MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

DIGITAL BACKEND PROGRAMMING AND RECEPTOR TESTS FOR THE BINGO RADIO TELESCOPE

Matheus Furlan Alpoin

Scientific Initiation Report funded by the PIBIC/CNPq program, advised by Dr. Carlos Alexandre Wuensche and Dr. Cesar Strauss

INPE
São José dos Campos
2021

PUBLISHED BY:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3945-6923/6921
Fax: (012) 3945-6919
E-mail: pubtc@sid.inpe.br

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

DIGITAL BACKEND PROGRAMMING AND RECEPTOR TESTS FOR THE BINGO RADIO TELESCOPE

Matheus Furlan Alpoin

Scientific Initiation Report funded by the PIBIC/CNPq program, advised by Dr. Carlos Alexandre Wuensche and Dr. Cesar Strauss

INPE
São José dos Campos
2021

Cataloging in Publication Data

Sobrenome, Nomes.

Cutter Digital Backend Programming and Receptor Tests for the BINGO radio Telescope / Nome Completo do Autor1; Nome Completo do Autor2. – São José dos Campos : INPE, 2021.

xvi + 40 p. ; ()

Dissertação ou Tese (Mestrado ou Doutorado em Nome do Curso) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, AAAA.

Orientador : José da Silva.

1. Palavra chave. 2. Palavra chave 3. Palavra chave. 4. Palavra chave. 5. Palavra chave I. Título.

CDU 000.000

ATENÇÃO! A FOLHA DE APROVAÇÃO SERÁ INCLUIDA POSTERIORMENTE.

Mestrado ou Doutorado em Nome do Curso

ACKNOWLEDGEMENTS

I thank God, in the first place, for all the good things that he has done in my life; my parents, for providing me all that I needed to keep going with the work. I also acknowledge my advisors, Dr. Carlos Alexandre Wuensche and Dr. Cesar Strauss, for the continued support and patience towards my work, and for always push me further; Fred Vieira, Telmo Machado and Márcio Gastaldi,for humbly helping me to build confidence in the project in many ways - this wouldn't have been possible without your help; Jefferson Lopes and Dr. Edmar Gurjão, for introducing me with kindness to the world of FPGA programming; Dr. Gabriel Rodrigues Hickel and Dr. Alan Pavan, for helping me get this opportunity at INPE; Michael Peel and Michael D'Cruze, for kindly answering all my questions regarding the designs; all the BINGO collaboration, for the trust placed upon me; and CNPq, for the financial support.

ABSTRACT

BINGO (Baryon Acoustic Oscillations Integrated Neutral Gas Observations) is a radio telescope designed to investigate baryon acoustic oscillations (BAOs) in the frequency range from 980 to 1260 MHz, allowing for the investigation of dark matter properties and other phenomena, such as Fast Radio Bursts (FRBs). In this scientific initiation project, the proposed goal was to develop a digital backend based on Field Programmable Gate Arrays for the BINGO Telescope, as well as testing the implemented backend with simulated signals. To this end, the group chose to make use of the Collaboration for Astronomy Signal Processing and Electronics Research) Toolflow, which aims at speeding up the design of digital signal processing algorithms and their compilation for FPGAs. In order to use it, the following softwares were installed: Ubuntu 14.04 in a virtual machine, MATLAB, Simulink, Xilinx ISE 14.7, Python and the Toolflow proper software, considering an implementation on a Reconfigurable Open Architecture Computing Hardware 2 (ROACH2) board. With the aid of a MATLAB/Simulink Test license, it was possible to compile a prototype spectrometer design and to implement it on the board. After this, a number of spectra generated from both a signal generator and a prototype radiometer chain were plotted, which permitted a preliminary assessment of the spectrometer's performace. Besides, field interference tests were also performed. The following steps of this project consist in installing the software environment for the SKARAB board and the configuration of a communication protocol. In a new work front, it is also possible to embed HDL code into the Toolflow's Simulink environment through the black-boxing process.

Keywords: BINGO. Radio Telescope. Digital Backend. FPGA. CASPER.

ESCREVER O TÍTULO EM INGLÊS PARA PUBLICAÇÕES ESCRITAS EM PORTUGUÊS E EM PORTUGUÊS PARA PUBLICAÇÕES ESCRITAS EM INGLÊS

RESUMO

O BINGO (Baryon Acoustic Oscillations Integrated Neutral Gas Observations) é um radiotelescópio projetado para estudar as oscilações acústicas de bárions (BAOs) na faixa de frequência de 980 a 1260 MHz, permitindo a investigação de propriedades da matéria escura e outros fenômenos, como Fast Radio Bursts (FRBs). Nessa iniciação científica, foi proposto o desenvolvimento de um receptor digital (digital backend) baseado na plataforma de Field Programmable Gate Arrays (FPGAs), através da implementação de um espectrômetro operacional para o BINGO e a realização de testes com sinais simulados. O grupo optou pelo uso do CASPER (Collaboration for Astronomy Signal Processing and Electronics Research) Toolflow, desenvolvido para acelerar o projeto de algoritmos de processamento digital de sinais de alto nível e a sua compilação em arquivos programáveis para FPGAs. Para utilizá-lo, realizou-se a instalação do sistema operacional Ubuntu 14.04 numa máquina virtual, seguida da instalação dos softwares MATLAB, Simulink, Xilinx ISE 14.7 e Python, bem como a parcela do Toolflow desenvolvida pela CASPER, tendo em vista uma placa ROACH2 (Reconfigurable Open Architecture Computing Hardware 2). Utilizando uma licença de testes, foi possível compilar um design de espectrômetro e implementá-lo. Foram plotados espectros gerados a partir de um gerador de sinais e do protótipo de receptor para o telescópio BINGO, permitindo realizar uma análise preliminar de sua performance. Além disso, foram realizados testes em campo para estudo de interferência. Os próximos passos deste trabalho consistem na configuração de um ambiente análogo para a FPGA SKARAB (Square Kilometre Array Reconfigurable Application Board), cuja transição é simples, e o estabelecimento de um protocolo de transmissão de dados. Além disso, numa nova frente de trabalho, desenvolver a incorporação de códigos em HDL dentro do ambiente do Simulink através do CASPER Toolflow a partir do chamado black-boxing.

Palavras-chave: BINGO. Rádio Telescópio. Digital Backend. FPGA. CASPER.

LIST OF FIGURES

Page

LIST OF ABBREVIATIONS

BINGO	–	Baryon Acoustic Oscillations from Integrated Neutral Gas Observations
BAO	–	Baryon Acoustic Oscillations
DSP	–	Digital Signal Processing
ASIC	–	Application Specific Integrated Circuit
CPU	–	Central Processing Unit
GPU	–	Graphical Processing Unit
FPGA	–	Field Programmable Gate Array
CASPER	–	Collaboration for Astronomy and Signal Processing Electronics
PSD	–	Power Spectral Density
ATS	–	Autocorrelation Spectrometers
DFT	–	Discrete Fourier Transform
FFT	–	Fast Fourier Transform
FTS	–	Fourier Transform Spectrometer
PFB	–	Polyphase Filter Bank
FIR	–	Finite Impulse Response
CLB	–	Configurable Logic Block
IOB	–	Input/Output Blocks
BRAM	–	Block Random Access Memories
DCM	–	Clock Management
ROACH2	–	Reconfigurable Open Architecture Computing Hardware 2
SKARAB	–	Square Kilometer Array Reconfigurable Application Board
HDL	–	Hardware Description Language
IP	–	Intellectual Property
XST	–	Xilinx Synthesis Technology
NCD	–	Native Circuit Description
VM	–	Virtual Machine
ADC	–	Analog-to-Digital Converter

CONTENTS

Page

1 INTRODUCTION

The BINGO (BAO from Integrated Neutral Gas Observations) telescope is an instrument designed to measure baryon acoustic oscillations (BAO) in the frequency range 980 MHz to 1260 MHz, which is equivalent to a redshift of 0.127 to 0.449 (ABDALLA et al., 2021). It has a cross-Dragone optical design, with two 40-m class reflectors and a focal plane covered by 28 corrugated feed horns. The telescope will be located in Serra da Catarina, in Aguiar (PB) and the construction of its subsystems and the site preparation are under way (WUENSCHE et al., 2021) (WUENSCHE et al., 2019).

Digital backends are a crucial part of the design of many telescopes, digitizing the observed signal and analyzing it in the realm of Digital Signal Processing (DSP) (STANKO et al., 2005) (NG et al., 2017) (SURNIS et al., 2019) (MCMAHON, 2011). According to the different purposes of each instrument, multiple designs can be deployed aiming at different ends, of which beamformers, autocorrelators spectrometers and Fourier Transform spectrometers can be mentioned (PRICE, 2016). Different platforms can also be chosen, ranging from Application Specific Integrated Circuits (ASICs), through Central Processing Units (CPUs), Graphical Processing Units (GPUs), and Field Programmable Gate Arrays (FPGAs), with the latter presenting some advantages in comparison with the others (D'CRUZE, 2018).

In this work, the first steps taken to implement a digital backend for the BINGO radiotelescope are described, envisioning a Fourier Transform spectrometer operating inside the BINGO bandwidth with an optimal configuration of 2048 channels and 0.1 s of integration time in a Field Programmable Gate Array (FPGA) board, by means of the Collaboration for Astronomy and Signal Processing Electronics Research (CASPER) Toolflow.

Firstly, the installation procedures of the software environment is described. After that, the first design compilations are explained in detail, followed by a brief discussion about the dependency of software licenses for the establishment of a functional environment. Finally, the initial tests of a preliminary spectrometer with both a signal generator and a prototype radiometer chain are shown.

2 BIBLIOGRAPHIC REVIEW

2.1 DSP Concepts

A spectrometer is an instrument used to measure and record the spectral content of signals (PRICE, 2016). Simply put, it consists of a digital sampler, which digitizes the signal, and a channelizer, that separates the digitized signal into discrete bins or channels (MCMAHON, 2011). In the context of radio astronomy, the main goal of the spectrometer is to obtain a Power Spectral Density (PSD), with units of power per unit frequency, so as to extract useful information about the received signal (D'CRUZE, 2018).

There are two main types of spectrometers used in radio astronomy. Autocorrelation spectrometers (ATS) obtain the PSD by taking the Fourier Transform of the autocorrelation function, which presupposes a signal that does not change its mean or variance and shows itself to be ergodic over time (PRICE, 2016). As it is not the main focus of this work, this type of spectrometer will not be further explained.

