



KIT  
Karlsruhe Institute of Technology



Institut für Technik der  
Informationsverarbeitung  
(ITIV)

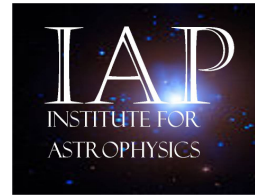
Head of Institute

Prof. Dr.-Ing. K. D. Müller-Glaser

Prof. Dr.-Ing. Dr. h. c. J. Becker

Prof. Dr. rer. nat. Wilhelm Stork

in cooperation with



# Prototype for a base station for supernova remnant HI line radio wave detector and analyzer (SRWDA)

Master Thesis

by  
Suheeb Kassar

Supervisor: M.Sc. M.Eng. Samir Mourad, *IAP (AECENAR)*

Main Referee: Prof. Dr. rer. nat. Wilhelm Stork, *ITIV (KIT)*

---



I hereby declare that I wrote on my own and that I have followed the regulations relating to good scientific practice of the Karlsruhe Institute of Technology (KIT) in its latest form. I did not use any unacknowledged sources or means and I marked all references I used literally or by content.

Karlsruhe, 6th of April 2014

-----

M. Suheeb Kassar

---

## **Abstract**

With the increasing dependence on the communication technology with wide bandwidth in more Applications and devices especially in wireless Field SDR (Software Defined Radio) is employed which provide flexibility in technical point of view (Software) respectively market challenges.

The advantages of SDR as multiband, multicarrier, multimode, Multirate and variable bandwidth enable to scan a wide range of frequencies with modified channel (modulation).

On the other hand SDR concept can be implemented in different reconfigurable Hardware such as FPGA und DSP which means that replacing another analog elements for example antenna in Hardware can be effected using software design concept in reprogrammable und reconfigurable Systems .

This thesis describes the design and implementation of radio wave detector and analyzer (SRWDA) for processing supernova radio signals collected using four antennas which gives also direction information of the signals

The implemented Hardware in Software reduce the cost of station and at same time the right choice for accelerated digital signal processing. The detected signal can be displayed on GUI to observe a desired signals.

The project was carried out in corporation with the Institute for Astrophysics (IAP). Some of the work was carried out in the IAP facility in Ras Nhache/Batroun/Lebanon.

# Table of contents

<b>ABSTRACT</b> .....	<b>II</b>
<b>TABLE OF CONTENTS</b> .....	<b>III</b>
<b>LIST OF FIGURES</b> .....	<b>V</b>
<b>LIST OF TABLES</b> .....	<b>VI</b>
<b>LIST OF FORMULAS</b> .....	<b>VII</b>
<b>LIST OF KEYWORDS</b> .....	<b>VIII</b>
<b>1 INTRODUCTION</b> .....	<b>9</b>
1.1 BACKGROUND .....	9
1.2 MOTIVATION .....	9
1.3 OBJECTIVES.....	<b>FEHLER! TEXTMARKE NICHT DEFINIERT.</b>
1.4 OUTLINE.....	9
<b>2 BASICS</b> .....	<b>10</b>
2.1 21-CM RADIO ASTRONOMY BASICS .....	10
2.2 RADIO WAVE RECEIVING .....	11
2.3 SOFTWARE DESIGNED RADIO.....	11
2.4 BASIC SDR RECEIVER ARCHITECTURE.....	12
<b>3 SYSTEM DESIGN</b> .....	<b>14</b>
<b>4 MECHANICS</b> .....	<b>FEHLER! TEXTMARKE NICHT DEFINIERT.</b>
4.1 MECHANICAL DESIGN .....	<b>FEHLER! TEXTMARKE NICHT DEFINIERT.</b>
<b>5 HARDWARE CONCEPTION</b> .....	<b>16</b>
5.1 PLATFORM SELECTION .....	16
5.1.1 Hackrf platform.....	16
5.1.2 BladeRF Platform .....	17
5.1.3 Ettus Research's USRP B100.....	17
5.1.4 Rtl_SDR(820).....	18
<b>6 HARDWARE REALIZATION</b> .....	<b>19</b>
6.1 HARDWARE OF USRP .....	21
6.1.1 USRP B100 as motherboard .....	21
6.1.2 WBX 50-2200 MHz Rx/Tx as daughterboard.....	22
6.1.3 FPGA .....	23
6.1.4 ADC converter .....	25
6.2 LOW NOISE AMPLIFIER LNA .....	25
6.3 LINE AMPLIFIER.....	29
6.4 ANTENNA .....	29
<b>7 SOFTWARE</b> .....	<b>31</b>
7.1 SOFTWARE DESIGN .....	<b>FEHLER! TEXTMARKE NICHT DEFINIERT.</b>
7.2 SOFTWARE IMPLEMENTATION .....	36
<b>8 INTEGRATION</b> .....	<b>FEHLER! TEXTMARKE NICHT DEFINIERT.</b>
<b>9 INTEGRATION TEST</b> .....	<b>FEHLER! TEXTMARKE NICHT DEFINIERT.</b>

---

<b>10</b>	<b>FUTURE WORK</b> .....	<b>40</b>
<b>11</b>	<b>LITERATURE</b> .....	<b>41</b>
<b>12</b>	<b>APPENDIX</b> .....	<b>42</b>
A.1	SOFTWARE .....	42
A.1.1	<i>System Software</i> .....	<i>Fehler! Textmarke nicht definiert.</i>
A.1.1.1	Example.c.....	<b>Fehler! Textmarke nicht definiert.</b>
A.2	HARDWARE.....	51
A.2.1	<i>Layout motherboard USRP B100</i> .....	51
A.3	LIST OF OFF-THE-SHELF ELECTRONIC COMPONENTS AND ASSEMBLIES .....	56
A.4	MECHANICAL COMPONENTS .....	57

## List of figures

Figure 1: hydrogen emission.....	1
Figure 2: Spain mechanisms .....	1
Figure 3: Radio system.....	1
Figure 4: Model of Software Radio [1] .....	1
Figure 5: Block scheme of typical SDR receiver.....	1
Figure 8: Overview and comparison of different platforms .....	16
Figure 9:USRP Instant SDR Kit .....	21
Figure 10: Architecure of B100 .....	22
Figure 11: RF Front-End diagram.....	23
Figure 12: Receiver Blocks.....	24
Figure 13: DDC Block.....	25
Figure 14: Schematic and platine of LNA .....	26
Figure 15: Test of LNA .....	26
Figure 16: Output signal of LNA .....	26
Figure 17 .....	1
Figure 18.....	<b>Fehler! Textmarke nicht definiert.</b>
Figure 19: Implementation of dial_tone example within Gnu Radio Companion.....	1

---

## List of tables



## List of formulas

Formel 1.1: block diagram of a generic radio.....

# List of Keywords

**ADC** Analog-to-Digital converter

**AF** Audio Frequency

**AGC** Automatic Gain Control

**AM** Amplitude Modulation

**BFO** Beat Frequency Oscillator

**CW** Continuous Wave

**DAB** Digital Audio Broadcasting

**DVB-T** Digital Video Broadcasting - Terrestrial

**RTL-SDR** Software Defined Radio Dongle

**FFT** Fast Fourier Transform

**FM** Frequency Modulation

**FPGA** Field Programmable Gate Array

**FSF** Free Software Foundation

**GRC** GNU Radio Companion

**IF** Intermediate Frequency

**IAP** Institute for AstroPhysics

**I/O** Input/output

**ISO** International Organization for Standardization

**LNA** Low Noise Amplifier

**RDS** Radio Data System

**RF** Radio Frequency

**SRD** Software Defined Radio

**SRWDA** supernova radio wave detector and analyzer.

**USB** Universal Serial Bus

# **1 Introduction**

## **1.1 Background**

The gained information about the universe has been interpreted using radio frequency electromagnetic signals that can be detected through radio digital receiver. One of the most important beams is HI line 21cm that carry information about hydrogen atoms in space to get different astronomic parameters of hydrogen atom as speed, direction or distance within the galaxy etc. The rotation curve of our galaxy has also been calculated using the 21-cm hydrogen line. It is then possible to use the plot of the rotation curve and the velocity to determine the distance to a certain point within the galaxy [1].

The digital technology in communication scope made possible to measurement a spectrum line and processes to get clean signals.

## **1.2 The SRWDA Project**

The radio astronomical IAP project supernova radio wave detector and analyser (SRWDA) aims to detect and analyze HI radio signals from supernova remnants. At the base station a set of antennas, which gives also direction information of the signals, is aimed to be connected to a computer which acts as Software Defined Radio (SDR). Afterwards an analysing program is aimed to be installed.

Later the detectors are planned to be installed on satellites in the IAP SRWDA-SAT project [9] to improve the resolution and to suppress disturbing signals from earth stations.

## **1.3 Objectives**

In this thesis it is aimed to build a first prototype for a ground station for SRWDA.

## **1.4 Outline**

This thesis presents the design and implementation of SDR (Software Defined Radio) receiver to detect spectral line 21 cm of hydrogen atom for radio astronomy using reconfigurable platform and open source software for implementing Digital signal processing.

Chapter 2 gives some scientific and technical background information.

Chapters 3 until the last chapter describe the successive development process of the prototype according to the V model.

Chap

## 2 Basics

### 2.1 21-cm Radio Astronomy Basics

The hydrogen gas is one important of the main materials that can be found in throughout of the space. The 1420 MHz radiation form hydrogen pass through the Earth's atmosphere and gives us a more complete map of the hydrogen than that of the stars themselves since their visible light won't penetrate the dust clouds. The radiation comes from the transition between the two levels of the hydrogen 1s ground state, slightly split by the interaction between the electron spin and the nuclear spin. In this procedure hydrogen in its lower state will absorb 1420 MHz.

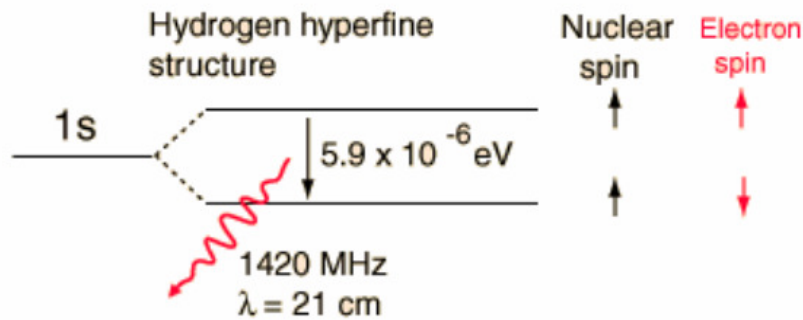


Figure 1: hydrogen emission

The electron moving around the proton can have a spin in the same direction as the proton's spin (i.e., parallel) or spin in the direct opposite direction as the proton's spin (i.e., anti-parallel). The energy state of an electron spinning antiparallel is slightly lower than the energy state of a parallel-spin. Since atoms always want to be in the lowest energy state possible, the electron will eventually flip to the anti-parallel spin direction if it were in the parallel spin direction. The energy difference is very small, so a hydrogen atom can wait on average a few million years before it undergoes this transition. The advantage for detecting the 21cm signal is to calculate the mass of galaxies, to put limits on any changes over time of the universal gravitational constant and to study dynamics of individual galaxies and to plot the rotation curve of our galaxy respectively the velocity [1].

This splitting of the hydrogen ground state is extremely small compared to the ground state energy of  $-13.6 \text{ eV}$ , only about two parts in a million. The two states come from the fact that both the electron and nuclear spins are  $1/2$  for the proton, so there are two possible states, spin parallel and spin antiparallel. The state with the spins parallel is slightly higher in energy (less tightly bound).

In visualizing the transition as a spin-flip, it should be noted that the quantum mechanical property called "spin" is not literally a classical spinning charge sphere. It is a description of the behaviour of quantum mechanical angular momentum and does not have a definitive classical analogy. The observation of the 21cm line of hydrogen marked the birth of spectral-line radio

astronomy. It was first observed in 1951 by Harold Ewen and Edward M. Purcell at Harvard, followed soon afterward by observers in Holland and Australia.

The prediction that the 21 cm line should be observable in emission was made in 1944 by Dutch astronomer H. C. van de Hulst [2]. A radio telescope is powerful tool, which can “see” radio waves emitted by radio sources in throughout the space.

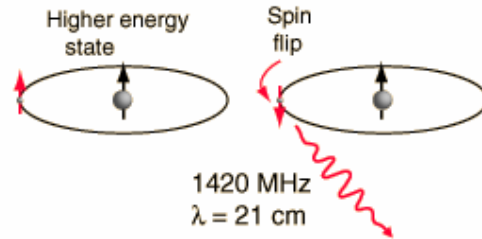


Figure 2: Spin mechanisms

## 2.2 Radio Wave Receiving

Firstly we want to present the simple radio system diagram and explain their components to get better understanding for complete system.

The electromagnetic signal has been received by an antenna and then converted into an electrical signal. This signal is normally very weak and disturbed because of many factors as atmosphere ... etc. therefor the noise must be removed from a signal amplified before it processing later. That will be happened in the next stage (RFF) that processes a signal for ADC converter. RFF consists of filter, amplifier und mixer to convert the radio frequency to lower frequency. The resulted analog signal from RFF has to be digitized using *Analog -Digital converter* for further processing in suitable hardware components as Digital Signal Processing (DSP), Field Programmable Gate Array (FPGA) or microprocessor [2].

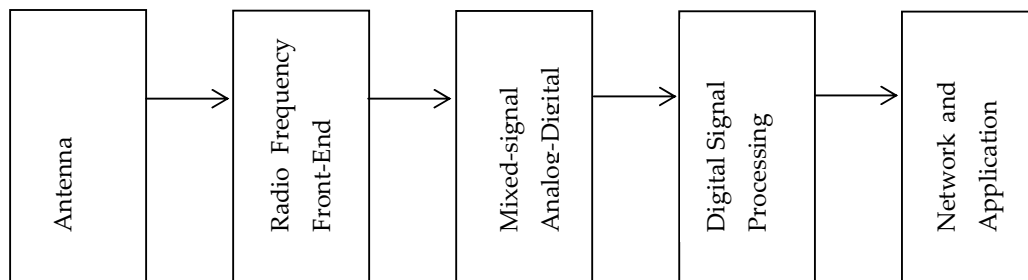


Figure 3: Radio system

Finally a processed radio signal can interface in modern system to a network or an application.

## 2.3 Software Designed Radio

The definition software radio was Joe Mitola in 1991 to refer to the class of reprogrammable or reconfigurable radios [3]. That means, the same hardware components can execute different

---

functions at different times. However, a radio defines in software its modulation, error correction, and encryption processes, exhibits some control over the RF hardware, and can be reprogrammed is clearly a software radio.

SDR is defined as a described radio in Software and whose physical layer behaviour can be significantly altered through changes to its software in which the receive digitization is performed at some stage downstream from the antenna, typically after wideband filtering, low noise amplification, and down conversion to a lower frequency in subsequent stages – with a reverse process occurring for the transmit digitization. Digital signal processing in flexible and reconfigurable functional blocks defines the characteristics of the radio [4].

Hardware components of the traditionally radio system consists of the mixers, filters, amplifiers and oscillators are replaced by software in SDR as indicated in the Figure 2.2. The hardware for signal processing should be selected for high speed signal processing for example as GGP, FPGA or DSP. The main advantage of an SDR is reconfigurability and customizable. The implementing radio functions in software are more flexible than in hardware for a radio device to be reconfigured for different use cases rather needing to redesign the hardware to support new functionality. With the rapid evolution of wireless protocols and standards, a hardware radio could be made obsolete due to the inability to conform to new standards or protocols. An SDR, however, could be reconfigured to support new standards that may not have existed at the time the device was built. Such flexibility is attractive to the manufacturers of the devices as it enables them to update their SDR product through software and not necessarily need to change the hardware of the radio. Such flexibility is helpful to fast update suchlike systems and allows manufacturer to redesign the system without changing the hardware architecture.

