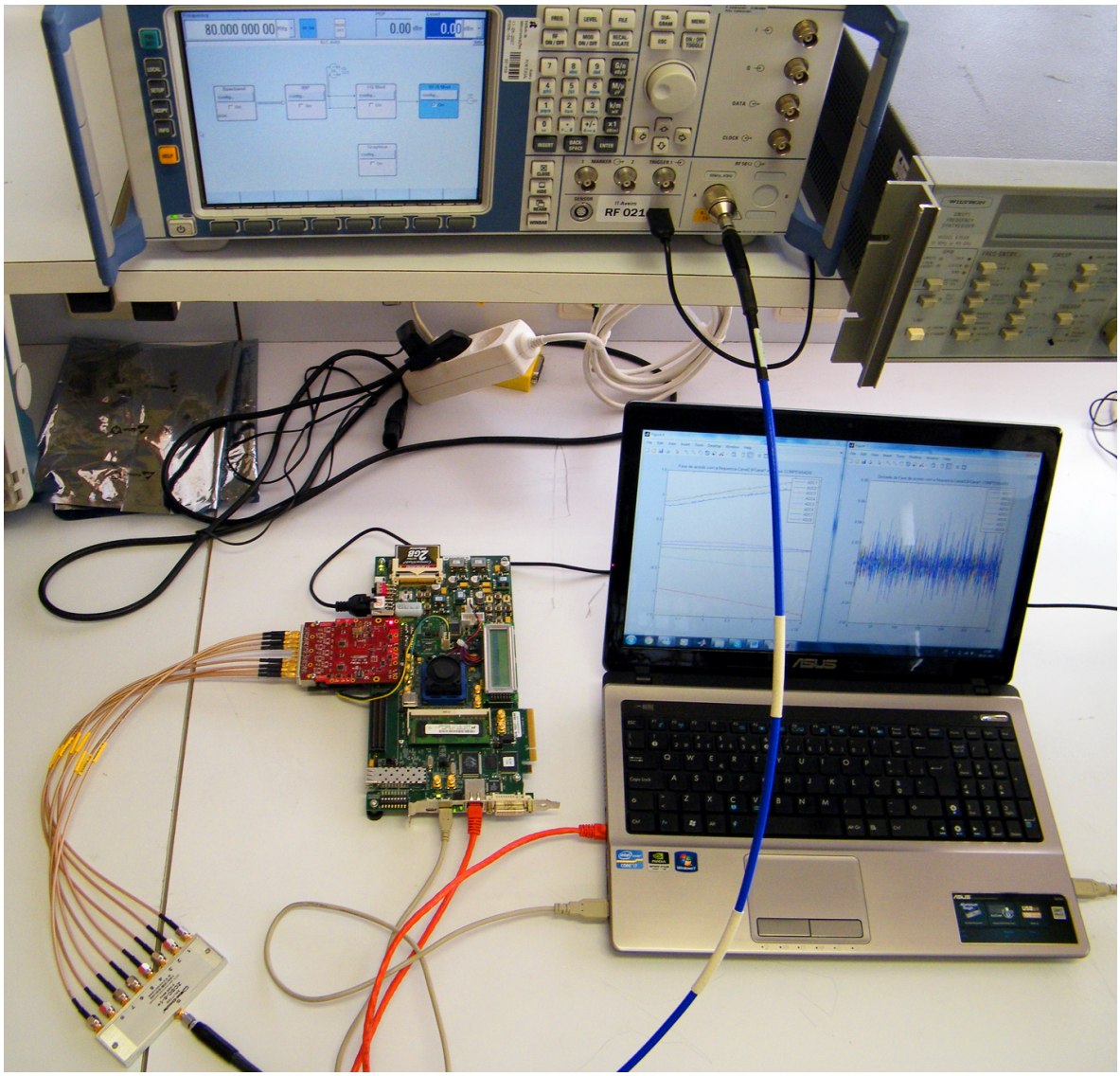


Figure 4.19: Practical setup for phase relationship measurement

4.8.2 Data Processing

To cancel the phase and amplitude imbalance from the power splitter, its transmission responses were measured, with a E8361C network analyzer, and used to equalize the above mentioned ADC frequency sweep responses by using:

$$\varphi(\mathrm{f}) = angle(\frac{V(f)}{S_{21}(f)})$$

Where f is the frequency of the sinusoid in which the ADC samples were acquired, V is the complex value of frequency response derived from the ADC samples and $S_{21}$the forward transmission coefficient of the power splitter acquired from the network analyzer.

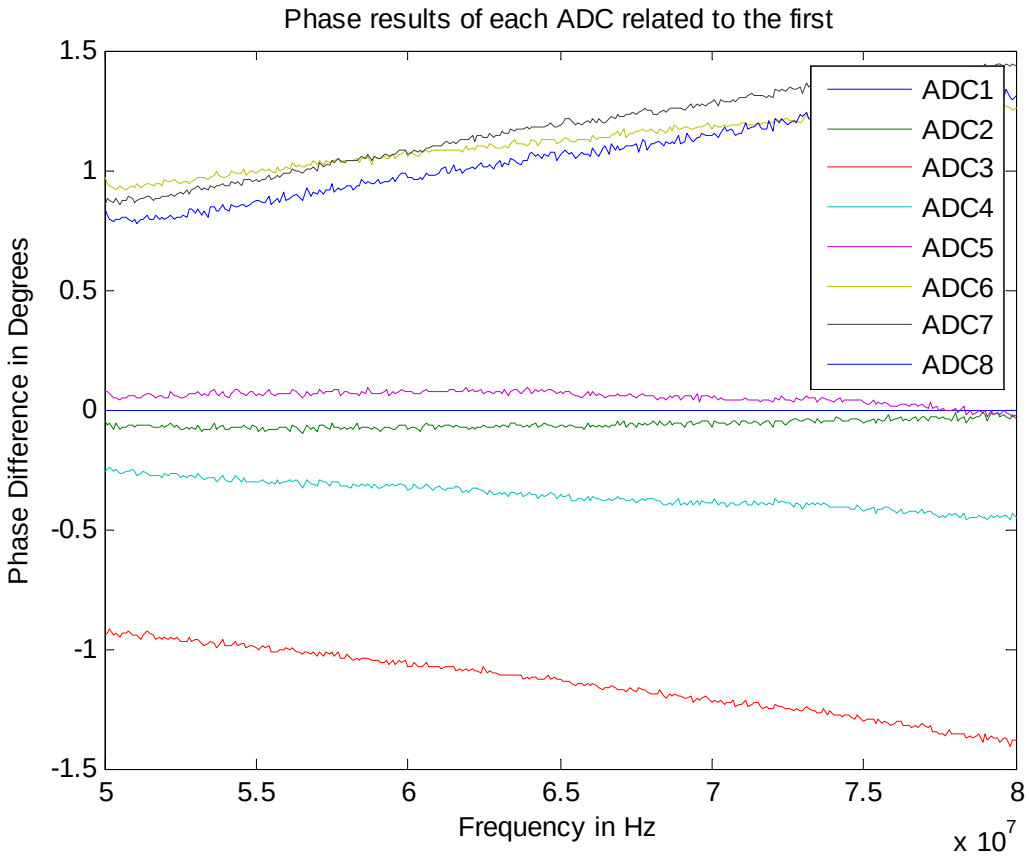


Figure 4.20: Phase results of each ADC related to the first

The resulting phase responses are plotted in Fig. 4.20, and it can be seen that some A/D converters present phase offsets a little higher than one degree for the range of frequencies in consideration.

4.9 Reconstruction of the original signal

Now that we have measured the phase relationship of the ADCs, we are now ready to try to reconstruct the signal that passes trough the filter bank. For this we need to measure the signal coming from the ADCs and offline implement the digital filter synthesis bank.

4.9.1 Creating the digital synthesis filters

The digital filter bank is synthesized from the measurements made from the analog filter bank with the Network Vector Analyzer, and the method used was the following:

The reconstructed signal $X(e^{j\omega})$ is the Fourier transform of input signal X splited in M sub-bands by the H_k band-pass filters. Each sub-band signal is then sampled at $1/M$ the effective Nyquist sampling rate $1/T_S$ of the input signal. Then, it is filtered by the digital synthesis filter bank F. Therefore, the Fourier transform for the output signal is as follows:

$$X(e^{j\omega}) = \frac{1}{MT_s}\sum_{p=-\infty}^{+\infty} X(\frac{j\omega}{T_S} - \frac{j2\pi p}{MT_s})\sum_{m=0}^{M-1} H_m(\frac{j\omega}{T_S} - \frac{j2\pi p}{MT_s})F_m(e^{j\omega})$$

Where p represents the aliasing components. As we want the filter bank to achieve almost perfect reconstruction, then the output signal needs to be a delayed version of the input signal and the distortion needs to be:

$$T(e^{j\omega_q}) = \frac{1}{MT_s}H(e^{j\omega_q})F(e^{j\omega_q})$$

Where $j\omega_q$ represents a set of arbitrary frequencies with q=0,1,... Q-1 Then the coefficients of the FIR filters can be evaluated from:

$$f_m = W^+F_m$$

Where$W^+$is the pseudoinverse of the Fourier transformation matrix W.