The second type of spectrometer, in its turn, is implemented by first converting the input $x(n)$ signal into the frequency domain $X(k)$ via a Discrete Fourier Transform with N channels, shown in 2.1, and taking its squared average just after that, as shown in 2.2. Therefore, this type of spectrometer is referred to as a Fourier Transform Spectrometer (FTS).

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-2\pi i n k/N} \tag{2.1}$$

$$S_{xx} = \langle |X(k)|^2 \rangle \tag{2.2}$$

A pure implementation of a FTS, however efficient, still leaves space to some undesirable effects. This effects correspond, chiefly, to spectral leakage and scalloping loss (LYONS, 2011). To counteract both, it is useful to implement a window on the input signal, which is normally applied, in radio astronomy implementations, by use of a Polyphase Filter Bank (PFB).

PFBs correspond to a implementation of a Finite Impulse Response (FIR) Filter which reduces the effect of scalloping loss while bringing a moderate computational improvement. It takes is central idea from a FIR, mathematically represented in equation 2.3. In this equation, $h(k)$ correspond to the K filter coefficients, which are applied to the original signal to obtain an output signal $y(n)$ by delaying each

input $x(k)$ by n.

$$y_n = \sum_{k=0}^{K-1} h(k)x(n-k) \tag{2.3}$$

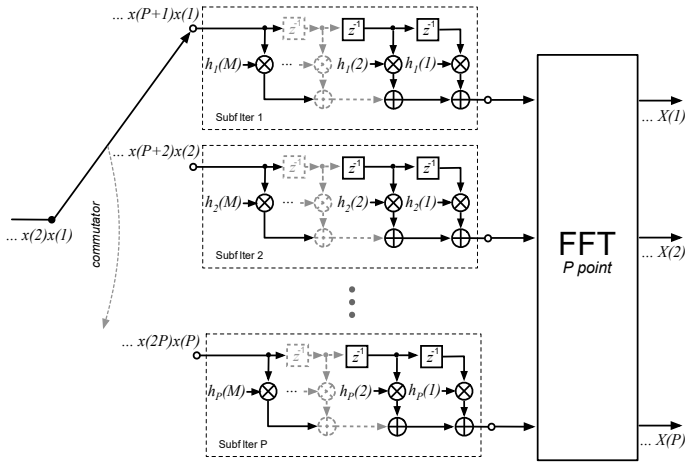
In order to change a FIR filter into a Polyphase FIR Filter, first it is necessary to apply a polyphase decomposition to an input signal according to 2.4, consisting in a decimation by P and a lag by p. This can be done by means of a structure of P discrete FIR filter branches, each possessing M taps, according to 2.5.

$$x_p(n') = (\downarrow P)(z^{-p})x(n) \tag{2.4}$$

$$y(n') = \sum_{p=0}^{P-1} \sum_{m=0}^{M-1} h_p(m)x_p(n'-m) \tag{2.5}$$

Finally, to obtain a PFB, the output of each branch is fed into the input of a DFT with P inputs, leading to 2.6. Graphically, this is shown in 2.1, taking into consideration that DFT are normally implemented with a Fast Fourier Transform (FFT) algorithm, which reduces its computational complexity and greatly improves efficiency.

Figure 2.1 - Polyphase Filter Bank (PFB)



SOURCE: (PRICE, 2016)

$$Y(k,n') = \sum_{p=0}^{P-1} \sum_{m=0}^{M-1} [h_p(m)e^{-2\pi ink/N}]x_p(n'-m) \tag{2.6}$$

2.2 Brief Introduction to FPGAs

FPGAs are reprogrammable semiconductor devices that can be tailored to a particular end. Unlike other devices, FPGAs do not run programs. Rather, they are "programmed" in their hardware connections, making them particularly useful for specific applications or when there is a need for changing the instrument after deployment in the field. As this process of setting up the hardware of an FPGA is inherently different from that of writing a program in a specific programming language, it is also labeled differently. This process is referred to as configuration of a FPGA.

The main components of an FPGA chip are listed below, with the structure of a Xilinx FPGA being shown in figure 2.2.

- **Configurable Logic Blocks - CLB:** essentialy consisting in Lookup Tables (LUT), Flip Flops and Multiplexer, the CLB is the basic logic resource of the FPGA, which can be configured via a bitstream;

- **Interconnect:** routing wires and interconnection switches which surround the and connect the CLBs according to the users' configuration. Xilinx FPGAs, such as the board used in this work, have dedicated matrices of switches to implmenet these interconnections;

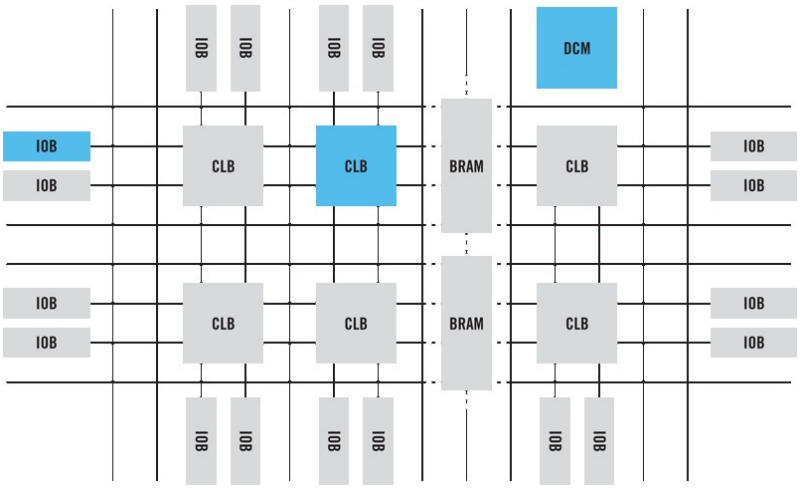
- **Input/Output Blocks - IOBs:** physical structures on the FPGA that permit the communication with external sources through different communication patterns;

- **Block Random Acces Memories - BRAMs:** embedded memory slics embedded within the FPGA chip that can be used as Random Access Memorie (RAM), Read-Only Memory (ROM), and First-In First-Out (FIFO);

- **Digital Signal Processors - DSPs:** blocks designed to particular digital signal processing operations;

- **Clock Management - DCMs:** specific resources inside the FPGA to handle problems related to clock throughout the whole chip.

Figure 2.2 - Basic Structure of a Xilinx FPGA



SOURCE: (URBINA, 2014)

In this work, a Reconfigurable Open Architecture Computing Hardware 2 (ROACH2) board was used. It is the predecessor to the Square Kilometer Array Reconfigurable Application Board (SKARAB), that will effectively be deployed on the BINGO site, and was used due to its availability at the time this work was realized and due to its similarities to SKARAB. Among other features, ROACH2 has the ones listed below, with a block diagram being shown in **??**

- Xilinx Virtex-6 XC6VSX475T-1FFG1759C chip;
- Standalone Power Pc - Processor

2.3 Introduction to the CASPER Toolflow

The Collaboration for Astronomy Signal Processing and Electronics Research (CASPER) is, as its name indicates, a workgroup made of multiple researchers working in various institutions throughout the world, with the goal of minimizing the obstacles related to the development of radio-astronomy signal processing software. To this end, CASPER aims to provide the scientific community with open-source hardware, software and programming tools that can be reused and collectively developed (HICKISH et al., 2016).

As mentioned in the introduction, expertise in FPGA Hardware Description Languages (HDL) represents the most significant barrier to overcome when developing applications for this class of platforms. Additionally, the domain of digital signal

processing (DSP) can easily turn out to be a daunting subject for those who carry little or no experience in the field.

CASPER provides a solution for both of this problems through the use of the CASPER Toolflow. This term might refer both to the whole set of software used to program the FPGA as well as the set of DSP libraries and tools maintained by CASPER within this set (HICKISH et al., 2016) (WERTHIMER, 2011). Its main goal is to provide a graphic environment that speeds up the design and compilation of highly complex DSP projects.

With the aid of the Toolflow, much of the challenging details of FPGA programming is spared from the user, such as ADC interfaces, Ethernet gateways, timing requirements and FPGA pin locations (HICKISH et al., 2016). The Toolflow is also arranged so as to allow for easy porting between compatible hardware without compelling the programmer to reorganize the whole design, but rather by simply altering a top-level parameter[1] (HICKISH et al., 2016).

On the other hand, CASPER is clear in stating that the Toolflow structure demands particular versions of each of its component software in order to function properly, and that the usage of different software versions or of different software configurations, as in the case of employing operational softwares other than the ones recommended by the collaboration, may lead to additional steps in the Toolflow configuration or to a non-functional Toolflow. In this sense, a compatibility matrix is provided for each of the boards supported by the collaboration. This report deals with the Toolflow installation and handling for the ROACH2 board, as it was the available platform during the period of its execution.

The next section explains, in simple terms, how the Toolflow works and how each software plays its role in the design process. It is worth mentioning that, even though ROACH2 utilizes a deprecated version of the flow in relation to the one employed by SKARAB, both of them fulfill the goal proposed by the CASPER philosophy.

2.3.1 Toolflow Inner Workings

The original CASPER Toolflow was MATLAB-based, that is, it operated by allowing users to develop their own designs inside Simulink graphical interface, which would be later used for generating a Xilinx Embedded Development Kit (EDK) Project that would be compiled, alongside with associated constraints, into a *.bof*

[1] https://casper-toolflow.readthedocs.io/en/latest/jasper_documentation.html

file. This file, which contained a programmable bitstream and meta-data, would then be uploaded and programmed onto the FPGA through the use of a Python-based communication library - allowing for real-time access to some software registers (HICKISH et al., 2016). This MATLAB-based Toolflow was the focus of the main aspects of this work until now, as it is supported by ROACH2.

More recently, in virtue of the new generation of FPGAs developed by Xilinx and the new software utilized to interact with them, a new Toolflow has been developed. Under the name of *jasper*, it was thought so as to be completely Python 2.7-based and is applied in the configuration of the newer CASPER boards, such as SKARAB. It still makes use of Simulink as the graphical entry tool, but deals with the compilation process in a different manner.