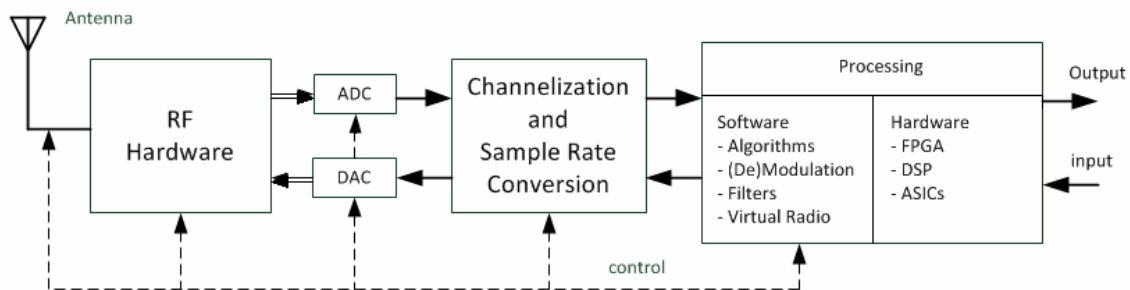


Figure 4: Model of Software Radio [1]

## 2.4 Basic SDR Receiver Architecture

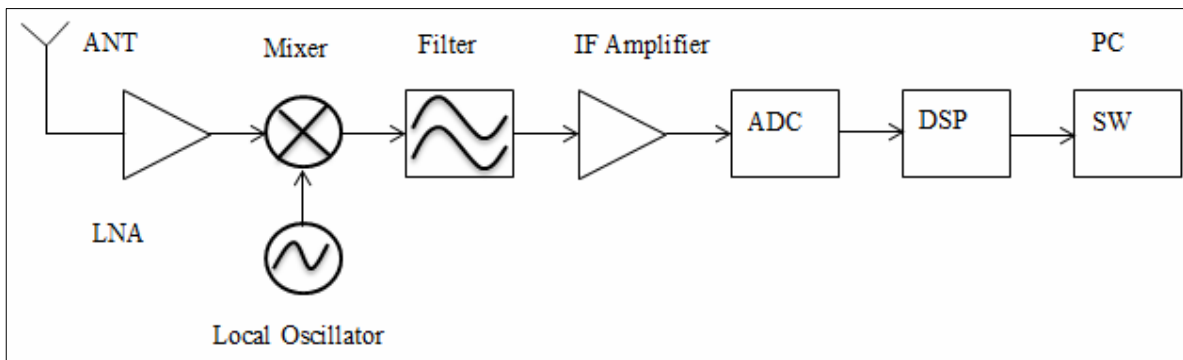
The most used radio receivers use the architecture of super heterodyne receiver, in which a received signal has been converted to a fixed intermediate frequency, which can be more processed than the original radio carrier frequency. A detected signal is picked up via the antenna, mixed with local oscillator to reduce frequency of an incoming signal to intermediate frequency (IF) for further processing, filtered due unwanted signals filtered using low pass filter and finally amplified with a low noise amplifier (LNA) for accommodation to ADC converter.

The digitized IF signal repeat oneself as spectral harmonic that can be placed near the baseband frequency, helping frequency translation and digitization to be carried out simultaneously. For next processing digital filtering and downconversion of sample rate (for receiving) are needed to change a sampled signal from high frequency to lower frequency allowing a hardware as DSP or FPGA to process it with suitable sample rate without loss of the information.

Typical SDR receivers have a simple und cheap direct-conversion or low intermediate frequency architecture. The ideal SDR receiver would have all the radio-frequency bands and modes defined software-wise, meaning it would consist only of an antenna, ADC and a programmable processor. Typically, RF front-end of a SDR will consist of antenna circuitry, amplifiers, filters, local oscillators and ADCs. When the signal is received, it is amplified and its carrier frequency downconverted to a low-intermediate frequency in order for ADC to perform digitization.

The processing is done by some of the computational resources at our disposition – mainly, General Purpose Processors (GPPs), Digital Signal Processors (DSPs) and Field Programmable Gate Arrays (FPGAs), whereas some of the future resources may include a combination of the aforementioned, thus extending the computational capacity. One of the most important aspects when deciding on a computational resource that is to be used in the system is its reprogrammability (important for implementation of new waveforms), therefore dedicated-purpose circuitry is generally avoided in SDRs.

The basic receiving process is illustrated in Figure 2.3 with its principle design constraints will be explained in this chapter.



**Figure 5: Block scheme of typical SDR receiver**

The important design parameters that have to be considered during design SDR receiver system are: input sensitivity, maximum expected input signal and blocker specifications.

### 3 System Design

The HI spectrum signal will be detected by the antenna which converts the electromagnetic signal to analog signal to process with SDR system (Hardware and Software). Because of the weakness of the incoming HI spectrum signal we have to use a low noise amplifier for filtering the incoming signals and amplifying the hydrogen line signal at 1420 MHz. The filtered signal is then available for further processing phase. It is important to integrate a line amplifier in the design taking into account the signal line loss by using coaxial cable. For further high signal processing SDR platform (SDR-hardware and-software) were needed to realize **SRWDA** model. Two designs have been developed in IAP institute. The figures 6 und 7 give both alternative designs explained.

A software-defined radio (SDR) system is a hardware and software co-design system which can tune to any frequency band and preform different communication functions (modulation, demodulation, etc.) by means of a programmable hardware, which is controlled by software. SDR platform software performs various amounts of digital signal processing in PC [1]. SDR has evolved with advancement in computing machines and high-speed analog-digital (A/D) and digital-analog (D/A) converters. Thus SDR allows a single device to support a wide range of capabilities previously available only through multiple products [1].

As explained signals received by an antenna should be digitized so that SDR may processes these digitized samples at RF frequencies (around GHz frequencies) to perform further signal processing steps (IF conversion, filtering, baseband conversion etc). But due to certain limitations like unavailability of A/D converters at very high frequencies (starting from a few GHz) and very high speed general purpose computers, digitization takes place after the IF or baseband demodulator stage. That typically means that hardware is still required to convert the signals of interest into and out of the “baseband” frequencies in the digital domain, but all of the complex processing performed at baseband is handled in the digital software domain [1]. Yet, such hardware is usually fairly simple comprising of a local oscillator and mixer, and a pair of low-pass filters.

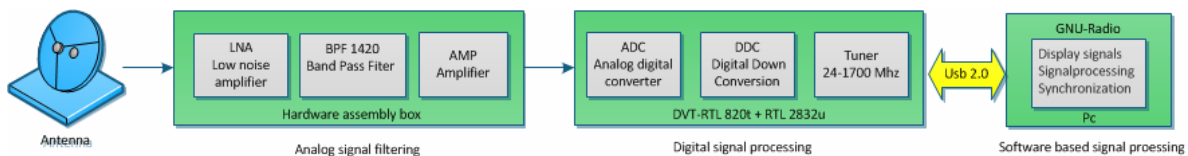


Figure 6: 1.System design overview

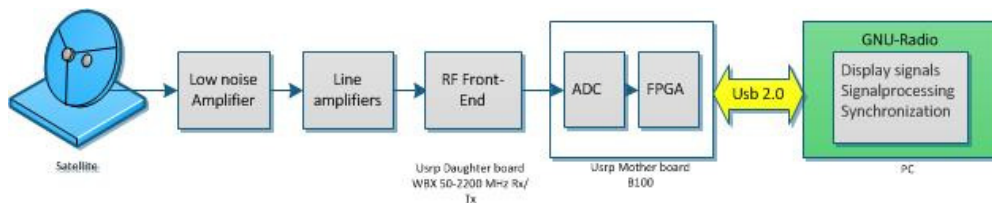


Figure 7: 2.System design overview



The digital signal will be sent via usb2.0 to host pc which software *Gnu-Radio* uses to control SDR hardware platform and transmit and receive data. GNU Radio is an open-source software toolkit. that, connected with hardware equipment such as USRP, allows for building Software Defined Radios connect with hardware. We will later go into details also hardware components and software respectively.

---

## 4 Hardware Conception

### 4.1 Platform Selection

This chapter describes the core idea for selecting receiver components to implement our system and the desired environment work. These are the criterions which have to be fulfilled by a suitable platform:

- The receiver system operates in 1420 MHz.
- Suitable for low-cost experimentation.
- Reprogrammability.
- Fully open source platform (hardware, software).

Couple of weeks of research narrowed the options down to the following SDR platforms:

SDR Platform	Hackrf	BladeRF	USRP(B100)	Rtl_SDR(E4000)
Frequency range	30 MHz – 6 GHz	300 MHz – 3.8 GHz	50 MHz – 2.2 GHz	52 – 2200 MHz
Bandwidth	20 MHz	28 MHz	16 MHz	3.57MHz
Simple size (ADC-DAC)	8 bit	12 bit	12 bit/14 bit	8 bit
Simple rate (ADC-DAC)	20 Msps	40 Msps	64 Msps/128 Msps	3.2Msps
Transmit?	Yes	Yes	Yes	No
Interface speed	USB 2 (480Mbit)	USB 3(5 gigabit)	USB 2 (480Mbit)	USB (480Mbit)
Open source	Everything (SW+HW)	HDL + Code Schematics	HDL + Code Schematics	-----
Supported OS	Linux, OS X, Windows	Linux, OS X, Windows	Linux, OS X, Windows	Linux, OS X, Windows
Supported software	Gnu radio	Gnu radio	Gnu-radio/ Labview	Gnu radio
Price	\$300	\$420	\$675	\$20

**Figure 6: Overview and comparison of different platforms**

#### 4.1.1 Hackrf platform

Hackrf is open source hardware to build SDR system that has been developed by Michael Ossmann HackRF operates from 30 MHz to 6 GHz, a wider range than any SDR peripheral available today. This range includes the frequencies used by most of the digital radio systems on Earth. It can operate at even lower frequencies in the MF and HF bands when paired with the Ham It Up RF upconverter. HackRF can be used to transmit or receive radio signals. It operates in

half-duplex mode: it can transmit or receive but can't do both at the same time. However, full-duplex operation is possible if you use two HackRF devices.

HackRF is designed primarily for use with a USB-attached host computer, but it can also be used for stand-alone applications with Jared's HackRF PortaPack, an add-on that gives HackRF an LCD screen, directional buttons, and audio ports

HackRF was designed to be the most widely useful SDR peripheral that can be manufactured at a low cost. The estimated future retail price of HackRF is \$300, but you can get one for even less by backing the Kickstarter project today. The most important goal of the HackRF project is to produce an open source design for a widely useful SDR peripheral. All hardware designs and software source code are available under an open source license. The hardware designs are produced in KiCad, an open source electronic design automation tool.

#### **4.1.2 BladeRF Platform**

bladeRF is a Software Defined Radio (SDR) platform make possible to enable a community of hobbyists, and professionals to explore and experiment with the multidisciplinary facets of RF communication. By providing source code modern radio systems will be simplified by covering everything from the RF, analog, and digital hardware design to the firmware running on the ARM MCU and FPGA to Linux kernel device drivers.

The bladeRF can tune from 300MHz to 3.8GHz without the need for extra boards. The current open source drivers provide support for GNURadio among other things, allowing the bladeRF to be placed into immediate use. This gives the bladeRF the flexibility to act as a custom RF modem, a GSM and LTE pico cell, a GPS receiver, an ATSC transmitter or a combination Bluetooth/WiFi client without the need for any expansion cards.

The bladeRF was designed to be highly integrated and fully reprogrammable. This means more than just providing source code to modify the host software. The USB 3.0 (Cypress FX3) microcontroller firmware is available to modify, as is the Altera Cyclone IV FPGA VHDL, bringing developers as close to the RF transceiver as possible.

#### **4.1.3 Ettus Research's USRP B100**

The Universal Software Radio Peripheral (USRP) is a digital acquisition (DAQ) system containing four 64 MS/s, 12-bit A/D converters (ADCs), four 128 MS/s, 14-bit D/A converters (DACs), and supports USB 2.0 interface or Ethernet(USRP N210). The USRP is capable of processing signals with 16 MHz of bandwidth.

The USRP takes daughter-cards to map the frequency ranges of interest into the "baseband" that is visible by the A/D hardware. The USRP is a most widespread SDR's in academic environments, Ettus products still count as the best-buy platforms for research. Ettus Research offers several platforms – USRP, USRP2 and USRP N210 that differ in the level of instantaneous bandwidth they can process; reprogrammability of the FPGA; type of interface to the computer (USB or Ethernet) and, of course, price..

---

#### 4.1.4 Rtl\_SDR (RTL2832+ E4000):

RTL-SDR is a very cheap software defined radio, that uses a DVB-T TV tuner dongle based on the RTL2832U chipset (USB devices intended to allow you to watch over-the-air DVB-T broadcast television) ,had a special “mode” that allow them to be used as crude SDR receivers. In this mode, the digital base-band samples bypass the DVB-T demodulator/decoder in the RTL2832U chip, and are sent over USB. Normally, these devices send partially-decoded MPEG transport frames over the USB, but in this “SDR” mode, they send raw I/Q base-band samples instead.

These “dongles” are typically shipped with one of two or three different tuner chips, and the most popular ones, E4000 tuner allows tuning to the hydrogen-line frequency of1420Mhz.

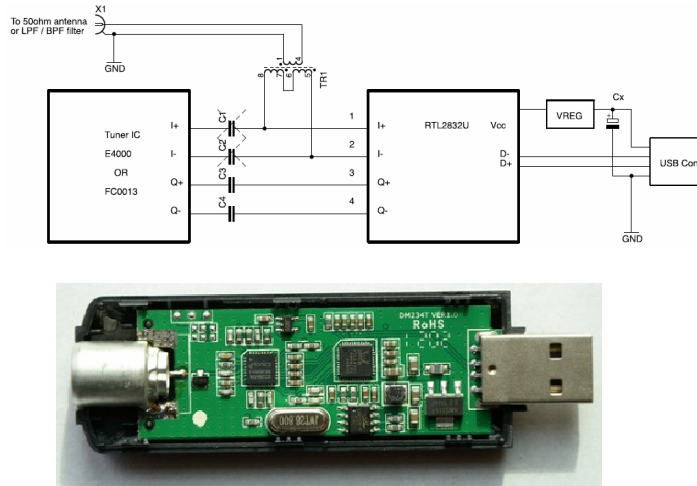
A driver library was quickly produced<sup>3</sup> based on the information gleaned from datasheets and a bit of reverse engineering. The driver library can be used stand-alone, or in concert with a “plug in” for GnuRadio that allows Gnu Radio applications to use the RTLSDR devices. The disadvantages by using Rtl\_SDR outweigh his advantages because of the following reasons:

- Dynamic range is quite limited, due to use of an 8-bit ADC, roughly 45dB SFDR.
- Temperature stability is poor, leading to gain drift, and frequency drift.
- Phase and amplitude balance is quite poor, leading to image problems.

The total cost was 30 \$.

## 5 Hardware Realization

### 5.1 Hardware of RTL SDR (RTL2832+E4000):



#### 5.1.1 RTL2832

The RTL2832U is a high-performance DVB-T COFDM demodulator that supports a USB 2.0 interface. The RTL2832U complies with NorDig Unified 1.0.3, D-Book 5.0, and EN300 744 (ETSI Specification). It supports 2K or 8K mode with 6, 7, and 8MHz bandwidth. Modulation parameters, e.g., code rate, and guard interval, are automatically detected.

The RTL2832U supports tuners at IF (Intermediate Frequency, 36.125MHz), low-IF (4.57MHz), or Zero-IF output using a 28.8MHz crystal, and includes FM/DAB/DAB+ Radio Support. Embedded with an advanced ADC (Analog-to-Digital Converter), the RTL2832U features high stability in portable reception.

The state-of-the-art RTL2832U features Realtek proprietary algorithms (patent-pending), including superior channel estimation, co-channel interference rejection, long echo channel reception, and impulse noise cancellation, and provides an ideal solution for a wide range of applications for PC-TV, such as USB dongle and MiniCard/USB, and embedded system via USB interface.