4.9.2 Material used

The material used for this report are:
- Agilent E8361C (Network Analyser)
- VNA calibration kit
- SMU200A signal generator
- GPIB/USB adapter
- Analog Analysis Filter Bank
- ML605 FPGA and FMC108 ADC bank
- CBL-3FT-SMSM+ (Coaxial cable of 50Ω with 3ft length)
- 6xSCA30316-18 (Coaxial cable of 50Ω SMA male to SSMA male 1.5ft length)
- ZX30-9-4-S+ (Directional coupler)

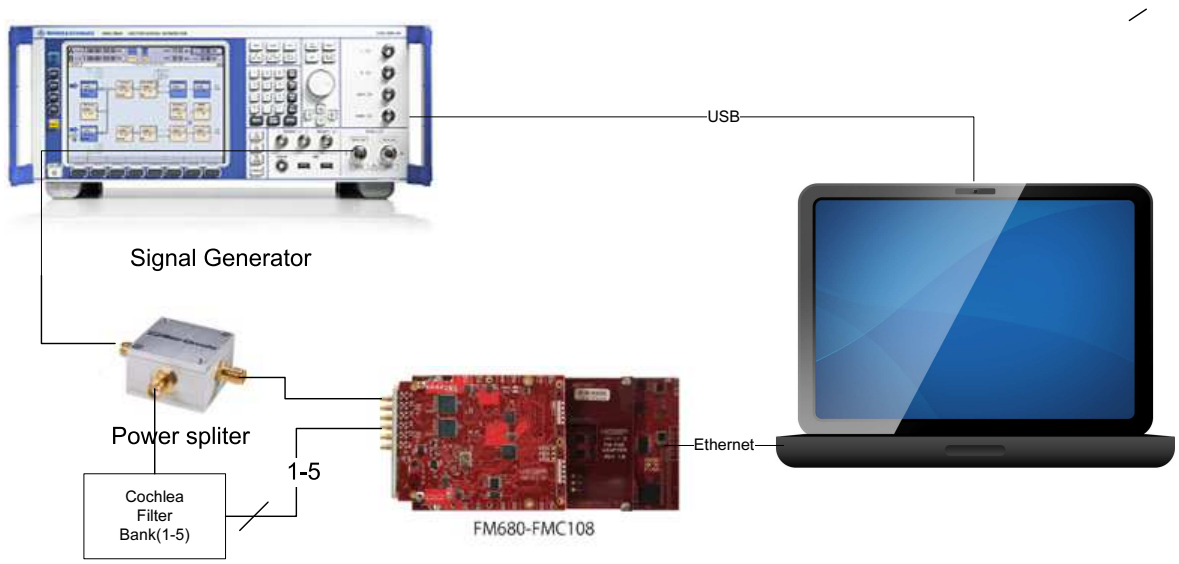


Figure 4.21: Experiment with the analog filter bank

4.9.3 Practical procedure

For this procedure we only used 5 channels of the analog filter bank, they are connected to 5 ADCs and one extra ADC is used as a reference of the original signal. The sampling frequency was of 10Mhz for each channel.

First the filter bank was measured with the network analyzer.

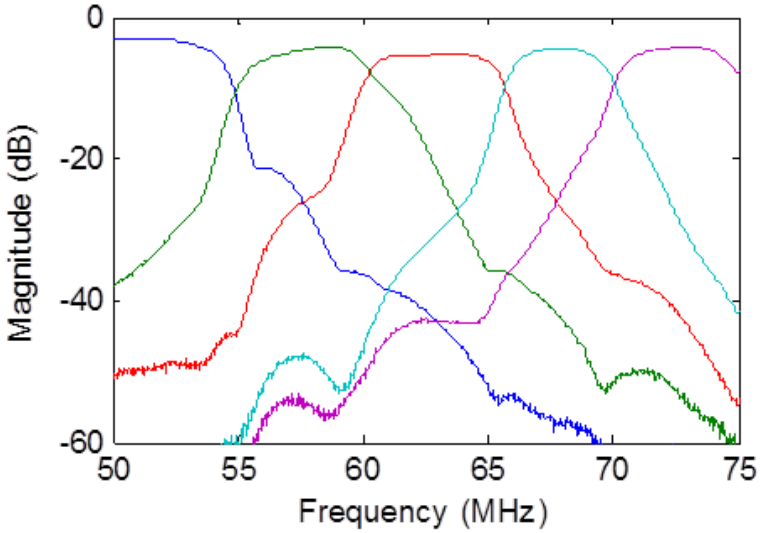
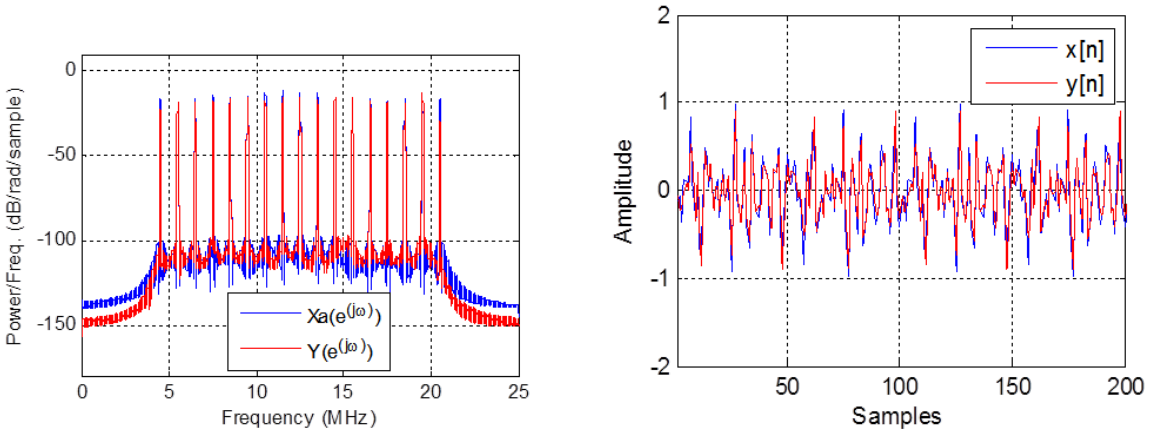


Figure 4.22: Complex magnitude of the S21 of the filter bank

A multi-sine signal with 17 evenly spaced tones from 1MHz and centered in 62.5MHz with random phase. This signal was generated in an E4433B signal generator. The received signal is then oversampled by the ADCs and sent to a computer where the samples are then used with the algorithm described previously.

and beyond.



(a) Original signal and reconstructed signal spectrum (b) Original and reconstructed signals in time domain

Figure 4.23: The results of the reconstructed signal

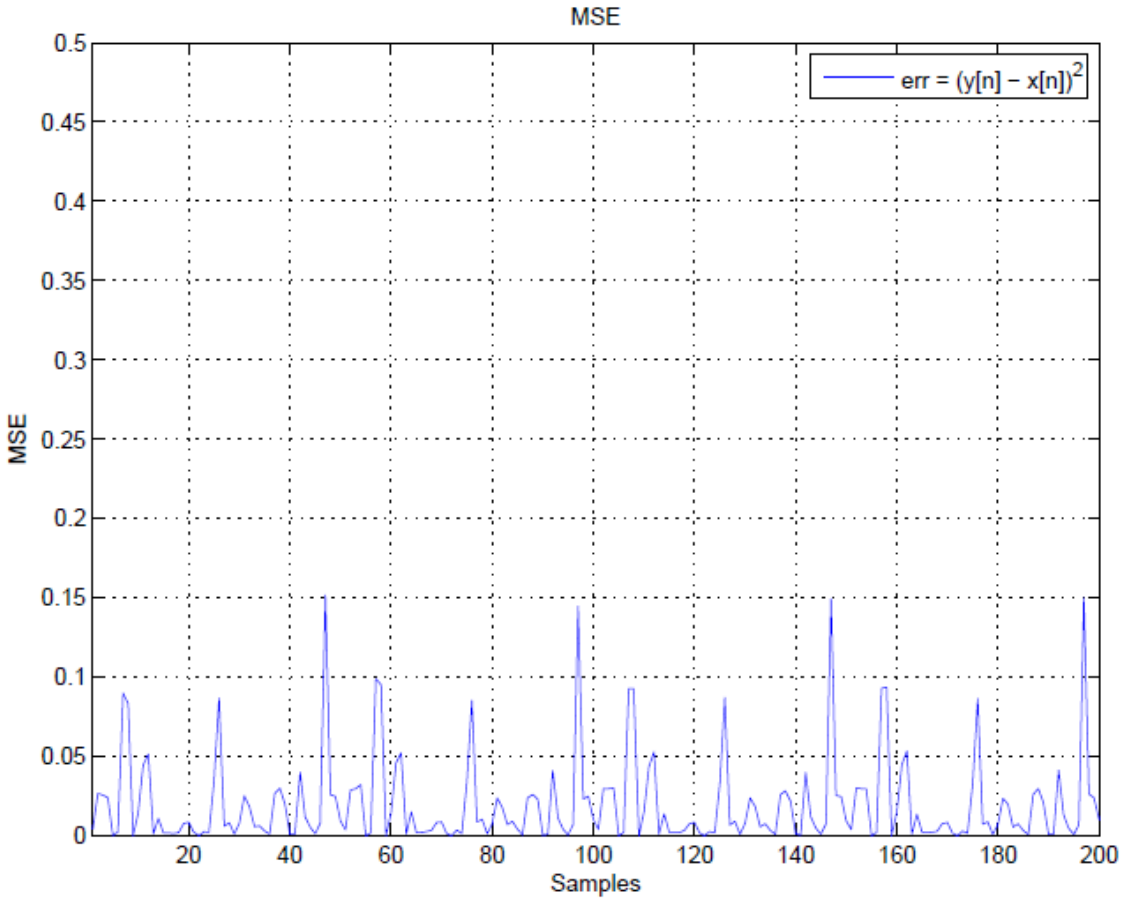


Figure 4.24: Mean Square Error of the reconstructed signal

By the reconstructed signal we can see that the signal is identical to the original but with some error. This source of errors can have 3 different origins:

- ADC-wise errors, that can be because of the ENOB or the phase difference between channels that will difficult the reconstruction;
- It can be because of the algorithm used that can have some imperfections;
- The measurement of the Cochlea is not perfect.