2.3.1.1 *bee_xps* Toolflow - ROACH2

The first and most important part of the original Toolflow in the user's perspective corresponds to the Simulink environment, on which instrument designs are developed in a high-level, model-based approach. CASPER provides the researcher with two Simulink libraries containing a variety of blocks to develop radio-astronomy instruments, which, albeit freely available for download, are dependent on paid Simulink libraries which were determined as part of this work and which will be discussed in the next sections. CASPER designs also depend on Simulink libraries provided by Xilinx environments, be it ISE or Vivado. These three libraries are listed below:

a) **CASPER XPS Blockset**: normally referred to as "Yellow Blocks" due to their color in the Simulink frontend, their basic functionality is to equip the researcher with hardware interfaces to the boards, allowing the designs to successfully communicate with parts of the board's architecture such as ADCs and BRAMs as well as making room for real-time interaction between a programmed design and the user through the use of software registers. They also work as a gateway between the FPGA environment and the Simulink environment, which handle data in different fashion - more on this below.

b) **CASPER DSP Blockset**: once again due to their color in the Simulink frontend, these are usually labeled as "Green Blocks". They allow the programmer to implement complex DSP functions, such as the Fast Fourier Transform (FFT), by simply adding a high-level interface (Simulink mask) and editing its parameters.

c) **Xilinx Blockset**: this set of blocks comes as part of the Xilinx software and is essential to the correct functioning of all CASPER-provided blocks mentioned above. Furthermore, it also contain basic logic elements which can be used within the design, such as counters, adders, multipliers, shifts and bit-slicers. The Gateway block provided by this library is responsible for doing the interfacing work mentioned in the CASPER XPS Blockset.

After a design is completed on the frontend Simulink environment, the next steps of the Toolflow come into play. These steps are illustrated below as a case example.

Firstly, the command *casper_xps* must be passed onto the MATLAB console. This command triggers the opening of a dialog box with a couple of options that demand some configuration before the Toolflow itself is able to operate, allowing the user to choose which of the flow steps will be part of the compilation process.

Once the compile button is pressed, the Toolflow operation, which is documented in real time within the MATLAB environment, is started. The compilation stages listed below are all carried out by Xilinx[2]:

a) **System Generation**: this stage inputs a Simulink Model (*.slx*) and translates it into hardware description language, in the form of VHDL and Xilinx CORE Generator Blocks. CORE Generator Blocks are Intellectual Property (IP) Cores optimized for Xilinx FPGAs and are included in ISE (XILINX, 2012);

b) **Synthesis**: the hardware description language of the previous stage is input into this one, which translates the behavioral implementation into a netlist of logic equations (*.ngc*) (XILINX, 2008). It consists of a structured netlist containing logical design data and constraints. This process is performed through the use of Xilinx Synthesis Technology (XST);

c) **NGD Build**: this stage of compilation translates the binary netlist output by the previous stage into a logical description of the circuit that can be implemented in the targeted FPGA (*ngd*). It is carried out through the NGDBuild software;

d) **MAP**: by inputting the *.ngd* file from the previous stage, the MAP program performs a design check and then maps the design logic to the components of the targeted FPGA (XILINX, 2008). It then outputs a mapped,

[2]https://casper.astro.berkeley.edu/wiki/Compilation_Stages

though not routed, Native Circuit Description (NCD) *.ncd* file. This is to say that the logical description is "translated" into the FPGA resources, but the placement of these resources is still to be performed;

e) **PAR**: the PAR program takes the *.ncd* file output by the previous stage and carries out the place and route task that was left undone by MAP, outputting a routed *.ncd* file that will then be used by the bitstream generator;

f) **BitGen**: This is the last stage handled by Xilinx software. It inputs the fully routed *.ncd* file and outputs a bitstream (*.bit*) that will be used in the FPGA configuration;

g) **Mkbof**: This stage, handled by CASPER software, takes the *.bit* file as an input and outputs a *.bof* file.

3 ROACH2 COMMISSIONING

The next subsections deal with the setup of the software environment and the first compilations done with ROACH2.

3.1 ROACH2 Setup: Toolflow Installation

3.1.1 Ubuntu 14.04 LTS

Firstly, the operating system (OS) needed to match the one required by ROACH2. CASPER official statements point that full support is offered only for the softwares shown in the compatibility matrix made available by CASPER, even though the Toolflow may work in similar software architectures.

The laboratory's computer, nicknamed as Hex, runs various partitions, including one that runs Red Hat as its main operating system while the other runs on Ubuntu 20.04.02 LTS. Initially, the idea of keeping the latter as the Toolflow operating system was considered, as well as the thought of continuing a previous work that had been done on the RedHat partition. However, in order to avoid risks, the decision was made to install Ubuntu 14.04 LTS on a Virtual Machine for ROACH2.

In this context, Oracle VM Virtual Box was chosen as the software of choice. Inside the Ubuntu 14.04.02 LTS machine, a disk with 50 GB of virtual space and 44 GB of effective space was mounted, which was later expanded with an additional 50 GB partition. A shared folder between the real and the virtual machine was also created, as well as a virtual network board.

3.1.2 Xilinx ISE 14.7

With regards to Xilinx softwares, a license for ISE 14.7 already existed in the Red Hat partition, on account of the previous work aforementioned. The task, then, was to copy this license onto the Ubuntu 20.04 LTS partition. After this was done, the installation process followed what was shown on the CASPER "Installing the Toolflow" website. The ISE 14.7 interface was not used during spectrometer compilations, as explained in the Toolflow workings, but rather summoned by the compilation process through system generator.

3.1.3 MATLAB 2013b

Most of the difficulties regarding this work were related, to a certain level, to MATLAB and Simulink licenses. During the bibliographic review, it soon became clear

that unavailable licenses at the time were required to compile the first models for the work and that it could not progress any further while those were not at the disposal of the workgroup.

In face of this, a split strategy was chosen to handle the problem: a member of the group investigated the possibility of compiling HDL code inside the CASPER Toolflow without having recourse to MATLAB and Simulink, while the main focus remained in investigating which licenses were needed and looking for means to obtain them.

At first, a license for MATLAB and the basic libraries of Simulink was obtained. This license provided the group with the possibility of installing any version of both softwares in two simultaneous machines at the same time. The installation process, then, was straightforward, and MATLAB 2013b was succesfully installed inside the VM environment.

However, after setting up the Toolflow proper software - which will be described in the next section - the dependency of the CASPER DSP Libraries came to light once again, as some blocks could not be configured within Simulink, with Simulink blocking the simple process of adding some others into the design, as it will be detailed in the MATLAB Toolbox Dependency subsection. Still, the first tutorial (Tutorial 1) made available by CASPER was compiled and programmed succesfully into ROACH2, as it will be explained in the sections below.

In a second moment, access to a trial version of MATLAB/Simulink with the needed Toolboxes was obtained, and a series of mock compilations were performed in order to estimate the dependency of the aimed designs at each one of them. At the end, a R2012a version was made available for the workgroup and some compilations were done with it, though the obtained results were far from Ideal.

3.1.4 Python

As shown in the compatibility matrix, Python 2.7 is the tested version for ROACH2 designs, being the environment by which real time communication can be made with the board. It was, therefore, necessary to install a virtual environment of Python 2.7 for both the VM and the real machine, with the reason for this decision being explained in the next subsection.

3.1.5 *mlib_devel* and *corr*

mlib_devel corresponds to the Toolflow proper, that is, the part of the Toolflow which is developed by the CASPER collaboration. For ROACH2, it consists mainly of MATLAB scripts for Simulink blocks and MATLAB object constructors, as well as Embedded Development Kits (EDK) projects, as it was explained in the introduction. It is available on GitHub and its installation on the VM, forking from the roach2 branch, was direct. It was found, however, that the roach2 branch did not possess a text file providing requirements for additional Python dependencies, as it was mentioned on the CASPER installation instructions. For SKARAB, however, these dependencies would present an additional step.

The main difficulties regarding installation came with the Python *corr* package, which was chosen after the installation of *casperfpga* did not go as planned. Both packages allow for the interaction with CASPER hardware in real time, providing the user with the ability to reconfigure firmware, as well as read and write registers, with *casperfpga* being an updated version of *corr* for more recent CASPER hardware.

Two main obstacles were found during this process: firstly, the laboratory's board firmware was deprecated, though still operational. This meant that *casperfpga* could not be used to communicate with it, as it was targeted at more up-to-date firmware. Our board, however, running an older firmware, required an older Python library to implement a control protocol it would understand. To solve this problem, *corr* should be installed. On account of this, the second main obstacle arose, as *corr* could not be installed inside the VM, once again due to software deprecation. It asked for a specific version of the Python library *matplotlib*, which in turn returned an error related to the *setuptools* library inside the VM's Python 2.7 virtual environment.

Thus, the found solution was to install *corr* on the real machine. In practice, this choice meant that, even though compilations were to be done inside the VM, all board to computer communications needed to be performed on the real machine. That is to say, extra time was added to the programming process, although it did not represent an impediment to the workflow. It is important to note, as a closing statement, that the group did not rule out the possibility of bypassing the incompatibility between *setuptools* and the VM's 2.7 Python installation, but rather chose a more expedient path to resolve it.

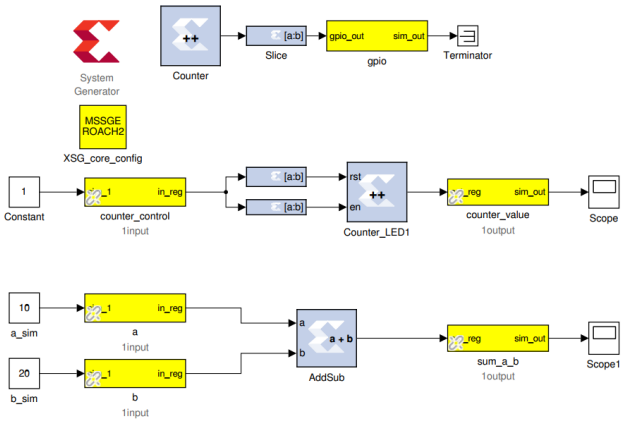
3.2 ROACH2 Setup: Compilation tests

CASPER provides a series of Tutorials which aim to introduce newcomers to the functioning of the Toolflow. They are comprised of functional models which can be fine-tuned for a particular application, providing a first implementation which can be used for initial tests as well as setting a core pattern to be followed in the development of more complex designs. This work in particular has highly beneffited from the Toolflow's flexibility, with the majority of testing being done on already existing designs, which were ported to our configurations. Below, we explain three of these tests.

3.2.1 CASPER Tutorial 1

The first tutorial provided by CASPER is a simple introduction to all the functionalities of the CASPER Toolflow. It does not make use of the CASPER DSP related Toolboxes and consists in a relatively short design, which reduces its compilation time. As it will be explained in the MATLAB Toolbox Dependency subsection, it was the only CASPER Tutorial to compile without demanding paid toolboxes. Its main structure is shown in Figure 3.1 below.