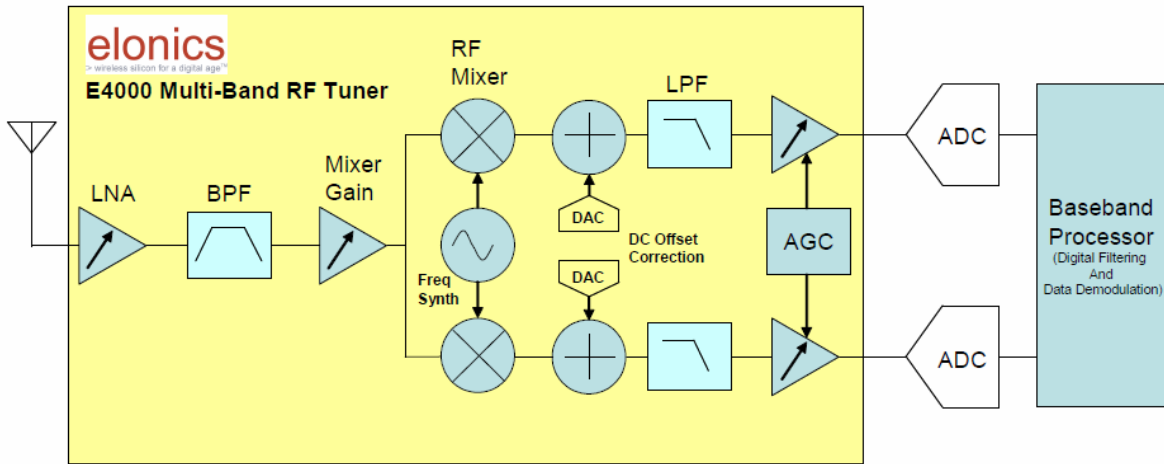
#### 5.1.2 Tuner E4000

The Elonics DigitalTune™ architecture provide solution around the ability to optimize each part of the tuner signal chain from input to output. The tuner must be able to cover the complete frequency spectrum required by the product, and output the

desired channel of interest all under control of the system controller. However, behind this apparently simple concept lies a hugely complex and challenging design problem. In order to

provide such a solution, each stage in the RF tuner signal chain must be capable of being modified to optimize the signal path characteristics for downstream processing, dependent on the broadcast standard and the desired signal characteristics chosen by the system designer.

Figure 4 shows the internal block diagram of the E4000 multi-standard RF tuner. Although the signal path looks conventional, the E4000 using the DigitalTune™ architecture provides unique flexibility at each stage in the process from input to output.



#### Benefits of Tuner E4000

- A single re-configurable RF tuner front
- Parametric performance comparable to single function tuners
- Low system power
- Small PCB
- Few external components
- Low system cost

#### Zero IF Architecture

A zero IF architecture is part of the DigitalTune™ concept. A homodyne or zero IF architecture as shown in Figure 7 employs a single stage to down convert the RF signal using a single sideband mixer to a baseband signal centred at DC. A subsequent low pass filter removes the higher frequency mixing products and attenuates unwanted out of band signals.

This filtered signal can then be digitised directly by a fast sampling ADC that has enough dynamic range to absorb both the wanted and unwanted signals. Final fine tune filtering can then be performed digitally.

The mixer output contains the sum and difference of the input signal frequencies to the mixer.

The advantages of a zero IF architecture are numerous if the intrinsic problems can be overcome. There is no image because the signal is mixed to baseband, and therefore no image filter is needed. Because there is no intermediate IF stage, there is also no requirement for a bandpass IF filter. Finally power consumption is reduced, not only due to the simplification of the signal chain but also because the signal amplification is done at lower frequencies

## 5.2 Hardware of USRP

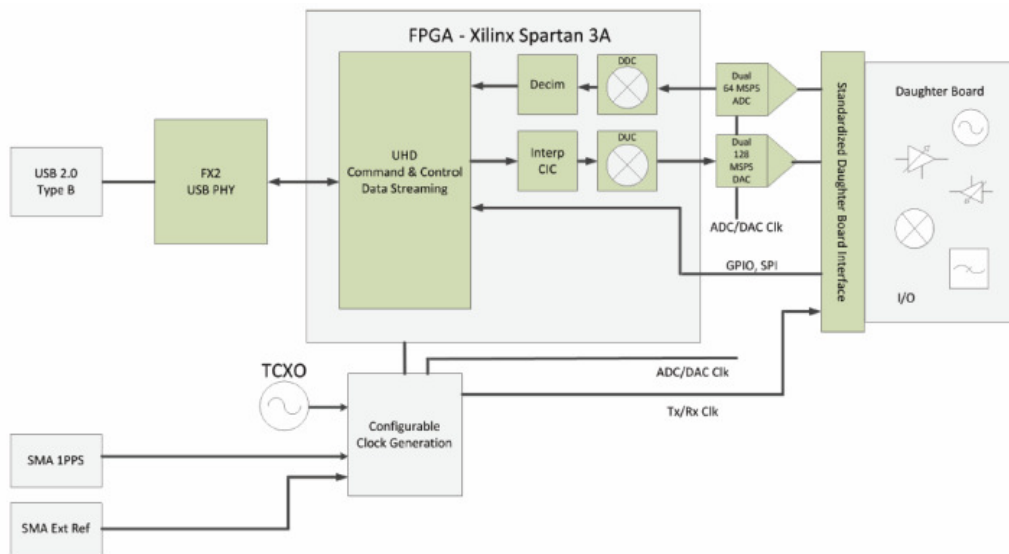
The Universal Software Radio Peripheral (USRP) is a low-cost SDR system developed by Ettus Research. The system consists of a motherboard with FPGA, 2 pairs of DACs and ADCs, digital downconverters and upconverters with programmable interpolation rates, and a the second part of the system is daughterboard which serve as RF front-end. The connection to the PC is done via USB2.0. Ettus company offers a SDR kit containing of USRP B100 as motherboard, WBX 50-2200 MHz as daughterboard which is suitable for low cost-experiment. The total price costs 590 euro. For more details see appendix A6.



Figure 7:USRP Instant SDR Kit

### 5.2.1 USRP B100 as motherboard

A USRP motherboard is the heart of software defined system provides subsystems where can be built reconfigurable structure for many application in wireless communication systems. The B100-hardware provides low-cost RF processing capability, and up to 16 MS/s of signal streaming through the USB 2.0 host interface. As shown in figure 3.2 the B100 consists of FPGA, ADC, DAC and a USB interface.



**Figure 8: Architecture of B100**

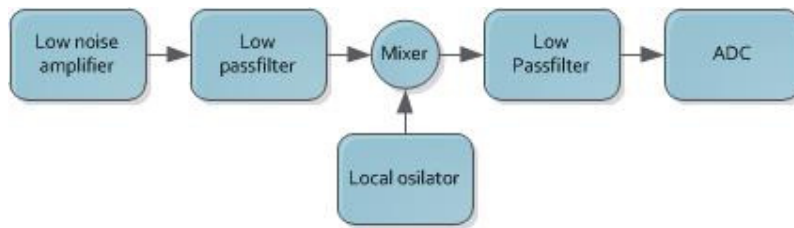
### 5.2.2 WBX 50-2200 MHz Rx/Tx as daughterboard

A new front-end circuit is used for analog operations such as up/down-conversion, filtering, and other signal conditioning. This modularity allows the USRP to serve applications that operate between DC and 6 GHz. The main function of RF Front-End is to reject undesired signal using filtering it after taking the signal from antenna and then converting the signal to a center frequency with an amplitude compatible for the analog digital conversion process. The Ettus research has developed many boards as RF so called daughterboard<sup>1</sup> which achieve our requirements and cover different frequency bands. For the project, receiver boards operating in 1.4 GHz (HL spectrum line) frequency band were needed, therefore two WBX 50-2200 MHz Rx/Tx daughterboard were chosen.

The WBX is a wide bandwidth transceiver that provides up to 100 mW of output power and a noise figure of 5 dB. The LO's for the receive and transmit chains operate independently, but can be synchronized for MIMO operation. The WBX provides 40 MHz of bandwidth capability and is ideal for applications requiring access to a number of different bands within its range - 50 MHz to 2.2 GHz [5]. This receiver architecture, shown in Figure 3.3, is implemented on the daughterboard WBX.

<sup>1</sup>Ettus Research Company offers different boards for various application areas. For selecting a suitable combination of boards Ettus Company recommends new users evaluate their application requirements against the specifications USRP devices.





**Figure 9: RF Front-End diagram**

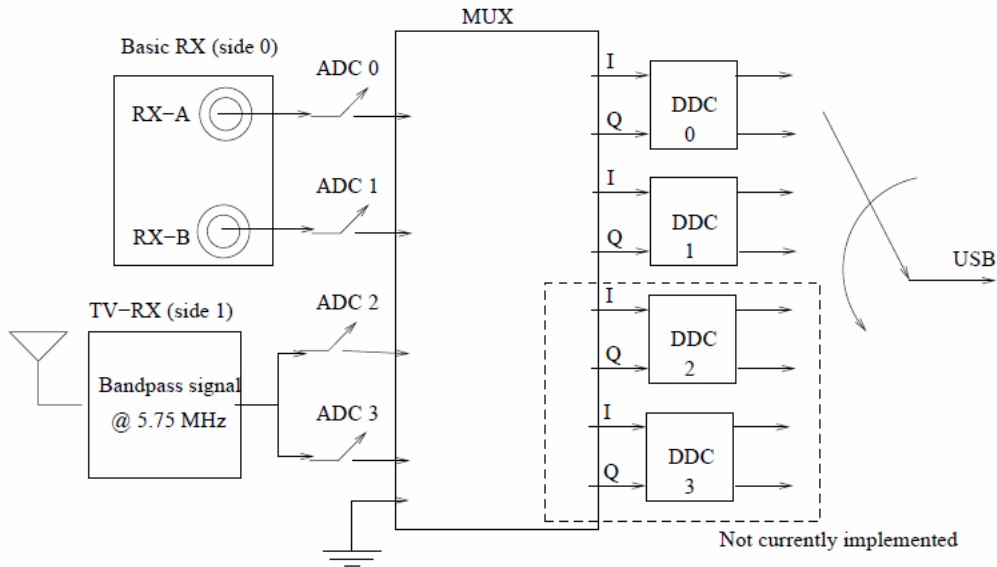
### 5.2.3 FPGA

The FPGA plays an important role in the Radio system, providing digital signal processing, as well as timing logic for clock, chip rate and time slot synthetization . The main functionality of FPGA is to perform high bandwidth math, and to reduce the data rates to something you can squirt over USB2.0. The FPGA connects to a USB2 interface chip, the Cypress FX2. Everything (FPGA circuitry and USB Microcontroller) is programmable over the USB2 bus.

USRP B100 is powered by Xilinx Spartan 3A-1400 FPGA[5] and has the following characteristics:

- Configurable Logic Blocks (CLBs) contain flexible Look-Up Tables (LUTs) that implement logic plus storage elements used as flip-flops or latches. CLBs perform a wide variety of logical functions as well as store data.
- Input/output Blocks (IOBs) control the flow of data between the I/O pins and the internal logic of the device. IOBs support bidirectional data flow plus 3-state operation. Supports a variety of signal standards, including several high-performance differential standards. Double Data-Rate (DDR) registers are included.
- Block RAM provides data storage in the form of 18-Kbit dual-port blocks.
- Multiplier Blocks accept two 18-bit binary numbers as inputs and calculate the product.
- Digital Clock Manager (DCM) Blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase-shifting clock signals.

The standard FPGA configuration includes digital down converters (DDC) implemented with cascaded integrator-comb (CIC) filters. CIC filters are very high-performance filters using only adds and delays. The FPGA implements 4 digital down converters (DDC). This allows 1, 2 or 4 separate RX channels. The receiver side includes 4 ADCs and 4 DDCs as shown in figure 3. Each DDC has two inputs I and Q. Each of the 4 ADCs can be routed to either of I or the Q input of any of the 4 DDCs.



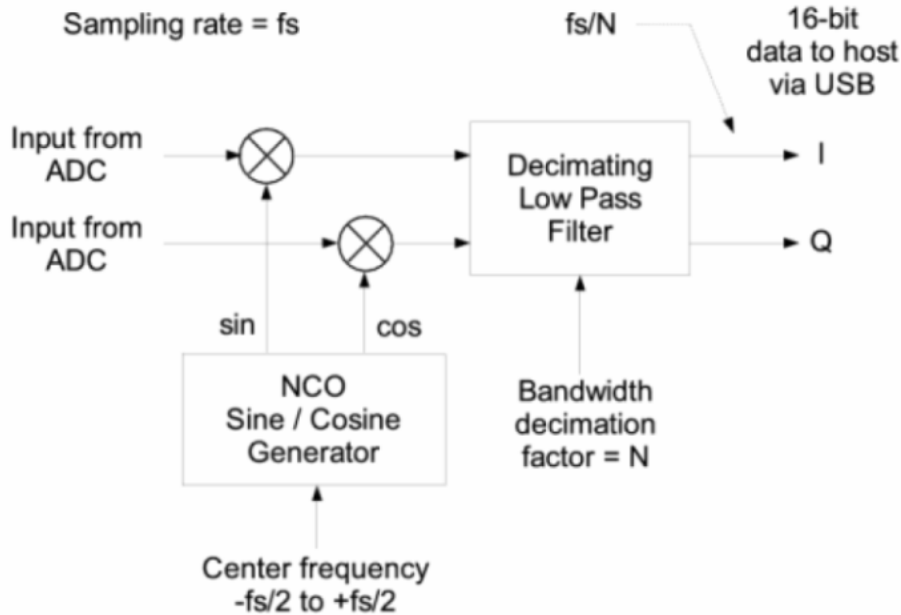
**Figure 10: Receiver Blocks**

The MUX is like a router or a circuit switcher. It determines which ADC (or constantzero) is connected to each DDC input. There are 4 DDCs. Each has two inputs. We can control the MUX using `usrp.set_mux ()` method in Python [6].

Each input (I0, Q0, I1 ... I3, Q3) can be connected with which ADC by using 4 bits (0, 1, 2, 3 or 0xf). For most real sampling applications, the Q input of each DDC is constant zero. So quite often we don't need to modify the standard configuration of the FPGA. Actually it is anticipated that the majority of USRP users will never need to use anything other than the standard FPGA configuration.

The platform contains two digital downconverters with programmable decimation rates, in charge of mixing, filtering and decimating arriving signals in the FPGA. Digital downconverters shift the frequency band of the incoming high sampling rate digitized signal to the baseband and lower the sampling rate without any information loss. First, the digitized stream is mixed with a digitized cosine (for I channel) and digitized sine (for Q channel), producing the sum and the difference components. These outputs are then put through the identical digital filters, filtering unwanted components. At this point, because the bandwidth of the signals we want to process has been reduced, sampling frequency can be loss lessly decimated. The DDC converts down the signal from the IF band to the base band. Second, it decimates the signal so that the data rate can be adapted by the USB 2.0 and is reasonable for the computers' computing capability. The Figure 3.4 shows the block diagram of the DDC. The complex input signal (IF) is multiplied by the constant frequency (usually also IF) exponential signal. The resulting signal is also complex and centered at 0. Then we decimate the signal with a factor N. Note that when there are multiple channels (up to 4), the channels are interleaved. For example, with 4 channels, the sequence sent over the USB would be I0 Q0 I1 Q1 I2 Q2 I3 Q3 I0 Q0 I1 Q1, etc. The FPGA's configuration data is stored

externally in PROM or some other non-volatile medium, either on or off the board. After applying power, the configuration data is written to the FPGA.



**Figure 11: DDC Block**

Finally I/Q complex signal enters the computer via the USB. Here starts the software fabric with signal processing.

#### 5.2.4 ADC converter

In SDR system, the acquiring data via antenna has to be digitalized immediately in the receiver chain. The B100 contains two 12 bit analog digital converters with sampling rate 64 MS/s. B100 ADC's full range is 2V peak-to-peak with the input 50 ohms impedance. ADC performs sampling and quantization of incoming signal at a certain sampling rate.

### 5.3 Low Noise Amplifier LNA

In wireless communications system, low noise amplifier (LNA) plays a significant role as the critical interface between the antenna and the electronic circuits. An LNA can be considered as front-end of the receiver channel. The noise of the total receive side is reduced by using an LNA which capture and amplify a very weak signal received by antenna in addition to associated random noise which the antenna presents to it. The main parameter of LNA is noise figure (NF) that defines the degradation of the signal-to-noise ratio (SNR) due to effect of atmosphere, thermal and other sources

Low noise amplifier has been developed in the Institute IAP with typical noise figure values 0.7 dB und gain 12 dB. The figure 7 shows a schematic of an LNA using monolithic amplifier „PGA-103+“ which is characterized through high dynamic range over a broad frequency range and with low noise figure [1]. In addition, the PGA-103+ has good input and output return loss over a broad

frequency range without the need for external matching components and has demonstrated excellent reliability.