Anyhow these results are very good and are more than enough to us to proceed to the next step in this thesis.

Chapter 5

Realtime implementation of the digital Syntheses Filter Bank on FPGA

5.1 Implementation of FIR filters

After all the firmware was successful modified to operate with simultaneous sampling and low inter-phase relationship, and tests were made to confirm that is possible to reconstruct the signal, we are now ready to implement real-time hardware that uses the data from the ADCs.

To equalize the response of the analog filter bank we are going to need to implement real time filters on the FPGA. There are two kinds of digital filters: IIR filters and FIR filters. We are going to use FIR filters not only because they are always stable but also because they are implemented with MAC (multiply-accumulate) units, and a Virtex-6 FPGA have specific DSP48E1 slices that implement high performance MAC operations that will give us more freedom for choosing the frequency of operation of the filters, their configuration and implementation.

So first we are going to implement the Xilinx LogiCORE IP FIR Compiler v6.3 with 10 coefficients, single rate, with the same data rate as the ADCs, with 16bit integer signed coefficients, the input data is from the ADC so is signed and at 14bit width. As for the output, to avoid major changes in the architecture of the hardware, it's going to be a signed 16bit width data. This way we are going to take advantage of the extra 2 bits that 4dsp was padding with 0s in the least significant bits to create frames of 64bit (4 samples for each frame).

We now need to choose where the filters shall be placed on the firmware, we decide do put them just after the IDDR block where the double data rate signal is passed to single data rate, in this location each ADC chip has their own clock so we need to choose the clock of the ADC that is being used also for the filter. This signal, that we are filtering, is the cha_sdr signal, we remove the padding that is no longer needed at the FIR input and the output of the filter will keep the same route as before to the first FIFO 16bit.

For testing we edited the FPGA VHDL in such a way that we cut off the input from the ADCs and created a sub-circuit only to create a unitary pulse every 32 clocks, this way, as the input is now predictable and unitary, we can debug with the output. If the output is exacly the coefficients of the filter and repeated every 32 clocks then the filters are indeed working. After the necessary debugging the filters where working as expected. This debugging sub-circuit was still used for the reconfigurable filters, to confirm that the change of coefficients

was working as expected.

5.2 Implementation of reconfigurable FIR filters via Serial Port

We kept the hardware as described in the previously subsection, with the input signal being the one generated by the pulse generator created by us.

As we then want FIR filters to be programmable to enable real-time filter adaptation techniques, we need to make the coefficients programmable. This is allowed from the Xilinx LogiCORE IP FIR Compiler v6.3, as it permits real time reloading of the coefficients.

To avoid bidirectional Ethernet communication while data burst is being transferred from the FPGA to the computer as it wasn't recommended by 4DSP because the Ethernet API was pretty basic, we decided to implement a secondary protocol to communicate with the board, this was implemented by the UART port on the development board. We will then use this port to reload new coefficients on the hardware. For communication we used also MATLAB. We also want to choose the coefficients and change them at any time for any of the filters, so the protocol that will be implemented needs to allow that.

5.2.1 Studying the reloadable coefficients on the FIR Compiler

We are using the Xilinx FIR Compiler v6.3 that enables reloadable coefficients. It works the following way:

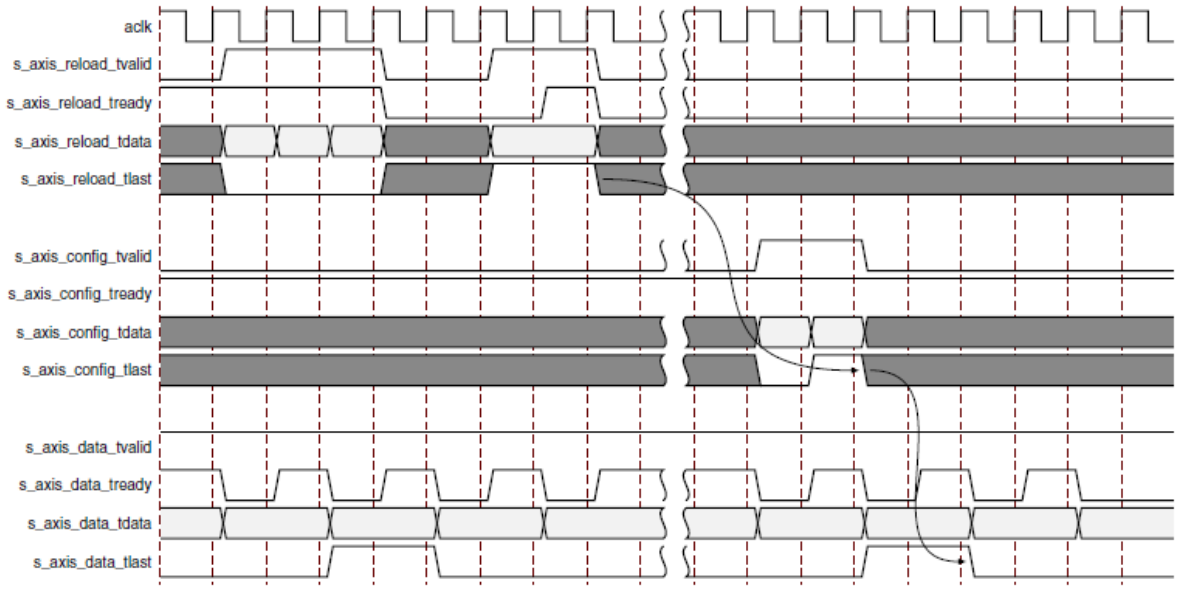


Figure 5.1: Interface timing for the reload of the filters[43]

So first it waits for the signal of the FIR Compiler s_axis_reload_tready be at '1', at that moment we are ready to configure it, so we activate the s_axis_reload_tvalid and put the data of the first coefficient in the s_axis_reload_tdata bus, each clock-cycle is then a different coefficient being written, if the reload_tvalid becomes '0', the data will not be read and we will need to wait for it to become '1' again to proceed with the reloading.

The arrow shows the situation of when we reach the last coefficient, in that case we should activate the s_axis_config_tlast bit. The second arrow then show a standard data streaming after the reload.

5.2.2 Serial Protocol

To implement the serial communication we need a protocol so we may know what filter we want to change, and what kind of change is it. We created a very simple protocol to take care of that. As we only want to send data to the FPGA, because we already are receiving all the data needed by Ethernet, this protocol is only 1-way, from the PC to the FPGA.

Each instruction is a 32bit frame, the 16 most significant bits are the number of the filter we want to change (in the actual state of the hardware we can only choose 1 of the 8 filters that are implemented one for each ADC channel), the 16 least significant bits are the coefficient that will be putted on that filter. As this is implemented using the RS-232 port, that only allows 1byte transfer each time[44], the first byte we send is the most significant 8bits the next one is the next most significant byte and so on until 4 RS-232 frames are sent.

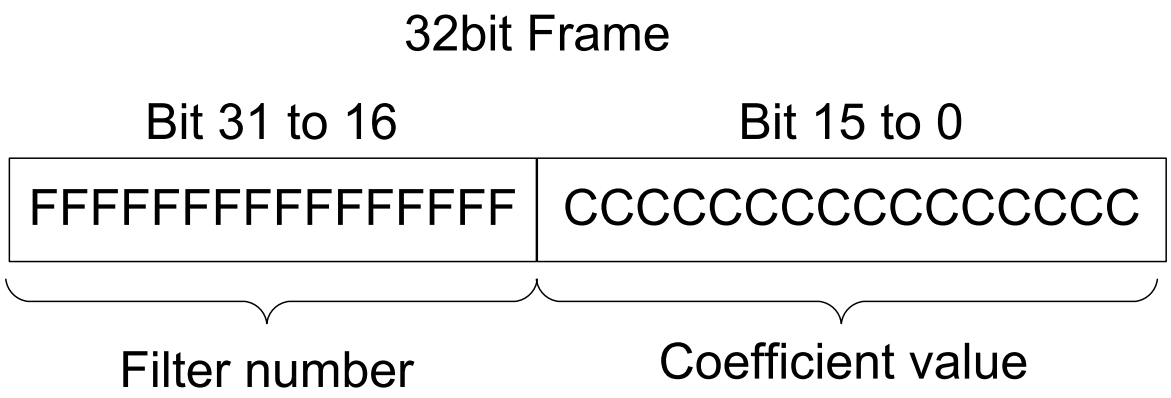


Figure 5.2: Protocol main frame structure

As each filter have more than 1 coefficient, the user need to send 'n' frames, each frame will then replace a coefficient by the following order: the first frame will replace the 'n' coefficient, the second frame will replace the 'n-1' coefficient and so on. This is due to the reloading order at the FIR COMPILER, that starts with the last coefficients first, and the order can't be change by the user.