Figure 3.1 - CASPER Tutorial 1

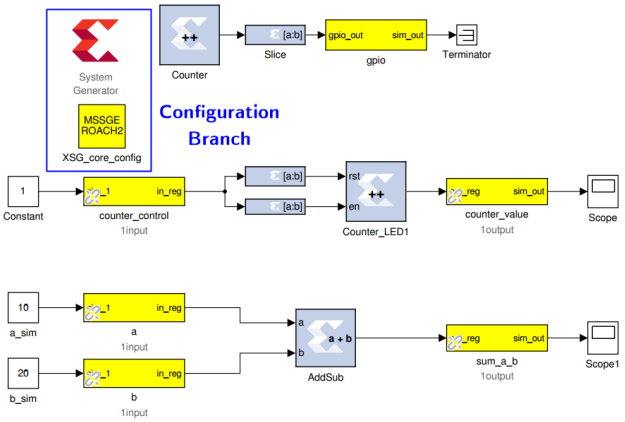


SOURCE: Author's work

The design can be split up into three different branches of functionality, paired together with the configuration blocks. In figure 3.2, this configuration branch is highlighted. The Xilinx System Generator Block handles the initial stage of the Toolflow compilation, translating the graphic design into a netlist targeted at the

FPGA chip to which it has been set through the use of the XSG_core_config block, which allows the user to choose on which of the CASPER supported boards the design will be implemented.

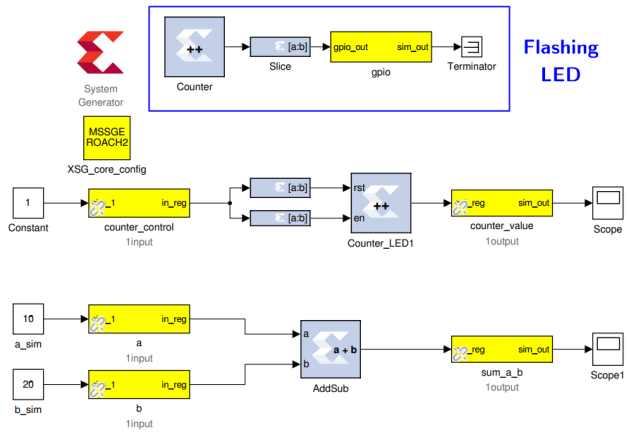
Figure 3.2 - CASPER Tutorial 1 (Configuration Blocks)



SOURCE: Author's work

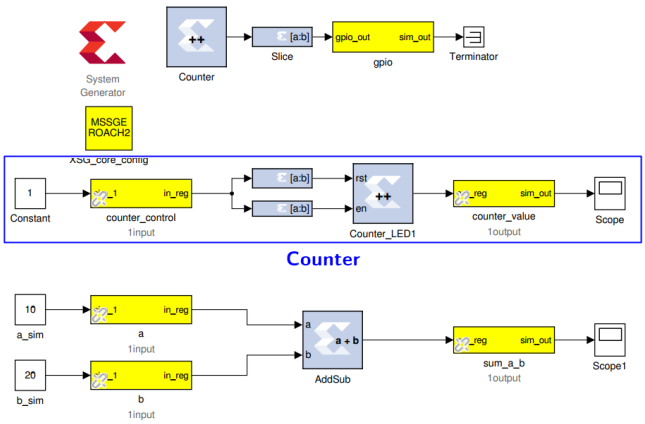
The first functionality of Tutorial 1, shown in 3.3, consists in a Xilinx Counter block, which, upon selection of the most significant bit (MSB), flashes a LED through the gpio block. This tutorial was remotely verified though the use of a camera pointing to the board in real-time.
The other two functionalities of the first CASPER tutorial are centered around two basic digital electronics blocks provided by Xilinx, that is, counting and adding. The counter branch, shown in 3.4, is comprised of a software register with I/O direction set as "From Processor" that inputs a 32-bit binary word into two bit slicers, which isolate the least significant bit (LSB) and the bit of weight 2^1, directing the former to the enable input and the latter to the reset input of the Xilinx Counter block. The output of this block is then directed to another software register set as "To Processor" which stores its values.
In like manner, the adder branch shown in Figure 3.5 takes the input values from two "From Processor" software registers and outputs them to a "To Processor" software register, which stores the result of the operation. In all of these branches, white Simulink blocks can be seen, such as constants, scopes and terminators. These blocks do not influence the final bitstream and are used solely for simulation purposes,

Figure 3.3 - CASPER Tutorial 1 (Flashing LED blocks)



SOURCE: Author's work

Figure 3.4 - CASPER Tutorial 1 (Counter Blocks)



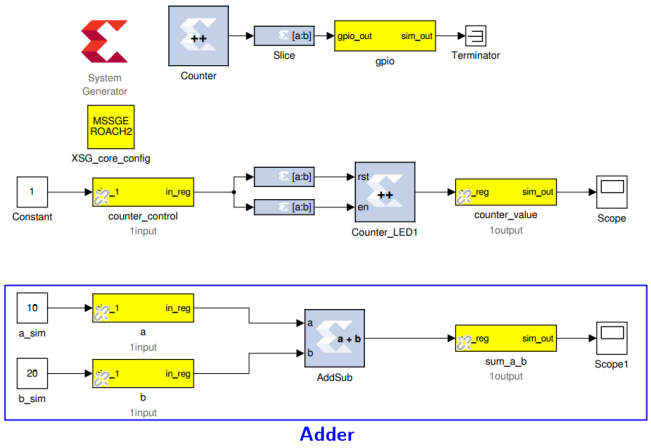
SOURCE: Author's work

which was not performed in this work.

3.2.2 CASPER Tutorial 3

Tests performed with CASPER Tutorial 3 formed the bulk of results in this work. This design corresponds to a complete, but simple, wideband spectrometer, serving as the prototype model to other spectrometer designs. It is composed of a central pipeline of data and some auxiliary blocks that either provide commands to central backbone or serve as extra functions to the design. Below, its main components are

Figure 3.5 - CASPER Tutorial 1 (Flashing LED blocks)



SOURCE: Author's work

explained in detail.

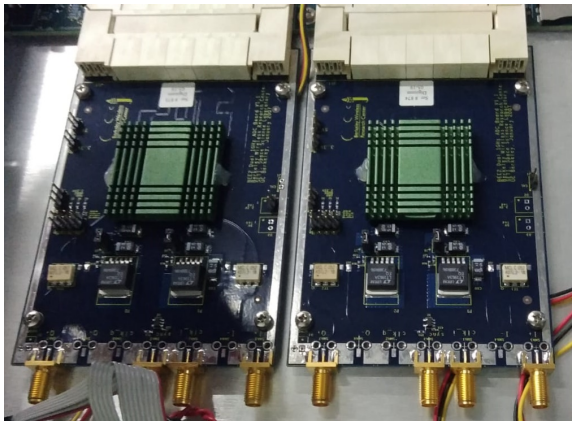
3.2.2.1 adc Block

The adc block interfaces with ROACH2's Analog-to-Digital Converter (ADC) cards. It is the entry point of data that was digitized from the board's analog inputs, being dependent on the ADC's configuration. The available ROACH2 board at the disposal of the workgroup was shipped with two ADC2x1000-8 cards, capable of sampling at 1 GSps for two streams and 2 GSps for a single stream, with a clock input ranging from 10 MHz - 1 GHz, 50 [Ω], 0 dBm. A picture of both ADC boards is shown in Figure 3.6, with the adc block being shown on Figure 3.7.
The parameterization of this block varied according to the purposes of the work. It is important to emphasize, primarily, that the clock rate of the adc block is set to be 4 times greater than that of the board. Therefore, it will output 4 parallel strings of data, each one with a data width of 8 bits, for every clock cycle of the board.

During the first compilation stages, for the sole purpose of testing the Toolflow Compilation Environment and the interface between the board and the laboratory computer, the adc clock was set to 800 MHz. After gathering information from a previous researcher, this value was changed to 900 MHz. Both compiled and operated successfully, and the reasoning behind this change is explained in more detail in the results subsection.

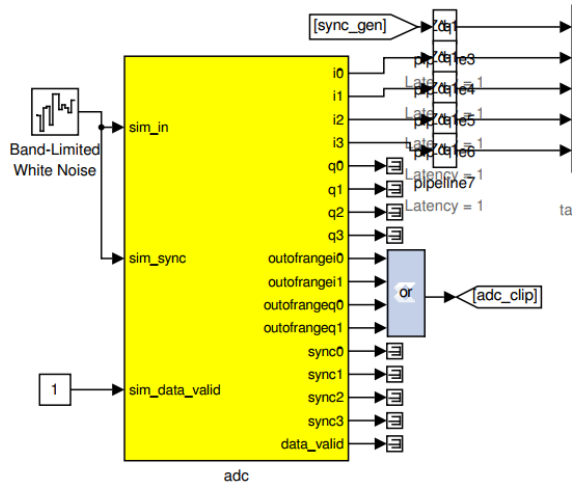
The inputs shown in Figure 3.7 do not influence in the final bitstream and were

Figure 3.6 - ADC Boards (ROACH2)



SOURCE: Author's work

Figure 3.7 - ADC Block (CASPER Tutorial 3)



SOURCE: Author's work

neither used nor investigated during this work. The outputs behave as follows:

- **iX (i3, ... , i0)**: in-phase sampled values, with 8 bits each, in the Fix_8_7 format (fixed binary point at the bit 7);
- **qX (i3, ... , i0)**: quadrature sampled values, with 8 bits each, in the Fix_8_7 format (fixed binary point at the bit 7). These inputs were not used in this work;
- **outofrangeiX/outofrangeqX**: a signal that represents when samples are

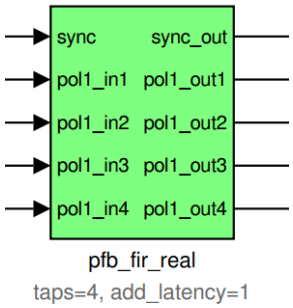
out of the range of the ADC, in boolean format;

- **syncX (sync3,..., sync0)**: a signal used to monitor if the sync pulse offset by X or 2X is high, if the two ADCs are interleaving or not, respectively. Each pulse is of the boolean type and was not utilized in this work;
- **data_valid**: a boolean signal to verify if the data is valid, which was also not used in this work.