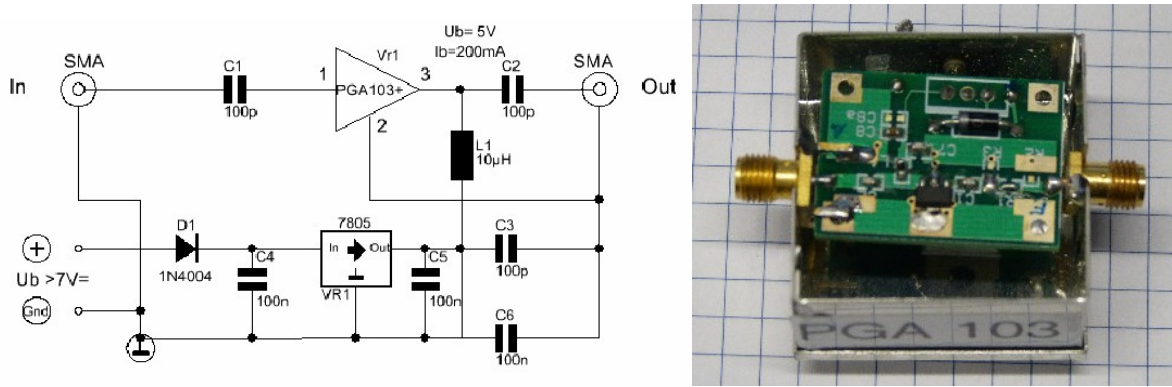


Figure 12: Schematic and platine of LNA

Furthermore, an LNA has been tested using 8620C Sweep Oscillator and the results are to see on Spectrum Analyzer (Anritsu MS 710E) as depicted in a figure 8

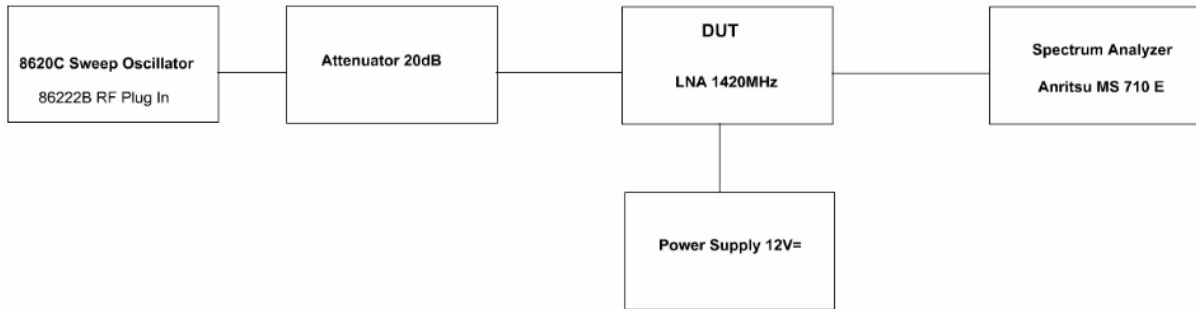


Figure 13: Test of LNA

The output signal of our test bed is to show in figure 9

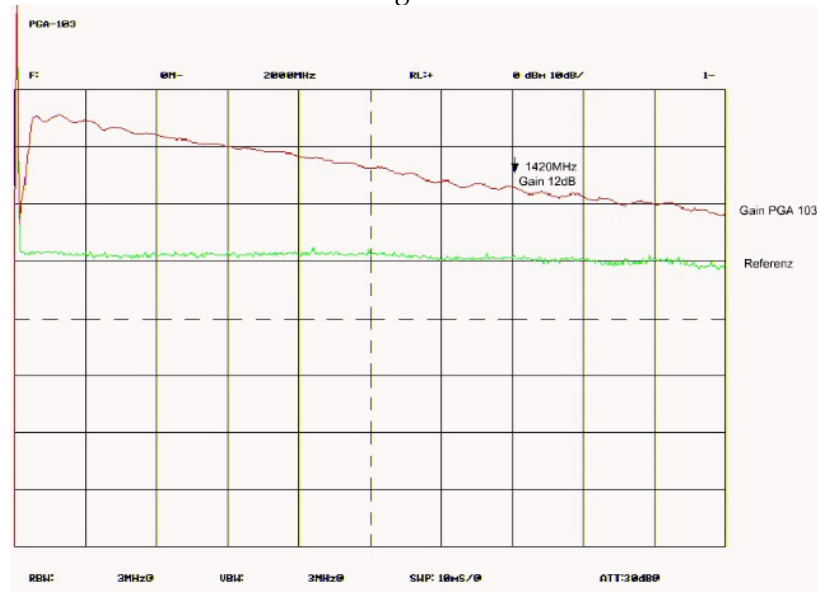


Figure 14: Output signal of LNA

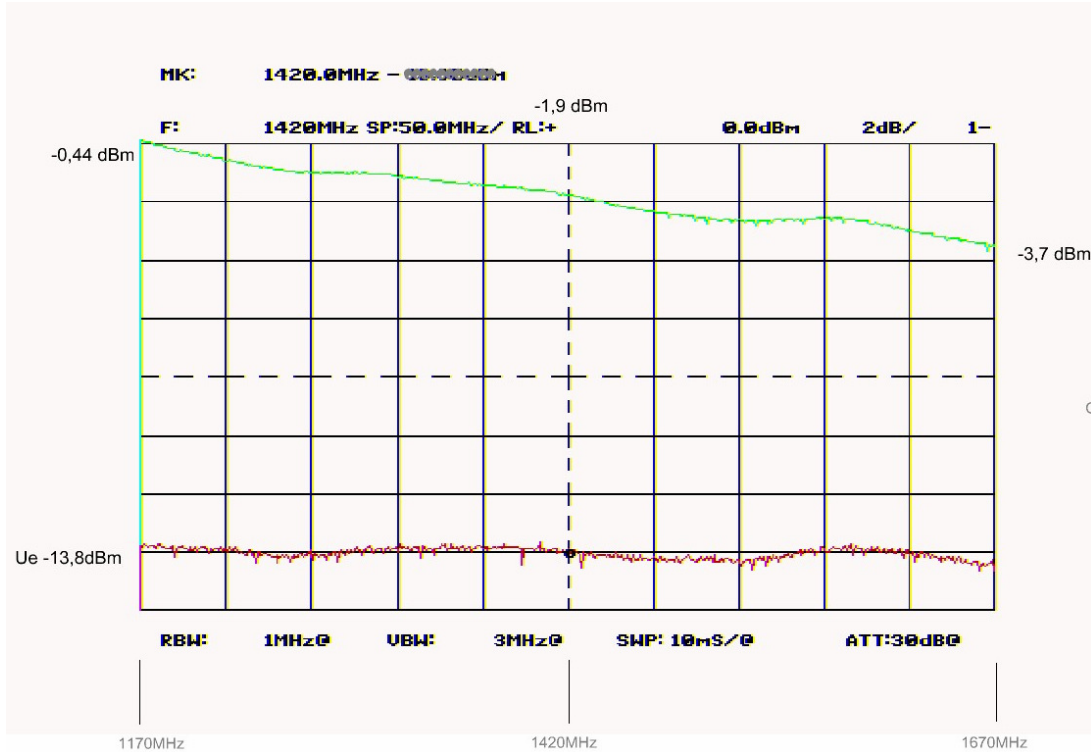


Figure 15 Output @ 2db

## 5.4 Hydrogen line filter

The frequency band to the observation of the hydrogen line which is worldwide exclusively at the radio astronomy's disposal suffices from 1400 to 1435 MHz. On the upper and lower side of this narrow frequency range strong transmitters work, such as L-band radar, mobile telephones, etc.. The antennas and preamplifiers don't have the selectivity to attenuate these strong signals. This was the reason for the development of this filter. Only the frequency range around 1420 MHz passes the filter. All other frequencies become effectively attenuated. This filter is a must to prevent interferences in an H-Line receiver.

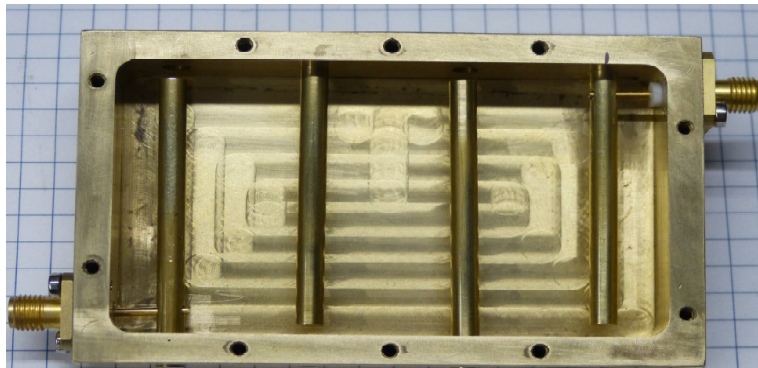


Figure 16

The installation of the filter should if possible at the receiver systems input. The insertion loss worsens however the complete noise figure of the radio astronomical receiver. The filter behind the preamplifier therefore is as recommended in the figure 8 shown commitments. Only if strong, disturbing transmitters lead to doubtful results, the filter should be used in front of the preamplifier. Please take care furthermore that no mechanical loads on the filter and his SMA connectors work. The filter must be installed in normal atmospheric humidity at a dry place, because the filter is not water protected. The preferred mounting place is in a temperature-stabilized case direct with the preamplifier at the feed of the antenna.

The following figure shows the frequency response of hydrogen filter

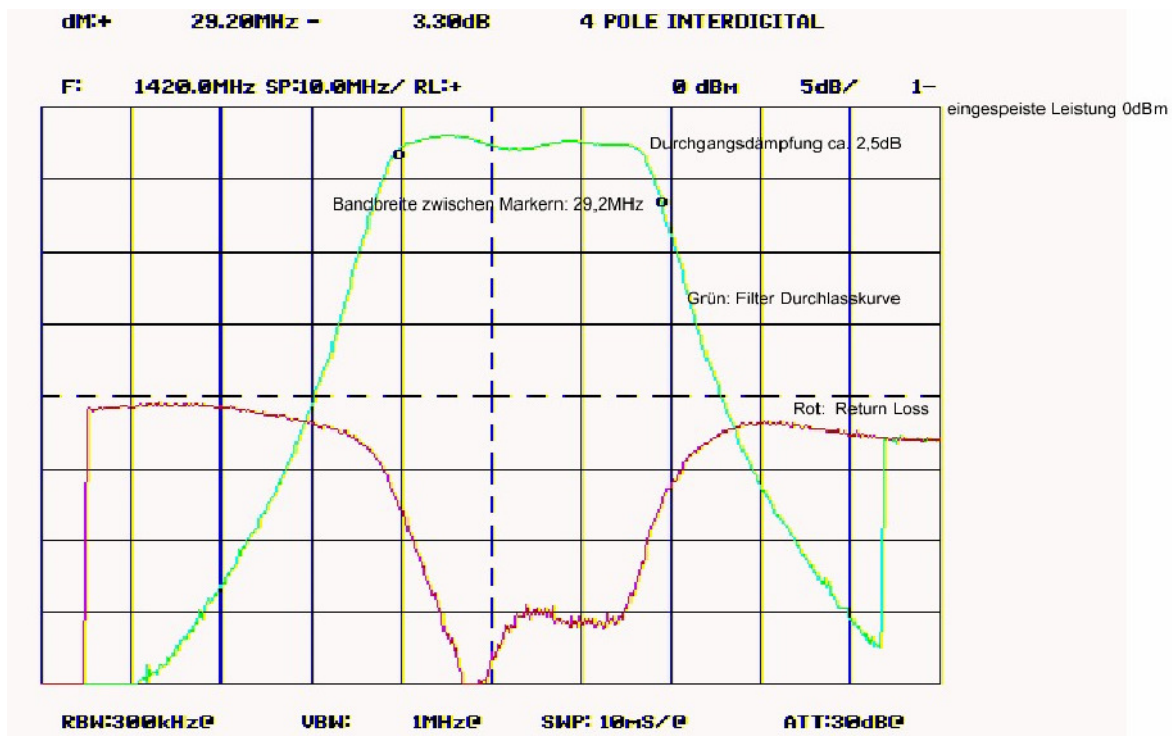


Figure 17 frequency response

## 5.5 Preamplifier:

The IF amplifier serves as adjustment the output of analog hardware with digital hardware so that adc converter can be able to work with high resolution. The next figures display the frequency response and hardware parts

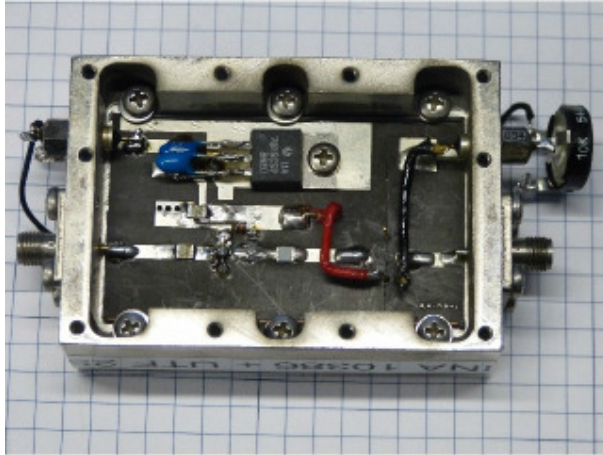


Figure 18 IF amplifier

The bandwidth of IF is 10- 1800 MHz.

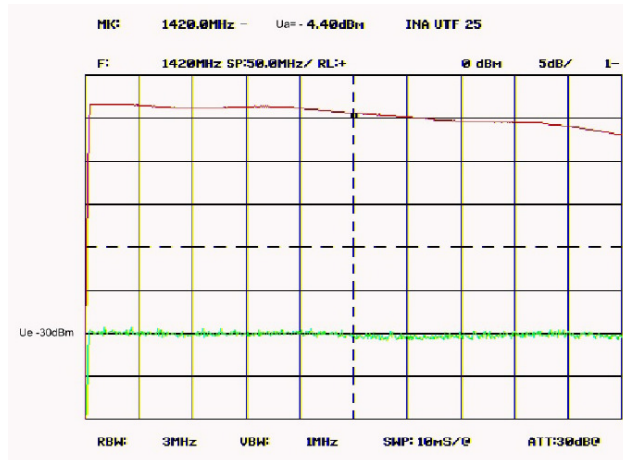


Figure 19 frequency response of IF amp



The feed horn is a corrugated aperture-controlled horn in which the phase variation over the aperture is small. By referring the design graph in [4, page 339], the directivity was calculated to be about 6 dB.

## 5.6 Antenna and Feed horn

Two parabolic mesh-surface reflector antennas were used in this project. The first antenna was thought for fixed receiver system without tracing the sun position having the diameter of 1.1 meters. Following is the parabola arc of the reflector. The F/D ratio was calculated to be about 0.5.





## 6 Software

### 6.1 Gnu radio

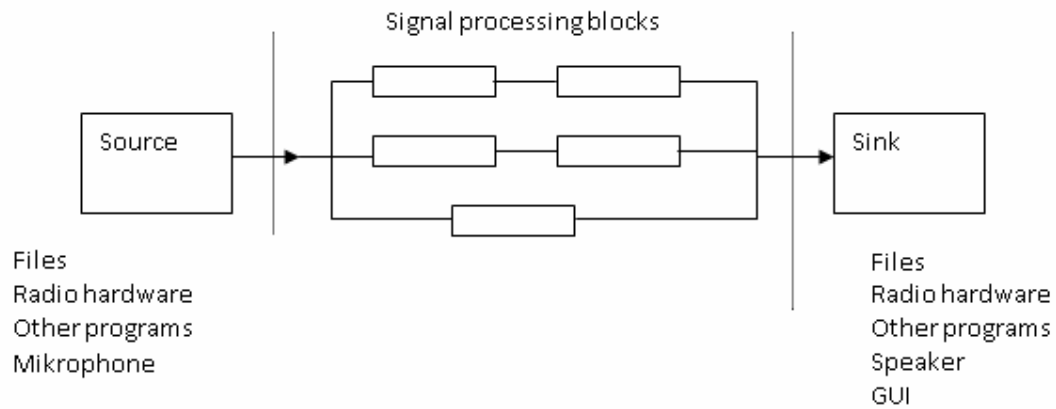
Radio is a free & open-source software development toolkit developed by Eric Blossom that is designed to couple PC host with hardware equipment (SDR platform) such as USRP, allowing for building SDR systems. GNU radio operates as stand-alone software package and provides signal processing blocks to implement software radios [3]. It can be used with readily-available low-cost external RF hardware to create software-defined radios, or without hardware in a simulation-like environment. Optimal operating system for building GNU Radio is Linux, but it can also be built on MS Windows using one of Linux-like environments such as Cygwin or MinGW/MSYS.