5.2.3 Implemented Hardware

To implement these kind of filters in a FPGA first we need to have a RS-232 controller. We use for the purpose an open core that will work as a black box, it provides a data bus and a Ready signal to know when the data is ready to be read. The clock provided to the core is derived from the system clock, and it provides a baud-rate of 115200bytes/s, works at 8 data bits for frame, 1 stop-bit, and no parity bits. The core it's connected directly to the pins of the chip that emulates a RS-232 port from the USB communication with the PC.

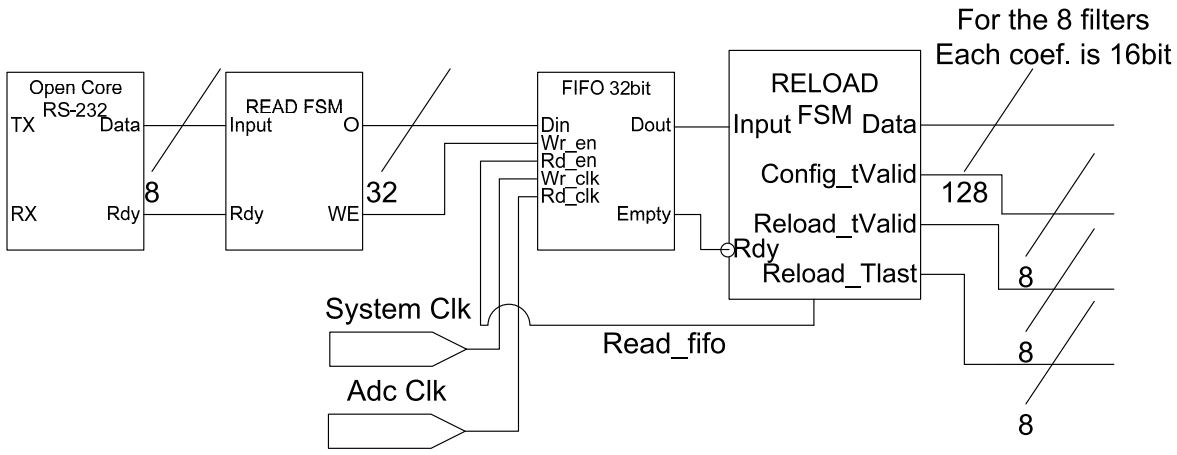


Figure 5.3: A schematic of the implemented system for the reloading filter coefficients

We then created a Moore Finite State Machine, the READ FSM that communicates with the RS-232 block and reads 4bytes of information and then translate it to a 32bit instruction. This 32 bit instruction is sent to a FIFO. This FIFO serves 2 proposes, it works as a clock buffer, between the System clock and the ADC clock domain in which the filters work, and can also store instructions to when what's ahead takes too much time to ask for new data.

The FIFO then outputs the data to the Reload FSM, this data is requested by the RELOAD FSM by the read enable bit in the FIFO. The state machine ask for the data if it detected that the FIFO is not empty. The state machine reads the instruction and interprets witch filter should be altered, activates the Reload_tValid signal for that filter and puts the first coefficient in the reload_tdata filter bus. It does this for the 'n' coefficients of the filter, if by some reason while writing at the coefficients of a filter, the FIFO is empty, witch means that the next instruction is not ready yet, in the next clock cycle, the Reload_tValid is disabled. While sending the last coefficient the Reload_Tlast is activated, in the next clock cycle is again disabled and the Config_tValid is activated for 1 clock cycle. This ends the process for one of the filters.

Part V

Conclusions and Future work

Chapter 6

Conclusions and Future work

As the results show this implementation was able to reconstruct the input signal with some error. To reduce this error we could use the FPGA implemented FIR filters to compensate the phase offset of the ADCs based on the channel with the most advanced phase. The actual filter implementation enables the phase compensation of the ADC channels but some extra work is still needs to be done to create the digital synthesis filters into the FPGA.

While the constrains where not met in the simulation tool the system still proved to be robust for the 250MSPS of sampling rate for each channel even with the altered firmware.

All the firmware only occupy 11% of the slices of the FPGA, what enables for even more features to be implemented in the future.

The reloadable FIR filters that were implemented work in real time at 250MSPS for each channel, and so, any other component at that frequency of operation that we want to implemented to process the data incoming from the ADCs would probably work as well using the same FPGA.

In conclusion, the system looks very promising, if any band is getting jammed with high power signals it still allows for other frequency bands to operate normally without saturating that band ADC. Also, each band allows for different gains, witch improves the overall dynamic range of the system. We can safely say that this architecture seems the way to go for the next generation radios.

For future work we want to: implement polyphasic filters with the same reloadable structure as those implemented for the digital synthesis filter. Enable unlimited sample streaming that would need a lower data-rate of data or an alternative to the ethernet communication that is used, because only a transfer of a limited number of samples is possible with this project, as a 2Gsps at 14bits each sample, stream is impossible via gigabyte Ethernet. Also we would like to implement a blind equalization system by creating a feedback via the filters coefficients and having a knowledge of the input signal, for even better results at the signal reconstruction.

Part VI

Annexes

Chapter 7

Annexes

7.1 Filtering

The filtering is an operation that lets you change the frequency behaviour of the input signal. In this thesis we split the input signal into M subsequent bands though the means of an analog channelizer with the same cut-off frequency so each channel of the channelizer is a pass-band filter for a certain frequency band.

The Fourier transform of the filter is:

$$Y(e^{jw}) = X(e^{jw}) * F(e^{jw}) \tag{7.1}$$

7.2 Decimation

The decimation is an operation of reducing the sampling frequency. This is created by discarding a value of the signal every M samples . This will be the operation that will introduce aliasing in the system so we'll focus more on it.

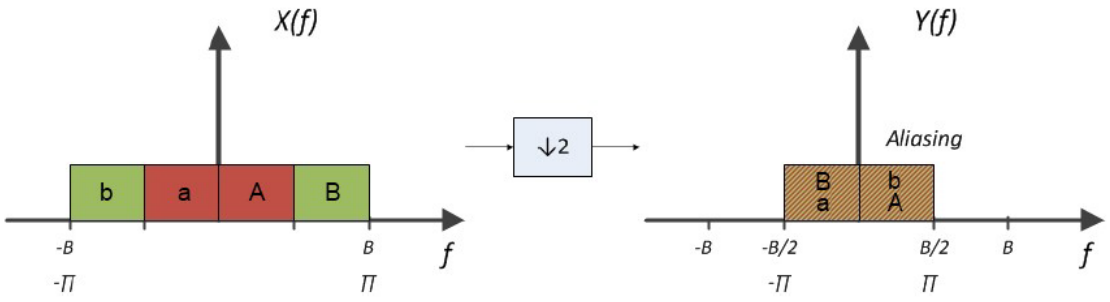


Figure 7.1: Aliasing provoked by decimation of 2

As an example we are going to study the simplest way of decimation. In a decimation of 2 we decrease the sampling frequency by half.

To make it easier to understand the effect of decimation by 2, we can say that the decimation is equivalent to alternately add and subtract the input signal itself and then divide by 2,

$$\begin{cases} x(n) = \{x(0), x(1), x(2), x(3), ...\} \\ x(n) * (-1)^n = \{x(0), -x(1), x(2), -x(3), ...\} \end{cases}$$

$$y(n) = [x(n) + (-1)^n * x(n)]/2 = \{x(0), x(2), ...\} \tag{7.2}$$

Whose Laplace transform of the output signal is:

$$Y(e^{jw}) = X(e^{jw}) + X(e^{j(w-\pi)}) \tag{7.3}$$

With the following Z transform:

$$Y(z) = X(z) + X(-z) \tag{7.4}$$

,with $z = e^{jw}$ and $e^{-j\pi} = -1$

7.3 Interpolation

Interpolation is an operation to increase the sampling frequency. Interpolation by 2 is created by adding between each value a new signal sample of value 0.

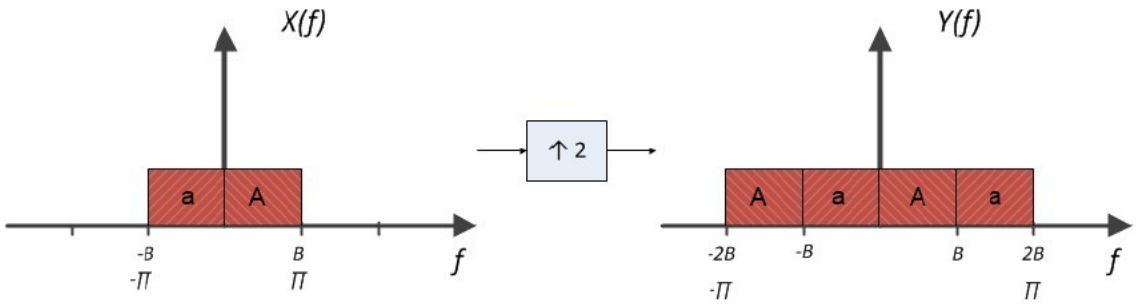


Figure 7.2: Changes in the spectrum of the signal caused by the interpolation

The Fourier transform of the interpolation by 2 is:

$$Y(e^{jw}) = X(e^{j2w}) \tag{7.5}$$

7.4 Equalization test with USRP for two analog filters, with the use of VNA measured filters in the frequency range to be used

This design is intended to construct a system based on the module TVRX2[45] connected to the USRP1[46] both from Ettus to be able to acquire, at the same time, two outputs of an RF filter bank. The original signal must be recovered digitally by using signal processing techniques that aim at a perfect reconstruction. We want to:

- Make measurements of filter bank in order to then calculate the coefficients of the synthesis filters that allow perfect reconstruction of the signal;
- Development and simulation in Matlab code that allows to know the frequency characteristics of the filter bank channels, to later obtain the analysis filter;
- Real time operation on the USRP;
- Validation of results.