3.2.2.2 pfb_fir_real Block

This block is responsible for implementing the frontend of the Real Polyphase Filter Bank described in the introduction. Its parameterization was left fixed for the purposes of this work, as the Hamming Window to which it was configured shows good performance for radio-astronomy applications. The setting of 4 taps was also left untouched, which did not harm the spectrometer performance, as it will be shown in the results subsections. Its inputs and outputs are listed below, with the block being shown in Figure 3.8:

Figure 3.8 - pfb_fir_real Block (CASPER Tutorial 3)



SOURCE: Author's work

- **Inputs:**
  - **sync:** this boolean input is obtained from the sync_gen block and indicates that the next clock cycle contains valid data;
  - **pol_in:** the inputs that will be processed by the FIR filter structure inside the pfb_fir_real block, inherited from the adc block. As there are parallel data streams, each of them is labeled according to its

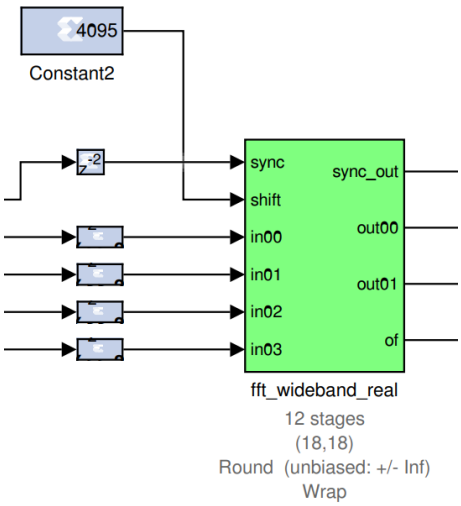
connection to the adc block - pol_in1 is connected, through a Xilinx Delay Block, to i0, while pol_in2 is connected to i1, and so forth;

- **Outputs:**
  - **sync_out:** does the equivalent work of the sync input to the next block in the chain;
  - **polX_outY:** the resulting processed signal from the pfb_fir_real block, consisting of filtered time-domain signals to be input to an FFT.

3.2.2.3 fft_wideband_real Block

The central core of the PFB implementation, the block shown in Figure 3.9 implements a real-sampled Fast Fourier Transform which operates with complex multipliers inside it. Besides, it has a modified biplex architecture to deal with parallel samples, outputting only the positive frequencies of the FFT. This leads to half the number of outputs compared to the inputs, with all inputs and outputs being listed below.

Figure 3.9 - fft_wideband_real Block (CASPER Tutorial 3)



SOURCE: Author's work

- **Inputs:**
  - **sync:** inherited input from the pfb_fir_real with the same function;

- **shift:** an unsigned binary input, in this work set by a Xilinx Constant Block, which sets the shifting schedule of each stage of the FFT. This means that the length of this word will depend on the size of the FFT - if an FFT has 2^N channels, the shift word will have N bits of length. To shift a stage, that is, to divide it by 2, its corresponding bit in the shifting word will be high;
- **in0X:** the input samples of the FFT. The first index number indicates the stream of parallel samples, which in this case correspond to a single stream, i. e., the zeroeth stream. The second index number specifies the input - input in00 will be connected, through a Xilinx Binary Shifter to pol1_out1 from the pfb_fir_real block, while input i01 will be connected to output pol1_out2, and so forth;

- **Outputs:**
  - **sync_out:**a boolean output indicating that the data will be valid in the next clock cycle;
  - **out0X:** the frequency domain output of the FFT block, in complex form. The zero index indicates that this output is linked to the zeroeth parallel stream, while the second index indicates if the streaming output correspond to the even-numbered or odd-numbered channels of the positive frequency channels, which will be accumulated into different BRAMs later in the design;
  - **of:** an unsigned stream of bits, whose length is dependent on the number of streams, indicating, in the case of a high value, the existence of a internal arithmetic overflow. Stream number 0, in this case, is related to the most significant bit, with the logic following this pattern as the number of parallel streams increase.

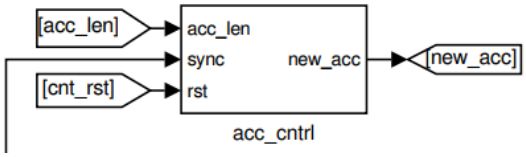
3.2.2.4 Accumulation Blocks

From this point onwards, the frequency domain is already available for analysis. However, for noise mitigation purposes, the data output by the FFT should be integrated, that is, accumulated, in time. After that, it should also be stored in BRAMs for real-time access of data.

The accumulation process is handed by two main blocks: the acc_cntrl block, shown in Figure 3.10 and the simple_bram_vacc block, one for each output stream of the

FFT block, shown in Figure 3.11. The acc_cntrl block takes three inputs and outputs a control signal for the accumulation process. These inputs and outputs are listed below:

Figure 3.10 - acc_cntrl Block (CASPER Tutorial 3)

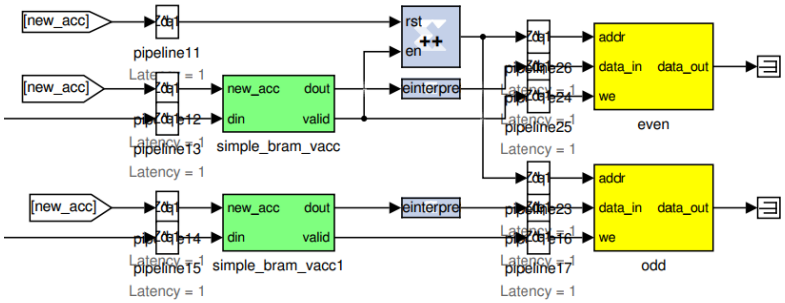


SOURCE: Author's work

- **Inputs:**
  - **acc_len** this user-defined parameter sets the length of the accumulation, in other words, how many samples it will integrate (which will be hereafter used as synonymous to accumulate) before outputting a value to be stored at the FPGA BRAM. The user sets this parameter through the use of a "To Processor" software register;
  - **sync:** the sync pulse inherited from the upstream blocks, which terminates in this block. Its role is, subsequently, performed by the new_acc output which will be explained below.
  - **cnt_rst:** an input which can be defined by the user via a "To Processor" software register to reset all the accumulation process done up until the moment of the command;
- **Outputs:**
  - **new_acc:** by taking the information from three block inputs, this output determines when to start a new integration after one has just been completed. It compares the acc_len value with a counter enabled by the sync input and sends a pulse everytime this operation is true.

The simple_bram_vacc block, in turn, integrates a defined length of data and outputs it when the process is finished. This means that the output has an energy-like character which differs from the power character of the input and, therefore, post-processing of the signal is needed if the aim is to obtain a power spectrum. Its inputs and outputs are listed below:

Figure 3.11 - simple_bram_vacc and bram Blocks (CASPER Tutorial 3)



SOURCE: Author's work

- **Inputs:**
  - **new_acc:** this input takes the output of the same name from the acc_cntrl block and outputs the already accumulated data, starting a new accumulation.
  - **din:** the din input receives the stream of data to be accumulated.
  - **cnt_rst:** an input which can be defined by the user via a "To Processor" software register to reset all the accumulation process done up until the moment of the command;
- **Outputs:**
  - **dout:** the output accumulated data;
  - **valid:** a pulse indicating that the data is valid;

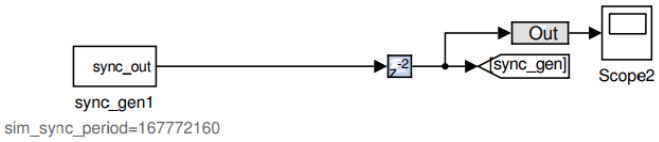
After the accumulation process, data is then sent to be stored in the BRAMs. The design is set so that, every time a new valid accumulation is completed, the already stored values are overwritten with new ones.

3.2.2.5 Software Registers and General Purpose Input/Outputs (GPIO)

In the above sections, a general explanation of the main data flow of Tutorial 3 was given. However, some important blocks are placed out of its main core, providing or retrieving useful information from its processing results. Besides the two configuration blocks already mentioned in Tutorial 1, Tutorial 3 makes use of specific software registers and Simulink Masks to successfully operate or give useful feedback to the programmer.

Firstly, the sync_gen block shown in Figure 3.12 serves the design as a source for the synchronization pulse, which enters the main core of the spectrometer through the pfb_fir_real block and is propagated until the acc_cntrl block. Its role is important in the sense that it makes up for the absence of a deployed on-site sync source, which would play this role for a fully operational design. Its configuration is based on the size of the FFT, the number of parallel streams, the number of taps in the PFB, the order of the reorder blocks inside the FFT block - related to bit unscrambling and biplex FFT configuration for the fft_wideband_real block - and a scaling factor to scale the pulse.

Figure 3.12 - sync_gen Block (CASPER Tutorial 3)



SOURCE: Author's work

In addition, a set of software registers are used to multiple functions. They consists in both "From Processor" and "To Processor" registers, as well as General Purpose Input/Outputs (GPIOs), and are listed below.

- **"From Processor" Registers:**
  - **acc_len:** takes the values for the accumulation length. It is a fixed value, but it should accomodate to the design number of channels. This is done via the acc_cntrl's block internal logic;
  - **cnt_rst:** a pulse to reset all counters in the design;
  - **gain:** an inherited input which was found to have no functionality on the implemented version of Tutorial 3, as it was aimed at designs which have a accumulator of 32 bits or less. It can be deleted if the programmer is willing to do it.
- **"To Processor" Registers:**
  - **sync_cnt:** counts the number of synchronization pulses issued by the board.

  - **acc_cnt:** counts the number of accumulations performed. Useful when analyzing data and to verify if the spectrometer is functioning correctly;

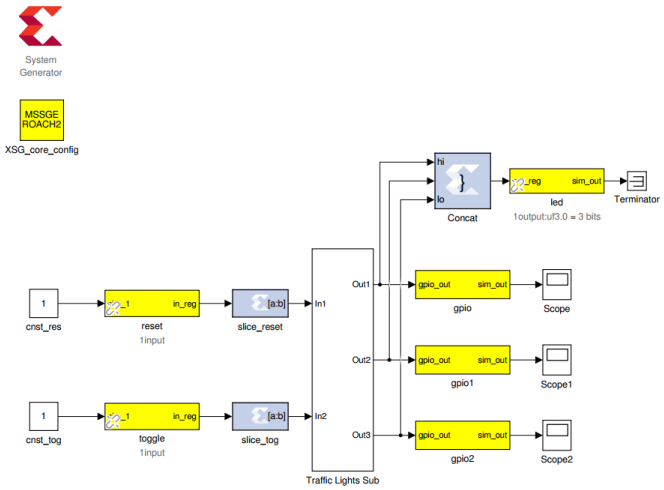
- **GPIOs:**

  - **led0_sync:** flashes a LED every time a sync pulse is emitted;
  - **led1_new_acc:** flashes a LED for every new accumulation that occurs;
  - **led2_adc_clip:** flashes a LED every time a signal exceeds the range of the LED.