Most of GNU Radio's applications are written in Python as communicator function between functional blocks, whereas every block is programmed in C++ for implementing signal processing blocks. Python commands are used to control all of platform for software software-defined parameters, such as transmit power, gain, frequency, antenna selection, etc., some of which can be modified while the application is being executed. That means developer can implement real time radio systems in this development environment.

GNU Radio has been structured on two main levels: signal processing blocks and flow graphs. Blocks are designed to have a certain number of input and output ports, consisting of small Signal processing components. When the blocks are appropriately connected, a flow graph is made.

GNU Radio it is very straightforward to modulate and simulate SDR systems. It has a same principle as Matlab/Simulink and can be categorized as sinks, sources, filters and other functional blocks.

- Sources blocks consist of outputs have no inputs and are used as the first element in building the flow graph.
  - Sinks consist of inputs and have no outputs and are typically the last element in building the flow graph.
  - Filters are all the in between blocks and consist of both inputs and outputs.
  - Flow-graph: the application is based on a flow-graph. Every flow-graph consists of intermediate blocks along with source and sink blocks.
  - Scheduler: It is created for each active block, which is based on steady stream of data flow between the blocks. It is responsible for transferring data through the flow-graph- it monitors each block for sufficient data at I/p and O/p buffers so as to trigger processing function for those blocks.
-



**Figure 20 functionality of gnuradio**

Various blocks such as different modulation/demodulation, filter, signal indicators, signal generator and widgets, etc. are written in C++ and already integrated within GNU Radio, while it is also possible to write and add new blocks. Graphical interfaces are such as FFT sink and oscilloscope supported in GNU Radio.

Two kinds of Flow graphs are available in GNU Radio either as hierarchical blocks or as top blocks. Top blocks are top level flow graphs that contain all other flow graphs and have no input/output (IO) ports.

Hierarchical blocks, on the other hand, contain a certain number of IO ports (used to connect to other blocks). All of the basic signal processing blocks are connected within hierarchical blocks and can that way be used as one block. Communication between blocks is achieved using data streams considering to be connected together using same data type. It means that data types between output of one block and input of the next have to be adequately set. GNU Radio supports different data types such as byte, short, integer, float and complex (8 float byte).

## 6.2 Gnu Radio Architecture

The basic architecture of Gnuradio shows in following figure 18 containing a complex flow-graph that consists of blocks modules and low-level algorithms. Every Block or algorithm is implemented in C++ and offers various signal processing functions (Channel Coding, Filters, Modulations technique etc). These functions can be converted (generated) automatically into python modules using python "wrapper"(Simplified Wrapper) and it serves as the interface compiler allowing the integration between C++ and Python language. As mentioned, the signal processing blocks are written in C++ so that script language python connets the blocks together to form the flow graph [1]. The generated blocks are used to buid a flow-graph model with the help of python. The python framework is responsible for communication of data through module buffers and creates a simple scheduler that helps to run blocks in a sequential order for signal iteration [1]

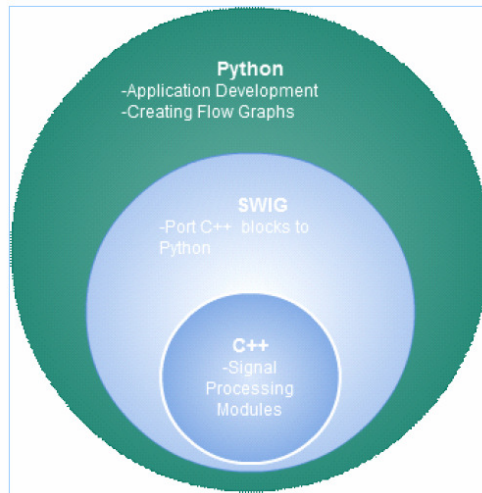


Figure 21 Software architecture

### 6.3 Gnu Radio Companion (GRC)

GNU Radio Companion is a graphical user interface for GNU Radio that allows building flow graphs by simply connecting visually-presented blocks. GRC is highly intuitive interface suitable for GNU Radio beginners that resembles Matlab Simulink's one, so anyone with some background in working with Simulink shouldn't have problems learning GRC as well.

From the Ubuntu terminal, GRC is started with command:

*gnuradio-companion .*

The dial\_tone example can be considered as “Hello world” of GNU Radio can be done using GRC as explained in Figure xx

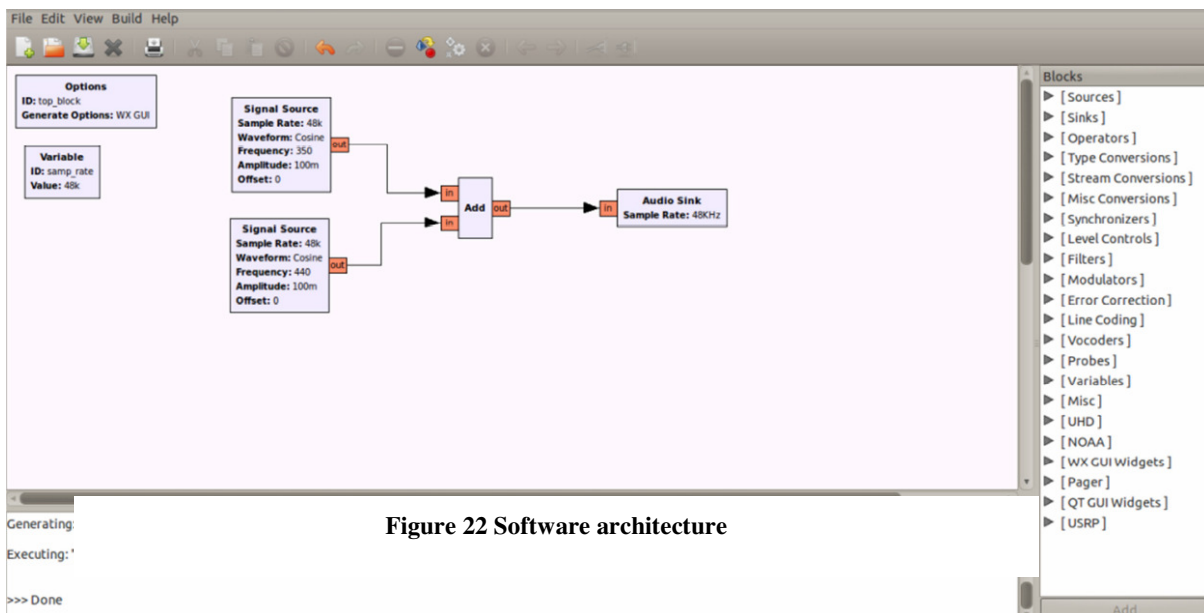




Figure 22 Software architecture

Figure 23: Implementation of dial\_tone example within Gnu Radio Companion

*Dial\_tone* example generates two sine waves of the same amplitude at frequencies 350 Hz and 440 Hz, which corresponds to the sound of US dial tone, and outputs them to the sound card.

One can translate this graph into the correspondent python code by pressing on the button “generate the flow graph ”. The generated code will be saved in the Linux Ubuntu under home/bin and then can be executed through the button „execute the flow graph ”.

The created code source of the *dial\_tone* example is given and explained below.

```
#!/usr/bin/env python
from gnuradio import gr
from gnuradio import audio

def build_graph():
    sampling_freq = 48000
    ampl = 0.1
    fg = gr.flow_graph()
    src0 = gr.sig_source_f(sampling_freq, gr.GR_SIN_WAVE, 350, ampl)
    src1 = gr.sig_source_f(sampling_freq, gr.GR_SIN_WAVE, 440, ampl)
    dst = audio.sink(sampling_freq)
    fg.connect((src0, 0), (dst, 0))
    fg.connect((src1, 0), (dst, 1))
    return fg

if __name__ == '__main__':
    fg = build_graph()
    fg.start()
    raw_input('Press Enter to quit: ')
    fg.stop()
```

The first line, `#!/usr/bin/env python`, points python to the location of python executable, and is a line that has to be added to every program that we want to run directly from terminal (as an executable).

Then, we define which modules to import – in this case, *gr*, which is always imported, and *audio*, which allows us to use audio sink.

After that, sampling frequency is set according to sound card’s specifications (48 kHz is the sampling frequency of majority of modern sound-cards), and the amplitude to 0.1.

*gr.sig\_source\_f* is used to create sine waves at the frequencies of 350 Hz and 440 Hz, where *\_f* extension, as previously explained, indicates that the produced data is of type float.

Then, the audio sink that writes received data to the sound card is created. It is worth mentioning that audio sink only accepts float data as an input.

*fg.connect* connects the flow graph’s blocks – the first sine wave is connected to the port 0 of the audio sink, while the second sine wave is connected to port 1.

## 6.4 Basic Blocks

### 6.4.1 RTL SDR source

The purpose of the Master thesis is converting a big part of the Hardware from receiver for registration of the HL 21 cm spectrum into software using GnuRadio platform. So many algorithms for signal processing and hardware functionality (filter, modulation etc...) can be made in GRC (*gnuradio-companion*). The RTL SDR Usb(2382u) conduces as Front-End to detect HI Signal and convert it to digital signal. GnuRadio recognizes the incoming data from RTL SDR using driver implemented by Osmsann [1] in GRC in order to recive IQ data so one inputs the desired parameters as center frequency, gain, decimation etc.. through the graphic user interface driver to tune the desired frequency. The RTL driver in GRC is given and explained as follows in Figure 18.

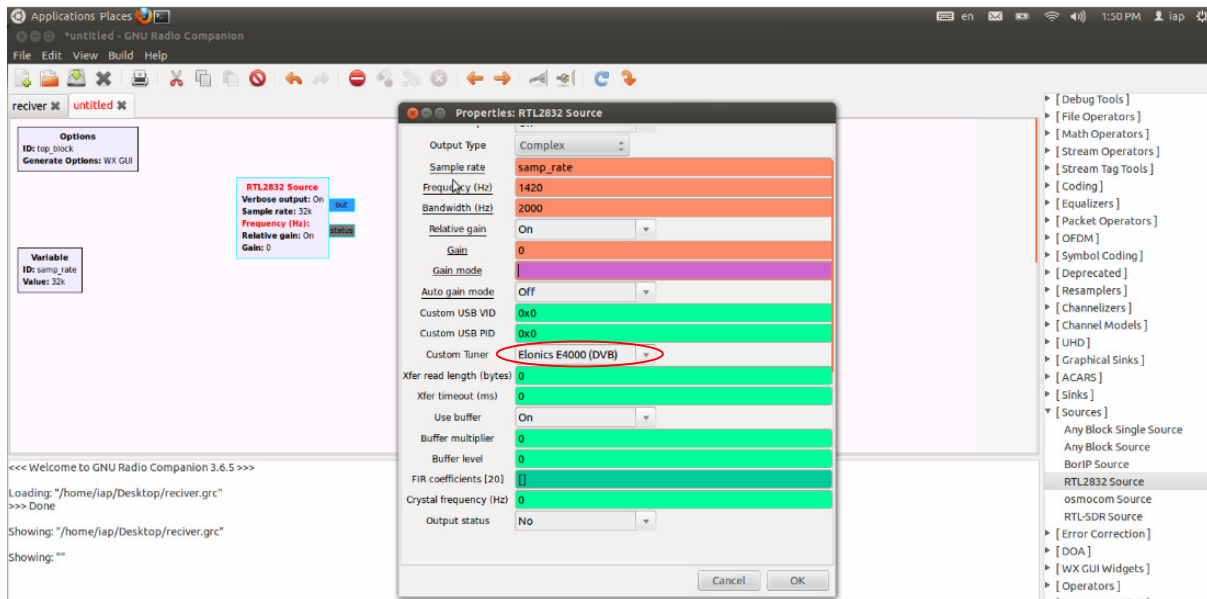


Figure 24 RTL SDR driver in GnuRadio

In the properties window of RTL2832 (our source hardware) one can select too tuner which RTL Usb contains (Elonics E4000 DVB).

### 6.4.2 WX GUI blocks

The WX GUI functions is very powerful tools to display and analyze a signal is to display in time domain and frequency domain. This block made us easy to measure and analyzes a hydrogen line 1 graphically. The python framework provides such wonderful tools in Gunradio to construct GUI tools. We used the following WX GUI blocks:

- FFT sink block

The function of FFT is used just as the signal sink ,based on fast Fourier transformation (FFT). It is defined in the module in python interface “wxgui.fftsink.py”. The function “make\_fft\_sink\_c()”helps as the interface to create an instance of the FFT sink.

#### **- Scope sink block**

WX GUI scope is very useful tools in Gnu Radio to display the waveforms in the time domain. It seems as software oscillograph.

### **6.5 Software design**

The HI 21 cm signals with other signals were obtained directly from RTL SDR Usb. The incoming digital data was filtered with FFT band-pass filter which filter the signal in frequency domain to get better removing frequently noise, whereas the parameter of this filter can be connected with GUI and then parameters of filter (center frequency, bandwidth, gain etc...) can be entered through it. Hereafter the filtered signal has been converted back to time domain then the power spectrum calculated and integrated to get average power. The averaging process ensures removing the random noise from the signal as much as possible.

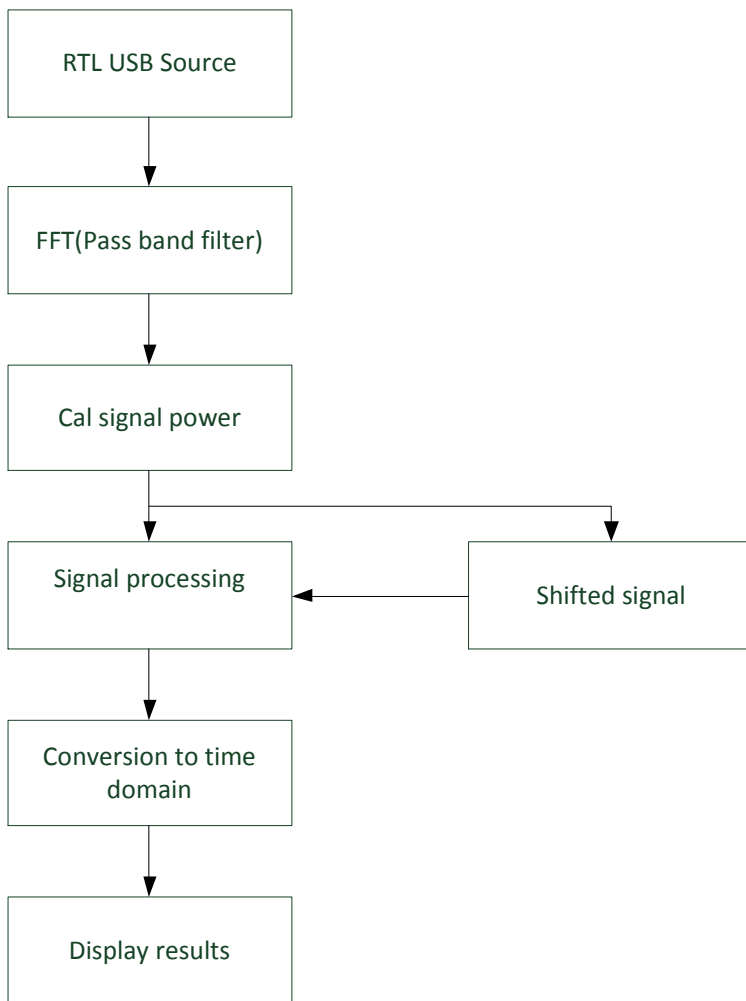
To make the signal detectable within desired range and probable noise a signal processing technique is performed. The main idea of this technique is comparing the spectrum at Hydrogen line with calculated spectrum created from shift the Hydrogen line signal a few Mhz. The exact spectrum is calculated and stored for a certain period of time. The help signal comes into existence using changing the local oscillator frequency of the mixer (IF stage) to a few MHz away from Hydrogen line and then spectrum is calculated at that frequency. The Spectrum at the Hydrogen line is named Signal and spectrum at some offset frequency from Hydrogen line is named Help.