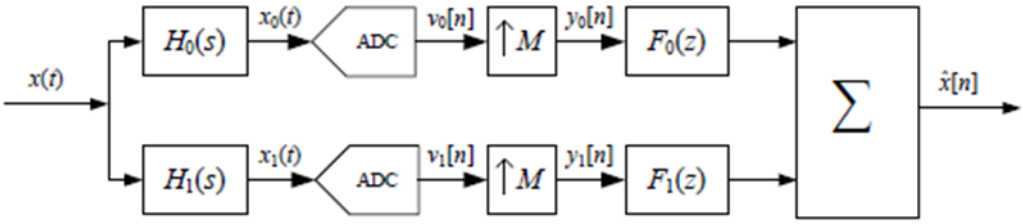


Figure 7.3: Hybrid Filter Bank

We intend to characterize the system to be possible to represent it in simulation and to be inverted in real time. In fact, we will acquire two channels, *x0* and *x1*, filtered by the analog filters *H0* and *H1* respectively.

These signals will be acquired by the USRP kit with the tvrx2 module, and the remaining processing will be performed digitally. To complete the work is necessary to obtain the characteristic of the analog filters *H0* and *H1* and perform a function in Matlab that allows us to do a digital approximation of those. This version of the digital filters will enable the creation of the filters *F0* and *F1* in such a way that we can archive a almost perfect reconstruction. After the calculation of this kind of filters it's possible to get the system running in real time. By taking as starting point the concepts presented, the scheme presented below allows the system operation in real time.

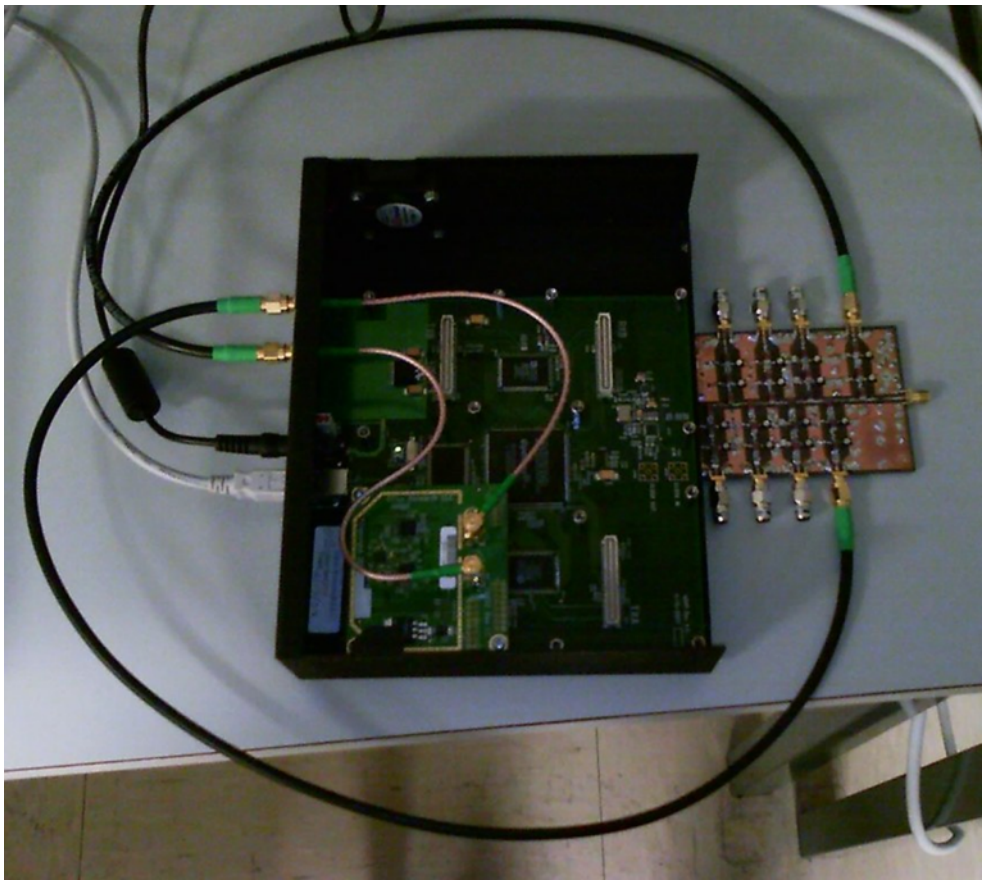


Figure 7.4: USRP and the TVRX2 board connected to 2 channels of the filter bank

7.4.1 Offline test

The first step to create a simulation, to test the system offline, is to measure the 2 channels, that are going to be used in the USRP, of the filter bank in a Vector Network Analyser. After we have the frequency response of those, we create a FIR filter digital approximation with a limited number of coefficients. To this purpose we created a Matlab routine that make that:

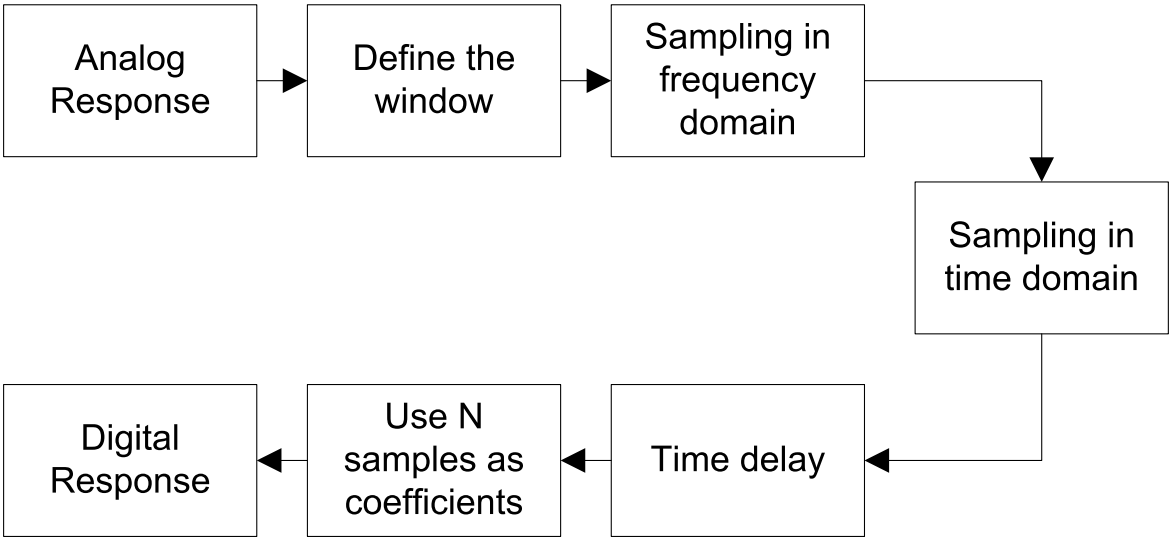


Figure 7.5: How to obtain a digital response of the filters

The response of the analog filters need to be limited in frequency, so we define a window that limits the response to then sample it. By sampling the signal in the frequency domain we will create a periodicity in the time domain response, the next sampling in the time response will then create a periodicity in the frequency response. In some cases the time impulse response may not be causal, in those cases a temporal delay is needed. And finally we select only N coefficients of the response and we now have the digital approximation of the filter.

After this we make the inversion of the analog filters (as we explained earlier) to get the *F0* and *F1*.

All we need now for a simulation is to obtain real data from the hardware, for that the GNURadio[47] software is used.

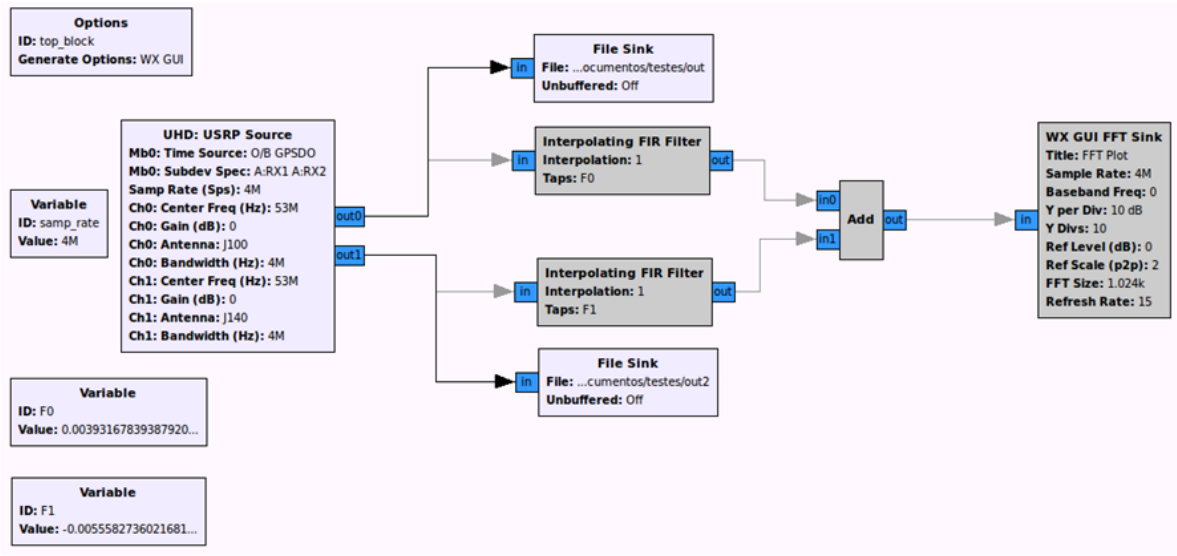


Figure 7.6: Sampling URSP (grey boxes are not in use)

This structure allows us to analyze the spectrum of the signal in real time and store the results so they can be subsequently used in Matlab, for that it was developed a routine that has as input the companion file from the GNURadio, converts it to a matrix in Matlab and prints then the results.