3.2.3 HDL Black Box Tutorial

In this tutorial, an alternative to the common approach of the Toolflow was explored. Here, a Verilog code was embedded (masked) inside a Simulink block and left to operate inside the Toolflow's Frontend Environment. To this end, the Xilinx Black Box was used and a simple traffic lights code was implemented, on which after a certain period of time, given in function of the clock frequencie, three LEDs would operate according to a traffic light. The test was monitored remotely and found to be successful, showing that this approach can be explored further in future works. Its main core is shown in Figure 3.13.

Figure 3.13 - HDL Black Box Tutorial



SOURCE: Author's work

3.2.4 MATLAB Toolbox Dependency

On the CASPER Documentation Wiki, the most recent entry related to the required MATLAB Toolboxes was considerably deprecated by the time of the beginning of this work, dating back to the R2008b version of MATLAB, released in 2008[1]. The Wiki article referred to a slightly updated entry, which contained, nonetheless, no useful information regarding which Toolboxes to install[2]. In addition, the more up-to-date CASPER ReadTheDocs[3] portal does not list any additional requirements for the CASPER frontend besides MATLAB and Simulink.

As such, the latest available information regarding this issue listed four Toolboxes as being needed for the correct functioning of the Toolflow. These were the Fixed Point Toolbox, the Signal Processing Blockset, the Signal Processing Toolbox and the Simulink Fixed Point.

For more recent versions of MATLAB/Simulink, these had been updated and, specifically for 2013b, the number of four Toolboxes could be reduced to three, namely, the Fixed-Point Designer Toolbox, the DSP System Toolbox and the Signal Processing Toolbox. We then required a test period in order to evaluate which Toolboxes were necessary for the compilation of our designs, and the results are summarized in the next subsections.

Initially, the methodology of testing revolved around Tutorials 1 and 3. By uninstalling and reinstalling the licenses one after another, it was then tested if the compilation of both these designs were possible. However, after reaching the conclusion that the first tutorial would compile in any configuration, given that it compiled with no toolboxes, the process then analysed if a set of Toolboxes was capable of compiling the third CASPER tutorial.

3.2.4.1 Fixed-Point Designer Toolbox

This Toolbox is listed as being used to perform operations with the UFix and the Fix data types, which are not available with the basic Simulink Fixed-Point Structure. However, as we were able to compile the first design, this remains unsolved for the purposes of this work. It is worth mentioning that, on the CASPER website, upon the absence of this toolbox, signals will be replaced by single-precision signals. In

[1] https://casper.astro.berkeley.edu/wiki/MSSGE_Toolflow_Setup

[2] https://casper.astro.berkeley.edu/wiki/MSSGE_Setup_with_Xilinx_14.x_and_Matlab_-2012b

[3] https://casper-toolflow.readthedocs.io/projects/tutorials/en/latest/

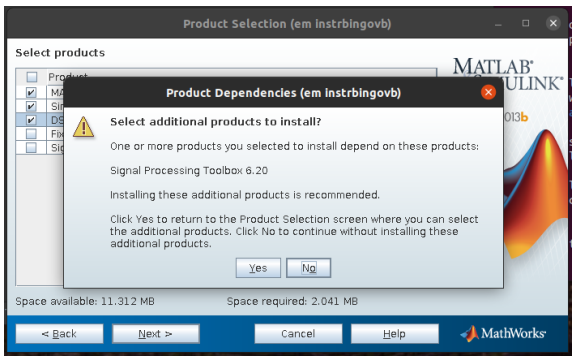
the end, as this did not seem to affect the results, this Toolbox was taken as not needed. However, further investigations should be made regarding this topic.

3.2.4.2 Signal Processing Toolbox and DSP System Toolbox

As the name implies, these two toolboxes are intended to provide the Digital Signal Processing capabilities for the frontend of the CASPER Toolflow, corresponding in the major part to the green boxes. From the beginning, it was expected that, without those toolboxes, more complex designs which relied upon DSP functions would not be compiled, and this was confirmed without issues, as it will be explained below.

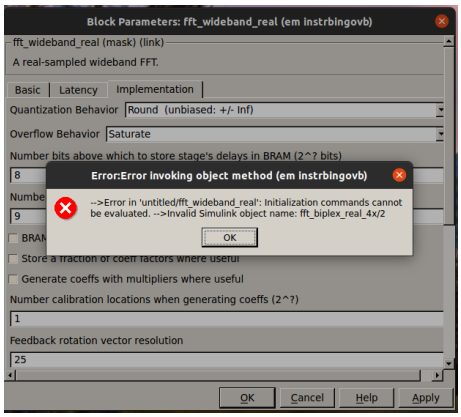
During the installation process, the DSP System Toolbox could not be installed without the concurrent installation of the Signal Processing Toolbox, as shown in 3.14. The contrary process, however, was permitted, but most of the DSP blocks relied upon the DSP System Toolbox. Without both Toolboxes, it was not possible to modify any parameters of the green boxes fft_wideband_real and pfbfirreal, which are critical for Tutorial 3. The error messages related to these impediments are shown in Figures 3.15 and 3.16.

Figure 3.14 - Signal Processing Toolbox and DSP System Toolbox



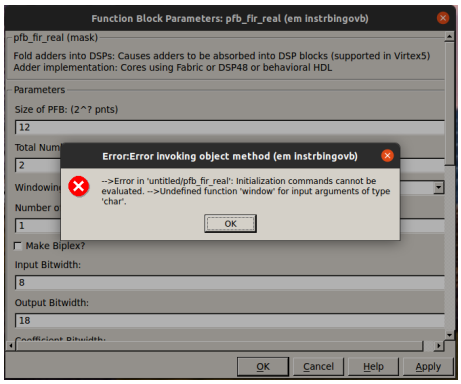
SOURCE: Author's work

Figure 3.15 - Error when Adding fft_wideband_real Block



SOURCE: Author's work

Figure 3.16 - Error when Adding pfb_fir_real Block



SOURCE: Author's work

4 ROACH2: Initial Tests

The initial goal of this project was to implement a spectrometer that could extract the spectral information of the signals from the BINGO radiotelescope. However, as the project developed, its focus shifted to a analysis of the environment on which this spectrometer should be implemented, given that the necessary licenses for the implementation and development were not at the disposal of the workgroup. Therefore, after its successful compilation, Tutorial 3 was used as a prototype spectrometer on which tests were to be performed.

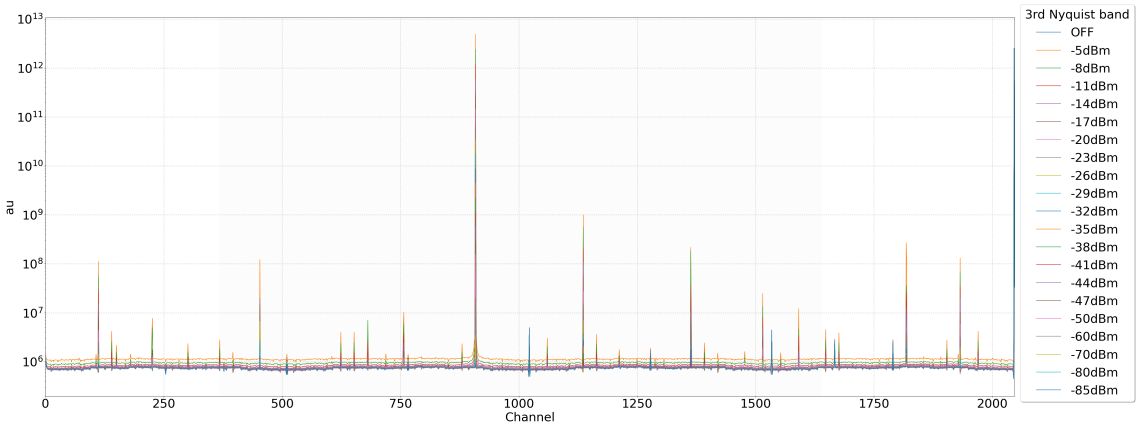
To all the the tests listed below, a version of Tutorial 3 compiled with a test license and clocked with 225 MHz for the board and 900 MHz for the ADC was implemented. This configuration meant that the band of interest was sampled in the 3rd Nyquist Zone of the spectrometer, with a spare band to either side of the band of interest. By doing this, it would be easier to quantify spurious signals outside the BINGO bandwidth.

4.1 Signal Generator Tests

In this test, a single frequency was input into ROACH2 with the use of a signal generator. Here, the group did some measurements with different frequencies, as well as some power measurements to quantify the spectrometer's linear region of operation. Below, in Figure 4.1, a single tone at 1100 MHz was input into ROACH2 with different power values. The values are output in energy-like, arbitrary units (au), due to the accumulation process, which integrates power measurements over time. The spikes seen in various channels were a major concern during the tests. As shown in figura 4.1, their intensity varied according to input power. This is explored in greater detail in figures 4.2 and 4.3. 4.2 corresponds to the spikes observed with an input signal possessing -5 dBm of power, while 4.3 shown the spikes observed when only a clock source is provided. After inquiring some members of the project, the workgroup reached the conclusion that the spikes are expected for this model of spectrometer and that they appear due to the unusual scenario of a single tone, high intensity pulse being input to the board. However, the group did not discard the hypothesis that the appearance of those spikes can be related to a synchronization issue, with this line of thought being open to future investigations.