### **6.6 Software Implementation**

The RF signal coming from Satellite, amplified through LNA, pass-filtered using interdigital 4 pole filter for passing signal 21 cm with bandwidth 20 Mhz finally through PGA circuit pre-amplified, is fed into RTL SDR which has E4000 chip as tuner and a RTL 2382 u chip that contains an IF stage for converting RF into IF and then baseband with a maximum bandwidth of 4 MHz This baseband signal is available through usb2.0 interface to GnuRadio running on a PC for further processing. The software for filters and FFT analyze are implemented in GnuRadio to calculate the power spectrum. In order to have two slightly different frequency signals (Signal and Reference), frequency switching is used. Instead, the whole band consisting of the signal as well as the

reference was obtained for processing. In our case this was as follows Signal at 1420.4 MHz with 2.5 MHz bandwidth (1419.15 – 1421.65) MHz.

Reference at 1417.9 MHz with 2.5 MHz bandwidth (1416.65 – 1419.15) MHz. Total bandwidth comprising of Signal and Reference thus became 5 MHz. The signal and reference signals are then filtered through separate filters, their FFT is calculated, averaged and finally the above mathematical equation is applied to detect the required signal.



### 6.6.1 RTL Source

The HL signal was acquired using determining the signal's parameters. The software diagram begins always with signal source (hardware, software). The important parameters are listed as follows.

Output type: IQ (In-phase and Quadrature) is used in signal kind of the typical kind signal in wireless communication

Frequency: Frequency: This is the frequency which will be down-converted to 0 Hz (baseband). It was set to the center of Reference frequency band i.e. 1417.9 MHz.

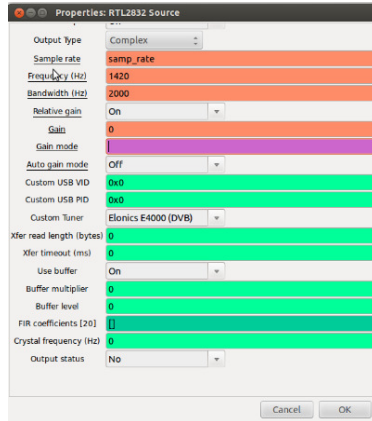
---

Sample rate:

Bandwidth: 4 MHz

Gain: 10 dB

Customer Tuner : Elonics E 4000 (DVB) is built on RTL SDR



### 6.6.2 FFT filter:

As explained the goal of using FFT filter is to remove noise in frequency domain. The suitable pass band filter was implemented in FFT block.

## 6.7 Test and Measurements

The setup of total system was realized on the high mountains for testing the whole system for HL1 detection. This testing in real environment would be validating that all the components are placed in the right order. (antenna, LNA, power supply etc).

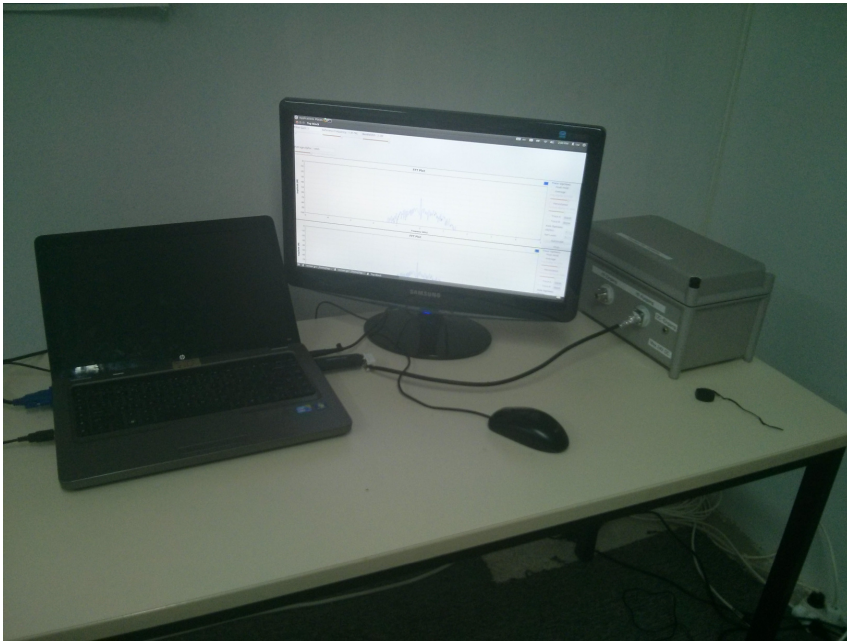
Many experiments were carried out for the first test in different directions and then power values of the signal were measured. As next step the antenna was orient in the direction of the sun and power spectrum has been observed and that noted whereby the power values was shifted compared to spectrum taken form many astronomical observatory in Heppenheim Germany [1] and in offline environment (simulation) therefore power spectrum has been recorded regarding to changing of preamplifier gain (hardware) and filters parameters (software). The results were relative acceptable as expected. Then the antenna has scanned the sky for many angle values inclusive in the direction of hydrogen line source. Figures 3-10, 3-12, 3-13 show the setup and the difference in the spectrum of (sig-ref/ref) without and with the antenna pointed towards the Hydrogen line source respectively. It can be seen from the figures 3-12 and 3-13 that the receiver worked properly as the signal is differentiable from noise.

Comparing the spectrum taken with the old configuration of SALSA ([11], figure 3-11) pointed toward the same direction of the galactic plane (Galactic longitude  $l=120$  degrees, Galactic latitude  $b = 0$ ) and the spectrum (figure 3-13), obtained with the upgraded digital software receiver pointing in the same direction, clearly shows the functionality of the upgraded digital receiver. The difference is spectrum can be explained by the approximate pointing during our experiment, but it



certainly looks like a detection! It would also have been useful to observe in a slightly larger or shifted (towards right) band [11].

The observed spectrum range seems far wider than the actual spectrum band needed to observe (within red circle in figure 3-13). This is due to the FFT Scope block of the GnuRadio, where no zooming on a particular section of the spectrum is possible; hence we have to see the whole band which is being translated from analog to digital domain through sampling. To observe only a small part of spectrum, one could decimate the (SIGNAL-REFERENCE)/REFERENCE signal to a point where the bandwidth is half the decimated sampling rate (Nyquist Theorem). Actually, a small decimation is performed here if we compare figures 3-13 and 3-9 to see the difference.



## 7 Future Work

Parts of the system are planned to be put on a satellite in a further step in the IAP SRWDA-SAT project [9]. The Gnu Radio receiver model will be converted to python code then stored on memory of board computer



**Figure 26 Import Receiver on Mockup model**



**Figure 25 Mock up model**

## 8 Literature

- [1] K. Rohlfs, Tools of radio astronomy, 2004.
- [2] E. Grayver, Implementing software defined radio, 2013.
- [3] I. Joseph Mitola, Software Radio Architecture: Object Oriented Approaches to Wireless Systems Engineering., John Wiley and Sons, 2000.
- [4] W. Tuttlebee, Software Defined Radio, England, 2002.
- [5] E. Comany, „<https://www.ettus.com/product/details/WBX>“.
- [6] Xilinx, „[http://www.xilinx.com/support/documentation/data\\_sheets/ds529.pdf](http://www.xilinx.com/support/documentation/data_sheets/ds529.pdf)“.
- [7] D. Shen, „The USRP Board,“ p. 7, Augst 2005.
- [9] IAP SRWDA-SAT project, see [www.aecenar.com/research](http://www.aecenar.com/research)

[www.astronomynotes.com](http://www.astronomynotes.com)

[www.aao.gov.au](http://www.aao.gov.au)

---

## 9 Appendix

### A.1 Software

#### Python code

```
#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: Top Block
# Generated: Tue May 6 15:46:55 2014
#####

from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import fft
from gnuradio import filter
from gnuradio import gr
from gnuradio import window
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.gr import firdes
from gnuradio.wxgui import fftsink2
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import ConfigParser
import baz
import wx

class top_block(grc_wxgui.top_block_gui):

    def __init__(self):
        grc_wxgui.top_block_gui.__init__(self, title="Top Block")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-
grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        #####
        # Variables
        #####
        self._variable_config_0_config = ConfigParser.ConfigParser()
        self._variable_config_0_config.read("default")
        try: variable_config_0 =
self._variable_config_0_config.getfloat("main", "key")
        except: variable_config_0 = 0
        self.variable_config_0 = variable_config_0
        self.samp_rate = samp_rate = 10000000
        self.low_cutoff = low_cutoff = 0.125
        self.high_cutoff = high_cutoff = 0.375
        self.gui_onoff = gui_onoff = False
        self.gain = gain = 1
        self.fft_size = fft_size = 1024
        self.dec = dec = 10
        self.avg_alpha = avg_alpha = 0.1
        self.Sig_minus_Ref = Sig_minus_Ref = 2500000
        self.Sig_freq = Sig_freq = 1420400000
```

```

self.Ref_freq = Ref_freq = 1417900000
self.N = N = 1024
self.BW = BW = 2500000

#####
# Blocks
#####
_gain_sizer = wx.BoxSizer(wx.VERTICAL)
self._gain_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_gain_sizer,
    value=self.gain,
    callback=self.set_gain,
    label="Filter Gain",
    converter=forms.float_converter(),
    proportion=0,
)
self._gain_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_gain_sizer,
    value=self.gain,
    callback=self.set_gain,
    minimum=1,
    maximum=32,
    num_steps=100,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.GridAdd(_gain_sizer, 0, 0, 1, 1)
_avg_alpha_sizer = wx.BoxSizer(wx.VERTICAL)
self._avg_alpha_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_avg_alpha_sizer,
    value=self.avg_alpha,
    callback=self.set_avg_alpha,
    label="Average Alpha",
    converter=forms.float_converter(),
    proportion=0,
)
self._avg_alpha_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_avg_alpha_sizer,
    value=self.avg_alpha,
    callback=self.set_avg_alpha,
    minimum=0.001,
    maximum=0.25,
    num_steps=100,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.GridAdd(_avg_alpha_sizer, 3, 0, 1, 3)
_BW_sizer = wx.BoxSizer(wx.VERTICAL)
self._BW_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_BW_sizer,
    value=self.BW,

```

---

```

        callback=self.set_BW,
        label="Bandwidth",
        converter=forms.float_converter(),
        proportion=0,
    )
self._BW_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_BW_sizer,
    value=self.BW,
    callback=self.set_BW,
    minimum=500000,
    maximum=2500000,
    num_steps=100,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.GridAdd(_BW_sizer, 0, 4, 1, 1)
self.wxgui_fftsink2_0_0_0 = fftsink2.fft_sink_c(
    self.GetWin(),
    baseband_freq=0,
    y_per_div=10,
    y_divs=10,
    ref_level=0,
    ref_scale=2.0,
    sample_rate=samp_rate,
    fft_size=1024,
    fft_rate=15,
    average=False,
    avg_alpha=None,
    title="FFT Plot",
    peak_hold=False,
)
self.Add(self.wxgui_fftsink2_0_0_0.win)
self.wxgui_fftsink2_0_0 = fftsink2.fft_sink_c(
    self.GetWin(),
    baseband_freq=0,
    y_per_div=10,
    y_divs=10,
    ref_level=0,
    ref_scale=2.0,
    sample_rate=samp_rate,
    fft_size=1024,
    fft_rate=15,
    average=False,
    avg_alpha=None,
    title="FFT Plot",
    peak_hold=False,
)
self.Add(self.wxgui_fftsink2_0_0.win)
self.wxgui_fftsink2_0 = fftsink2.fft_sink_c(
    self.GetWin(),
    baseband_freq=0,
    y_per_div=10,
    y_divs=10,
    ref_level=0,
    ref_scale=2.0,
    sample_rate=samp_rate,

```

```

        fft_size=1024,
        fft_rate=15,
        average=False,
        avg_alpha=None,
        title="FFT Plot",
        peak_hold=False,
    )
    self.Add(self.wxgui_fftsink2_0.win)
    self.single_pole_iir_filter_xx_0_0 =
filter.single_pole_iir_filter_ff(0.1, 1024)
    self.single_pole_iir_filter_xx_0 =
filter.single_pole_iir_filter_ff(0.1, 1024)
    self.rtl2832_source_0 = baz.rtl_source_c(defer_creation=True,
output_size=gr.sizeof_gr_complex)
    self.rtl2832_source_0.set_verbose(True)
    self.rtl2832_source_0.set_vid(0x0)
    self.rtl2832_source_0.set_pid(0x0)
    self.rtl2832_source_0.set_tuner_name("e4k")
    self.rtl2832_source_0.set_default_timeout(0)
    self.rtl2832_source_0.set_use_buffer(True)
    self.rtl2832_source_0.set_fir_coefficients([])

    self.rtl2832_source_0.set_read_length(0)

    if self.rtl2832_source_0.create() == False: raise
Exception("Failed to create RTL2832 Source: rtl2832_source_0")

    self.rtl2832_source_0.set_sample_rate(samp_rate)

    self.rtl2832_source_0.set_frequency(1420000000)

    self.rtl2832_source_0.set_auto_gain_mode(False)
    self.rtl2832_source_0.set_relative_gain(True)
    self.rtl2832_source_0.set_gain(27)

    self._gui_onoff_check_box = forms.check_box(
        parent=self.GetWin(),
        value=self.gui_onoff,
        callback=self.set_gui_onoff,
        label="check to parameter",
        true=True,
        false=False,
    )
    self.Add(self._gui_onoff_check_box)
    self.freq_xlating_fir_filter_xxx_0 =
filter.freq_xlating_fir_filter_ccc(1,
(gr.firdes.complex_band_pass(gain, samp_rate, low_cutoff, high_cutoff, BW, 0
, beta=6.76)), -2500000, samp_rate)
    self.fft_vxx_0_0_0_0 = fft.fft_vcc(1024, True,
(window.hamming(N)), False, 1)
    self.fft_vxx_0_0_0 = fft.fft_vcc(1024, True,
(window.blackmanharris(1024)), True, 1)

```

---

```

        self.fft_vxx_0_0 = fft.fft_vfc(1024, True,
(window.blackmanharris(1024)), 1)
        self.fft_filter_xxx_0_0_0 = filter.fft_filter_ccc(1,
(gr.firdes.complex_band_pass(gain, samp_rate, low_cutoff, high_cutoff, BW, 0
, beta=6.76)), 1)
        self.fft_filter_xxx_0_0 = filter.fft_filter_ccc(1,
(gr.firdes.complex_band_pass(gain, samp_rate, low_cutoff, high_cutoff, BW, 0
, beta=6.76)), 1)
        self.fft_filter_xxx_0 = filter.fft_filter_ccc(1,
(gr.firdes.complex_band_pass(gain, samp_rate, low_cutoff, high_cutoff, BW, 0
, beta=6.76)), 1)
        self.blocks_vector_to_stream_0 =
blocks.vector_to_stream(gr.sizeof_gr_complex*1, 1024)
        self.blocks_stream_to_vector_0_0 =
blocks.stream_to_vector(gr.sizeof_gr_complex*1, 1024)
        self.blocks_stream_to_vector_0 =
blocks.stream_to_vector(gr.sizeof_gr_complex*1, 1024)
        self.blocks_keep_one_in_n_0 =
blocks.keep_one_in_n(gr.sizeof_gr_complex*1, 5)
        self.blocks_divide_xx_0 = blocks.divide_ff(1024)
        self.blocks_conjugate_cc_0 = blocks.conjugate_cc()
        self.blocks_complex_to_mag_squared_1_0 =
blocks.complex_to_mag_squared(1024)
        self.blocks_complex_to_mag_squared_1 =
blocks.complex_to_mag_squared(1024)
        self.blocks_add_const_vxx_0 = blocks.add_const_vff((-1)*N)
        _Ref_freq_sizer = wx.BoxSizer(wx.VERTICAL)
        self._Ref_freq_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_Ref_freq_sizer,
            value=self.Ref_freq,
            callback=self.set_Ref_freq,
            label="Reference frequency",
            converter=forms.float_converter(),
            proportion=0,
        )
        self._Ref_freq_slider = forms.slider(
            parent=self.GetWin(),
            sizer=_Ref_freq_sizer,
            value=self.Ref_freq,
            callback=self.set_Ref_freq,
            minimum=1417400000,
            maximum=1418400000,
            num_steps=100,
            style=wx.SL_HORIZONTAL,
            cast=float,
            proportion=1,
        )
        self.GridAdd(_Ref_freq_sizer, 0, 1, 1, 3)

#####
# Connections
#####
        self.connect((self.fft_vxx_0_0_0_0, 0),
(self.blocks_complex_to_mag_squared_1, 0))
        self.connect((self.blocks_add_const_vxx_0, 0),
(self.fft_vxx_0_0, 0))