7.4.2 Simulation Results

The simulation work includes the development, testing of routines in Matlab and the realization of some simulations. Here we analyze the results by dividing those into three cases:

- Testing the function of the approximation of the digital filters;
- Simulation of the system in Matlab;
- Simulation with samples obtained with the USRP.

The chosen procedure allows to prove the validity of the results and therefore confirming that the earlier steps are correct. The first step is to test the algorithm of the sampling of the filters, for this purpose a Butterworth bandpass filter of 2nd order is generated at the simulation with the following response:

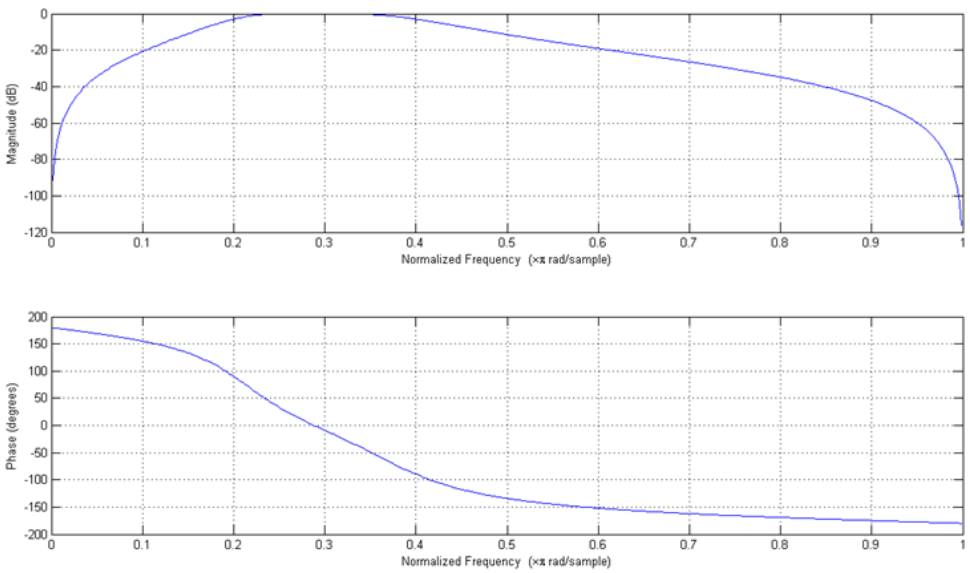


Figure 7.7: A Butterworth bandpass digital second order filter

We now sample this filter with the developed function to obtain a FIR filter with 128 coefficients. Getting the following response:

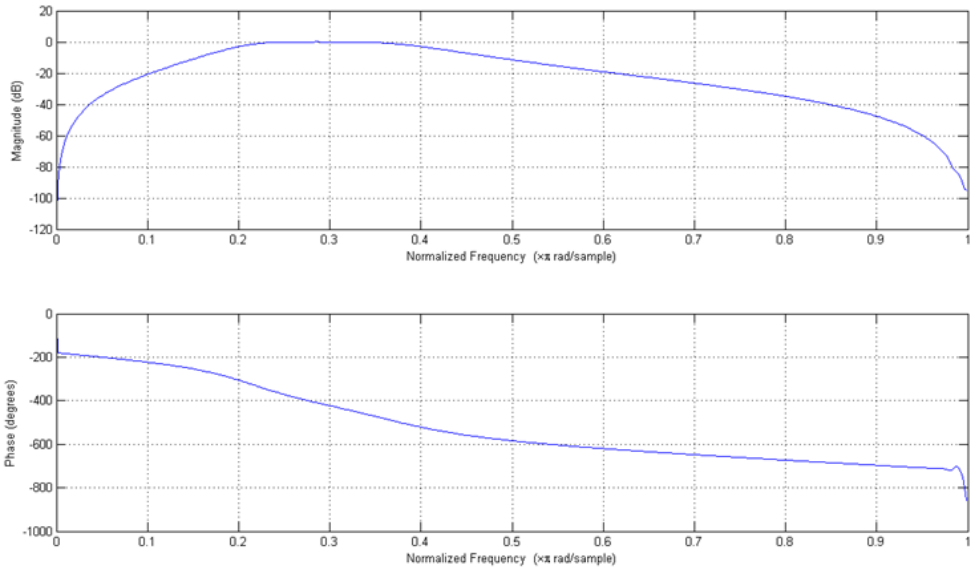


Figure 7.8: Approximation to a 128 coefficient FIR filter

It is apparent that the responses are very similar, the main difference being the phase response. These results prove the validity of the sampling routine and therefore allow to advance into the development of the project. Having tested the function of approximation of the filters, we can now test all the system in simulation. The response obtained of the analog filters with the VNA (Vector Network Analyzer) between 53-55MHz is shown in the figure:

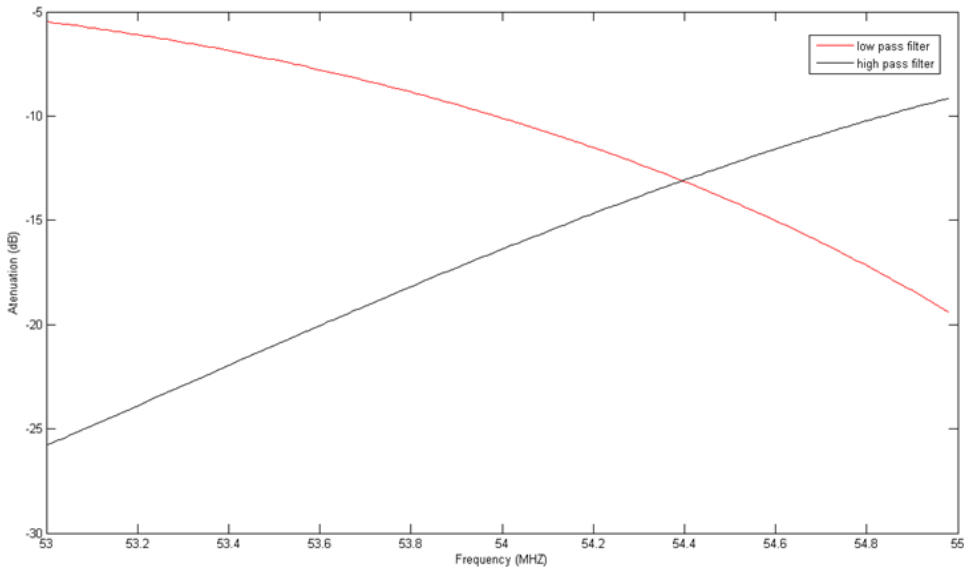


Figure 7.9: The VNA filter response

Using the filters inversion routine to obtain the synthesis filters, *F0* and *F1*, we got the following response:

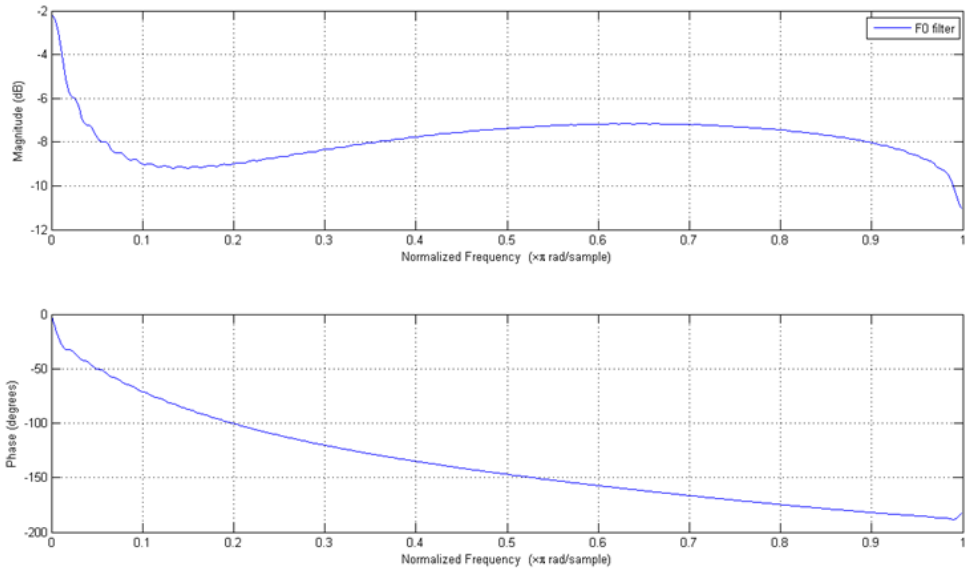


Figure 7.10: Obtained *F0*

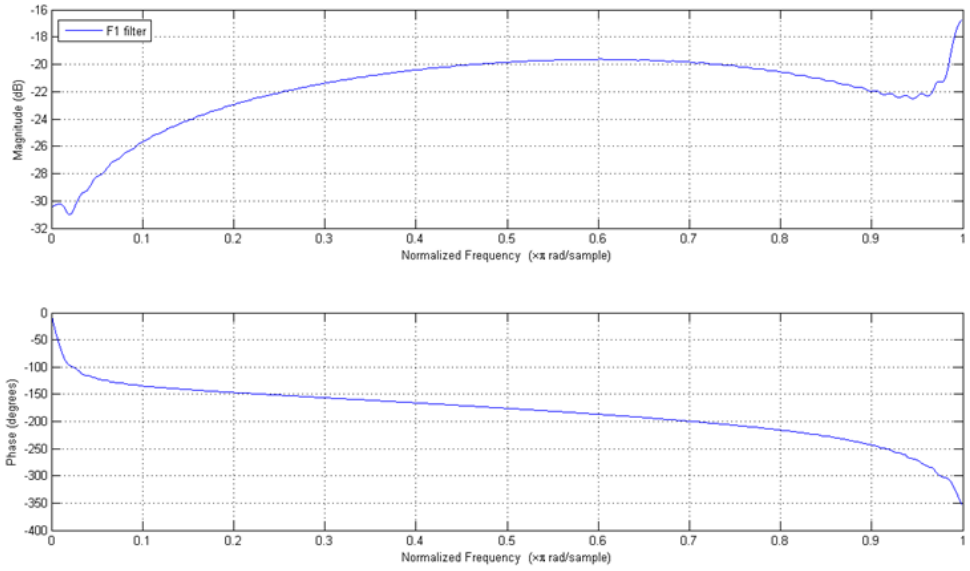


Figure 7.11: Obtained *F1*

It is apparent that the filters response follows the equations that ensure perfect reconstruction in QMF filters,

$$\begin{cases} F_0(z) = H_1(-z) \\ F_1(z) = H_0(-z) \end{cases} \tag{7.6}$$

The filter *F0* is similar to *H0,* and *F1* is the high pass version of *H0.* But in reality, the filter *F0* for low frequencies does not follow exactly *H0* and *F1* has a peak to higher frequencies that was not suppose to have.

To test now all the system we generate in Matlab a signal with 100 sinusoids ranging from 53 to 55MHz. As would be expected the system shows an almost perfect reconstruction, being the reconstructed signal the following:

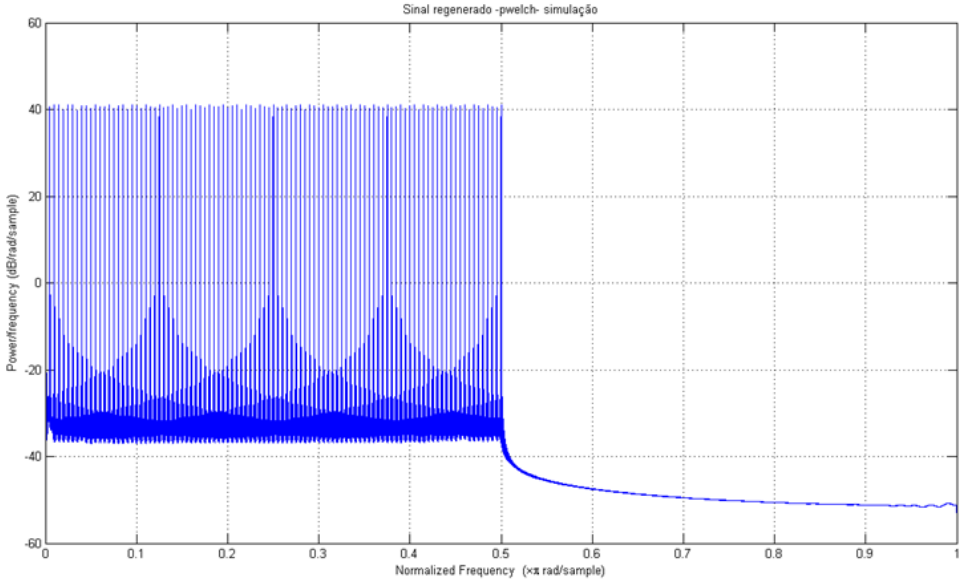


Figure 7.12: Simulation of the signal reconstruction

Managing to archive almost perfect reconstruction we can say that the filters are being reversed correctly.

The last test to be carried out in this simulation is to test it with a real signal obtained with the USRP kit generated by a signal generator. With this final simulation it's possible to test the analog-digital interface, and the reversal of the filters in real practical conditions. In the signal generator is generated the same signal as used earlier, the sampled signal coming from the analog filters is the following:

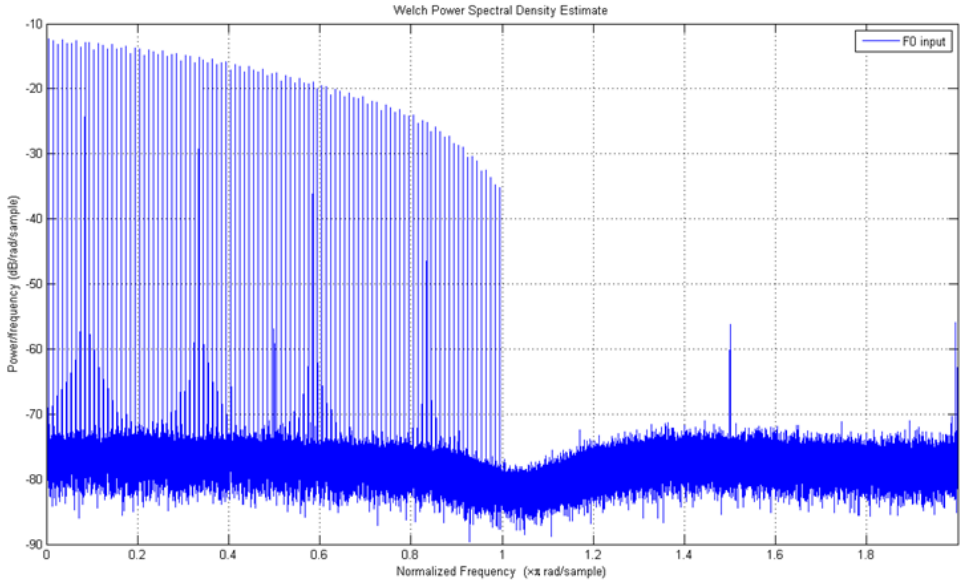


Figure 7.13: Signal acquired from the first channel

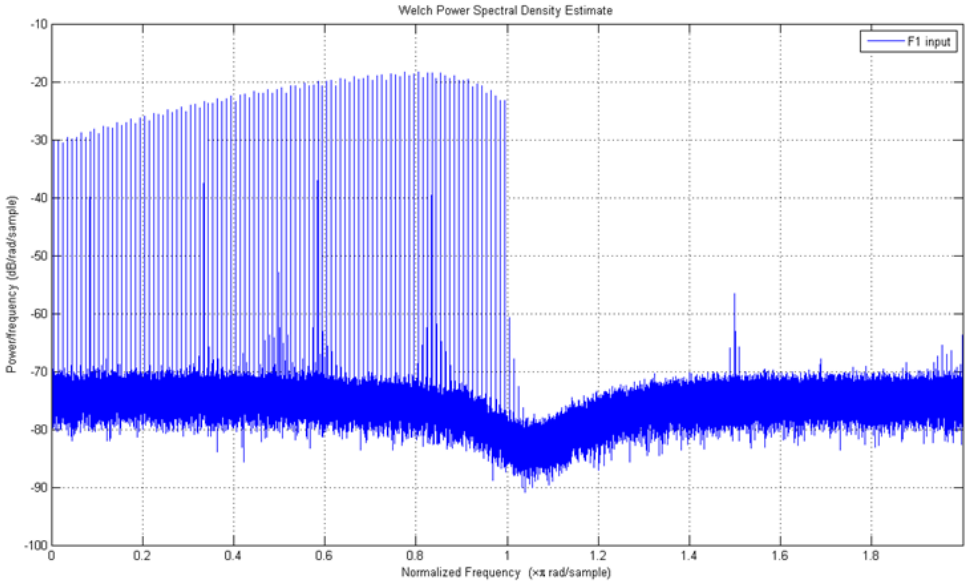


Figure 7.14: Signal acquired from the second channel

It should be noted that the signals present the form of the filters registered by the VNA (Vector Network Analyzer) as we were expecting. And the recovered signal is:

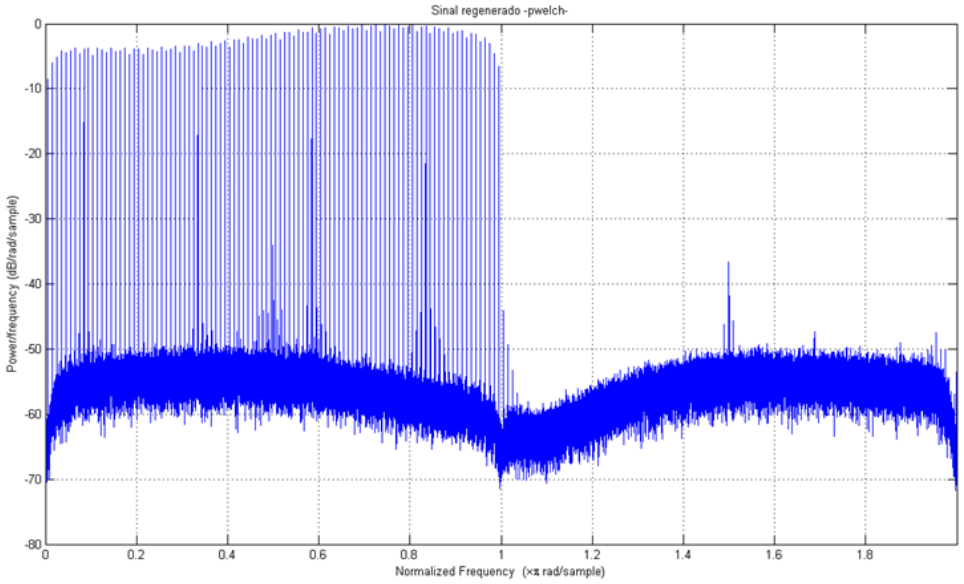


Figure 7.15: Reconstructed signal

So we can tell that the signal is recovered without a great precision.