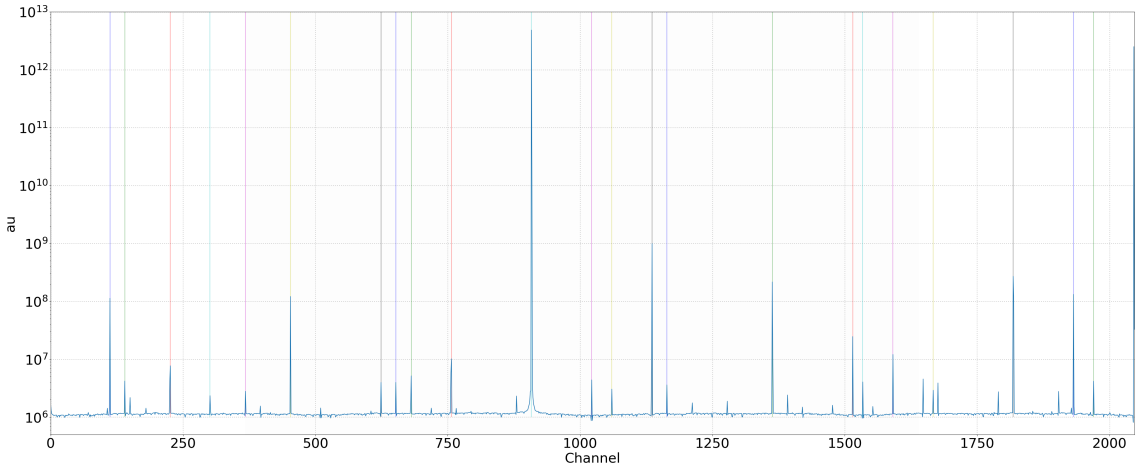
Some tests were also made by changing the integration time of the accumulators. Considering that the ADCs were configured to operate at 900 MHz and that the FFTs were configured to 2048 channels, meaning that 4096 channels were needed to

Figure 4.1 - Power Measurements at 1100 MHz



SOURCE: Author's work, in collaboration with F. Vieira

Figure 4.2 - Power Measurements at 1100 MHz (-5 dBm)

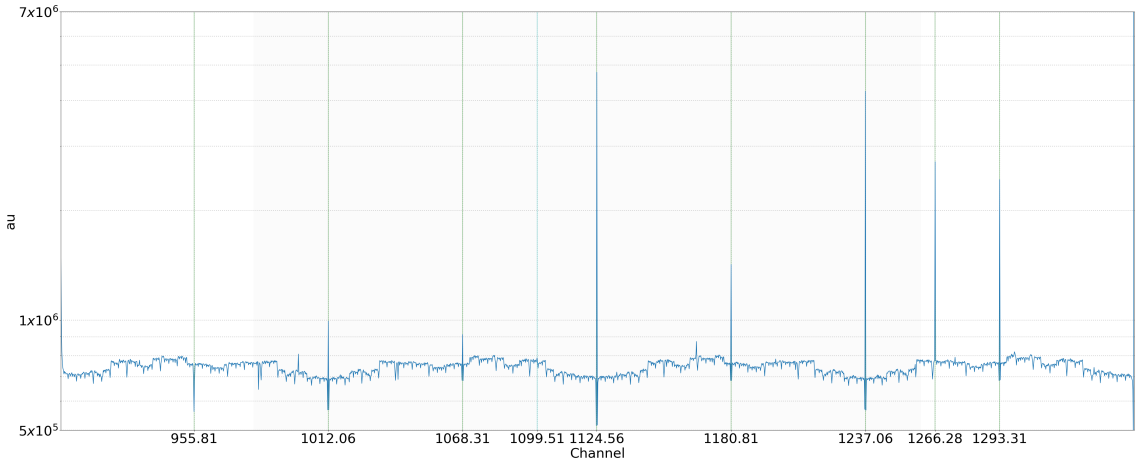


SOURCE: Author's work, in collaboration with F. Vieira

output a spectrum, the team estimated that, for approximately 1 s of accumulation, the acc_len parameter would have to be set to $9 \cdot 10^8/4096$, that is, the number of samples digitized in 1 s ($9 \cdot 10^8$), divided by the number of samples needed to output a spectrum (4096). The results are shown in Figure 4.4 for three integration times:

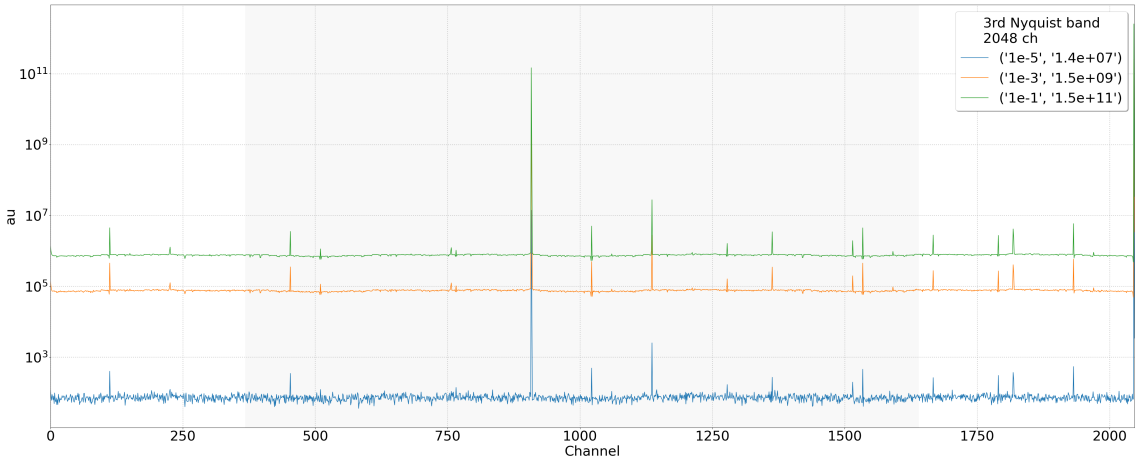
In order to determine the linear regime of the spectrometer, multiple measurements were done, with 89 different power inputs at channel 954, corresponding to the frequency of 1100 MHz. To convert the arbitrary units to a power figure, all measurements were divided by their integration time. The results are shown in 4.5.

Figure 4.3 - Power Measurements at 1100 MHz (No signal)



SOURCE: Author's work, in collaboration with F. Vieira

Figure 4.4 - Integration Time Tests (-20 dBm)

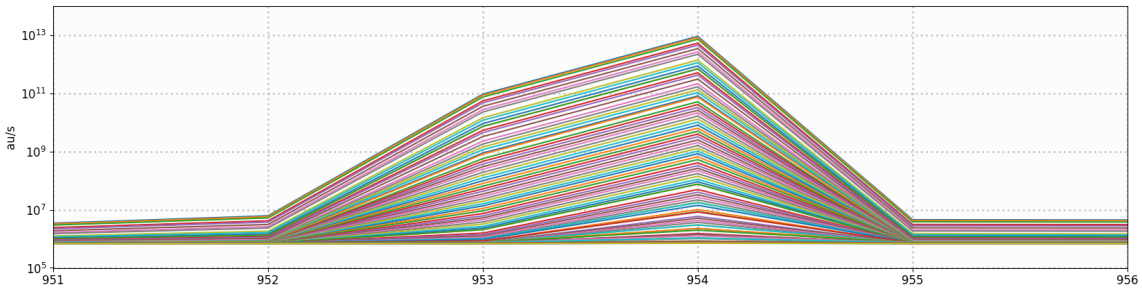


SOURCE: Author's work, in collaboration with F. Vieira

From the above measurements, a linear fit was applied to determine the linear regime of the implemented spectrometer. First, the power measurements in au were plotted in a logarithmic scale, with the horizontal axis showing the input power of the signal. Then, a fit was applied from au, shown in Figure 4.6 and in equation 4.1. After that, the same fit was applied from dBm, as shown in figure 4.7 and in equation 4.2.

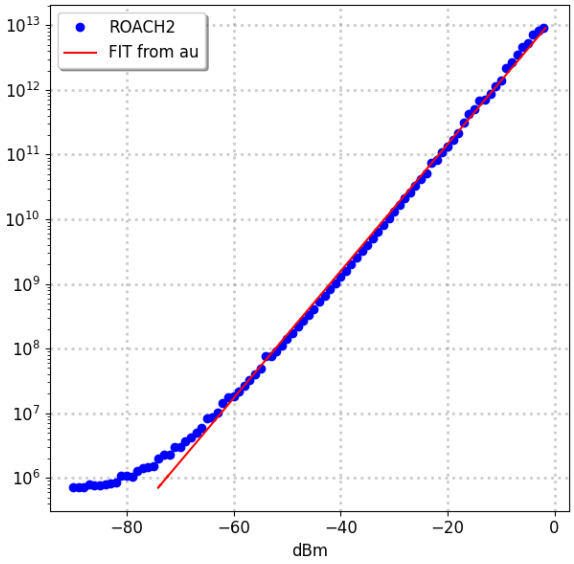
$$\text{dBm} = 10.187945 \cdot \log(\text{au}) - 133.69334364 \tag{4.1}$$

Figure 4.5 - 89 Different Power Measurements (1100 MHz)



SOURCE: Author's work, in collaboration with F. Vieira

Figure 4.6 - Fit from au to dBm



SOURCE: Author's work, in collaboration with F. Vieira

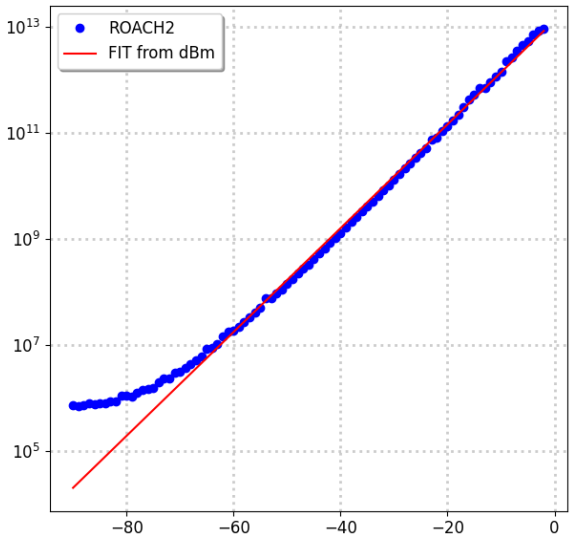
$$\text{au} = 10^{0.09797413 \cdot \text{dBm} + 13.11627086} \tag{4.2}$$

In conclusion, the linear regime of the spectrometer was found to be inside the -60 dBm to -2 dBm range. However, further measurements should be made in different frequencies to increase the number of available data and the accuracy of these fits.

4.2 Prototype Radiometer Chain

Upon finishing the tests with the signal generator, the workgroup then proceeded to test the spectrometer with a prototype radiometer chain. The chain had 52 dB of

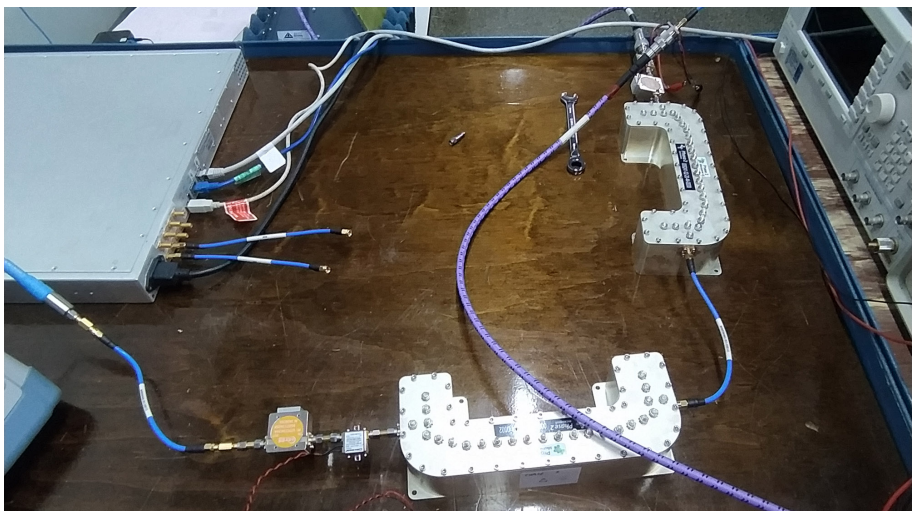
Figure 4.7 - Fit from dBm to au



SOURCE: Author's work, in collaboration with F. Vieira

total gain, with a filter to limit the input signal inside the BINGO bandwidth and a primary Low Noise Amplifier (LNA) with 38 dB of gain and a noise figure of 0.35 dB. A picture of the chain is shown in 4.8.

Figure 4.8 - Prototype Radiometer Chain

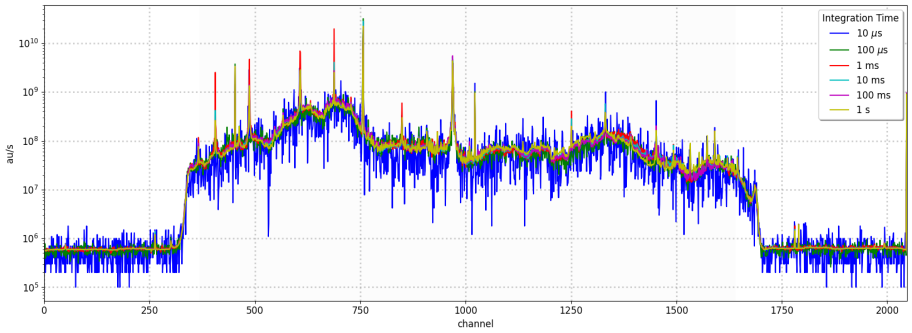


SOURCE: F. Vieira

Measurements were made with the chain to assess the difference between different

integration times, ranging from 1 s to 10 μs. These are shown in figure 4.9, with the units converted to power.

Figure 4.9 - Integration Time Tests (Radiometer Chain)

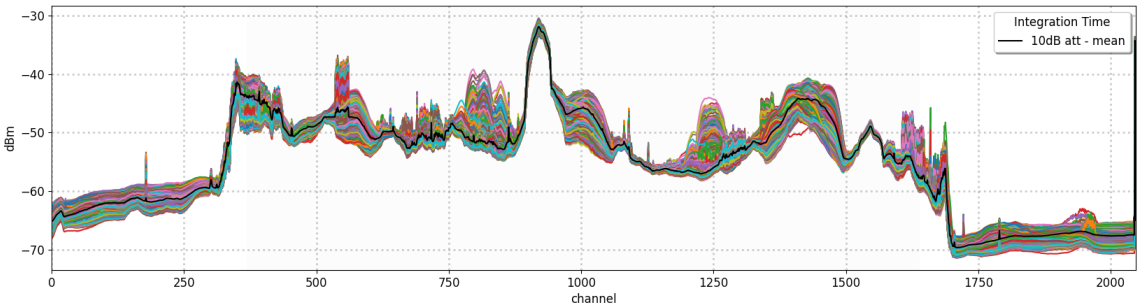


SOURCE: Author's work, in collaboration with F. Vieira

4.3 BDA Site Measurements

With the aid of the implement spectrometer, the group performed interference tests in the Brazilian Decimetric Array (BDA) site. This was done through an omnidirectional antenna which stayed on for two distinct periods: from 8 pm until 1 am; and from 5 am until 10 am, accumulating a number of 17601 spectra for the former and 17812 spectra for the latter. The results are shown in Figures 4.10 and 4.11, below.

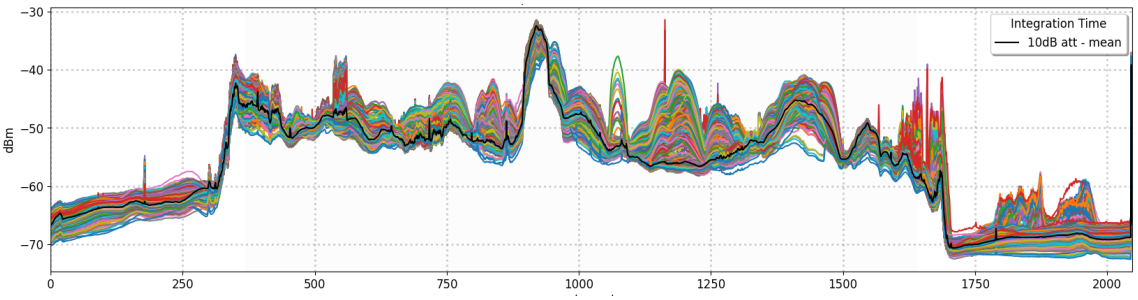
Figure 4.10 - Omnidirectional BDA Measurements (17601 spectra from 8pm to 1am)



SOURCE: Author's work, in collaboration with F. Vieira

In both figures above, the black line indicates the mean of all accumulated spectra. Signals that deviate from this mean with high intensity are interpreted as interfer-

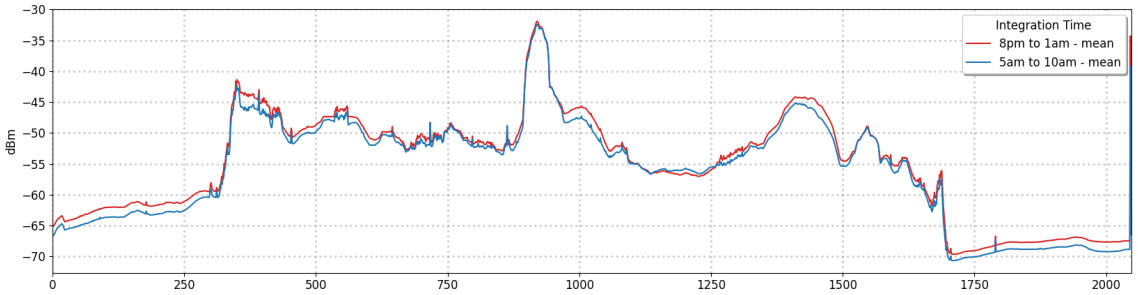
Figure 4.11 - Omnidirectional BDA Measurements (17682 spectra from 5am to 10am)



SOURCE: Author's work, in collaboration with F. Vieira

ence signals. Both means are shown in Figure 4.12 below.

Figure 4.12 - Omnidirectional BDA Measurements (Means)



SOURCE: Author's work, in collaboration with F. Vieira

4.4 MATLAB 2012a Test Attempt

The group did, however, attempt to modify the parameters of the spectrometer to obtain a more precise benchmarking of the number of channels, the number of taps and the window type applied in the PFB frontend. However, these compilations were performed with a poorly tested version of MATLAB 2012a, which did not show the same result pattern as the ones shown above, even for spectrometers compiled with the same parameters in both versions. Therefore, they are not included in this work.

5 CONCLUSIONS

At the end of this work, expertise related to installing the adequate environment for the development of spectrometers using the CASPER Toolflow was obtained. Besides, it was also verified that MATLAB/Simulink licenses are necessary for a conventional compilation of the Toolflow, and that the most critical of the paid Toolboxes are the DSP System and the Signal Processing Toolbox, though further investigation should be made regarding the Fixed-Point Designer Toolbox. Future work regarding the software environment is related to the installation of the required software for the SKARAB board and investigation of the possibility of embedding HDL inside the Simulink environment,

It was also possible to make an assessment of the performance of a prototype spectrometer, which is now functional. Further work should be done in optimizing this design, such as establishing a communications protocol and fine-tuning the design for the BINGO site. In summary, even though the goals of the project could not be completely fulfilled, it can be considered a success in various aspects.

REFERENCES

ABDALLA, E. et al. The BINGO Project I: Baryon Acoustic Oscillations from Integrated Neutral Gas Observations. **arXiv e-prints:2107.01633**, 2021. 1

D'CRUZE, M. J. **Spectral Line Surveys with Jodrell Bank Telescopes**. 279 p. Thesis (PhD in Science and Engineering) — University of Manchester, Manchester, 2018. 1, 3

HICKISH, J. et al. A Decade of Developing Radio-Astronomy Instrumentation using CASPER Open-Source Technology. **Journal of Astronomical Instrumentation**, v. 5, n. 4, p. 1641001–12, dec. 2016. 6, 7, 8

LYONS, R. G. **Understanding Digital Signal Processing**. 3. ed. Ann Arbor: Pearson, 2011. 3

MCMAHON, P. L. Adventures in radio astronomy instrumentation and signal processing. **arXiv e-prints:1109.0416**, sep. 2011. 1, 3

NG, C. et al. CHIME FRB: An application of FFT beamforming for a radio telescope. In: **2017 XXXIInd General Assembly and Scientific Symposium of the International Union of Radio Science (URSI GASS)**. [S.l.: s.n.], 2017. p. 1–4. 1

PRICE, D. C. Spectrometers and polyphase filterbanks in radio astronomy. **arXiv e-prints:1607.03579**, jul. 2016. 1, 3, 4

STANKO, S.; KLEIN, B.; KERP, J. A field programmable gate array spectrometer for radio astronomy. First light at the Effelsberg 100-m telescope. **Astronomy and Astrophysics**, v. 436, n. 1, p. 391–395, jun. 2005. 1

SURNIS, M. P. et al. GREENBURST: A commensal Fast Radio Burst search back-end for the Green Bank Telescope. **Publications of the Astronomic Society of Australia**, v. 36, p. e032, 2019. 1

URBINA, K. V. C. **Wideband Digital Spectrometers on the ROACH Field Programmable Gate Array**. 91 p. Thesis (Master of Science with Mention in Physics) — Universidad de Concepción, Concepción, 2014. 6

WERTHIMER, D. The CASPER collaboration for high-performance open source digital radio astronomy instrumentation. In: **2011 XXXth URSI General Assembly and Scientific Symposium**. [S.l.: s.n.], 2011. p. 1–4. 7

WUENSCHE, C. A. et al. The BINGO telescope: a new instrument exploring the new 21 cm cosmology window. In: **Journal of Physics Conference Series**. [S.l.: s.n.], 2019. (Journal of Physics Conference Series, v. 1269), p. 012002. 1

_____. The BINGO Project I: Baryon Acoustic Oscillations from Integrated Neutral Gas Observations. **arXiv e-prints:2107.01633**, jul. 2021. 1

XILINX. **Development System Reference Guide 10.1**. [online], 2008. 500 p. 9

_____. **System Generator for DSP**. [online], oct 2012. 424 p. 9