```



```

        self.connect((self.blocks_divide_xx_0, 0),
(self.blocks_add_const_vxx_0, 0))
        self.connect((self.single_pole_iir_filter_xx_0, 0),
(self.blocks_divide_xx_0, 1))
        self.connect((self.blocks_keep_one_in_n_0, 0),
(self.wxgui_fftsink2_0, 0))
        self.connect((self.fft_vxx_0_0, 0),
(self.blocks_vector_to_stream_0, 0))
        self.connect((self.blocks_complex_to_mag_squared_1, 0),
(self.single_pole_iir_filter_xx_0, 0))
        self.connect((self.blocks_conjugate_cc_0, 0),
(self.freq_xlating_fir_filter_xxx_0, 0))
        self.connect((self.blocks_vector_to_stream_0, 0),
(self.blocks_conjugate_cc_0, 0))
        self.connect((self.single_pole_iir_filter_xx_0_0, 0),
(self.blocks_divide_xx_0, 0))
        self.connect((self.blocks_complex_to_mag_squared_1_0, 0),
(self.single_pole_iir_filter_xx_0_0, 0))
        self.connect((self.fft_vxx_0_0_0, 0),
(self.blocks_complex_to_mag_squared_1_0, 0))
        self.connect((self.fft_filter_xxx_0, 0),
(self.blocks_keep_one_in_n_0, 0))
        self.connect((self.freq_xlating_fir_filter_xxx_0, 0),
(self.fft_filter_xxx_0, 0))
        self.connect((self.rtl2832_source_0, 0),
(self.fft_filter_xxx_0_0, 0))
        self.connect((self.fft_filter_xxx_0_0, 0),
(self.wxgui_fftsink2_0_0, 0))
        self.connect((self.rtl2832_source_0, 0),
(self.fft_filter_xxx_0_0_0, 0))
        self.connect((self.fft_filter_xxx_0_0_0, 0),
(self.wxgui_fftsink2_0_0_0, 0))
        self.connect((self.fft_filter_xxx_0_0_0, 0),
(self.blocks_stream_to_vector_0, 0))
        self.connect((self.blocks_stream_to_vector_0, 0),
(self.fft_vxx_0_0_0_0, 0))
        self.connect((self.fft_filter_xxx_0_0, 0),
(self.blocks_stream_to_vector_0_0, 0))
        self.connect((self.blocks_stream_to_vector_0_0, 0),
(self.fft_vxx_0_0_0, 0))

```

```

def get_variable_config_0(self):
    return self.variable_config_0

```

```

def set_variable_config_0(self, variable_config_0):
    self.variable_config_0 = variable_config_0

```

```

def get_samp_rate(self):
    return self.samp_rate

```

```

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate

```

```

self.freq_xlating_fir_filter_xxx_0.set_taps((gr.firdes.complex_band
_pass(self.gain,self.samp_rate,self.low_cutoff,self.high_cutoff,self.BW,
0 ,beta=6.76)))

```

---

```

self.fft_filter_xxx_0.set_taps((gr.firdes.complex_band_pass(self.gain, self.samp_rate, self.low_cutoff, self.high_cutoff, self.BW, 0, beta=6.76)))

self.fft_filter_xxx_0_0_0.set_taps((gr.firdes.complex_band_pass(self.fgain, self.samp_rate, self.low_cutoff, self.high_cutoff, self.BW, 0, beta=6.76)))
self.rtl2832_source_0.set_sample_rate(self.samp_rate)

self.fft_filter_xxx_0_0.set_taps((gr.firdes.complex_band_pass(self.gain, self.samp_rate, self.low_cutoff, self.high_cutoff, self.BW, 0, beta=6.76)))
self.wxgui_fftsink2_0.set_sample_rate(self.samp_rate)
self.wxgui_fftsink2_0_0_0.set_sample_rate(self.samp_rate)
self.wxgui_fftsink2_0_0.set_sample_rate(self.samp_rate)

def get_low_cutoff(self):
    return self.low_cutoff

def set_low_cutoff(self, low_cutoff):
    self.low_cutoff = low_cutoff

self.freq_xlating_fir_filter_xxx_0.set_taps((gr.firdes.complex_band_pass(self.gain, self.samp_rate, self.low_cutoff, self.high_cutoff, self.BW, 0, beta=6.76)))

self.fft_filter_xxx_0.set_taps((gr.firdes.complex_band_pass(self.gain, self.samp_rate, self.low_cutoff, self.high_cutoff, self.BW, 0, beta=6.76)))

self.fft_filter_xxx_0_0_0.set_taps((gr.firdes.complex_band_pass(self.fgain, self.samp_rate, self.low_cutoff, self.high_cutoff, self.BW, 0, beta=6.76)))

self.fft_filter_xxx_0_0.set_taps((gr.firdes.complex_band_pass(self.gain, self.samp_rate, self.low_cutoff, self.high_cutoff, self.BW, 0, beta=6.76)))

def get_high_cutoff(self):
    return self.high_cutoff

def set_high_cutoff(self, high_cutoff):
    self.high_cutoff = high_cutoff

self.freq_xlating_fir_filter_xxx_0.set_taps((gr.firdes.complex_band_pass(self.gain, self.samp_rate, self.low_cutoff, self.high_cutoff, self.BW, 0, beta=6.76)))

self.fft_filter_xxx_0.set_taps((gr.firdes.complex_band_pass(self.gain, self.samp_rate, self.low_cutoff, self.high_cutoff, self.BW, 0, beta=6.76)))

self.fft_filter_xxx_0_0_0.set_taps((gr.firdes.complex_band_pass(self.fgain, self.samp_rate, self.low_cutoff, self.high_cutoff, self.BW, 0, beta=6.76)))

self.fft_filter_xxx_0_0.set_taps((gr.firdes.complex_band_pass(self.gain, self.samp_rate, self.low_cutoff, self.high_cutoff, self.BW, 0, beta=6.76)))

```

```

gain,self.samp_rate,self.low_cutoff,self.high_cutoff,self.BW, 0
,beta=6.76))

def get_gui_onoff(self):
    return self.gui_onoff

def set_gui_onoff(self, gui_onoff):
    self.gui_onoff = gui_onoff
    self._gui_onoff_check_box.set_value(self.gui_onoff)

def get_gain(self):
    return self.gain

def set_gain(self, gain):
    self.gain = gain
    self._gain_slider.set_value(self.gain)
    self._gain_text_box.set_value(self.gain)

    self.freq_xlating_fir_filter_xxx_0.set_taps((gr.firdes.complex_band
_pass(self.gain,self.samp_rate,self.low_cutoff,self.high_cutoff,self.BW,
0 ,beta=6.76)))

    self.fft_filter_xxx_0.set_taps((gr.firdes.complex_band_pass(self.ga
in,self.samp_rate,self.low_cutoff,self.high_cutoff,self.BW, 0
,beta=6.76)))

    self.fft_filter_xxx_0_0_0.set_taps((gr.firdes.complex_band_pass(sel
f.gain,self.samp_rate,self.low_cutoff,self.high_cutoff,self.BW, 0
,beta=6.76)))

    self.fft_filter_xxx_0_0.set_taps((gr.firdes.complex_band_pass(self.
gain,self.samp_rate,self.low_cutoff,self.high_cutoff,self.BW, 0
,beta=6.76)))

def get_fft_size(self):
    return self.fft_size

def set_fft_size(self, fft_size):
    self.fft_size = fft_size

def get_dec(self):
    return self.dec

def set_dec(self, dec):
    self.dec = dec

def get_avg_alpha(self):
    return self.avg_alpha

def set_avg_alpha(self, avg_alpha):
    self.avg_alpha = avg_alpha
    self._avg_alpha_slider.set_value(self.avg_alpha)
    self._avg_alpha_text_box.set_value(self.avg_alpha)

def get_Sig_minus_Ref(self):
    return self.Sig_minus_Ref

def set_Sig_minus_Ref(self, Sig_minus_Ref):

```

---

```

        self.Sig_minus_Ref = Sig_minus_Ref

def get_Sig_freq(self):
    return self.Sig_freq

def set_Sig_freq(self, Sig_freq):
    self.Sig_freq = Sig_freq

def get_Ref_freq(self):
    return self.Ref_freq

def set_Ref_freq(self, Ref_freq):
    self.Ref_freq = Ref_freq
    self._Ref_freq_slider.set_value(self.Ref_freq)
    self._Ref_freq_text_box.set_value(self.Ref_freq)

def get_N(self):
    return self.N

def set_N(self, N):
    self.N = N
    self.blocks_add_const_vxx_0.set_k((-1)*self.N)

def get_BW(self):
    return self.BW

def set_BW(self, BW):
    self.BW = BW

    self.freq_xlating_fir_filter_xxx_0.set_taps((gr.firdes.complex_band_pass(self.gain,self.samp_rate,self.low_cutoff,self.high_cutoff,self.BW, 0 ,beta=6.76)))

    self.fft_filter_xxx_0.set_taps((gr.firdes.complex_band_pass(self.gain,self.samp_rate,self.low_cutoff,self.high_cutoff,self.BW, 0 ,beta=6.76)))
    self._BW_slider.set_value(self.BW)
    self._BW_text_box.set_value(self.BW)

    self.fft_filter_xxx_0_0_0.set_taps((gr.firdes.complex_band_pass(self.gain,self.samp_rate,self.low_cutoff,self.high_cutoff,self.BW, 0 ,beta=6.76)))

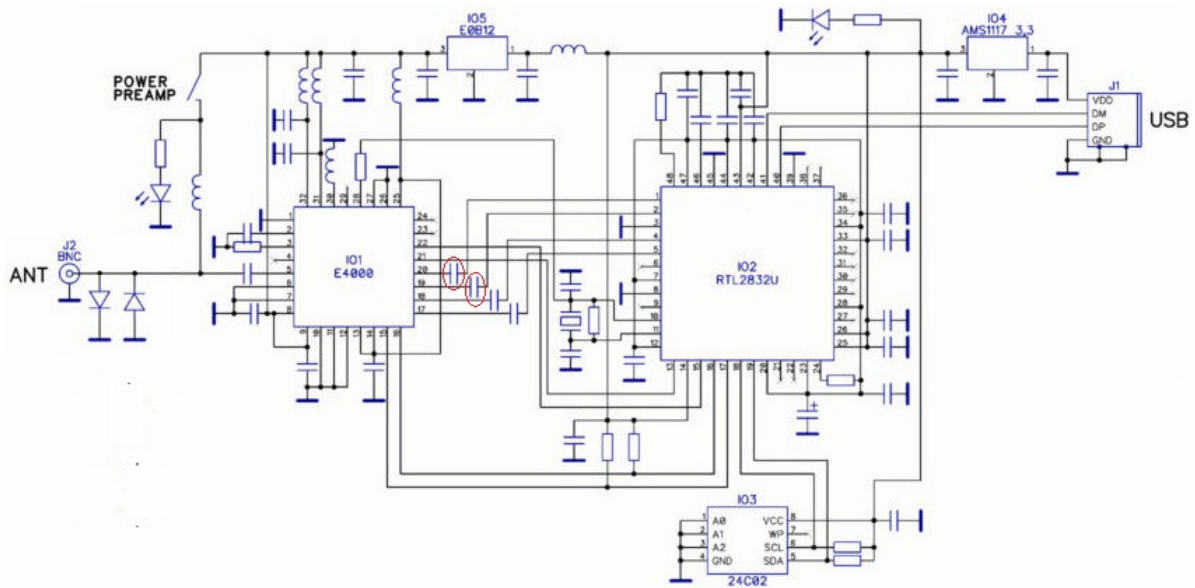
    self.fft_filter_xxx_0_0.set_taps((gr.firdes.complex_band_pass(self.gain,self.samp_rate,self.low_cutoff,self.high_cutoff,self.BW, 0 ,beta=6.76)))

if __name__ == '__main__':
    parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
    (options, args) = parser.parse_args()
    tb = top_block()
    tb.Run(True)

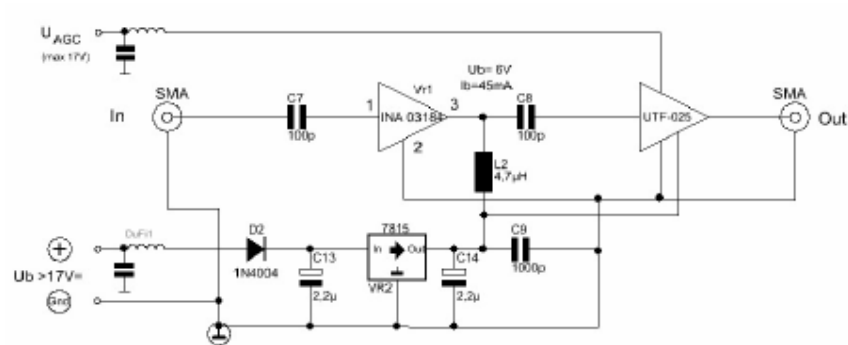
```

## A.2 Hardware

### SDR RTL 2382+4000



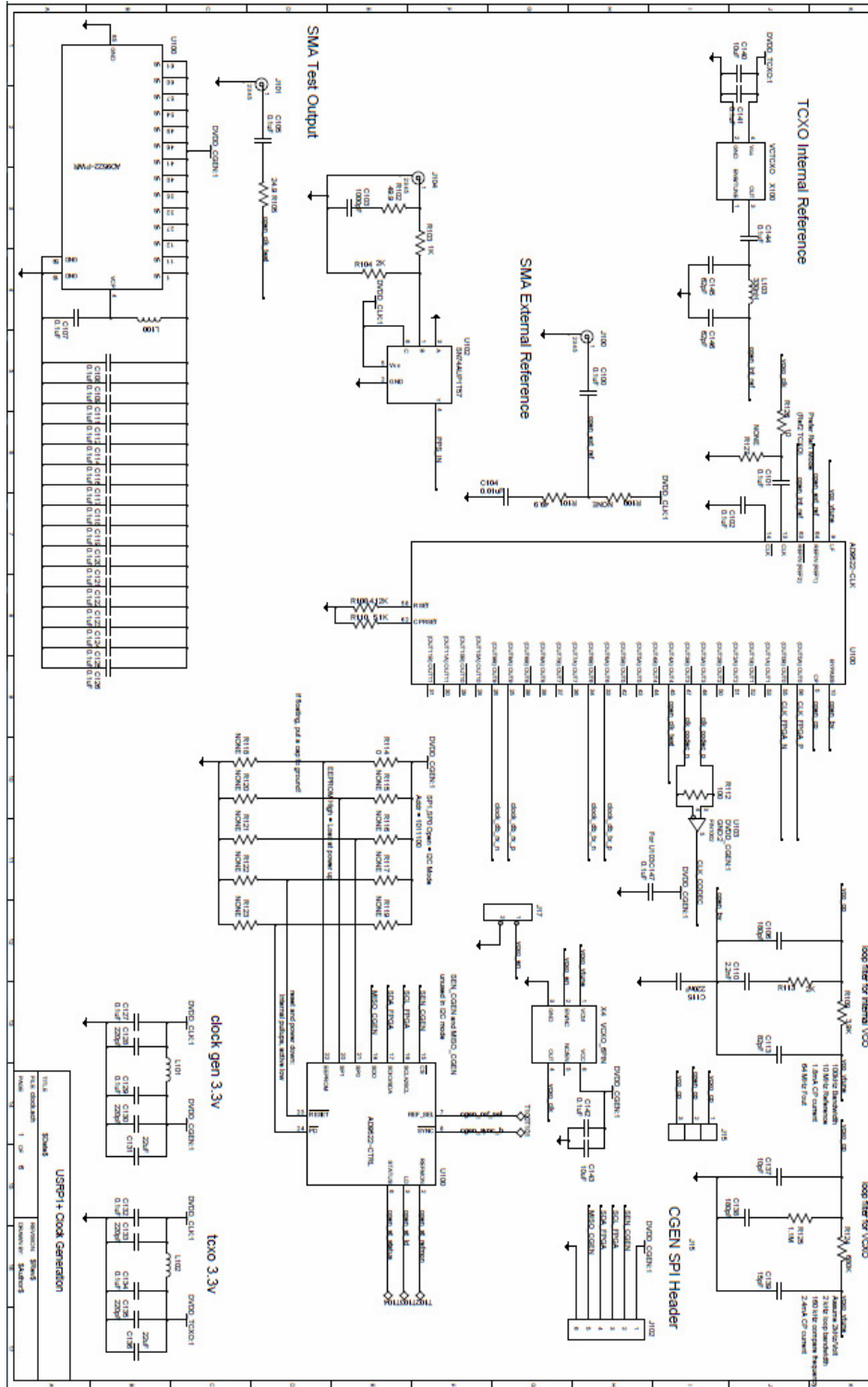
### IF Amplifier

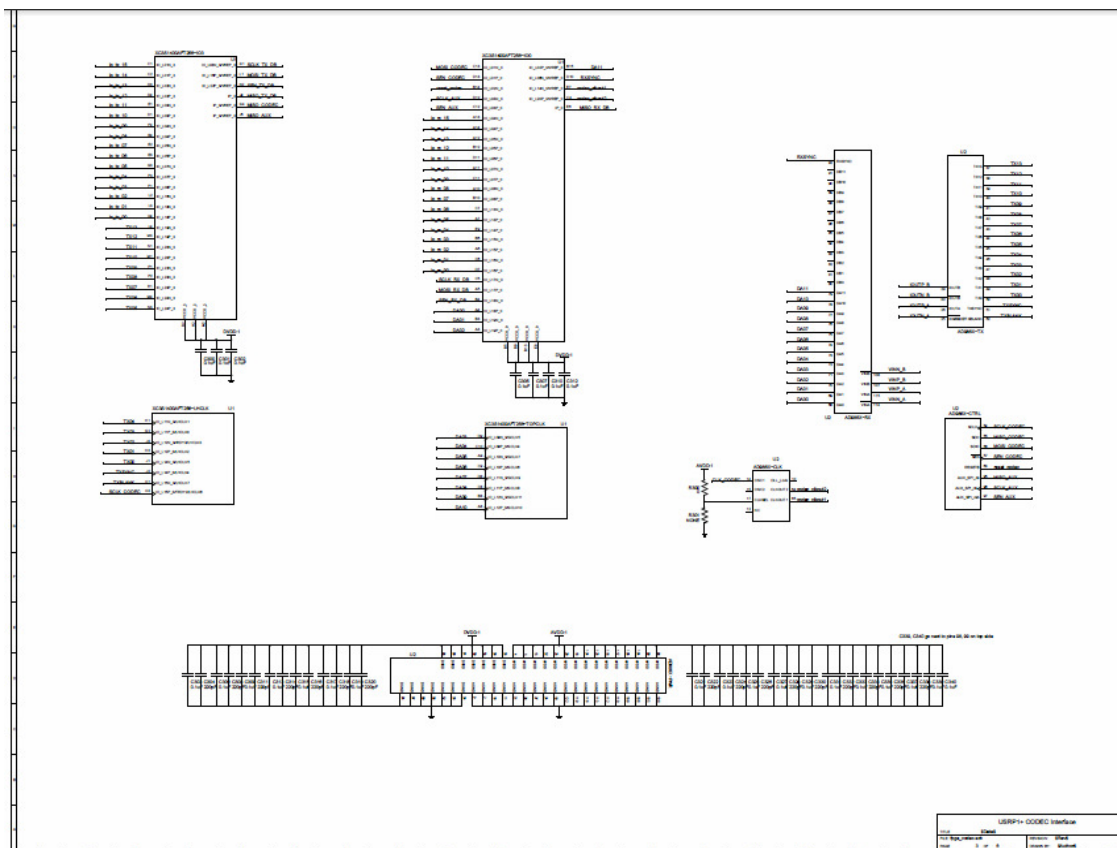
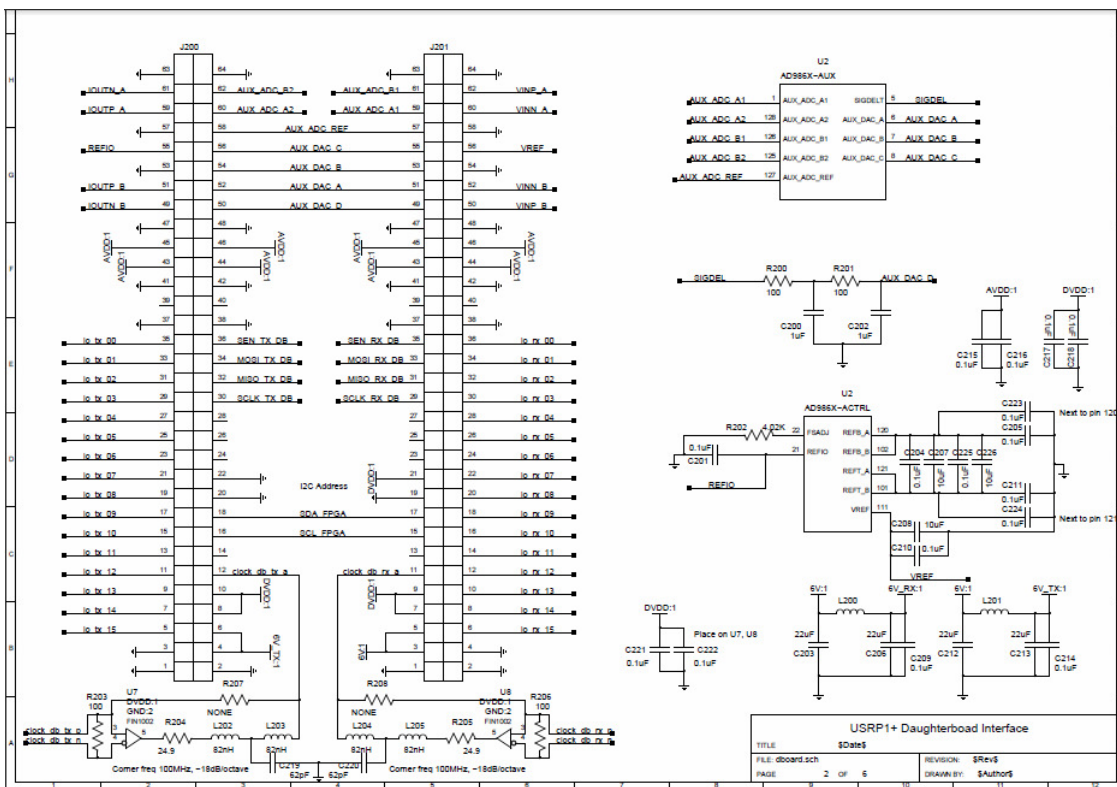


# Low noise amplifier

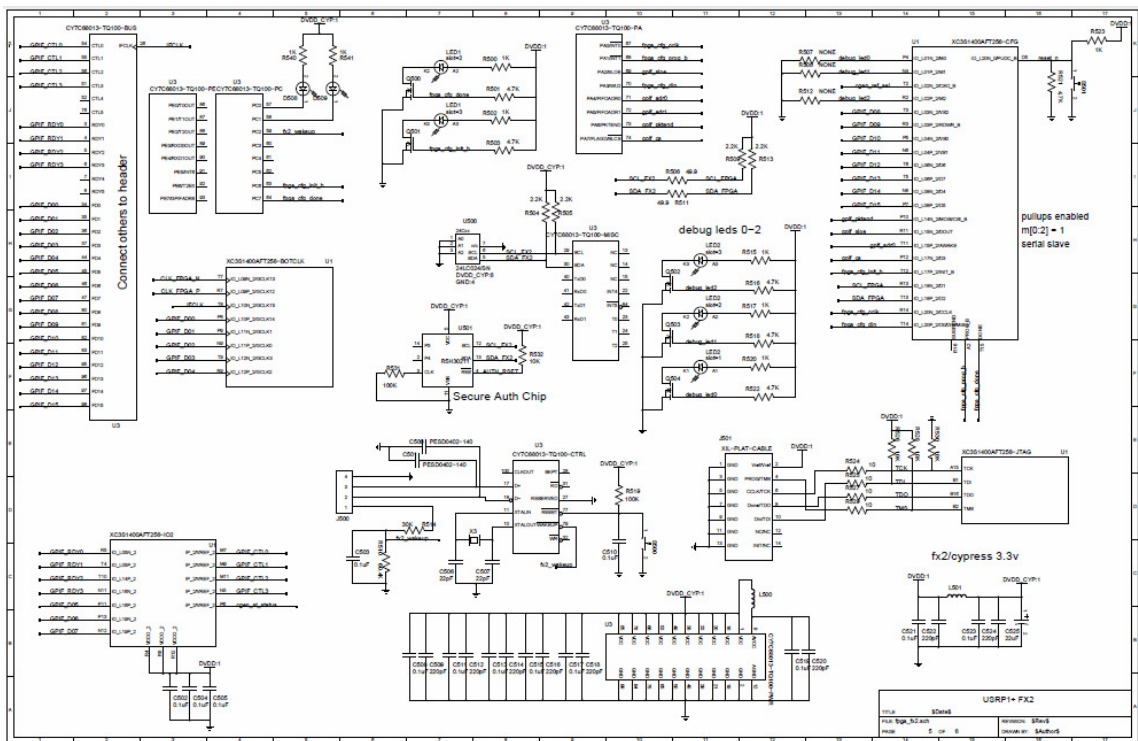
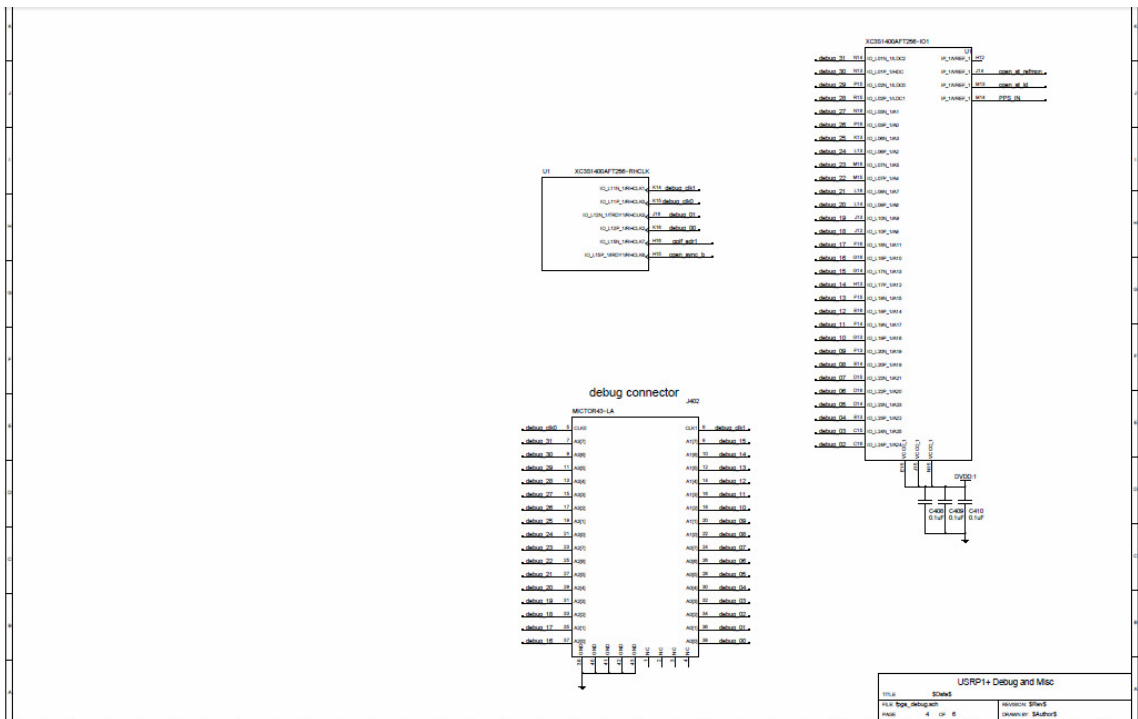
1	A	B	C	D	E	F																							
2																													
3	<p>Low Noise Amplifier PGA 103+</p> <p>Gain @1400 MHz 12dB</p> <p>Noise Figure @1400MHz 0,7 dB</p> <p>P1dB 22,5dBm typ. @ 2 GHz @5V</p> <p>High IP3, 45dBm typ. @2GHz, 5V</p>																												
4				<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th colspan="2">Änderungen</th> <th>Datum</th> <th>Name</th> <th>Bezeichnung:</th> <th>Blattzahl:</th> </tr> </thead> <tbody> <tr> <td>Datum</td> <td>Name</td> <td>gez.:</td> <td></td> <td rowspan="2" style="text-align: center;">LNA PG103+</td> <td></td> </tr> <tr> <td></td> <td></td> <td>gepr.:</td> <td></td> <td>Blatt-Nr.:</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td>Zeichnungs-Nr.:</td> <td></td> </tr> </tbody> </table>			Änderungen		Datum	Name	Bezeichnung:	Blattzahl:	Datum	Name	gez.:		LNA PG103+				gepr.:		Blatt-Nr.:					Zeichnungs-Nr.:	
Änderungen		Datum	Name	Bezeichnung:	Blattzahl:																								
Datum	Name	gez.:		LNA PG103+																									
		gepr.:			Blatt-Nr.:																								
				Zeichnungs-Nr.:																									

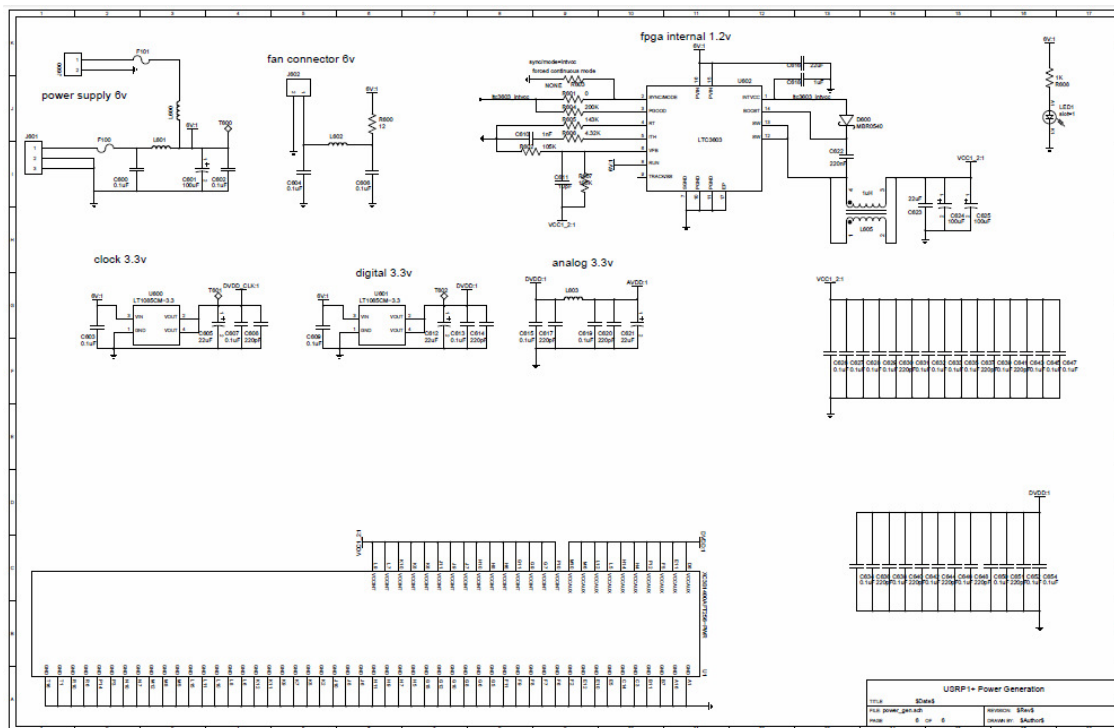
# Motherboard USRP B100




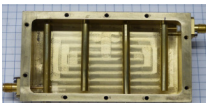












### A.3 List of off-the-shelf electronic components and assemblies

Receiver box		250 €
Hydrogen filter		
Power supply		22 €
Low noise Amplifier		120 €

Line Amplifier		3.80 €
Antenna		120 €

## A.4 Mechanical Components

### Hydrogen Filter