Applying the filters in real time in the GNURadio Companion we then have the following results:

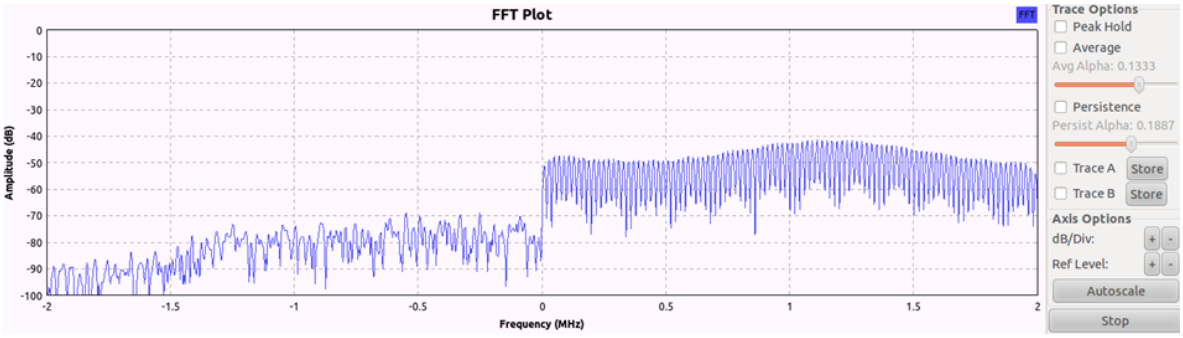


Figure 7.16: Real time reconstruction

This is the final result, as we can see by this result we didn't archive perfect reconstruction, the result shows some reconstruction, with a maximum difference of near -10dB at the bandwidth we are using, but not ideal. This results then had lead to a reformulation at the synthesis filter algorithm calculation.

Bibliography

[1] FCC. Rf exposure info 1. [Online]. Available: https://fjallfoss.fcc.gov/oetcf/eas/reports/ViewExhibitReport.cfm?mode=Exhibits&RequestTimeout=500&calledFromFrame=N&application_id=875932&typ=8374&fcc_id=A3LSGHI757

[2] J. Mitola, "The software radio architecture," *Communications Magazine, IEEE*, vol. 33, no. 5, pp. 26 – 38, may 1995.

[3] S. Forum, "Software defined radio technology for public safety," Tech. Rep., 2006, pp. 20 and 71. [Online]. Available: http://www.sdrforum.org/pages/documentLibrary/documents/SDRF-06-P-0001-V1_0_0%20_Public_Safety.pdf

[4] Ettus research llc. [Online]. Available: http://www.ettus.com/

[5] Wireless open-access research platform. [Online]. Available: http://warp.rice.edu/

[6] Texas instruments adc. [Online]. Available: http://www.ti.com/product/adc12d1800

[7] D. Albuquerque, J. Vieira, N. Carvalho, and J. Pereira, "Analog filter bank for cochlear radio," in *RF Front-ends for Software Defined and Cognitive Radio Solutions (IMWS), 2010 IEEE International Microwave Workshop Series on*, feb. 2010, pp. 1 –4.

[8] E. B. Online. basilar membrane: analysis of sound frequencies. [Online]. Available: http://www.britannica.com/EBchecked/media/537/The-analysis-of-sound-frequencies-by-the-basilar-membrane

[9] ——. ear: hearing mechanism. [Online]. Available: http://www.britannica.com/EBchecked/media/536/The-mechanism-of-hearing

[10] M. D. Hauser, *The evolution of communication.* MIT Press, 1998.

[11] D. Albuquerque, J. Vieira, N. Carvalho, and J. Pereira. Cochlear radio an analog to digital converter system for sdr. [Online]. Available: http://uaonline.ua.pt/upload/med/med_1888.pdf

[12] Xilinx. Virtex-6 fpga, configurable logic block - user guide. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug364.pdf

[13] A. Oliveira, "Sda lecture01 slides 2012/13."

[14] Xilinx. Ise tutorial. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/ise_tutorial_ug695.pdf

[15] ——. Ml605 development board. [Online]. Available: http://www.xilinx.com/products/boards-and-kits/EK-V6-ML605-G.htm

[16] ——. Virtex-6 family overview. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf

[17] ——. Getting started with the xilinx virtex-6 fpga ml605 evaluation kit. [Online]. Available: http://www.xilinx.com/support/documentation/boards_and_kits/ug533.pdf

[18] ——. Ml605 evaluation kit hardware setup guide. [Online]. Available: http://www.xilinx.com/support/documentation/boards_and_kits/xtp084.pdf

[19] 4DSP. 4dsp company info. [Online]. Available: http://www.4dsp.com/company.php

[20] ——. Fmc108. [Online]. Available: http://www.4dsp.com/FMC108.php

[21] TexasIntruments. Dual channel 14-/12-bit, 250-/210-msps adc with ddr lvds and parallel cmos outputs ads62p49 datasheet. [Online]. Available: http://www.ti.com/lit/ds/symlink/ads62p49.pdf

[22] AnalogDevices. 1.2 ghz clock distribution ic, pll core, dividers, delay adjust, eight outputs ad9510 datasheet. [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/AD9510.pdf

[23] ——. Spi and i2c -compatible, 10-bit digital temperature sensor and 8-channel adc adt7411 datasheet. [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/ADT7411.pdf

[24] J. P. R. Magalhães. Info. [Online]. Available: http://wiki.ieeta.pt/wiki/index.php/Magalh%C3%A3es-2011a

[25] J. M. N. Vieira. Matlab num instante. [Online]. Available: http://www.ieeta.pt/~vieira/MyDocs/MatlabNumInstante.pdf

[26] S. Soldado, J. Vieira, D. Albuquerque, and T. Monteiro, "Controlling the reconstruction error in hybrid filter banks," in *Signal Processing Advances in Wireless Communications (SPAWC), 2011 IEEE 12th International Workshop on*, june 2011, pp. 46 –50.

[27] NI. History of gpib. [Online]. Available: http://www.ni.com/white-paper/3419/en

[28] Matlab. Communicating with instruments from matlab using visa. [Online]. Available: http://www.mathworks.com/products/instrument/supported/visa.html

[29] R&S. Matlab toolkit for r&s signal generators. [Online]. Available: http://www2.rohde-schwarz.com/file_11743/1GP60_8E.pdf

[30] Agilent. Io libraries suite. [Online]. Available: http://www.home.agilent.com/en/pd-1985909-pn-E2094/io-libraries-suite-162

[31] Mini-Circuits. Directional coupler zx30-9-4 datash. [Online]. Available: http://217.34.103.131/pdfs/ZX30-9-4.pdf

[32] Agilent. E8361c network analyzer. [Online]. Available: http://www.home.agilent.com/en/pd-1349333-pn-E8361C/pna-microwave-network-analyzer

[33] Mini-Circuits. Cbl-1.5ft-smsm+ datasheet. [Online]. Available: http://217.34.103.131/pdfs/CBL-1.5FT-SMSM+.pdf

[34] ——. Cbl-3ft-smsm+ datasheet. [Online]. Available: http://217.34.103.131/pdfs/CBL-3FT-SMSM+.pdf

[35] R&S. Smu200a. [Online]. Available: http://www2.rohde-schwarz.com/product/SMU200A.html

[36] Tektronix. Dpo/dsa/mso70000 digital & mixed signal oscilloscope. [Online]. Available: http://www.tek.com/oscilloscope/dpo70000-dsa70000-mso70000

[37] 4DSP, *4DSP FMC108 Documentation CD111 - Reference Firmware For ML605*, r1.0 ed.

[38] ——, *4DSP FMC108 Documentation SD060 - FMC108 Star A/D Daughter Card*, r1.0 ed.

[39] Arnaud. (2012, April) Official 4dsp staff response to if there as a way to simultaneous sample the 8adc. IT forum at 4DSP. [Online]. Available: http://www.4dsp.com/forum/index.php?board=161.0

[40] Xilinx. Virtex-6 fpga selectio resources - user guide. [Online]. Available: www.xilinx.com/support/documentation/user_guides/ug361.pdf

[41] Mini-Circuits. Power splitter/combiner zfrsc-183 datasheet. [Online]. Available: http://research.physics.illinois.edu/bezryadin/labprotocol/ZFRSC-183.pdf

[42] Fairview. Ssmc cables. [Online]. Available: http://www.fairviewmicrowave.com/ssma_ssmb_ssmc_sma_cables.htm

[43] Xilinx. Logicore ip fir compiler v6.3. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/fir_compiler/v6_3/ds795_fir_compiler.pdf

[44] ST. An1797 - application note (communication with a pc using rs232). [Online]. Available: http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/APPLICATION_NOTE/CD00015428.pdf

[45] Ettus. Tvrx2 overview. [Online]. Available: https://www.ettus.com/product/details/TVRX2

[46] ——. Usrp overview. [Online]. Available: https://www.ettus.com/product/details/USRP-PKG

[47] GNURadio. Companion. [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioCompanion