TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Page

LIST OF ABREVIATIONS

ACARS	Aircraft Communications Addressing and Reporting System
ADS-B	Automatic Dependent Surveillance-Broadcast
ATC	Air Traffic Control
ADC	Analogic to Digital Converter
CPU	Central Processing Unit
CRC	Cyclic Redundancy Code
CRIAQ	Consortium de Recherche et d'Innovation en Aérospatiale du Québec
DAC	Digital to Analog Converter
DF	Downlink Format
DME	Distance Measurement Equipment
DPSK	Differential Phase Shift Keying
DSP	Digital Signal Processor
FAA	Federal Aviation Administration
FMS	Flight Management System
FPGA	Field Programmable Gate Array
GNSS	Global Navigation Satellite System
GPS	Global Positionning System
GRC	GNU Radio Companion
GUI	Graphical User Interface
ICAO	International Civil Aviation Organization
OMAP	Open Multimedia Application Platform
PAM	Pulse Amplitude Modulation
PCIE	Peripheral Component Interconnect Express
PI	Parity/Identity
PPM	Pulse Position Modulation
RF	Radio Frequency
RTCA	Radio Technical Commission for Aeronautics
SDAR	Software Defined Avionics Radio

SDR	Software Defined Radio
SNR	Signal to Noise Ratio
SPI	Special Pulse Identification
SSR	Secondary Surveillance Radar
SWIG	Simplified Wrapper and Interface Generator
TACAN	Tactical Air Navigation System
TMS	Transponder Mode S
UHF	Ultra High Frequency
VDL	VHF Data Link
VHF	Very High Frequency
VOR	VHF Omnidirectional Radio Range
SWAP	Size Weight and Power

INTRODUCTION

The need of the aviation industry for innovative ways of integrating current and future avionics systems while taking into consideration size, weight and power (SWaP) constraints is the key factor that enables this project. The introduction of modern processing units that incorporate Field Programmable Gate Arrays with traditional top of the line microprocessor units offer new opportunities with regards to what can be achieved by tight cooperation of these two systems, suddenly tasks that would seem impossible or very complicated to achieve appear now attainable by performing the proper task division and communication routines between them.

Challenges still remain that will put to the test the existing cooperation schemes between the two processing units with regards of timing and efficiency; systems such as the ones here described have already proved their efficiency in fields such as telecommunications and image processing industries, however not much work has been done in the field of avionic navigation systems.

The benefits of the integration of electronic navigation systems and other avionics are not only the reduction in SWaP but also the potential increase in reliability that entirely programmable systems may bring to scene in terms of configurable redundancy and by homogenizing the hardware building blocks of future avionics systems.

This project consists of the definition, design and implementation of a reusable architecture that results solid enough to incorporate modular applications, which shall be able to be loaded and unloaded from the system as the requirements of the mission change. A prototype including a graphical user interface needs to be developed in order to interact with more than one system at a time. It will be showcased as a proof of concept and prepared to perform flight-tests in a real life scenario, the information and results collected will be made available for improvements and future research on the subject. The architecture will also take into account that it will be used for research and development, so it must allow to a group of students to perform their work in a jointly but asynchronous way.

In order to achieve the desired result, a definition of tasks was carefully planned and executed in an orderly way, the work here presented is composed of 6 chapters; the first one introduces the project, its scope, challenges and objectives. Chapter 2 presents the current trends of research in topics related to the one studied by this project, Chapter 3 is an analysis of the scenarios that were considered for the development of the presented architecture and provides the reader with an analysis of such scenarios. Chapter 4 features the resulting architecture and explains the way in which it was implemented. Chapter 5 incorporates the GUI developed for this project, which results crucial for the system´s multi-application operation; finally Chapter 6 covers the system's operation and its achieved performance.

CHAPTER 1

PROJECT PRESENTATION

1.1 Context Overview

The work presented in this document was developed at LASSENA (**L**aboratoire des technologies **S**patiales, **S**ystemes **E**mbarques, **N**avigation et **A**vionique). Because of this, the entire research process is centered on avionics navigation systems and navigation environment.

Even when the technologies used and/or developed in this project could be analyzed from a more general point of view, the avionics and navigation systems approach is always privileged and their analysis is performed from that perspective, the goal of the project was to find and develop means of integrating the different avionics systems that LASSENA had developed into a scalable platform that allowed for future expandability and the aggregation of new systems as they are developed at the laboratory in future projects.

1.2 Research Problematic

The number of avionics systems required by regulation on civilian and commercial aircrafts has been rising steadily since 1950, when the term was coined (Collinson R. , Introduction to Avionics Systems, 2011), and it is only expected to grow in the forthcoming years as the amount and complexity of airborne systems increases.

Up to this day, each time that a new communication technology has been introduced to the navigation environment, the formula has been usually to add new equipment, which requires more space in the already space-restricted avionics systems installation in aircraft cabins. “Equipment is usually at the duplex level of redundancy, for the case of modern airliners

VHF radios are at a triplex level of redundancy" (Collinson R. , Introduction to Avionics Systems, 2011) with each device making use of its specific radio and antenna pair.
This is only understandable given the required level of security and integrity at which the aerospace industry operates and due to the considerable complexity of each modern communication device that needs to be installed into an aircraft.

The advent of modern Field Gate Programmable Arrays (FPGA)-Microprocessor cooperative embedded systems gives us a new opportunity to test the extent to which is currently possible to integrate two or more different airborne communication systems into a single generic reprogrammable universal communication device. The following paragraphs introduce the various motivations that enable this research.

1.3 Motivation and Objectives

The presence of multiple burdensome radio communication devices in airplanes is a strong argument to develop an integration architecture that reduces the excessive infrastructure and replaces it by a more flexible Software Defined Radios (SDR) based on shared and reconfigurable hardware.

This project will explore the creation of a standard integration platform that permits the reutilization of hardware components by means of in site re-configurability, providing multiple simultaneous radio operation in order to accommodate future software defined functionalities.

The project's main objective is to serve as a proof of concept by demonstrating the integration of three independent avionics subsystems such as: TMS (Transponder Mode S), ADS-B (Automatic Dependent Surveillance-Broadcast) and DME (Distance Measurement Equipment) into a state-of-the-art prototyping Software Defined Radio platform. To achieve this objective, a highly optimized tight integration between software and FPGA bitstream image will be designed and implemented to allow the greatest possible extent of cooperation between the FPGA and i7 processing units in such a way that concurrent operation of all the three systems is attained.

As mentioned in (LASSENA, 2013) the successful implementation of the work previously described will enable for future research areas such as:

- The development of a test bench for avionics communication systems and performance analysis, which permits achieving a high level of reliability for the SDR communication system's applications;
- The analysis and study of the integration architecture of multiple individual SDA modules into a single hardware;
- The leverage of significant interoperability issues (partial reconfiguration, on-the-fly reconfiguration, resource sharing, etc.);
- Performance analysis of the integrated avionic SDR in real-life scenarios.

The system developed in this project was conceived in several incremental steps, the SDAs previously developed at LASSENA went through a revision and test phase to assess the possible ways of integration into a single scalable platform. After being separately evaluated into their original development platform (Zepto, USRP) they were ported to the defined integration system and their capabilities were re-assessed accordingly.

It was until this point when an integration effort was started. Tests in the laboratory were performed in an incremental way until the full system was completed. A full testing documentation package (available in Annexes I to V) was prepared for the selected prototypes to be tested in flight; before the execution of the flight tests, ground tests were also performed in the plane and in-ramp where applicable.

1.3.1 AVIO-505 Project

The AVIO-505 project is an industry-academia collaborative research effort between LASSENA and the Consortium de Recherche et d'Innovation en Aérospatiale du Québec (CRIAQ) the work developed during this master's project is framed as part of the AVIO-505 project, which has a much broader scope. A brief description of the project is presented:

AVIO-505 project aims to establish new design methods and digital signal processing techniques for robust and efficient universal navigation and communication equipment in the fields of aeronautics and aerospace. The project anticipates the integration of multiple navigation and communication systems in a single hardware element minimizing connectors, antennas, cable length, electromagnetic interference (EMI) and system footprint. The project's goal is "to digitize the radiofrequency (RF) signal in proximity to the antenna and to transmit the baseband signal to a generic radio for further digital signal processing" The proof-of-concept demonstrator will be evaluated in-laboratory and in-flight using simulation equipment and a flight test platform under real operating conditions in order to characterize protocols and system performance. "The developed technologies will also be applicable to ground or airborne infrastructure." (LASSENA, 2013).

The context overview and motivations that enable this research have been briefly introduced by these first pages. The following chapter presents a literature review, which covers the development of avionics systems and the historical integration efforts that have been introduced since their appearance in the aviation industry.

CHAPTER 2

LITERATURE REVIEW

2.1 Reconfigurable Avionics Systems

Military investment in modular radio components during the 90’s (Upmat, 1995) enabled technologies that permitted to bring most of the signal processing components of radio system into a computer (Bivona, 2009). These efforts pioneered the study of Software Defined Radio Systems, later, by the beginning of 2000 efforts such as (Eyermann, 1999) continued advancing the research in avionics related fields. Finally the Spectrumware project served as a root for two different SDR development branches, Vanu Radio (J. Chapin, 2002) and GNU Radio.

For the last 30 years approaches that aim at facilitating the integration of avionics systems have been under development; the two primary widely accepted guidelines for avionics architectures in the US are: an Open Systems Architecture (OSA) directive from the Department of Defense (DoD) and an Integrated Modular Avionics (IMA) standards guideline from DO-297 (Gask, 2015).

There is a recent interest from avionics manufacturers to develop upcoming avionics systems able to perform expected growing software volume and capability. This need is a challenging one as software exhibits some difficulties to meet increasingly stringent performance requirements while keeping other requirements in terms of reliability, continuity, availability and integrity. In (Strunk, Knight, & Aiello, 2004), a flexible architecture based on distribution of function and assured reconfiguration that can react to failures in both hardware and software is introduced. The authors manage to architecture reconfigurable software satisfying safety properties and enhancing analysis capabilities for critical safety properties and reduce certification costs for much of the system.

In (Hodson, 2007) a reconfigurable avionics for exploration is fully described. The goal of avionics in exploration is to provide scalable interoperable avionics for a wide class of space exploration missions to optimally satisfy or outperform the highly demanding specifications while at the same time minimizing the cost of development, deployment, and maintenance of avionics in the space vehicles. This is added to the fact that space missions develop for several years during which the needs evolve at a high pace while the hardware equipment cannot be updated after the mission has started. It is for this reason that reconfigurable software avionics play a fundamental role, since software is the only component of the avionics that can be updated to meet the new requirements.

An interesting approach to reconfiguration, software defined avionics is presented in (Suo, An, & Jihong Zhu, 2011). As the main concern of SDA relies on reliability, this work explores a new technique to realize fault-tolerance and respond to changes in external environment. The technique is applied to the design of Integrated Modular Avionics. The main novelty of the work refers to their innovative approach to address the safety problem of Avionics reconfiguration. Traditional analysis approaches mainly focus on single component failure. Therefore, they might underestimate the influence of a design flaw. On the contrary, the authors propose the use of System-Theoretic Process Analysis (STPA) to carry out a hazard analysis. They focus on the interaction between human operator and equipment, and define two criteria to decide the level of autonomy: the Failure Degree, and the Time budget. System dynamics are applied to analyze and model human factors such as mental workload, situation awareness and complacency.

Integrated modular avionics face several challenges in order to enable their operation in aerospace systems (Gask, 2015). Firstly, multi-core systems do not currently meet the aviation standards in terms of reliability and integrity. It is critical to leverage chips with many cores (some of which are heterogeneous) for affordability and size, weight, and power (SWAP) constraints. This cannot be achieved without the insertion of new and refined software standards for improved reuse. Therefore software-defined avionics require necessarily a new mind-set in the regulatory agencies only to be achieved after SDRs have proven its reliability for the aviation sector. In addition, the unification of mixed networking technologies is important to support bandwidth and connectivity flexibility. For instance, the use of fiber optics can improve significantly both the weight and the available throughput required by these systems. Therefore, signals do not necessarily need to be processed at the antenna and distributed avionics systems can be deployed while decreasing dramatically the weight of the cabling. Software defined avionics can also provide greater autonomy and improved safety and availability since they can run self-check routines and self-diagnose operational problems. Trying to meet this objective with conventional hardware-based systems is, nowadays, beyond the state-of-the-art technology.

We can notice that two variants of the SDR implementation exist, one in which most of the processing takes place on the FPGA, and the computer only executes control and display tasks and another in which the signal processing tasks are carried out by a computer where we can find a general purpose processor. In this latter case, the FPGA only performs basic signal processing such as filtering and down sampling (Debatty, 2010). In the last years a notorious increase in Lines of Code (LOC) has been observed in avionics systems, where systems have moved from 100K LOC to 400K LOC. It is expected that in the next decades the complexity will continue increasing exponentially. There has been as well a generalized focus in Size, Weight and Power reduction for avionics systems development (Gask, 2015) to the extent of prototyping entire avionics systems in SoC (System on a Chip); this approach has found limitations for certification of safety-critical embedded systems (El Salloum, 2013).

Future avionics systems will certainly base their hardware in multicore general-purpose processor because of their reduced cost and high performance. According to (Gask, 2015), it is expected that by 2016-2018, there will be on-chip multicore processors with 16 or more cores on each die integrated with on-chip transformational multi-Teraflop General Purpose Graphics Processing Units (GPGPUs). The aviation sector cannot disregard the computational power of this equipment and will definitely have to adapt its mainstream to the use of these devices from which software defined avionics will be their major beneficiary.

2.2 "SDR", "SDA" and "SDAR"

During the next pages the acronyms SDR, SDA and SDAR are frequently used throughout the text; it is important though to clearly identify what is referred to when making use of these terms. For this reason they are defined and put into context for the scope of this project:

Software Defined Radio (SDR): According to ITU-2009 definition, “a radio transmitter and/or receiver employing a technology that allows the RF operating parameters including, but not limited to, frequency range, modulation type, or output power to be set or altered by software, excluding changes to operating parameters which occur during the normal pre-installed and predetermined operation of a radio according to a system specification or standard.” (International Telecomunication Union (ITU), 2009).

Software Defined Application (SDA): A set of software instructions or computer program that interacts with the SDR components either to obtain/send information through them or to perform its configuration with the final goal of replicating specific equipment functionality.

Software Defined Avionics Radio (SDAR): It consists of the application of the SDR concept to the field of avionics. An SDAR can optimally perform any avionics application, including but not limited to communications, navigation and surveillance systems. SDARs promise to eliminate unnecessary redundancies in the avionics equipment and therefore minimize the weight and fuel consumption of aircraft operations, which in turn translates to a greener aviation.

2.3 Selected Avionic Systems for the Integrated Architecture

As briefly mentioned in the previous chapter, this Master's research project has been developed within the pale of collaborative research and development project AVIO-505. Therefore, its main objective is given by the definition of the AVIO-505 project, which aims at demonstrating through a proof-of-concept prototype the feasibility of Software Defined Avionics Radio (SDAR). This prototype must perform a functionality of each of the three main functions required by aviation, i.e. Communications, Navigation, and Surveillance. For the communications systems, a wideband radio system was selected in addition to satellite communications equipment. As regards the surveillance systems, the nineties-developed system transponder mode-S (TMS) and the NextGen keystone system named Automatic Dependent Surveillance-Broadcast (ADS-B) were selected. Finally, for the last function, which is navigation, the selected system is the distance measuring equipment (DME). This project will integrate these three avionic systems into a single piece of hardware; contributing in this way to the simplification of the intricate avionics systems interconnection that exists nowadays. The following sections are devoted to briefly describing the particularities of each of the selected systems:

2.3.1 Distance Measuring Equipment

The Distance Measuring Equipment is an avionics navigation equipment that measures the geometrical distance between the aircraft and a ground station (Collinson R. P., 2013). As the location of the ground station antenna is known, this information can be used in combination with other in order to determine the aircraft position. From one DME measurement, the aircraft can be located within a sphere around the ground station. Combined with another DME measurement, or VOR, the uncertainty can be reduced to a circumference (half a circumference when the VOR and DME stations are collocated, which is usually the case) normal to the ground plane. Thus, only the altitude is needed in order to determine the aircraft position. The airborne DME equipment operates very similarly to radar equipment. A pair of short pulses is omnidirectionally transmitted and received by the ground station. After introducing a known deterministic delay, the ground station retransmits this pair of pulses at a different frequency that determines the full-duplex channel. The ground station acts, thus, as a transponder. Finally, the reply from the ground is received at the aircraft and the measured is between the interrogation and the reply is measured, from which the distance or slant range is determined. As it can be seen, the DME is similar to a secondary radar, with two differences, the radar is not on the ground but airborne, since the antenna used is not a scanning one but omnidirectional, no angular information can be extracted from the measurement. The interrogation is transmitted at frequencies within the range 1025MHz to 1150MHz, while the replies can be received from 960 MHz to 1215 MHz. As it can be seen, both frequency overlay and proper channel frequency planning must be done in order to prevent self-interference. Another source of intra-system interference is produced by multiple aircraft interrogating the same ground station, which can only operate in one frequency channel. The medium access is done by collision using low duty-cycles and using a randomly generated interrogation pattern. Therefore, only replies following the same random pattern unique for every aircraft are identified as own. The use of different transmission schemes called mode X and mode Y doubles the number of channels. (LASSENA, 2013) (Spitzer, 2014) (Collinson R. , Introduction to Avionics Systems, 2011).

2.3.2 Automatic Dependent Surveillance – Broadcast

The Automatic Dependent Surveillance – Broadcast is the keystone of the NextGen and SESAR programs at the USA and the European Union, respectively, both programs being harmonized by ICAO. ADS-B is the modernization of the Air Traffic Management (ATM) Surveillance system currently based on the ATCRBS and the Transponder Mode-S briefly described below (Collinson R. P., 2013). When both are installed in an aircraft, the TMS and ADS-B equipment must be part of a single piece of hardware equipment (RTCA Special Committee 170, 1992). Through ADS-B, all the air traffic participants (not only aircraft, but also ground vehicles, UAS, parachutes, obstacles, etc.) broadcast their 3D position, latitude, longitude and height to the rest of participants. The system is therefore “dependent” on an additional navigation system providing it with this 3D position. Usually, the navigation system used is GPS, which dramatically diminishes the uncertainty of the aircraft position, from hundreds of meters achieved by secondary radar-based traffic control to few meters provided by the GPS system, especially in those areas where an augmentation system such as WAAS (Wide Area Augmentation System) or EGNOS (European Geostationary Navigation Overlay Service) has been approved for its use in aviation. This will enable separation limits reduction between aircraft in safe conditions, which directly translates to an increased capacity (in flights per hour) of the air transportation system. The ADSB system operates at frequency 1090 MHz, (the same as TMS) which is why the systems need to be synchronized when collocated. Traffic control can therefore be automated, although such automation is not part of ADS-B. Currently, computers on the ground integrate all the information and provide controllers with timely warnings of potential problems. (LASSENA, 2013) In the USA, the ADS-B is also enabled to operate at the Universal Access Transceiver (UAT) frequency 978 MHz.

2.3.3 Transponder Mode-S

The Mode-S (S for Select) of the secondary surveillance radar (SSR) was designed as an evolutionary addition to the Air Traffic Control Radar Beacon System (ATCRBS) to provide the enhanced surveillance and communication capability required for Air Traffic Control (ATC) automation; the spec was delivered to FAA in 1975 (Freeman, 1995) (Collinson R. P., 2013). However, the promised automation has not been yet achieved, which is one of the main motivations of ADS-B. A transponder Mode S performs all the functions of Mode A (squawk code only) and C (squawk, pressure and altitude) transponders and has data link capability. This capability is the one exploited by ADS-B for its operation, although the use of software defined avionic radios would have enabled much more flexible (and long-term) solutions. Some of the TMS specific advantages over Modes A and C are:

- Since aircraft are selective interrogated, as opposed to Modes A and C, the channel is less saturated which increases the capacity of the system. However, it is expected that modern ADS-B can produce an increase in the 1090 MHz channel saturation again, going back to the times of the modes A and C transponders;
- The accuracy of TMS is improved when compared to Modes A and C, thanks to the use of monopulse techniques and increased bandwidth. However, ADS-B (when based on GPS) overpasses by far this improvement and makes TMS obsolete as a source of traffic positioning information;
- High degree of data integrity in ground-to-air, air-to-ground and air-to-air data link, as CRC codes are used with a probability of an undetected bit error lower than 10^{-7}. (RTCA Special Committee 209, 2011);
- In TCAS equipped aircraft, the TCAS transmits coordination/interrogations to the other aircraft via the Mode S link in order to ensure the selection of complementary Resolution Advisories. (LASSENA, 2013).

2.3.4 The Wide Band Radio communications systems

Avionic communications systems, such as ACARS, VDL (VHF Digital Link), or voice communications, use little bandwidth and represent no challenge for the AVIO-505 project. Similar considerations apply to aircraft communications through satellites, currently using mainly the L-band frequencies. On the other hand, Ku and Ka-band satellite communications systems definitely pose a meritorious challenge for the AVIO-505 project but the development of a proof-of-concept-prototype would suppose a prohibitive increase in the project’s budget. As an alternative, a custom communications system was selected and specified. Among these specifications, one can find high throughput (4 Mbps minimum) and increased robustness thanks to the use of Adaptive Modulation and Coding (ACM), which is a technique widely spread in satellite communications.

2.4 Software Integration Tools

An overview of the two main tools used for this project´s development is presented here; the intention is to take the reader into context not to have a detailed explanation of them. For further information regarding these tools refer to their original sources in the reference section.

2.4.1 GNU Radio

GNU Radio is a free Software Development Toolkit (SDK) that has greatly accelerated the development of avionics systems in the AVIO-505 project. This SDK provides the avionics engineer with a set of signal processing libraries including many of the most common signal processing tasks such as filtering of phase lock loops. It also provides the capability of real-time scheduling and a very useful feature for this master project which is the capability of dynamic core affinity for the individual tasks during runtime. Another important feature is that it provides compatibility with a number of software radios using readily available, low-cost external RF hardware and commodity processors. Moreover, SDR manufactures are not unaware of the popularity of GNU Radio in the SDR developer community and very often provide a GNU Radio plugin to interface with their hardware. Such is the case with at least the following three Nutaq devices: ZeptoSDR, PicoSDR and PicoDigitizer. GNU Radio is widely used in hobbyist, academic and commercial environments to support wireless communications research as well as to implement real-world radio systems.

GNU Radio applications are primarily written using the Python programming language. However, there is a complementary python application, GNU Radio Companion (GRC), which provides the user with a developing environment similar to Simulink's environment.

All the performance-critical signal processing path is implemented in C++. GRC allows the C++ blocks to be interconnected in a graphical interface and it is the tool the one that generates proper Python code for the engineer. (GNU Radio, 2015)
However, GRC offers very limited capabilities with regards to this master's project. First, many of the avionic systems performs tasks that go much beyond merely signal processing. All these tasks need therefore to be implemented in python, which is not allowed by GRC. Second, other tasks related to the SDARs and the reconfigurability aspect can only be implemented outside GRC, specifically dynamic core reassignment, dynamic radio reconfiguration, inter-systems interaction, etc.

2.4.2 Xilinx System Generator and Nutaq's Model Based Design Kit

Most of the FPGA programming has been done at the highest possible level of abstraction by means of Nutaq's Model Based Design Kit (MBDK) for Xilinx's System Generator (Xilinx, 2015). Since one of the main objectives of the AVIO-505 project is to prove that avionics systems can be designed and implemented exclusively in the software domain, it did not make sense to use low-level tools to program them, in which case, all of the benefits of the SDAR during the design and development stages would have just vanished. Nutaq's MBDK is a plugin thought for its use along with Xilinx's System Generator. The kit allows the user to program the FPGA just by interconnecting elementary (and not so elementary, e.g., FIR filters or DDS's) signal processing blocks that perform the desired tasks for the FPGA. It is a system-level modeling tool that facilitates FPGA hardware design. It extends Simulink in many ways to provide a modeling environment that is well suited to hardware design. This provides the engineer with high-level abstractions that can be automatically compiled into an FPGA. With these tools, the design phase of the digital signal processing performed by FPGA devices can be carried out at a higher level, which results in reduced development time. Although not required in this project (which was intended), the tool also provides access to underlying FPGA resources through low-level abstractions.

2.5 The Integration Platform

A state of the art SDR system was provided by Nutaq®, which is one of the strategic partners of the AVIO-505 project, the SDR system comprises a fully integrated solution for embedded signal processing. The following section introduces the system by depicting its building blocks and better explaining its main capabilities.

2.5.1 Nutaq’s Software Defined Radio (PicoSDR2x2-E)

A top-view diagram of NUTAQ´s PicoSDR2x2-E, showing its building blocks and the way in which they interconnect is presented in Figure 2.1. Section 2.5.1.1 outbreaks more in depth each of the relevant system´s hardware components; the hardware component’s limitations are consequently addressed in section 3.3, where an overall consideration of the system´s capabilities is performed.

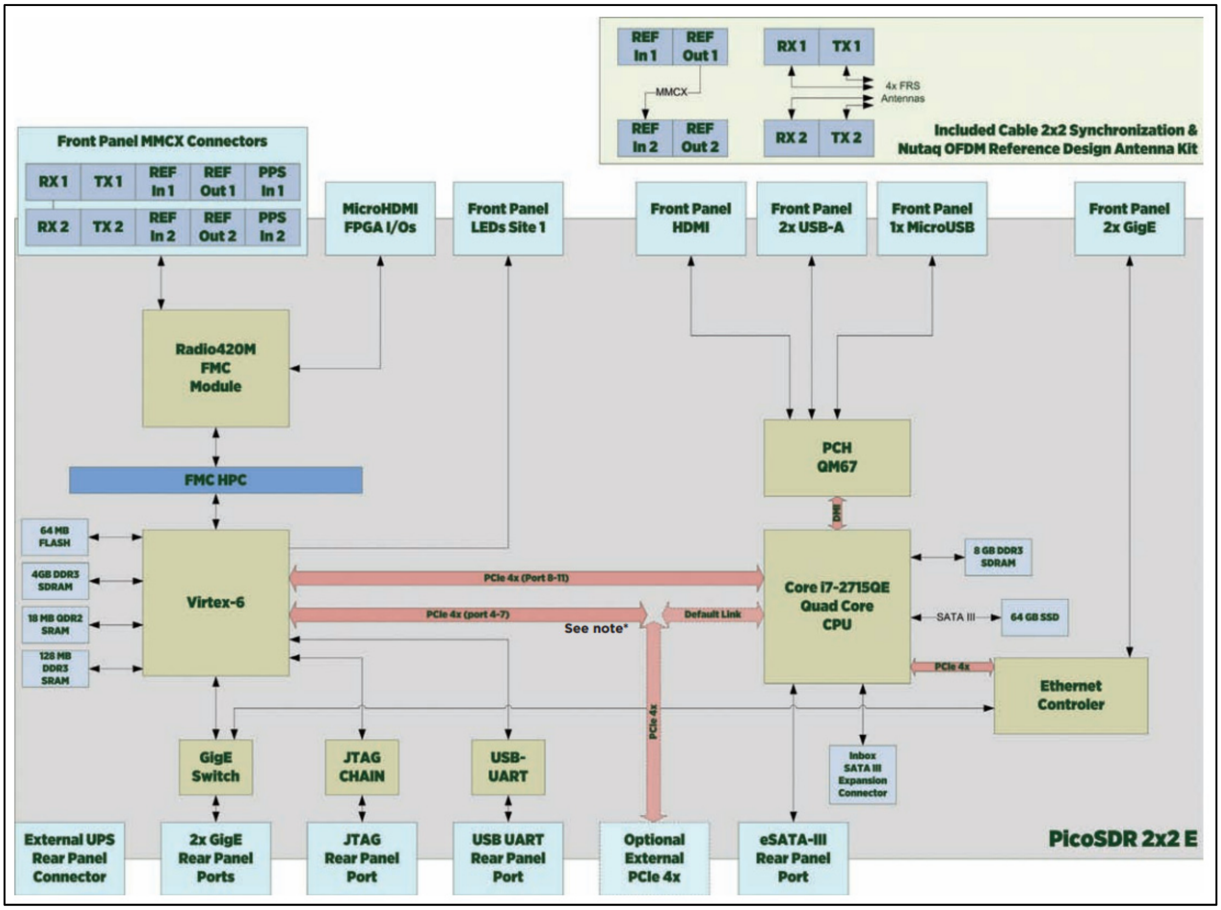


Figure 2.1 PicoSDR Building Blocks Interconnection (NUTAQ, 2014)

2.5.1.1 **Internal Hardware Components and Specifications**

An overview of the crucial PicoSDR hardware components used during the development of this project is presented in the following sections. The overview permits the reader to have a better idea of the general system´s capabilities. The PicoSDR components, which are essential for the project development are:

1. The radio card (Radio 420M FMC),
2. FPGA’s Mezzanine card (Perseus 6010),
3. FPGA type and family (Xilinx Virtex 6),
4. The embedded host system (SAMC-514).

2.5.1.2 **Radio 420M FMC**

The Radio420X FPGA Mezzanine Card (FMC) is a powerful SDR RF transceiver module designed around the Lime Microsystems LMS6002D RF transceiver IC, it is able to operate in multiple modes, multiple standards and multiple bands, supporting broadband coverage, TDD (Time Division Duplex) and FDD (Frequency Division Duplex) all of it while maintaining full duplex operation. Its selectable transceiver's bandwidth covers from 1.5 to 28 MHz, which makes it suitable for a large number applications (both broadband and narrowband). It incorporates multiple references and synchronization modes that allow the Radio420X to be appropriate for multimode SDR applications. As mentioned in (NUTAQ, 2014), other suitable applications include:

1. MIMO systems,
2. Cognitive radios,
3. LTE, WiMAX,
4. White space,
5. Wi-Fi,
6. GSM,
7. WCDMA,
8. Signal intelligence (SIGINT).

2.5.1.3 **Perseus 6010**

The Perseus 6010X is an advanced mezzanine card (AMC or AdvancedMC) designed by Lyrtech®. It is designed around the Xilinx Virtex-6 FPGA, allowing great flexibility to select between different Virtex 6 parts for the boardand up to 4 GB of external memory. It benefits from multiple high-pin-count, modular, add-on FMC-based I/O cards. "The Perseus is intended for high-performance, high-bandwidth, low-latency processing applications"

(NUTAQ, 2014). The card is fully supported by Nutaq's advanced software development tools, which aids with design when aiming to reduce "size, complexity, risks and costs associated to leading- edge telecommunications, networking, industrial, defense and medical applications" (NUTAQ, 2014).

2.5.1.4 **Nutaq's Real Time Data Exchange (RTDEx)**

RTDEx is an FPGA soft-core developed by NUTAQ, which "allows high-speed transfers of data between a host PC and an FPGA device, or between FPGA devices. The RTDEx uses the Gigabit Ethernet-base channel or PCI Express Gen1 4x. The PCI Express is only usable with the Linux operating system on an embedded AMC processor" (NUTAQ, 2014). Being the main communication protocol used to transfer data between the host system and the FPGA (the other being custom register direct access), it resulted a crucial tool for the integration of the system. Sections 4.1.1, 4.2.3.2, 4.2.3.3 and 4.2.3.6.3 contain further information on this protocol and its utilization in the project; in the same way Appendix VI is entirely dedicated to the tests performed for the protocol characterization.

2.5.1.5 **Xilinx Virtex 6**

FPGA family developed by Xilinx® "The **Virtex-6** family is built on a 40 nm process for compute-intensive electronic systems" (PR Newswire, 2009).

2.5.1.6 **SAMC-514**

The SAMC-514 processor module is an AMC form factor unit developed by the Russian company Scan Engineering Telecom ("CJSC" by its Russian Accronym). The module is based on a high-performance Intel Core i7 (Sandy Bridge) processor; the SAMC-514 unit combines a wide range of inter-module interfaces with large amounts of RAM. (Setdsp, 2014).

The literature review presented in this chapter covers the avionics systems integration schemes and its development from the early 90´s to the current day. It introduces the reader to the more recurrent terms used throgouth the work and defines them in the cases where the terms are not taken from previous works. A brief presentation of the tools in which the integration architecture, product of this work, is to be integrated and presents in the same way the more important system platform components' specifications and its capabilities.

CHAPTER 3

ARCHITECTURE ANALYSIS

In this chapter, the hardware capabilities of the integration platform presented in Section 2.5 are taken into consideration for the first time to explore possible solutions to the given problem. The system's required tasks are defined from a top level perspective and two different approaches for the integration of the avionics applications addressed in Chapter 2 are presented.

3.1 System's tasks identification

The system's processing functionality can be divided in two main functional areas:

1. Digital Signal Processing (DSP) tasks,
2. Control tasks.

These two areas can be divided further into more concise functions that will be later assigned to one of the two processing units found in the PicoSDR2x2-E:

1. i7 (CPU),
2. Virtex-6 (FPGA).

Table 3.2 presents the system's tasks organized by functional area and shows which of the sub-systems in the PicoSDR are capable of executing such tasks.

Table 3.1 Processing tasks and available system's capability

Processing tasks and. available sub-system's capability			
	System's Tasks	System's Capability	
		i7	Virtex-6
		(CPU)	(FPGA)
DSP tasks	SDA Incoming signal conditioning	✓	✔
	SDA Signal processing	✔	✔
	SDA Outgoing signal Conditioning	✓	✔
Control tasks	Graphical user interface (GUI) presentation	✔	
	SDA selection and routing	✔	✔
	Rx gain control	✔	✓
	Tx gain control	✔	✓

As seen in Table 3.1 just the GUI presentation is restricted to be executed by the CPU. For all of the other tasks, decisions have to be taken regarding their assignation to at least one of the processing sub-systems. It is important to mention that for some of the tasks, processing can occur in both of the sub-systems at different stages of the signal processing, so the assignation of one of them to a processing unit does not excludes it from being assigned to the other one as well.

Figure 3.1 shows a detailed diagram of the tasks introduced by Table 3.1.

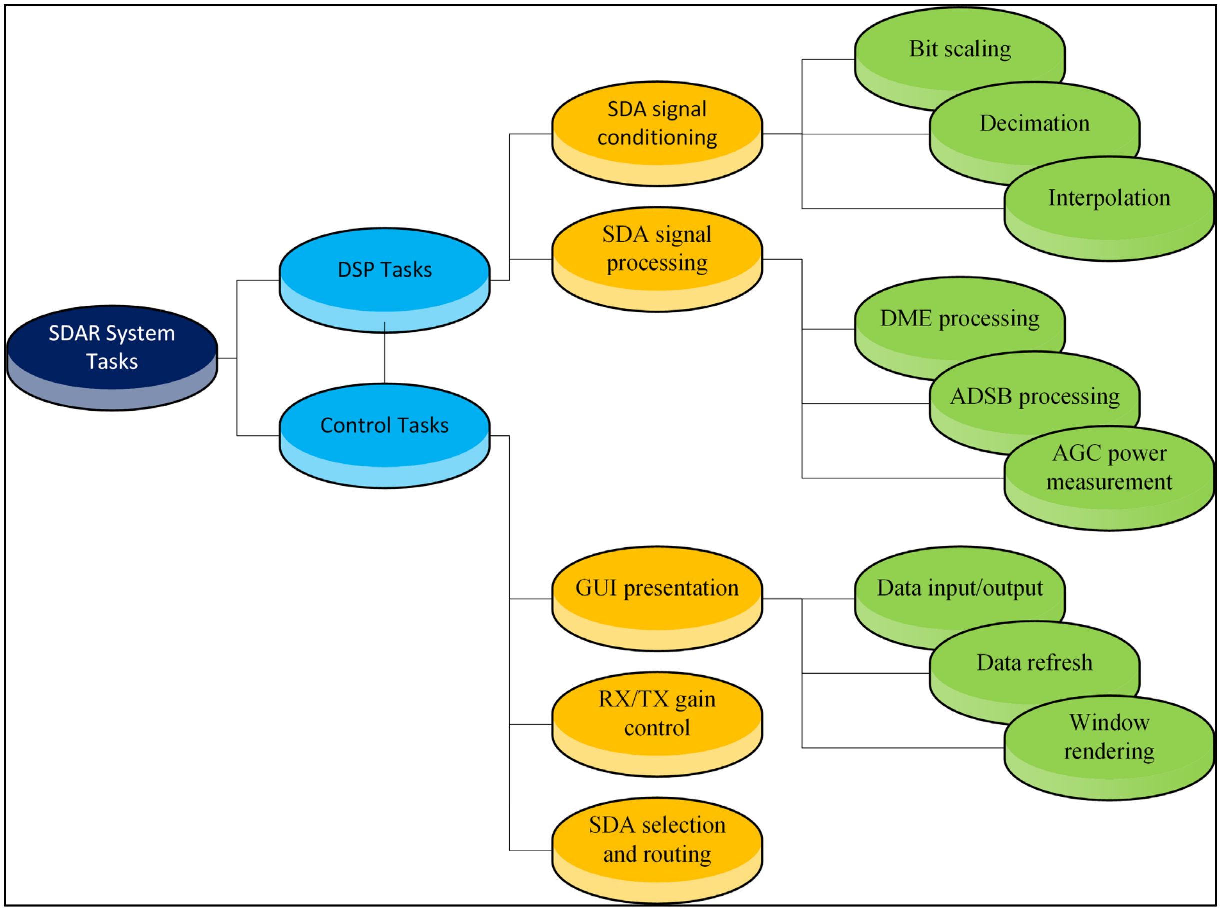


Figure 3.1 Identified SDAR System Tasks

3.2 Integration Architecture Approaches

The model considers the seamless integration of SDAs systems whose processing logic may come from three different design approaches based on the processing tasks that are assigned to each of the processing units of the PicoSDR:

1. **[TYPE I]** -PURE CPU SDA: Where the FPGA is used in a pass-through configuration and the processing happens almost entirely at the CPU level;
2. **[TYPE II]** -HYBRID CPU-FPGA SDA: Where the logic is balanced between the FPGA signal processing capabilities and the CPU;

3. **[TYPE III]** -PURE FPGA SDA: Where the whole SDA is hardware defined and just the user interface information is sent to the CPU for presentation and user interaction[1].

The architecture receives baseband inputs and it outputs the processed stream through RTDEx-Ethernet interface or directly through one of the peripherals if the entire signal processing is performed inside the FPGA. The architecture is planned to be compatible with as many pure FPGA SDA as the FPGA fabric can fit in. For the applications that make use of RTDEx (TYPE I and TYPE II SDAs), the limit is imposed either by the maximum number of RTDEx channels that can be simultaneously open or by RTDEx's upper throughput limit (whichever is reached first) [2].

3.2.1 Full FPGA Solution

It is possible to define the entire SDAs' logic by making use of a combination of System Generator® blocks and hand written/modified VHDL code. Unsurprisingly this approach requires depth knowledge of VHDL and certain characteristics of the defined applications would be tied to the specific model and limitations of the FPGA for which the design is originally made. This would also result in the application with the highest overall performance; since there is no computer processor involved in the flow path of the signal, the hardware-described implementation would result in deterministic latency of the system. The only limit to this approach would be the space (in logic gates) provided by the specific FPGA model in which the SDAs would be defined.

[1] From this point on, when referring to the SDAR system classification, the labels: TYPE I, TYPE II and TYPE III are used.

[2] RTDEx nominal upper-limit is 1 Gbps, however as shown in APPENDIX VI by the tests performed in the lab, the actual limit is 553.208*2 Mbps.

Since each system to be defined needs to be hardware defined from scratch and very low design reusability can be implemented, this approach would also result in the longest development time when compared to the other options evaluated here.

3.2.2 Full CPU-Based Solution

By using the FPGA in bypass mode (i.e., IQ samples from the receiver's ADC are made available at the CPU, and IQ samples from the CPU are directly pushed to the transmitter's DACs), a fully functional SDA can be designed in GNU Radio. As a matter of fact, this was exactly the approach taken in LASSENA to prototype and build other functional SDA systems making exclusive use of GNU Radio. These are SDAs where the CPU of the host system performs the entire signal processing. This approach has shown to provide a lot of flexibility when compared to solutions based exclusively on FGPA's fabric impacting also positively the development time of the system as a whole. An additional advantage to the faster development time is that there is no need to deal with hardware (timing, latencies, etc.) issues. The design task is also enormously facilitated by the set of libraries provided by GNU Radio SDK.

3.3 Hardware Limitations

Once that Nutaq's PicoSDR2x2-E was selected as the project's development platform; an analysis of the hardware components that are part of the unit is needed to understand the limitations and capabilities of the system. Such an analysis is presented in the following sections.

3.3.1 Radio 420M FMC

The Radio420M FPGA Mezzanine Card (FMC) is based on the lime LMS6002D RF transceiver IC, which supports broadband coverage, as well as TDD (Time Division Duplex) and FDD (Frequency Division Duplex) full duplex modes of operation (NUTAQ, 2014) The RF transceiver instantaneous bandwidth covers from 1.5 to 28 MHz. which can be tuned within the frequency range from 300 MHz to 3.8 GHz. Two LMS6002D are included in the Radio420M FMC, which provides it with 2×2 MIMO operation. (Jalloul, 2014) Therefore, the system is limited to two systems working simultaneously, as only two transceivers are physically included in the platform front-end.

3.3.2 Perseus 6010

The Perseus AMC connects to a computing system a.k.a. 'Host' to exchange information and take advantage of its processing power. Just two interfacing options are provided to interconnect the 'Host' and Perseus systems:

1. 4xPCIe, which provides a dedicated connection to the embedded SAMC-514 system ('Host') in the default configuration;
2. 1x GigE, which allows a connection to an external 'Host'.

Note: Additionally the PCIe interface can be also used to interface to an external 'Host' running a Linux system but extra equipment is required (NUTAQ, 2014).

Since Perseus 6010 is built around the Virtex 6 FPGA family, it supports only the 4 following parts:

5. 6010: LX240T,
6. 6011: LX550T,
7. 6012: SX315T,
8. 6013: SX475T.

The newest Virtex 7 family parts and development tools like the Xilinx 'Vivado suite' are not supported. This means that the system is limited to use the older Xilinx 'ISE Design

Suite' which impedes the user to acces the latest development capabilities introduced by Xilinx.

Other important limitations addressed in (NUTAQ, 2015), include:

1. Up to 4 GB, 64-bit DDR3 SDRAM SODIMM;
2. 64MB NOR flash memory (16 bits) for FPGA images, MicroBlaze boot code and user code;
3. Mid-size AMC (allows component heights maxed at 11.65 to 14.01 mm);
4. Two GigE ports;
5. Serial RX/TX—Mini-B USB (UART).

3.3.3 SAMC-514

The SAMC-514 comes with an Intel 4C i7 processor, clocked at 2.1GHz, a 64GB SSD SATA drive and 8GB DDR2 SDRAM SODIMM. These are the three main factors that impact the overall system's response and set the limit to the applications that can be executed in it (NUTAQ, 2014).

All of the limitations here presented were assessed when designing the system´s behavior and defining its expected capabilities; their specifications give us a first glance of the maximum performance that can be attained by the system as a whole. The following section proposes an SDAR architecture based on the hardware presented in this chapter.

CHAPTER 4

THE PROPOSED SDAR ARCHITECTURE

Taking into account the considerations presented on the previous chapter, a suitable architecture needed to be implemented. After analyzing both of the solutions proposed in Chapter 2, a combined FPGA/CPU solution, was developed and implemented on the PicoSDR; the result is a design that takes the best from each of the two alternatives presented and mixes them in a functional way. This chapter explains the details of the implemented solution.

4.1 SDAR Architecture Details

To make the integration scheme as comprehensive as possible, instead of describing specific hardware into the FPGA for each of the desired SDAs in a given configuration, a cooperative scheme comprising the CPU and FPGA communicating through RTDEx has been sought. In such design data enters the FPGA from Radio420M's ADCs and is routed to the current user-selected channel(s) through RTDEx. Once in the CPU side, GNU Radio will take care of processing the data and will output it back to the RTDEx bus where the FPGA will make use of a series of demux blocks to route the signal to the right radio output. A conceptual representation of the system is presented in Figure 4.1.

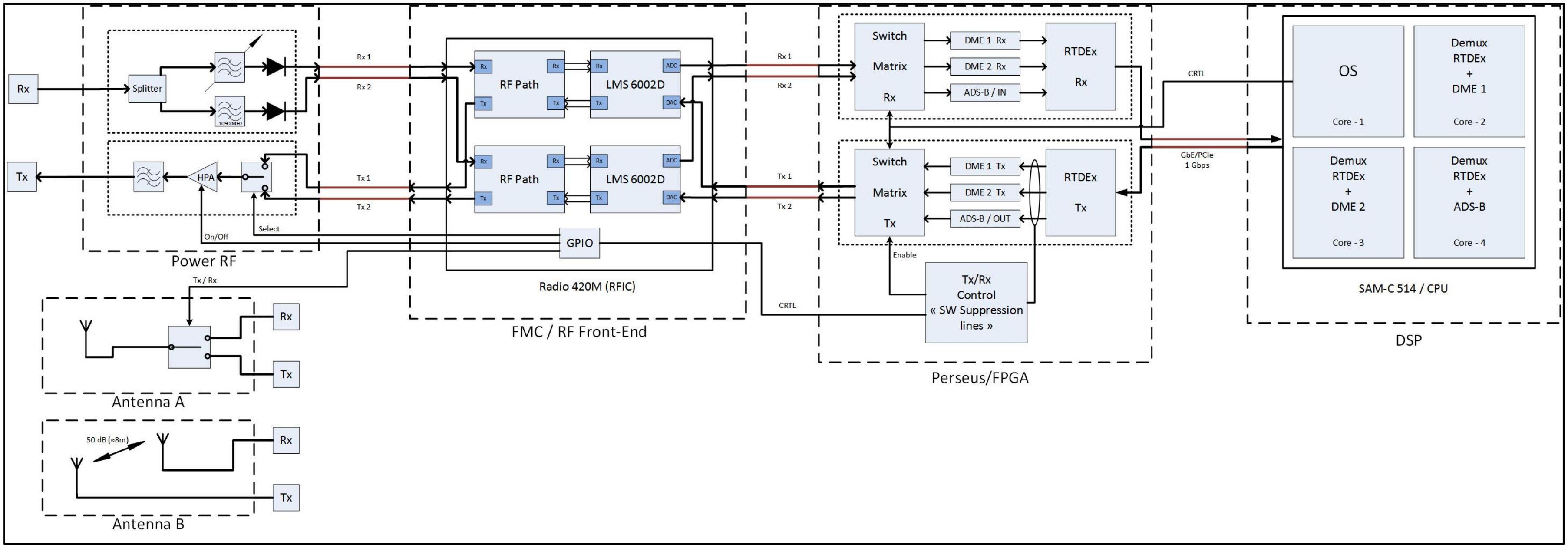


Figure 4.1 Conceptual System´s Representation

To achieve this tight cooperation between CPU and FPGA, and to allow the concurrent operation of SDA types I, II and III, Nutaq's 1Gbps-Ethernet-RTDEx interface is used. The ethernet implementation of Nutaq´s RTDEx was selected over its PCI version because the PCIe interface requires costly additional interconnection hardware to be used with an external (non-embedded) host CPU system. This hardware needs to be installed in each host system to be used for development. Therefore, to facilitate the development of the different SDAR modules by a team composed of several students, it is highly advisable that a common RTDEx interface is used in the design regardless whether the system operates the embedded CPU or an external host. This can only be achieved by using the Ethernet interface.

Using the PicoSDR2x2-E, there is a maximum of two Radio420 interfaces, which means two independent Tx/Rx transmission paths. To be able to have more than two SDA's concurrently working in the system, switching and control logic needs to be implemented at both the FPGA and CPU to perform the adequate signal routing to the desired radio interface. Note that only two systems will be operational at a given time due to the radio limitation, but these two systems can be chosen from a set of N systems concurrently running on the embedded CPU.

A minimum of three independent SDA's is needed in the final implementation to proof this project's concept. The original planning considered a TMS, an ADS-B and a DME system to be integrated together, however due to TMS signal timing constraints explained by (Jalloul, 2014), TMS was identified as a 'TYPE III' system and a full redesign was required. Since the redesign of the entire TMS application clearly falls out of the scope of this project, an extra DME system was selected to replace the missing TMS SDA[3].

The final SDA configuration is presented in Figure 4.2, it includes one ADS-B and two independent DME systems working concurrently.

[3] TMS FPGA implementation is left as future work. See 'Recommendations' section.

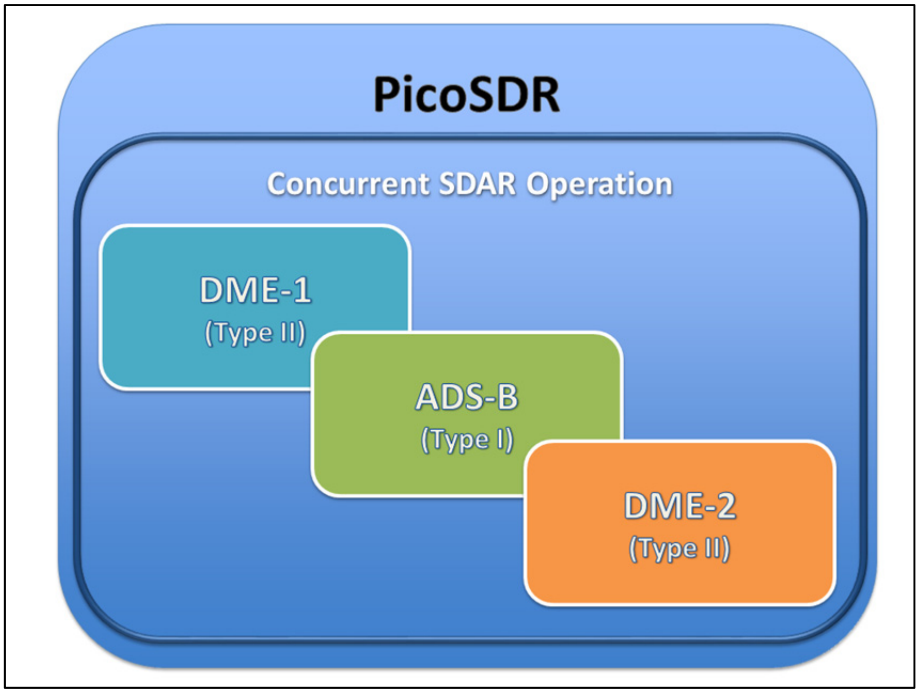


Figure 4.2 Final SDA configuration Implemented

4.1.1 RTDEx Considerations

To achieve the concurrent operation of the three systems, an exclusive RTDEx channel, out of the eight provided by Nutaq's implementation, was assigned to each of the three SDA systems to be integrated. This solution allows for up to eight concurrent systems, which is more than required at the current implementation time. In the future this limitation can be overcome by additionally multiplexing more than one system channel through a single RTDEx channel.

ADS-B SDAR executes its entire processing logic in the CPU side of the platform while DME balances its processing logic load between the CPU and FPGA making them TYPE I and TYPE II SDAs respectively, as defined in the introduction of section 4.

Since the required sampling rates of DME and ADSB SDARs are not the same, (1Msps for DME, 4Msps for ADSB) and because Radio420's sampling frequency cannot be changed dynamically once initialized; both Radio420 devices have to be configured to sample at the highest sampling frequency of the SDAs that were included in the design (4Msps for this case). Adequate decimation, interpolation and filtering have to be implemented in the system to cope for these differences in sampling rates between SDAs; further details are available in section 4.2.1.2.

Based on the fact that RTDEx uses a 1Gbps link, the available data rate was initially assumed to be close to the 888Mbps[4] however; the tests performed in Appendix VI show that the actual full throughput provided by the RTDEx medium is 553*2 Mbps.

For the selected configuration (Two DMEs + one ADSB), three concurrent RTDEx channels processing 4MSps each need to be in continuous operation. The whole system throughput is calculated below:

$$Throughput\ per\ channel = Sampling\ rate\ \cdot Sample\ Size$$

$$Throughput\ per\ channel = 4MSps\ \cdot 32Bits = 128Mbps \qquad (4.1)$$

$$Total\ througput = Througput\ per\ channel\ \cdot Number\ of\ channels$$

$$Total\ througput = 128\ \cdot 3 = 384\ Mbps \qquad (4.2)$$

Which represents 34.71% of the actual RTDEx limit per channel (553Mbps) found by the tests described in Appendix VI.

Additional throughput optimization can be attained by using Time Domain Multiplex (TDM) techniques and suitable interleave of all the three system channels through a single RTDEx channel.

[4] Due to network procol overhead. (Woligroski, 2009)

4.1.2 Tools for the Implementation

For the first part of the implementation, which is related to the entire processing executed at the CPU level, “GNU Radio” and “Nutaq’s GNU Radio plugin” were the main tools used during this phase. “GNU Radio Companion” was also used to create other prototypes that were lately abstracted to python code and defined exclusively there.

For the FPGA’s logic implementation, System Generator and Simulink were the tools of choice. Combined with Nutaq’s MBDK (Model Based Development Kit), they allowed access to specific PicoSDR peripherals and configuration parameters.

The Nutaq’s CLI (Command Line Interface) was indispensable to perform signal capture at fast rates, when DME signals from the IFR-6000 were captured and stored on PicoSDR’s DDR3 RAM and then replayed in Simulink for offline simulation purposes, more information regarding this data capture and replay is available in section 4.2.3.5.

4.1.3 FPGA/CPU Tasks Separation

To take full advantage of the processing power offered by the PicoSDR2X2-E, a separation of tasks was carefully planned between the i7 CPU and the Virtex-6 FPGA. The challenge was to maximize CPU’s digital signal processing (DSP) capability with GNU Radio by setting it free from the non-essential DSP tasks required by the system. The resulting task allocation is presented in Table 4.1.

Table 4.1 SDAR Final tasks to sub-system allocation

Final task to sub-system allocation			
	System's Tasks	System's Capability	
		i7	Virtex-6
		(CPU)	(FPGA)
DSP tasks	SDA Incoming signal conditioning	✔	
	SDA Signal processing	✔	✔
	SDA Outgoing signal Conditioning	✔	
Control tasks	Graphical user interface (GUI) presentation	✔	
	SDA selection and routing		✔
	Rx Automatic Gain Control (AGC)		✔
	Tx gain control	✔	

It can be observed that for the SDA configuration here presented (exclusive use of **Type I** and **Type II** applications), the control-related tasks are assigned to the FPGA while most of the signal processing stays in the CPU side.

It must be noticed than an effort was made to include the signal conditioning logic into the FPGA to free up more CPU resources. Nonetheless a multi-rate RTDEx implementation results out of this scheme and problems with RTDEx latency and sync parameters were observed when trying to concurrently execute systems with different sample rates.[5]

The three systems send data concurrently to the three open RTDEx channels, which will then be switched to/from the Radio interfaces by the FPGA's logic upon user request via custom FPGA registers accessible from the Nutaq's Python API.

[5] Decimation and interpolation are performed by GNU Radio's logic, it is left for future work to incorporate this signal conditioning at the FPGA level so that just the essential info can be sent to the CPU for processing. See 'Recommendations' section.

Figure 4.3 presents a simplified block diagram that shows the signal flow in the system. To be able to efficiently switch from a running SDA to another one at runtime, the control logic was implemented at the FPGA level, which allowed us to relief the CPU from the burden of the switching task; this is explained further in section 4.2.3.1.

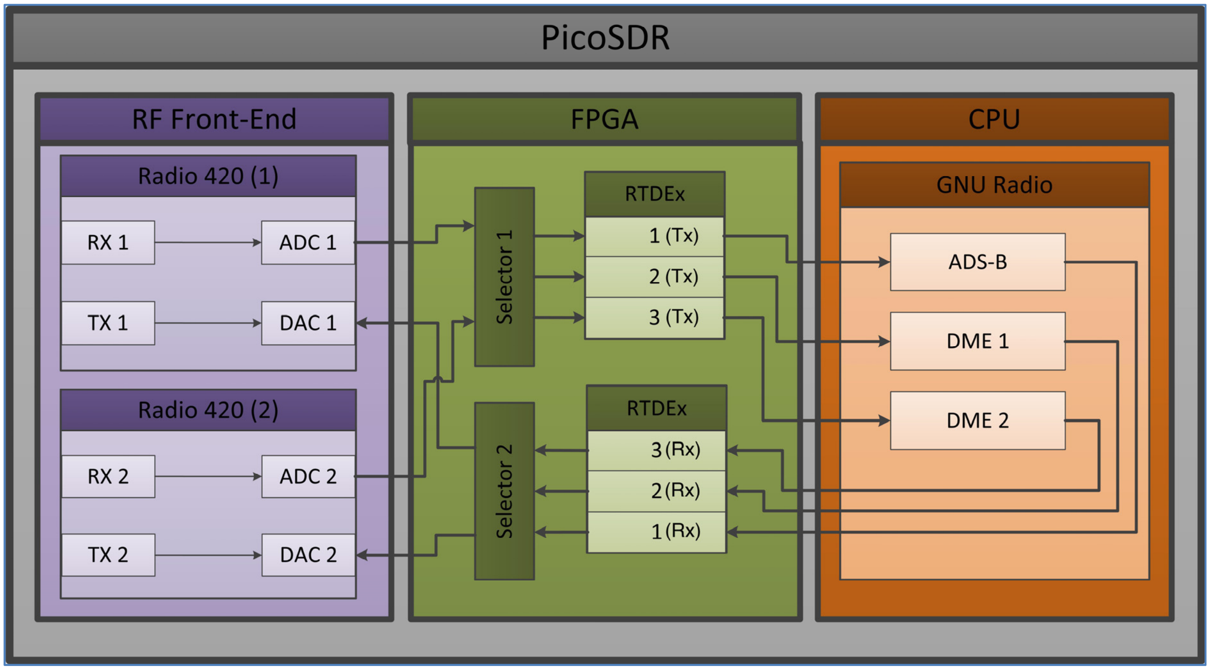


Figure 4.3 Simplified signal flow Diagram

4.2 Proposed SDAR Implementation

The implementation details are presented in this section; the specific modifications to Nutaq's libraries as well as the deviations from GNU Radio normal operation modes, task scheduling and core affinity assignation are described. In the second part a closer look to the FPGA design is presented, section 4.2.3.5 clarifies the way in which the design was simulated in Simulink, which was fundamental to test the AGC and selector's functionality before loading them into the FPGA.

4.2.1 GNU Radio Implementation

The GNU Radio suite provides a block oriented platform to perform signal processing in a multi staged fashion, where each specialized block applies only the desired modifications to the signal received at its input and passes it to the next block in the chain. As explained with more detail in section 4.2.2, the entire signal flowchart is embedded into the outermost block in the flow diagram hierarchy[6], which GNU Radio refers to as 'Top Block'. The 'Top Block' class contains (among other information) the information needed by the system to create the required connections, both inter-block and with the specific hardware platform from where the signals are read and written (PCIe or Ethernet interfaces on the PicoSDR).

At execution time, GNU Radio will create one main thread (in the 'main' function call) for the 'Top Block', which in turn will be responsible of creating a sub-thread for each of the blocks defined in the signal flow chart and synchronizing the signal processing of the GNU Radio application defined in that 'Top Block'.

In this way each 'Top Block' contains a 'main function' call, which creates its own thread and is in-charge of processing the entire logic defined in that specific 'Top Block'.

While the described strategy normally works well for a broad range of applications and is the GNU Radio default operation mode, we will see in the following section, that due to the concurrent SDA execution nature of this project, it proved to lack flexibility to accommodate the future modularity aggregation feature desired in the system.

[6] Using GNU Radio v3.7.3 (Supported by NUTAQ at the time of development).

4.2.1.1 **Problems Found with GNU Radio**

A good number of difficulties were faced during the GNU Radio implementation phase for this project. This sub-section exposes the more significant problems found and describes the way in which they were finally solved, circumvented and/or avoided.

4.2.1.1.1 **GNU Radio Companion**

GNU Radio Companion (GRC) "is a graphical tool for creating signal flow graphs and generating flow-graph source code. It facilitates the digital signal processing generation by abstracting the application and providing a block-oriented design interface that allows the user to describe the desired signal flow for the application of interest, shortening in this way the development time." (GNURadio, 2015)

For these reasons 'GNU Radio Companion' with Nutaq's plugin for GNU Radio were used for the initial GNU Radio's design phase of this project but the practicality offered by the graphical user interface came with some limitations in flexibility. This became obvious when attempting a calibration routine to determine latency of the DME submodule in the SDAR. This latency is measured as the propagation time of the signal from the moment it is generated by GNU Radio engine to the moment it is received at PicoSDR Rx antenna. In order for the system to be able to measure this time in a real life scenario without impeding the surrounding navigation equipment, a change in the transmission channel is required. Nutaq's plugin for GNU Radio supports a runtime change of Radio420's parameters from GRC, however, checking for DME calibration procedure to be completed requires polling a function at a specific rate, which cannot be changed at runtime. As a consequence, the check for DME calibration would have been running for the whole operation of the system. For this reason, it was chosen to implement the frequency change in the python language inside the code of the DME 'Top Block' class.

Because of this limitation the GRC tool was left aside and from this point onwards the entire GNU Radio development effort took place directly in the Python programming language. This proved to be an advantageous decision later, when working on more complex features of the system such as: SDA concurrency, Automatic Gain Control (AGC) at the receiver end, and a graphical user interface (GUI) for controlling the system. In all of these examples, trying to adapt GRC to perform the desired functionalities would have resulted in a cumbersome implementation and more time consuming than developing the desired behaviour directly in the Python language.

The GRC tool proved to be very useful though as an aid to quick-prototype some of the functionalities of the system that were previously included and enhanced in the Python program.

4.2.1.1.2 **Multiple SDAs - Single PicoSDR**

As expressed in section 4.2.1, the first problem to overcome and probably the most important one for the GNU Radio implementation section was the intrinsic one-to-one ‘Top Block’ to ‘main function’ relationship required by GNU Radio Companion.

Having each independent SDA defined into a GNU Radio ‘Top Block’, which contains the definition for the hardware interfaces that it uses on PicoSDR, the challenge was to avoid interface redefinition (which would result in a runtime error) and to be able to instantiate several SDAs to be executed concurrently by a single program. Since the hardware platform from where all of our applications read their data from and write to, is a single PicoSDR device; a way of defining the shared hardware resource from more than one GNU Radio ‘Top Block’ was needed and not provided by the default GNU Radio implementation.

If the entire definition of the system and its SDAs would need to be done in a single ‘Top Block’, the modularity of the system would be compromised because the developer would be forced to modify and thoroughly review the whole system design each time a modification were made to any part of the system (even a single SDA for instance).

This limitation was resolved by creating a multi 'Top Block' hierarchical architecture where a main 'Top Block' is used to create as many instances of the desired SDAs as needed. Each SDA is made available to the main 'Top Block' as an imported 'class' from its respective SDA Python module so that the modular property of the system design is respected. From this point onwards the main 'Top Block' is able to launch and control each SDA in an independent thread that can be handled without interfering with any other component of the system. A diagram clarifying this work-around is provided in Figure 4.7

4.2.1.1 **Task scheduling and Core Assignation**

Traditionally the Linux operating system used "O(1) scheduler" to handle the processes that are assigned to it in a "constant time"; however since kernel v2.6.23 it has been replaced by the "Completely Fair Scheduler (CFS)[7]" which has shown to have a better efficiency without significantly compromising performance (C. Wong, 2008). In addition to the Linux scheduler and because of the specialized duties that are performed by GNU-Radio, it implements its own schedulers: TPB (Thread-Per-Block) and STS (Single-Threaded-Scheduler) [8], which attempt to optimize throughput (The Free & Open Software Radio Ecosystem, 2016); contrastingly, the default Linux scheduler evenly distributes hardware interrupts among the available number of cores to obtain a better performance for the general purpose case. As a side effect, unexpected hardware interrupts in a core that performs signal processing result in a performance decrease for the application. To solve this problem, the measures presented in Appendix I were taken to isolate the cores that will perform the signal processing. Along with the core isolation, GNU-Radio core affinity was used to allow a better control over the

[7] CFS has an O (Log N) response time being no longer a "constant time" scheduler.

[8] GNU Radio allows the user to choose between the Linux OS default scheduler, TPB and STS for each project.

CPU cores that execute each of the SDAs defined in an SDAR. Core affinity assignation is explained in section 4.2.3.6.4.

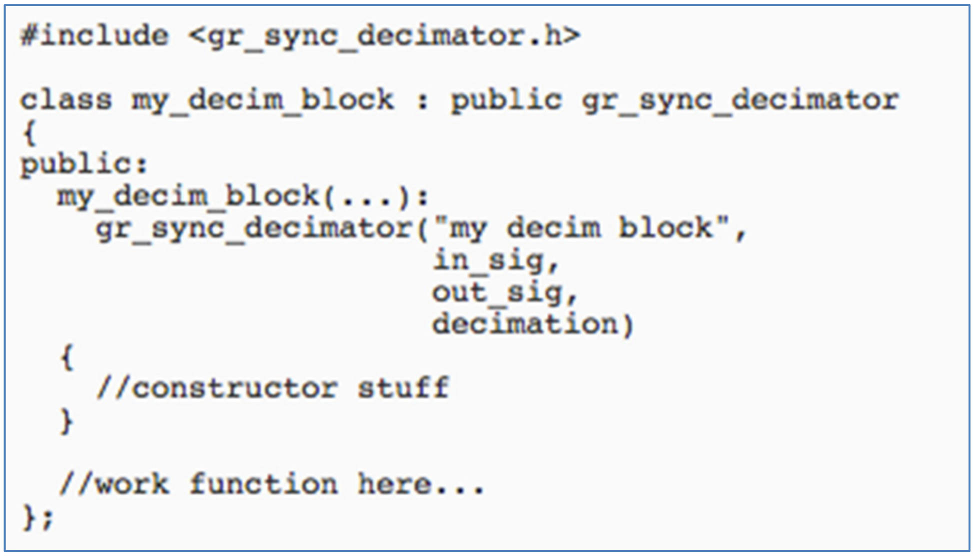
4.2.1.2 **Decimation and Interpolation**

To cope for different sampling rates of the applications that may conform a specific configuration of the SDAR architecture; decimation and/or interpolation are used accordingly by means of GNU Radio. The ‘Decimator’ and ‘Interpolator’ ‘Blocks’ are used for this purpose.

A decimation block is a fixed rate-type block in which the number of input items is a fixed multiple of the number of output items. In this block type the ‘decimation factor’ is passed as an argument to the block ‘constructor’. The number of input items is given by equation 4.3.

$$\mathit{number\ of\ input\ items} = \mathit{number\ of\ output\ items} \cdot \mathit{decimation\ factor} \qquad (\ 4.3\ )$$

Figure 4.4 shows an example of C++ code for a decimator block.



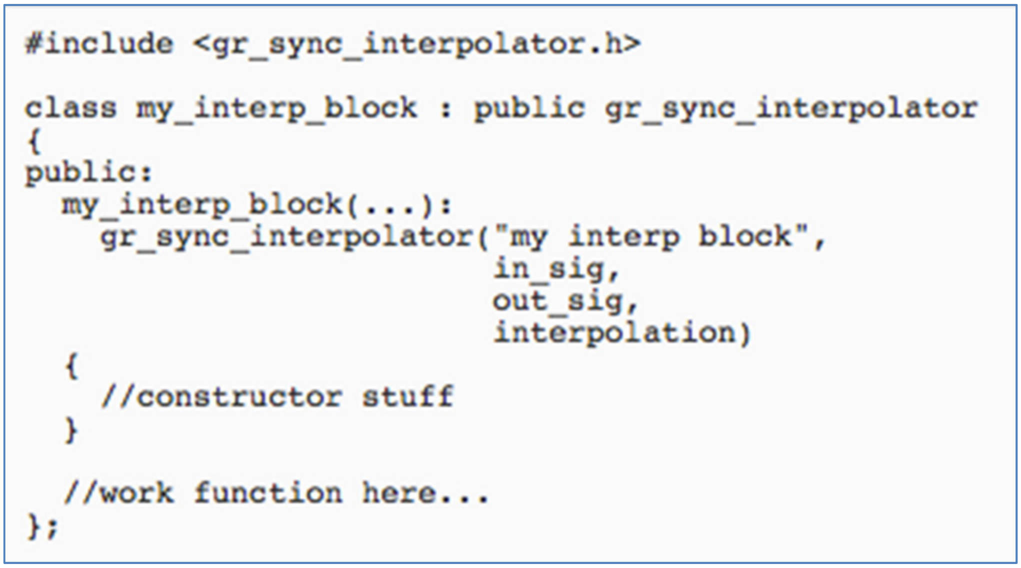
```
#include <gr_sync_decimator.h>

class my_decim_block : public gr_sync_decimator
{
public:
  my_decim_block(...):
    gr_sync_decimator("my decim block",
                      in_sig,
                      out_sig,
                      decimation)
  {
    //constructor stuff
  }

  //work function here...
};
```

Figure 4.4 C++ Decimator Block Example
(GNU Radio, 2013)

The interpolation block is the counterpart of the ‘Decimation Block’ in which the number of output items is a fixed multiple of the number of input items. In this case the block ‘constructor’ takes the interpolation factor as a parameter. The number of input items is then given by equation 4.4.

$$number\ of\ input\ items = number\ of\ output\ items\ /\ interpolation\ factor \qquad (\ 4.4\ )$$

Figure 4.5 shows an example of C++ code for an interpolation block.

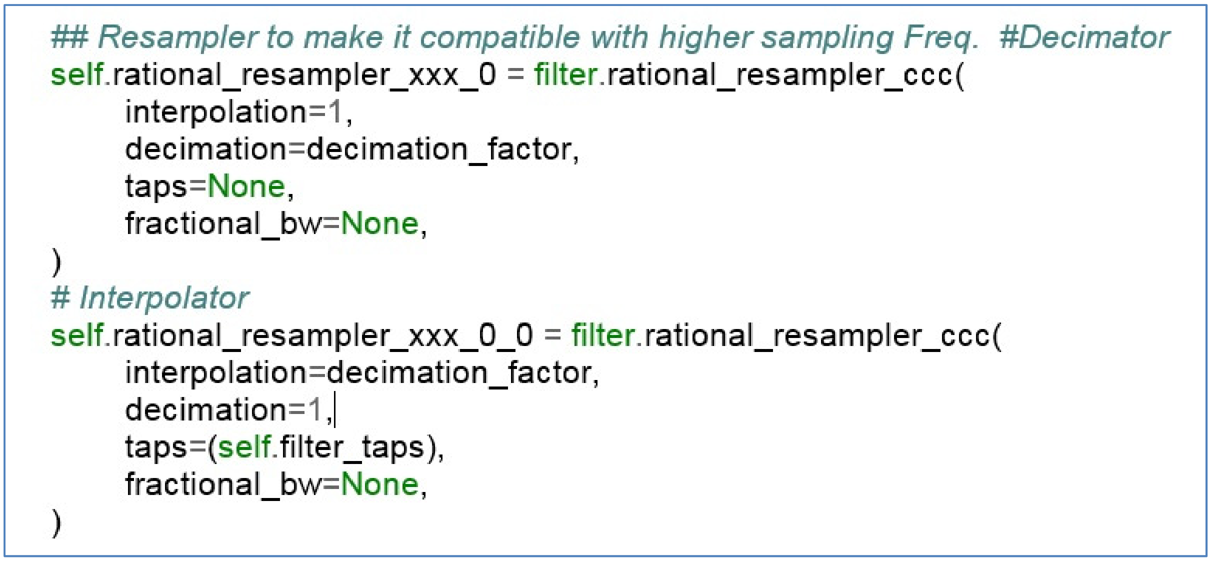


```
#include <gr_sync_interpolator.h>

class my_interp_block : public gr_sync_interpolator
{
public:
  my_interp_block(...):
    gr_sync_interpolator("my interp block",
                         in_sig,
                         out_sig,
                         interpolation)
  {
    //constructor stuff
  }

  //work function here...
};
```

Figure 4.5 C++ Interpolator Block Example
(GNU Radio, 2013)

For the configuration presented in Figure 4.2, the ADS-B requires the highest sampling frequency compared to the other (two DME) SDAs included in this configuration. 4Msps is thereafter set to be the overall sampling rate of the system.

The DME SDA is conceived to work with a sampling rate of 1Msps hence requiring a 4 to 1 decimator for its correct operation Figure 4.6 shows a code snippet with the implemented decimator and interpolator blocks.



```
## Resampler to make it compatible with higher sampling Freq.  #Decimator
self.rational_resampler_xxx_0 = filter.rational_resampler_ccc(
        interpolation=1,
        decimation=decimation_factor,
        taps=None,
        fractional_bw=None,
)
# Interpolator
self.rational_resampler_xxx_0_0 = filter.rational_resampler_ccc(
        interpolation=decimation_factor,
        decimation=1,
        taps=(self.filter_taps),
        fractional_bw=None,
)
```

Figure 4.6 Decimator, Interpolator and Filter Implementation

It is the responsibility of the designer to verify that all the SDA selected in a specific configuration are compatible (share a single sampling rate) after applying correct decimation and/or interpolation.

The utilization of this technique allows for a transparent operation between SDAs with different sampling rates as it is demonstrated in Chapter 6.

4.2.1.3 **The Automatic Gain Control**

To make better use of the three configurable amplifiers provided in the PicoSDR and permit for dynamic sensitivity adjustment based on the signal power at the reception antenna, an Automatic Gain Control feature for the DME SDA was added.

It uses a total dynamic range of 61 dB in reception and prevents signal saturation of the PicoSDR receivers when the transmitting source is either too close to the Rx antenna or too powerful.

This feature was originally implemented in Python as an independent thread of the DME program and proved acceptable performance when a single DME SDA was executed in the PicoSDR however, for the integration of multiple SDAs into a single system, improvements to the original AGC were required since the power of the processor could be better used if allocated to other SDAs instead. These improvements are explained in section 4.2.3.4.

4.2.2 The GNU Radio Result

The final GNU Radio Python application is divided in three files (python modules):

1. multi_sda.py,
2. lib_dme.py,
3. lib_adsb.py.

The code contained in ‘multi_sda’ is responsible for the general application configuration and control. It implements the GNU Radio ‘Top Block’ that defines the desired system SDA configuration.

To account for the greatest deal of flexibility, the SDAs to be instantiated are made available to ‘multi_sda´s’ ‘Top Block’ as independent imported GNU Radio applications (each one of them containing its own ‘Top Block’). To reinforce the system’s modularity a naming convention was adopted where each of the SDA python modules is prefixed by the word ‘lib’.

As explained in section 4.2.1.1.2, a multi ‘Top Block’ hierarchical structure is derived from the described implementation where the control of the system is held by multi_sda’s ‘Top Block’. The child threads retain control of their respective application and are in turn just accountable to the ‘Top Block’ that created them. In this way the SDAs can still take full advantage of GNU Radio libraries’ functionality. Figure 4.7 depicts the system’s ‘Top Blocks’ structure and lists the responsibilities assigned to each of the created threads.

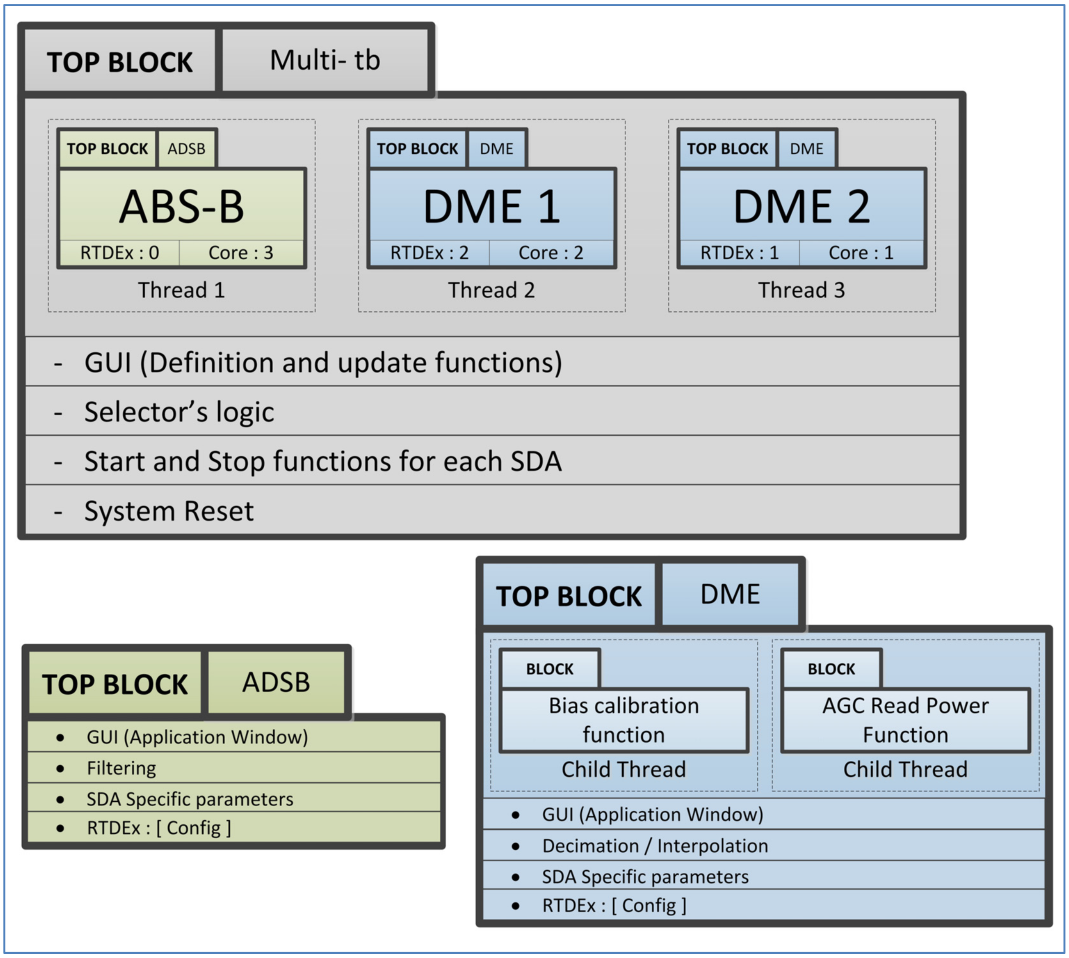
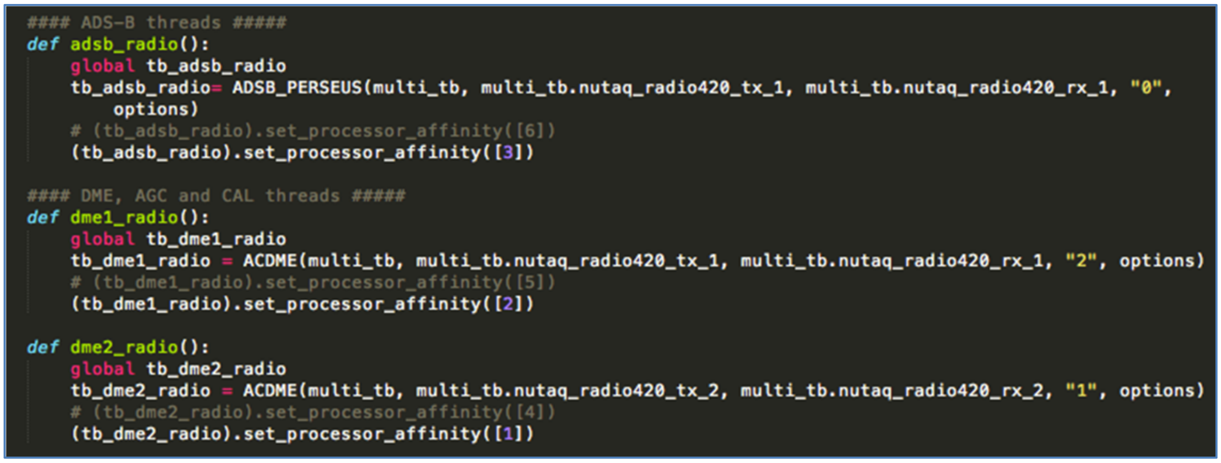


Figure 4.7 GNU Radio Top Blocks Division

All of the SDAs to be instantiated need to be imported into the 'multi_sda' module before they can be used, once there, a thread will be created for each of them, these threads will be under complete control of 'multi_sda's 'Top Block' and let the designer to configure:

1. The RTDEx channel number to be used by the SDA,
2. The radio card where the SDA will be initially started,
3. The thread's core affinity.

Figure 4.8 shows a code snippet where the definition and configuration of the instantiated SDAs are performed.



```
#### ADS-B threads #####
def adsb_radio():
    global tb_adsb_radio
    tb_adsb_radio= ADSB_PERSEUS(multi_tb, multi_tb.nutaq_radio420_tx_1, multi_tb.nutaq_radio420_rx_1, "0",
        options)
    # (tb_adsb_radio).set_processor_affinity([6])
    (tb_adsb_radio).set_processor_affinity([3])

#### DME, AGC and CAL threads #####
def dme1_radio():
    global tb_dme1_radio
    tb_dme1_radio = ACDME(multi_tb, multi_tb.nutaq_radio420_tx_1, multi_tb.nutaq_radio420_rx_1, "2", options)
    # (tb_dme1_radio).set_processor_affinity([5])
    (tb_dme1_radio).set_processor_affinity([2])

def dme2_radio():
    global tb_dme2_radio
    tb_dme2_radio = ACDME(multi_tb, multi_tb.nutaq_radio420_tx_2, multi_tb.nutaq_radio420_rx_2, "1", options)
    # (tb_dme2_radio).set_processor_affinity([4])
    (tb_dme2_radio).set_processor_affinity([1])
```

Figure 4.8 SDAs thread creation and configuration

In this way the final system exerts a great deal of control while providing the designer with the desired flexibility and modularity that were sought since the initial stages of the project.

4.2.3 FPGA Implementation

As previously explained in section 4.1.3 most of the control and signal routing tasks are assigned to the FPGA, to be able to perform such tasks, an automatic two-way 4-bit system selector was designed to route the signals from Radio420's ADCs to the RTDEx bus and back to the Radio420's DACs. The details of this implementation are presented below.

4.2.3.1 System's FPGA Selector's Implementation

By using a single FPGA custom register, which is readily accessible from GNU Radio thanks to Nutaq's GNU Radio Plugin, an asynchronous SDA selection can be invoked from a GNU Radio Python program. For the configuration here presented, three SDAs needed to be distinguished, since the three of them are continuously and concurrently executed by the CPU, a fourth routing path that disconnects the RTDEx channels from the radios was also created. The result is a 2 to 4 and 4 to 2 system selector that based on the user input at ‘Custom register 28’ automatically configures the selected SDA input to its corresponding output.

A group of 4 ‘selector bits’ is responsible of the system’s routing configuration. Figure 4.9 shows the logic that selects the radio and routes its incoming signal to any of the 4 available paths (3 for SDAs and one path for the disconnected one).

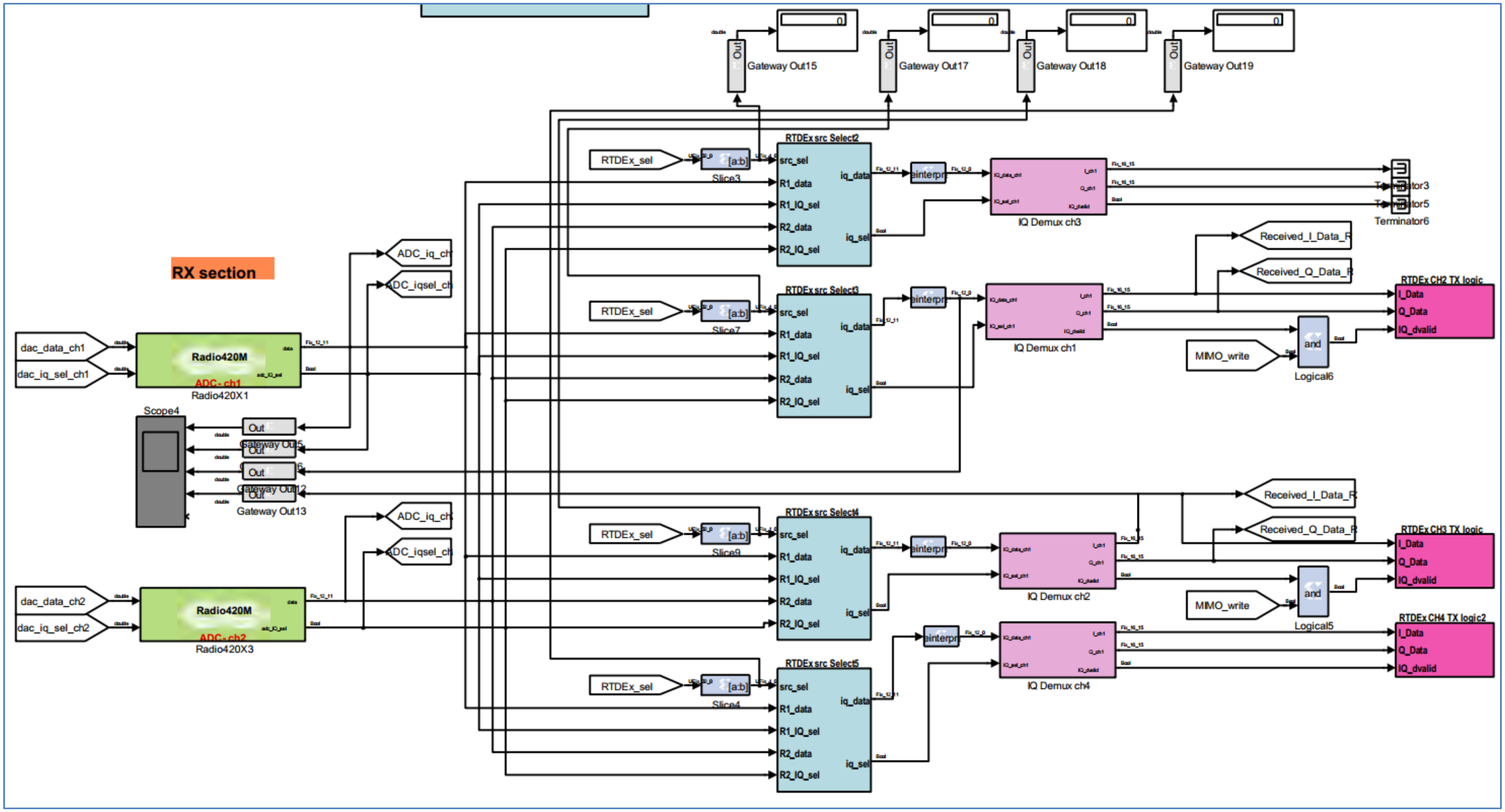


Figure 4.9 Radio Selector Implementation

4.2.3.2 **Radio420 to RTDEx**

For the Radio420-to-RTDEx direction, four identical 2 to 1 mux select blocks were implemented. They allow to route the signals acquired from the Radio420's ADC to the appropriate RTDEx channel based on the user's input. Register 28 input is 'sliced' in four bits, which allow selecting radio number 1 or 2 as a source for each available RTDEx channel. Since the four blocks are identical, just 'source selector number 4' is presented in Figure 4.10.

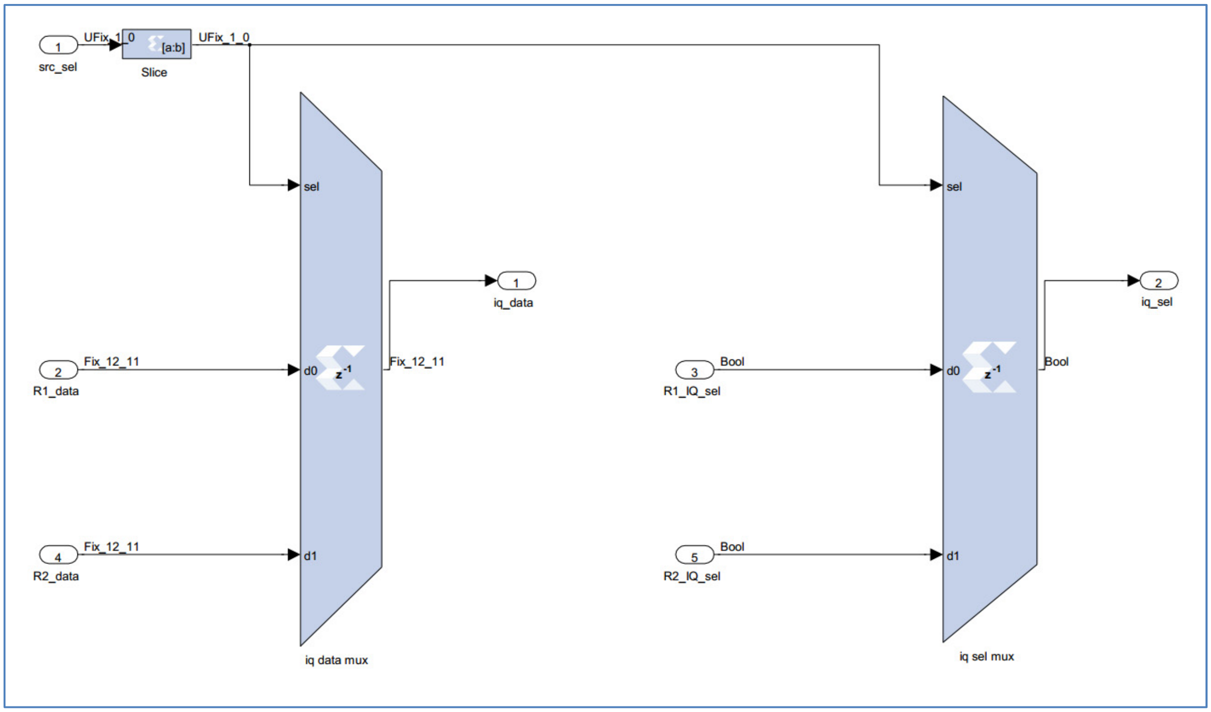


Figure 4.10 RTDEx src Select 4

4.2.3.3 **RTDEx to Radio420**

In the case of the RTDEx-to-Radio420 direction, two identical 4 to 1 mux select blocks were implemented. They allow routing the signals coming from the four available RTDEx channels to the desired Radio420 card based on the user's input. Register 28 value is 'sliced' in two halves (two bits each), which allow selecting radio number 1 or 2 as a source for each available RTDEx channel. Since the two blocks are identical, just 'source selector number 1' is presented in Figure 4.11.[9]

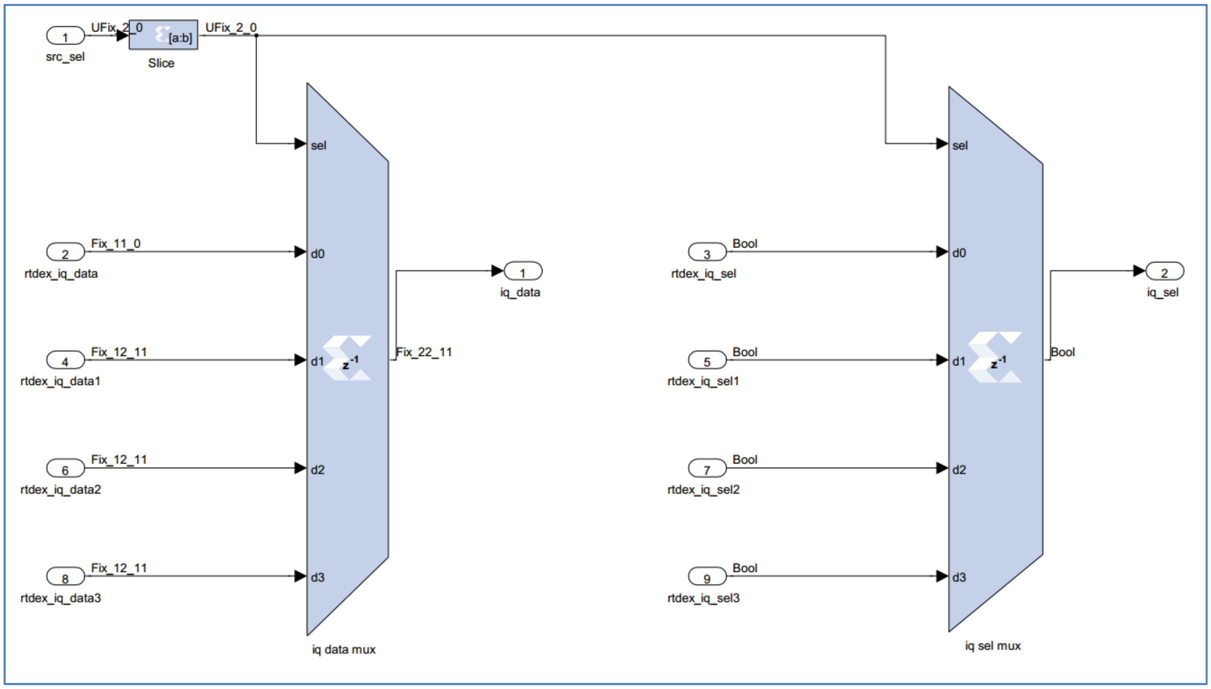
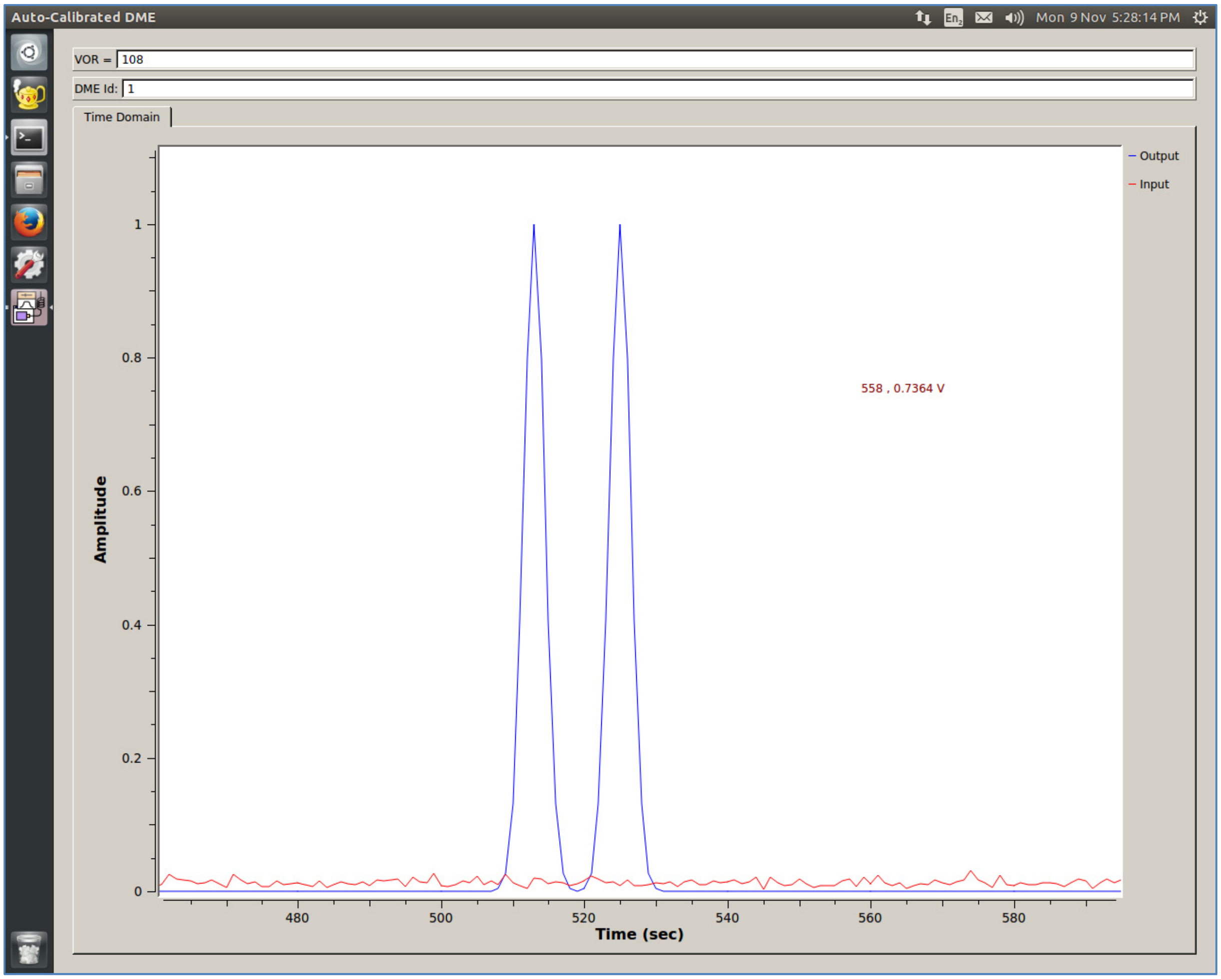


Figure 4.11 RTDEx src Select 1

4.2.3.4 **Improved Automatic Gain Control**

To further reduce the processor time used by the DME SDA, a final improvement was performed to DME's AGC; it consists of transferring all the signal power calculation logic to the FPGA. Due to the DME signal characteristics, we needed to calculate the peak-power of the signal instead of the more conventional average-power calculation performed by other AGCs. In our system with every valid pulse received, the AGC will re-adjust the input gain to keep the peak power level at 75% of the amplifier's saturation level. Figure 4.11 shows a captured mode X DME pulse, it can be observed that the pulse separation is 12µs.

[9] The rest of the modules for sections 4.2.3.2 and 4.2.3.3 are included in Appendix V.


VOR = 108

DME Id: 1

Figure 4.12 DME characteristic pulse (Mode X)

By making use of Nutaq's designed average power AGC presented in ¨An FPGA-based AGC Algorithm Using System Generator¨ (NUTAQ, 2015) and by modifying it to calculate the signal's peak power, a fully functional peak-power AGC was conceived.

Figure 4.13 presents the power calculation flowchart for the Rx path of one of the radios. This scheme is duplicated to calculate the signal's peak power on the second radio.

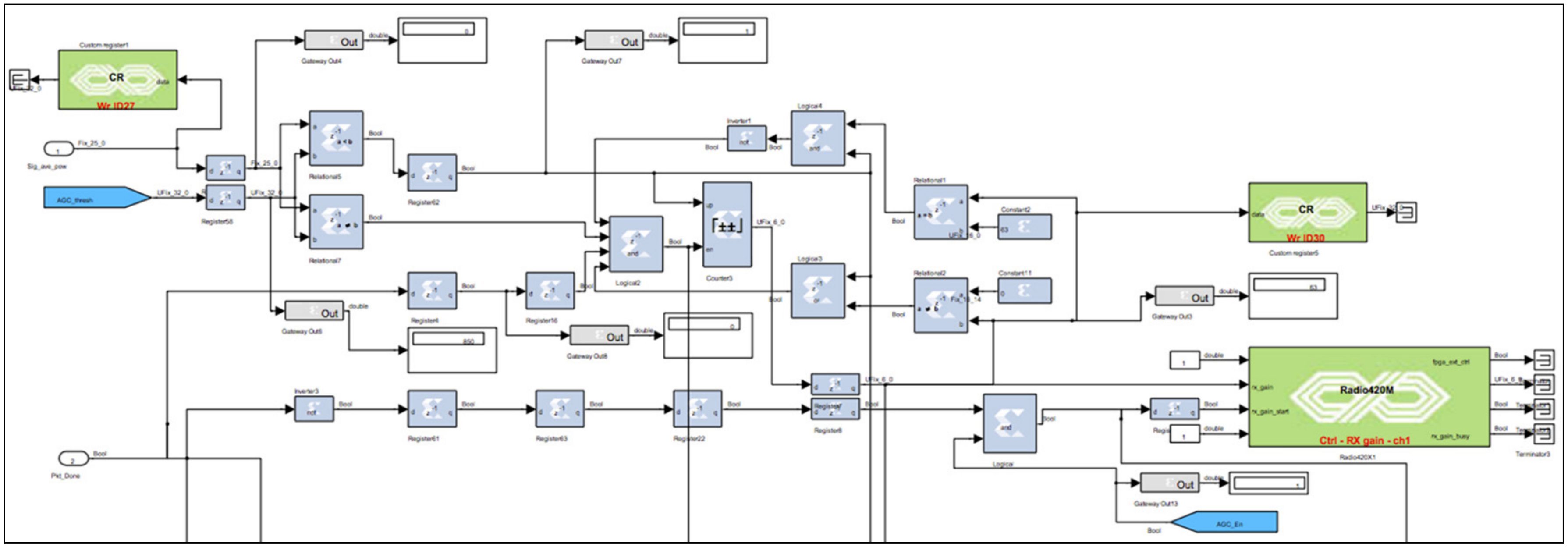


Figure 4.13 Received Power & AGC for Radio 1

4.2.3.5 **FPGA’s Logic Simulation in Simulink**

To verify the proper operation of the designed AGC offline simulation was performed in Simulink. Two groups of pulses from IFR-6000 were taken at different transmission powers to confirm that the right peak power was held for every DME pulse received. A capture of Simulink scopes in the offline simulation mode is presented in Figure 4.14.

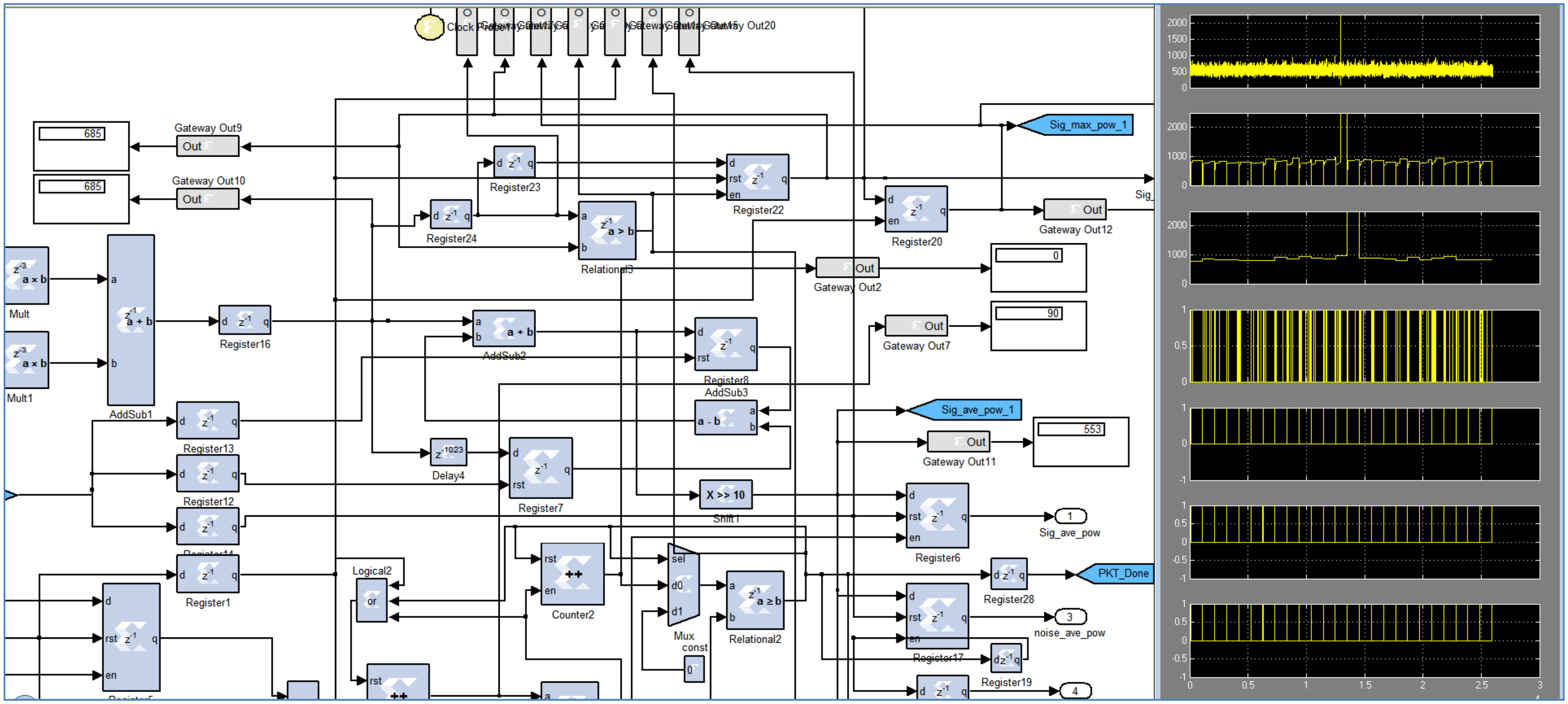


Figure 4.14 AGC’s Peak Power Calculation in Simulink

The system selector functionality was also tested during the offline simulation to confirm the expected behaviour. Figure 4.15 shows probes of the selector's inputs while inserting test values[10] to the user's accessible selector register (FPGA's custom register 28).

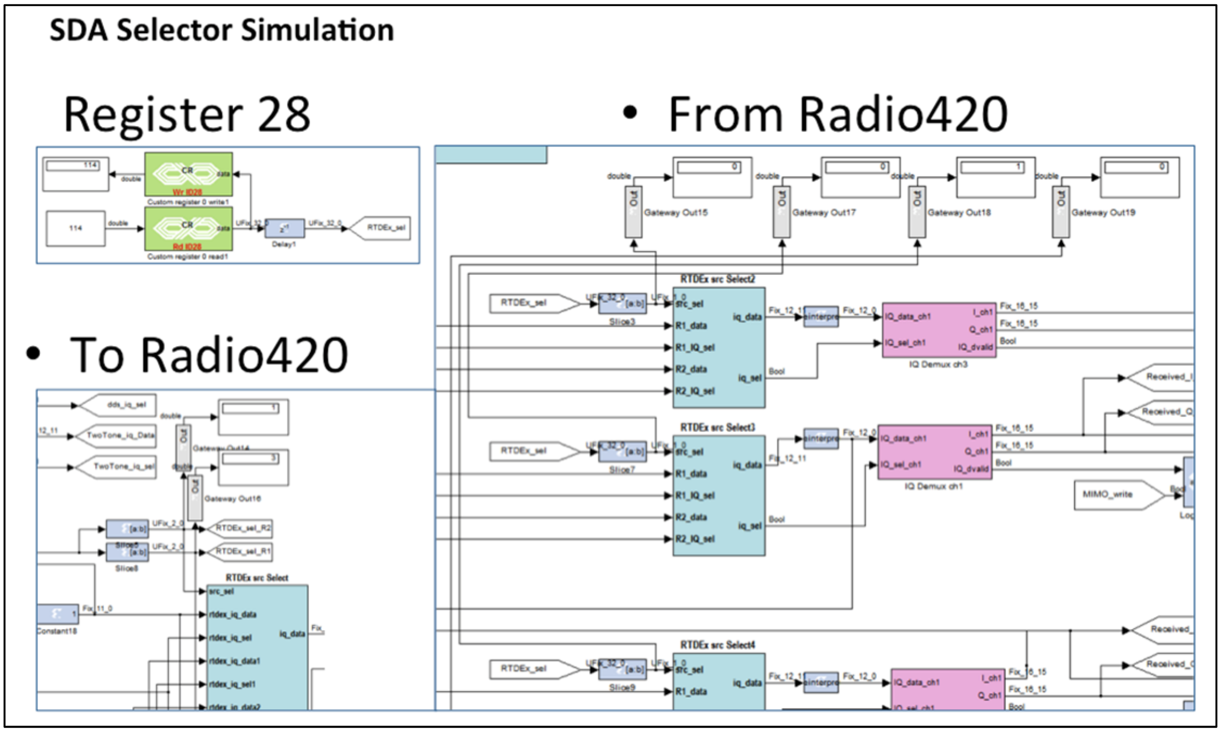


Figure 4.15 System Selector Probes for Simulation

4.2.3.6 **Problems Found**

The problems related to the implementation of the FPGA section of the project are presented in this section. A brief description of the problem and the way in which each of them was solved is described in the following lines.

[10] Appendix VII contains the truth tables of the SDAs selector where all the possible values are presented.

4.2.3.6.1 **Clock drifting**

A drifting in the measurements of one of the DME SDAs was observed while performing the test of the integrated SDA system, after investigating the problem it was found that each Radio420 card was configured to make use of its own oscillator to sample the input, as a result of the physical differences between the oscillators, one of the measurements drifted apart from the other. To resolve this problem one of the Radio420's ADC was configured to take the clock source from the oscillator of the other Radio420 card as an external reference. Figure 4.16 shows the way in which the Radio420 clock references are connected to solve the problem.

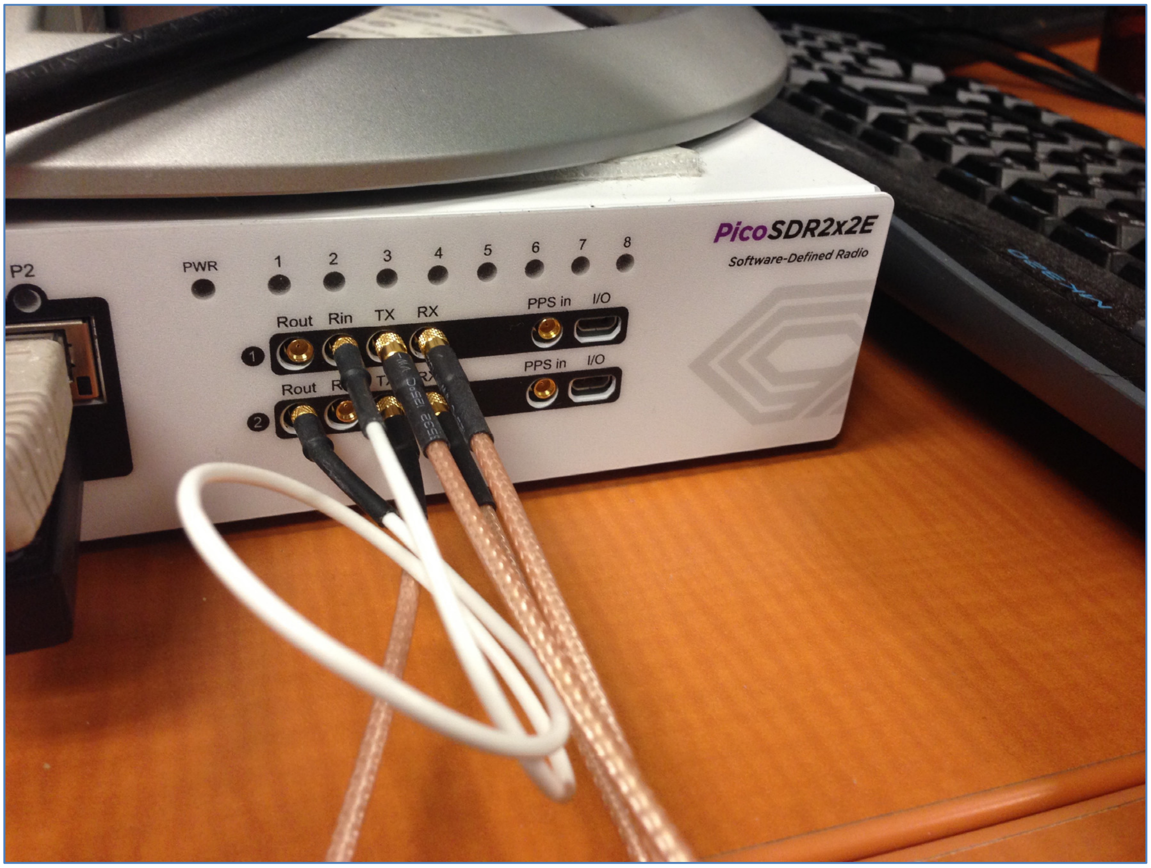


Figure 4.16 Radio 1 Reference-In Connected to Radio 2 Reference-Out (White cable)

4.2.3.6.2 **Dealing With Latency Issues**

As mentioned in section 4.2.1.1.1 latency measurement is of outmost importance for SDARs that rely on the propagation time of the signal to perform its measurements. The measured latency needs to be subtracted from the total signal's propagation time for the system to be able to provide the correct measured distance. It is equally important that once measured, this latency remains constant (or with minimal variation) since any variation in time will adversely affect the measured distance. For the DME SDA when sampling at 1MSps, a difference of one microsecond (1 sample) translates into an error of ≈150 meters.

4.2.3.6.3 **RTDEx Limitations**

When performing the first tests of the integrated DME SDA, unstable distance measurements were observed. Further investigation showed that this instability was caused by lost/dropped samples at the Rx side of the RTDEx bus. Figure 4.18 shows two pulses with constant separation being sent from a signal generator through RTDEx bus without performing any additional processing; Figure 4.17 depicts the test setup.

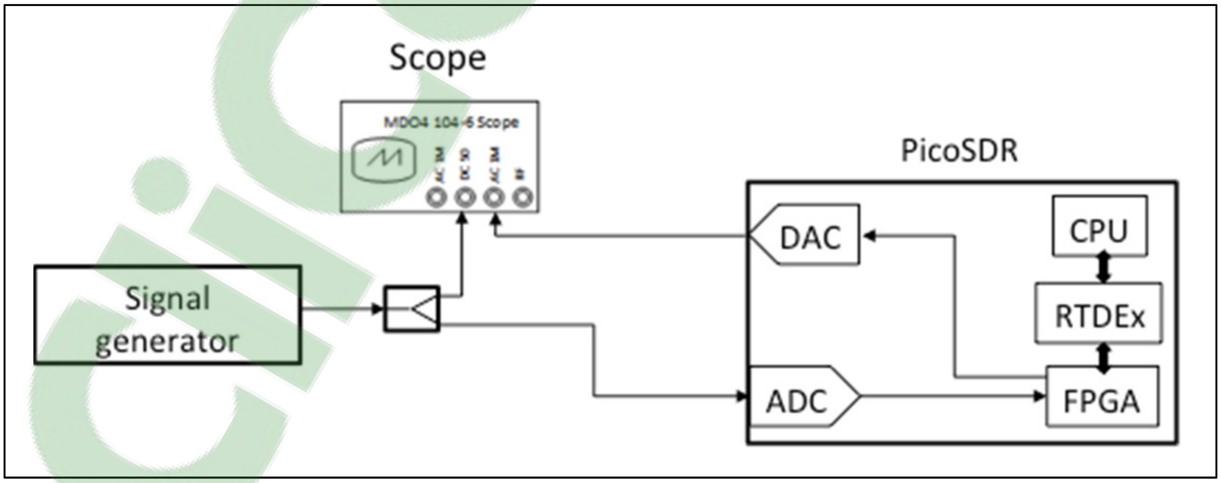


Figure 4.17 RTDEx Stability Measurements Setup

It can be observed in Figure 4.18 that the second pulse is at different positions at t0 and t1. This needed to be fixed for the DME SDA measurements to be stable and reliable.

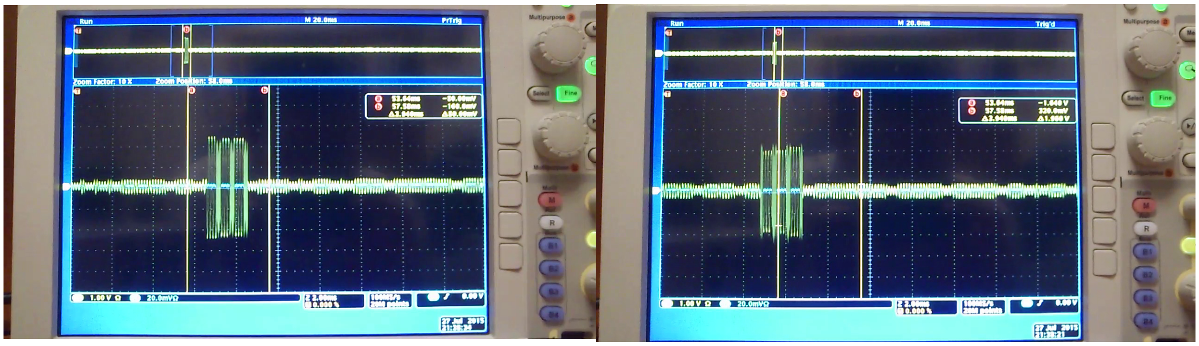


Figure 4.18 RTDEx instability measurement @ 10MSPs – t=t0 (left), t=t1 (right)

The problem is indeed caused by the RTDEx Rx buffer being overflown when the CPU is not able to consume all the samples that are put in the reception buffer by the FPGA; after careful analysis, it was determined that the CPU being interrupted at unpredictable times was triggering this issue, the problem was solved by performing core isolation at the CPU level. The CPU isolation techniques applied are included in Appendix I.

4.2.3.6.4 **GNU Radio thread to core assignation**

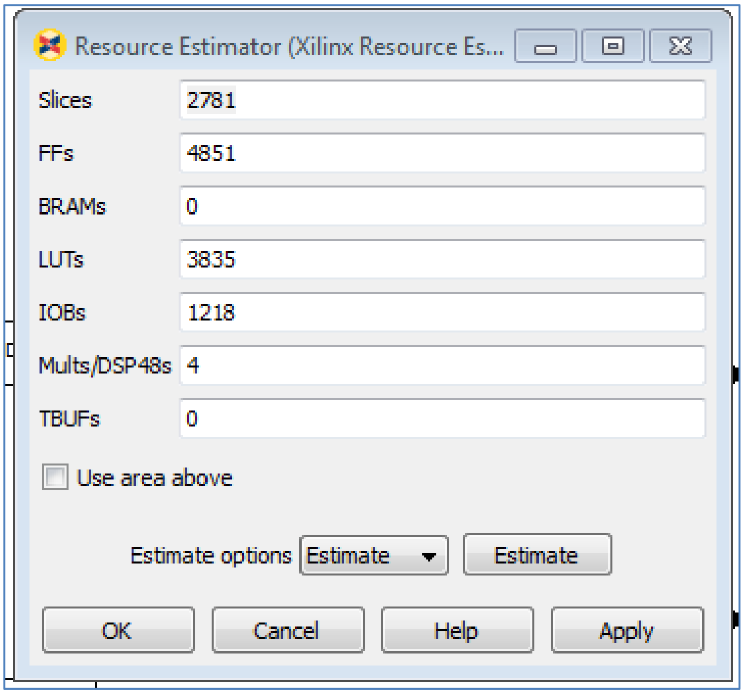
With all the IRQs mapped to 'core 0' the next step was to guarantee that the signal processing threads in GNU Radio would not get also assigned to this core. Proper thread separation and GNU Radio core assignation routines that make use of the GNU-Radio "set_processor_affinity" method were used in the Python program to achieve this.

Through the "set_processor affinity" method the designer is able to pin a GNU-Radio Block task to a core, a group of cores or to be set back to use the standard Linux kernel scheduler. This mechanism allowed to further control the assignement of tasks, making it possible to make use of an entire uninterrupted core to execute a single SDA; which greately simplifies the design, control and performance of the whole architecture.

4.2.4 FPGA Implementation Results

With the entire logic designed in system generator an estimation of the FPGA resources utilization was performed. The results are shown in Figure 4.19, and can be compared to Figure 4.20 where the total number of resources available in the Virtex 6 FPGA is presented (XC6VSX475T was the FPGA part used for this specific implementation).

For instance it can be observed that 2781 out of 74400 slices were used for this specific project, which corresponds to the 3.73% of the FGPA total slices.



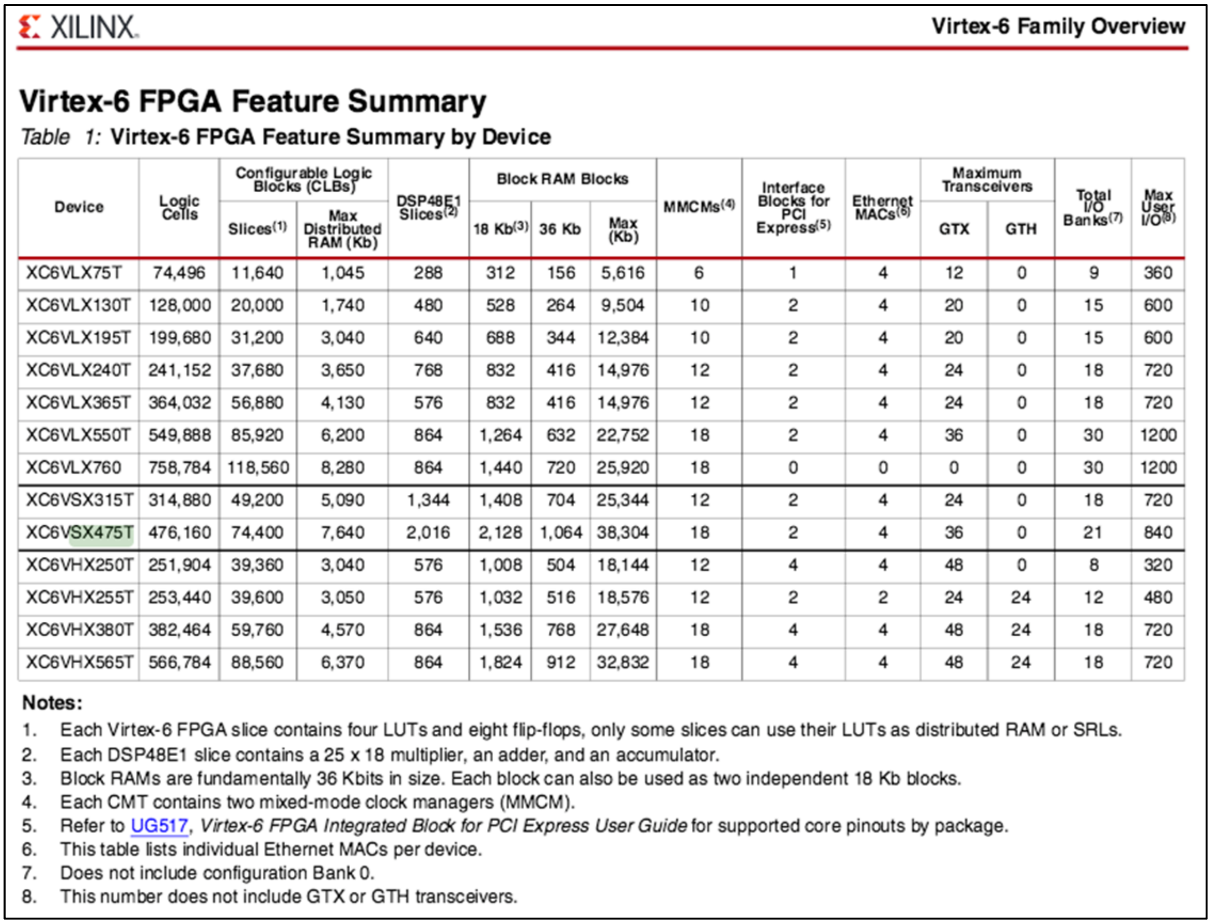
Resource Estimator (Xilinx Resource Es...

Slices	2781
FFs	4851
BRAMs	0
LUTs	3835
IOBs	1218
Mults/DSP48s	4
TBUFs	0

□ Use area above

Estimate options: Estimate

Figure 4.19 FPGA’s resources utilization



Virtex-6 Family Overview

Virtex-6 FPGA Feature Summary

Table 1: **Virtex-6 FPGA Feature Summary by Device**

Device	Logic Cells	Configurable Logic Blocks (CLBs)		DSP48E1 Slices(2)	Block RAM Blocks			MMCMs(4)	Interface Blocks for PCI Express(5)	Ethernet MACs(6)	Maximum Transceivers		Total I/O Banks(7)	Max User I/O(8)
		Slices(1)	Max Distributed RAM (Kb)		18 Kb(3)	36 Kb	Max (Kb)				GTX	GTH		
XC6VLX75T	74,496	11,640	1,045	288	312	156	5,616	6	1	4	12	0	9	360
XC6VLX130T	128,000	20,000	1,740	480	528	264	9,504	10	2	4	20	0	15	600
XC6VLX195T	199,680	31,200	3,040	640	688	344	12,384	10	2	4	20	0	15	600
XC6VLX240T	241,152	37,680	3,650	768	832	416	14,976	12	2	4	24	0	18	720
XC6VLX365T	364,032	56,880	4,130	576	832	416	14,976	12	2	4	24	0	18	720
XC6VLX550T	549,888	85,920	6,200	864	1,264	632	22,752	18	2	4	36	0	30	1200
XC6VLX760	758,784	118,560	8,280	864	1,440	720	25,920	18	0	0	0	0	30	1200
XC6VSX315T	314,880	49,200	5,090	1,344	1,408	704	25,344	12	2	4	24	0	18	720
XC6VSX475T	476,160	74,400	7,640	2,016	2,128	1,064	38,304	18	2	4	36	0	21	840
XC6VHX250T	251,904	39,360	3,040	576	1,008	504	18,144	12	4	4	48	0	8	320
XC6VHX255T	253,440	39,600	3,050	576	1,032	516	18,576	12	2	2	24	24	12	480
XC6VHX380T	382,464	59,760	4,570	864	1,536	768	27,648	18	4	4	48	24	18	720
XC6VHX565T	566,784	88,560	6,370	864	1,824	912	32,832	18	4	4	48	24	18	720

Notes:

1. Each Virtex-6 FPGA slice contains four LUTs and eight flip-flops, only some slices can use their LUTs as distributed RAM or SRLs.
2. Each DSP48E1 slice contains a 25 x 18 multiplier, an adder, and an accumulator.
3. Block RAMs are fundamentally 36 Kbits in size. Each block can also be used as two independent 18 Kb blocks.
4. Each CMT contains two mixed-mode clock managers (MMCM).
5. Refer to UG517, *Virtex-6 FPGA Integrated Block for PCI Express User Guide* for supported core pinouts by package.
6. This table lists individual Ethernet MACs per device.
7. Does not include configuration Bank 0.
8. This number does not include GTX or GTH transceivers.

Figure 4.20 Virtex 6 FPGA Feature Summary
(Xilinx, 2015)

A top-level view of the implemented architecture is provided in Figure 4.21, the different parts of the system have been highlighted using a color code:

1. Custom registers section appears in [blue];
2. FPGA and RTDEx Configuration appear in [red];
3. Power measurement and AGC section appears in [green];
4. SDA selector appears in orange [orange].

The entire system implementation by module is presented in in Appendix V.

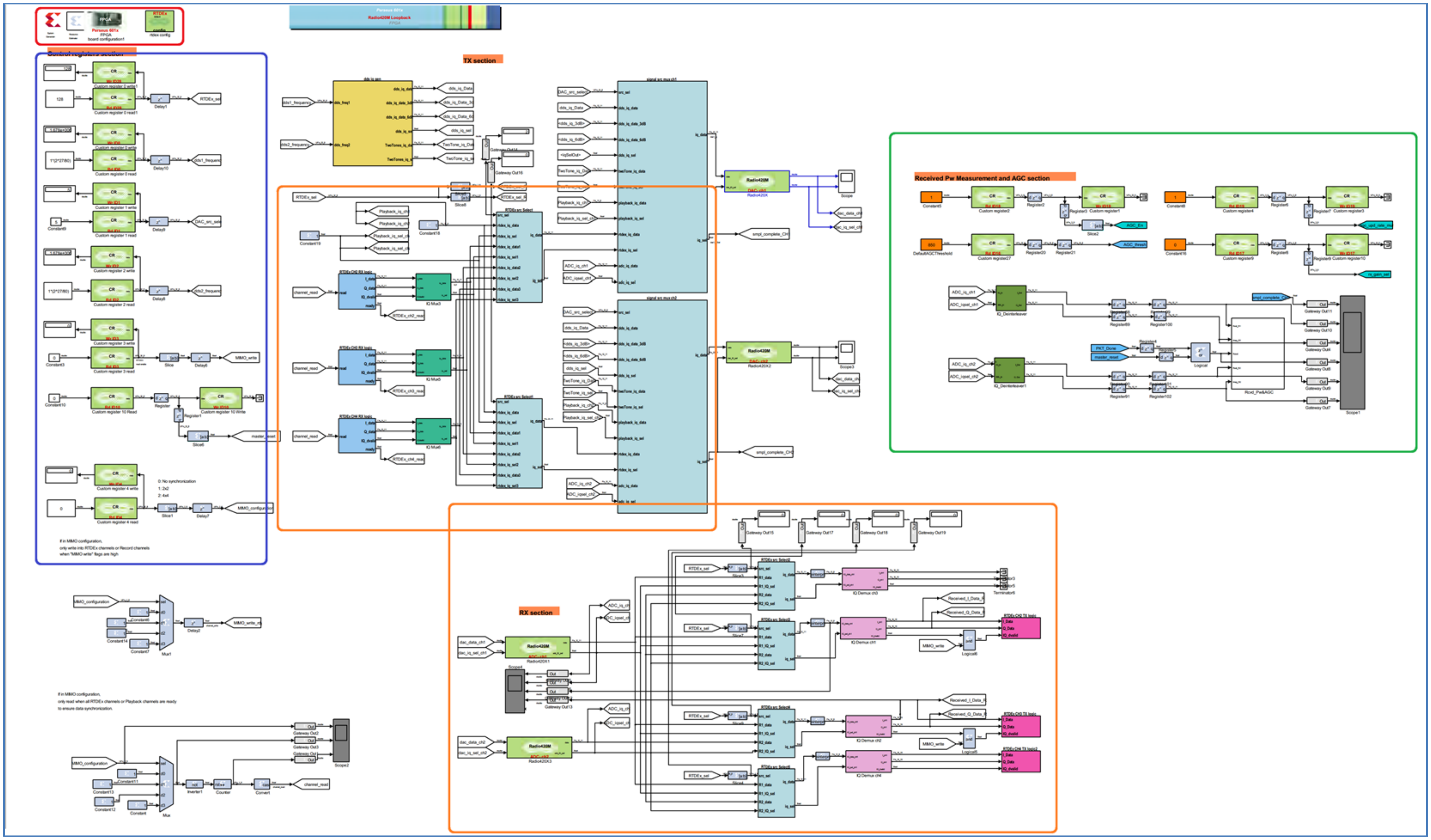

Control registers section
TX section
Received Pw Measurement and AGC section
RX section

Figure 4.21 Full System Implementation in the FPGA

As presented in the previous pages a fully functional architecture that allows the integration of TYPE I, II and III SDAs was conceived, designed and implemented. Functional verification was performed through offline simulation where applicable. The following two chapters of this document focus respectively on the usability and functional testing of the architecture here presented.

CHAPTER 5

THE USER INTERFACE

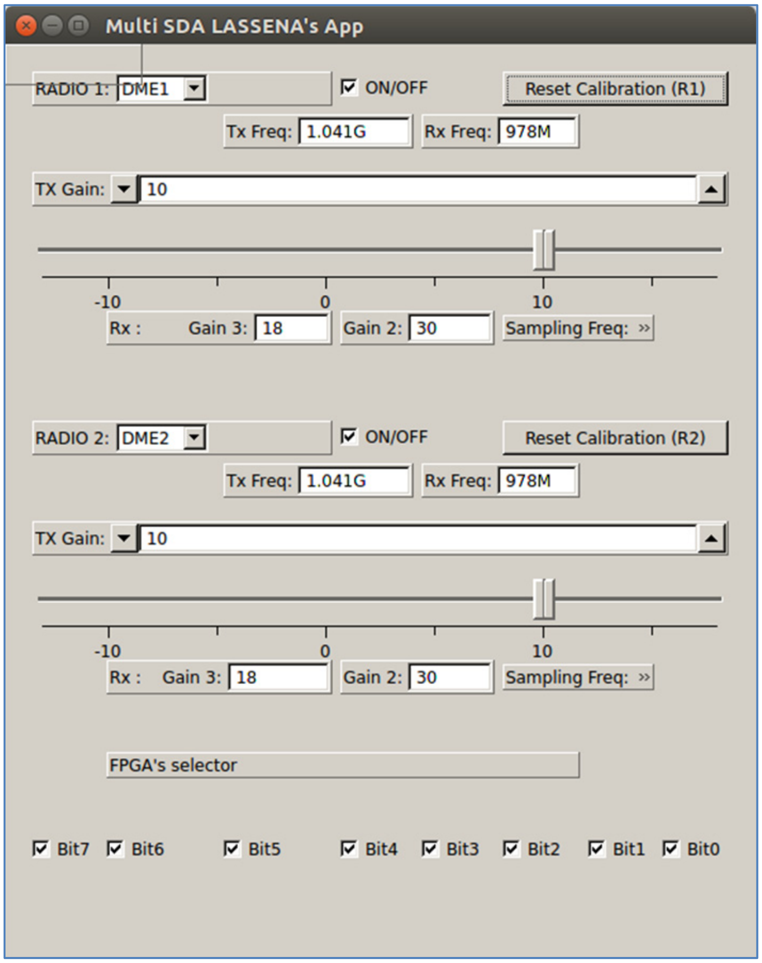
In order to be able to control the integrated SDAR modules, a Graphical User Interface (GUI) was designed using Python's QT library for GNU Radio. It was conceived to allow the final user of the system to interact with it and configure it at runtime. The GUI allows the user to:

1. Select the SDAR systems that will operate on the two RF transceivers available at the Radio420M FMC. Currently, these systems must be selected among 2 DME's and the ADS-B Out module. The architecture is ready for future developments of the TMS and the WBR modules;
2. Adjust the signal output power level;
3. Monitor the gains of the two amplifiers in the Radio420M receiver chain that are used to implement an Automatic Gain Control (AGC) circuit with more than 70 dB of dynamic range;
4. Tune the DME modules to any of the frequency and mode operation channels as described in the (DO-189);
5. Turn ON/OFF any of the two transceiver chains included in the Radio420M FMC;
6. Monitor the activity of the different SDA's once they are operating through a different window pop-up;
7. Program the information broadcast by ADS-B through its application specific GUI.

The GUI's structure, control interface and data display elements are presented in this section.

5.1 GUI’s Structure

The GUI is composed of two parts, the ‘SDR control section’ and ‘Application specific’ windows. The control section contains information related to the two radio systems available in the PicoSDR. This window allows the user to control the overall multi-SDA system functionality; it presents the SDAs that are currently available in the system and lets the user decide which of the radios is used by each of them. A screenshot of this window is presented in Figure 5.1. The SDAR switching and handling occurs in the background, resulting in this way, transparent to the user.



RADIO 1: DME1 ☑ ON/OFF

Tx Freq: 1.041G Rx Freq: 978M

TX Gain: 10

Rx : Gain 3: 18 Gain 2: 30 Sampling Freq: »

RADIO 2: DME2 ☑ ON/OFF

Tx Freq: 1.041G Rx Freq: 978M

TX Gain: 10

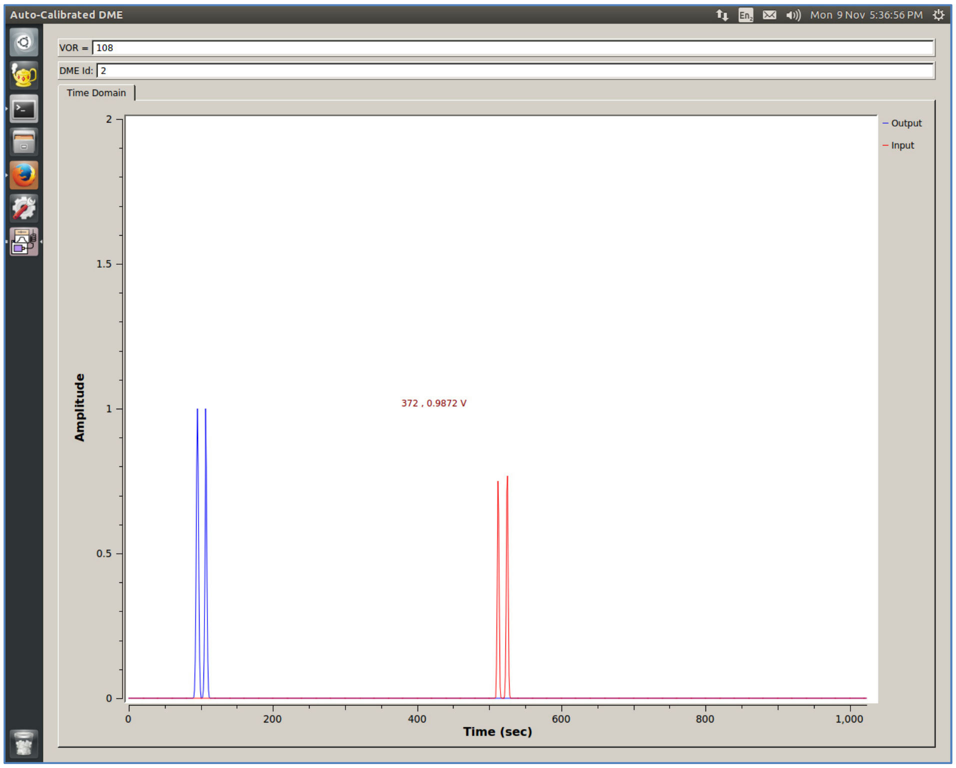
Rx : Gain 3: 18 Gain 2: 30 Sampling Freq: »

☑ Bit7 ☑ Bit6 ☑ Bit5 ☑ Bit4 ☑ Bit3 ☑ Bit2 ☑ Bit1 ☑ Bit0

Figure 5.1 SDR’s Control Window

As previously mentioned, the second part of the GUI consists of ‘Application specific’ windows that are particular to each independent SDA. These windows are defined in the SDA’s code itself and are not linked in any way to any other part of the system; they are only shown whenever the corresponding SDA is active, contributing in this way to simplify the overall system’s operation complexity. The structure is designed in this way to stay consistent with the sought modularity of the system. Figure 5.2 and Figure 5.3 show the application windows for the two SDAs developed for this project.

For the DME specific application window, the user is provided with information about the identification of the DME module (1 or 2), as well as a time scope that allows to see the transmitted and received baseband (demodulated and and modulating) signals. The user can also select the VHF Omnidirectional Range (VOR) frequency associated with the DME channel in operation.



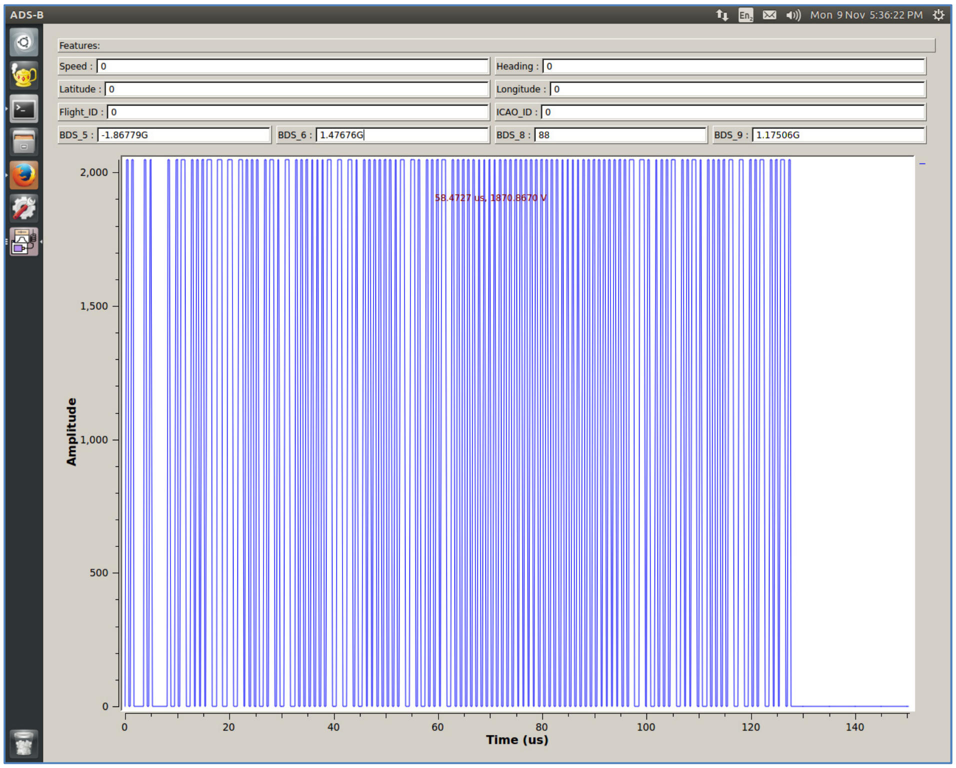
VOR = 108

DME Id: 2

Figure 5.2 DME specific application window

With regards to the ADS-B specific application window, the user can visualize the baseband modulating signal with the 1090 Extended Squitter (ES) message using Pulse Position Modulation (PPM) as specified in (DO-181). The user is also presented with the current information being transmitted in ADS-B message, namely:

1. The speed,
2. The heading,
3. The latitude,
4. The longitude,
5. The flight ID,
6. The ICAO ID,
7. The BDS number.



Features:

Speed : 0
Heading : 0
Latitude : 0
Longitude : 0
Flight_ID : 0
ICAO_ID : 0
BDS_5 : -1.86779G
BDS_6 : 1.47676G
BDS_8 : 88
BDS_9 : 1.17506G

Figure 5.3 ADS-B specific Application window

5.2 User Control

The following actions are enabled through the 'SDR Control Section' of the GUI interface.

Actions available per Radio420 card:

1. SDA selection and switching;
2. Radio Tx/Rx paths enable/disable control (ON/OFF switch);
3. Manual Tx Gain configuration;
4. Display of the currently selected AGC's gain (Rx path);
5. ADC's configured sampling frequency display;
6. Tx and Rx currently selected frequencies display;
7. SDA selective restart.

5.3 Data Display

Consistently with the previously described structure of the system, the general SDR related data is presented to the user by the 'SDR Control Section' of the GUI; data related to the state of the SDAs is left to be presented by each of them in their own 'Application Specific' windows.

As mentioned before in the introduction of this section, the GUI developed for this project was created to simplify the configuration and provide a more intuitive operation of the system. Its modularity helps reducing the amount and complexity of data presented to the user while maintaining the system's overall flexibility.

A dedicated thread is created for the refreshing and capture of data for each 'TopBlock' that implements a GUI window. The thread is automatically assigned with a lower priority than the rest of the system’s tasks to guarantee that the important signal processing tasks are not significantly affected. The refresh rate is fast enough to be imperceptible to the user.

The GUI functionality can be observed in Chapter 6 where the results of the system operation are presented.

CHAPTER 6

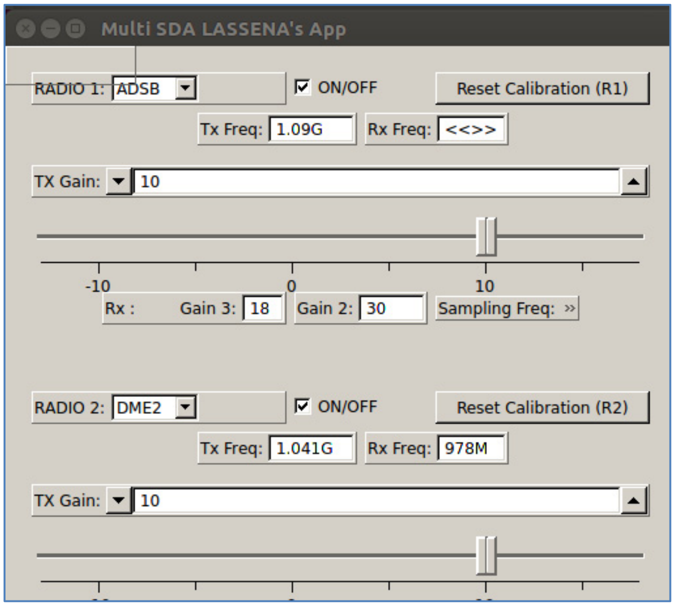
SYSTEM OPERATION RESULTS

This section presents the system in operation while making use of the three functional parts described in the previous sections:

1. [SECTION 4.2.1] GNU Radio Implementation,
2. [SECTION 4.2.3] FPGA Implementation,
3. [CHAPTER 5] The User Interface.

6.1 Multi-SDA System´s Interface Operation

Figure 6.1 depicts the ‘SDR Control Window’ while it selects where ADS-B has been selected to operate making use of Radio1 and DME2 SDA has been selected for concurrent operation in Radio2.



RADIO 1: ADSB ☑ ON/OFF

Tx Freq: 1.09G Rx Freq: <<>>

TX Gain: 10

Rx : Gain 3: 18 Gain 2: 30 Sampling Freq: »

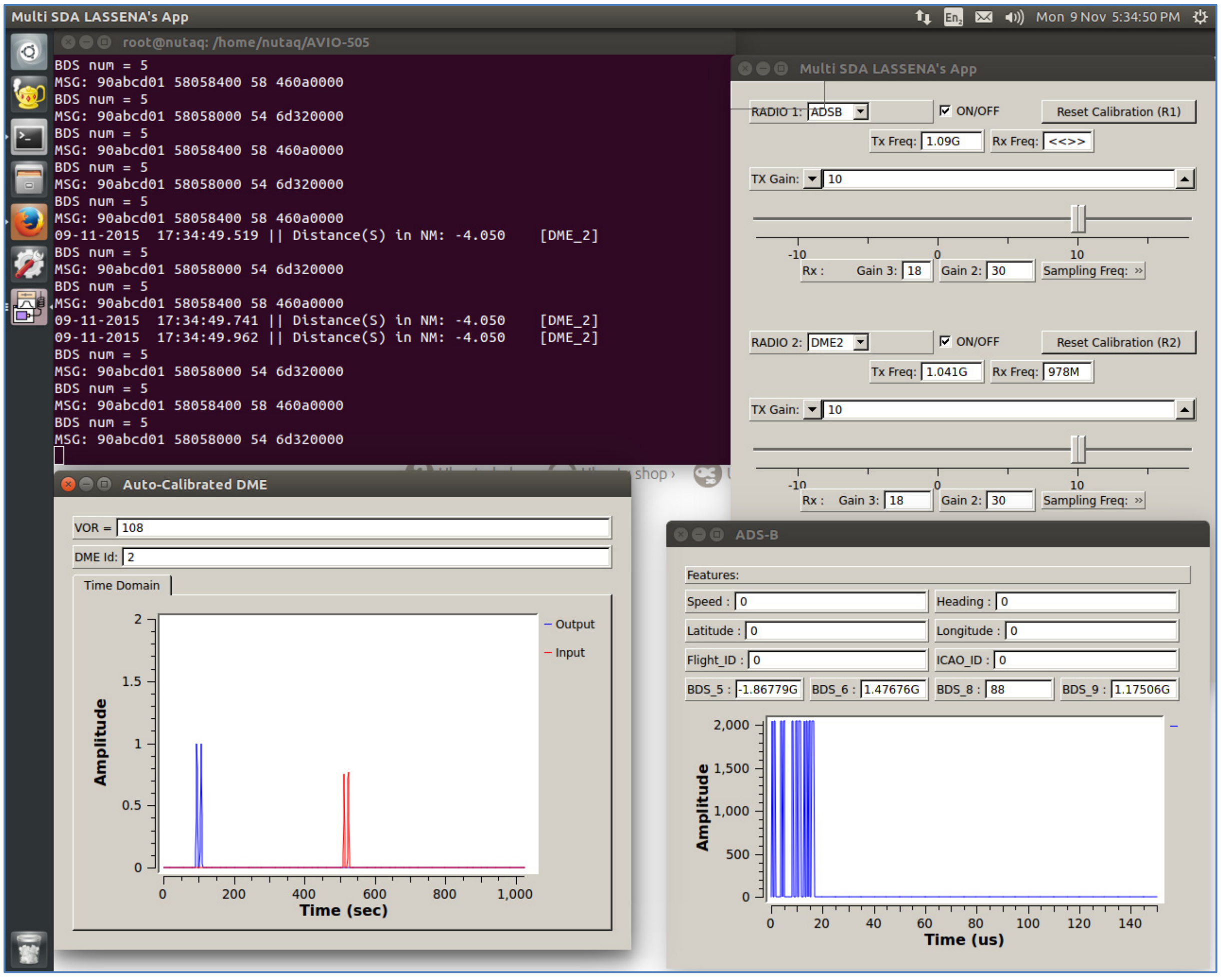
RADIO 2: DME2 ☑ ON/OFF

Tx Freq: 1.041G Rx Freq: 978M

TX Gain: 10

Figure 6.1 SDR Control Window in Operation

Figure 6.2 depicts the system in operation on an Ubuntu 14.3 Linux operating system. DME and ADS-B application windows are displayed in the image.



```
BDS num = 5
MSG: 90abcd01 58058400 58 460a0000
BDS num = 5
MSG: 90abcd01 58058000 54 6d320000
BDS num = 5
MSG: 90abcd01 58058400 58 460a0000
BDS num = 5
MSG: 90abcd01 58058000 54 6d320000
BDS num = 5
MSG: 90abcd01 58058400 58 460a0000
09-11-2015  17:34:49.519 || Distance(S) in NM: -4.050    [DME_2]
BDS num = 5
MSG: 90abcd01 58058000 54 6d320000
BDS num = 5
MSG: 90abcd01 58058400 58 460a0000
09-11-2015  17:34:49.741 || Distance(S) in NM: -4.050    [DME_2]
09-11-2015  17:34:49.962 || Distance(S) in NM: -4.050    [DME_2]
BDS num = 5
MSG: 90abcd01 58058000 54 6d320000
BDS num = 5
MSG: 90abcd01 58058400 58 460a0000
BDS num = 5
MSG: 90abcd01 58058000 54 6d320000
```

RADIO 1: ADSB ☑ ON/OFF
Tx Freq: 1.09G Rx Freq: <<>>
TX Gain: 10
Rx : Gain 3: 18 Gain 2: 30

RADIO 2: DME2 ☑ ON/OFF
Tx Freq: 1.041G Rx Freq: 978M
TX Gain: 10
Rx : Gain 3: 18 Gain 2: 30

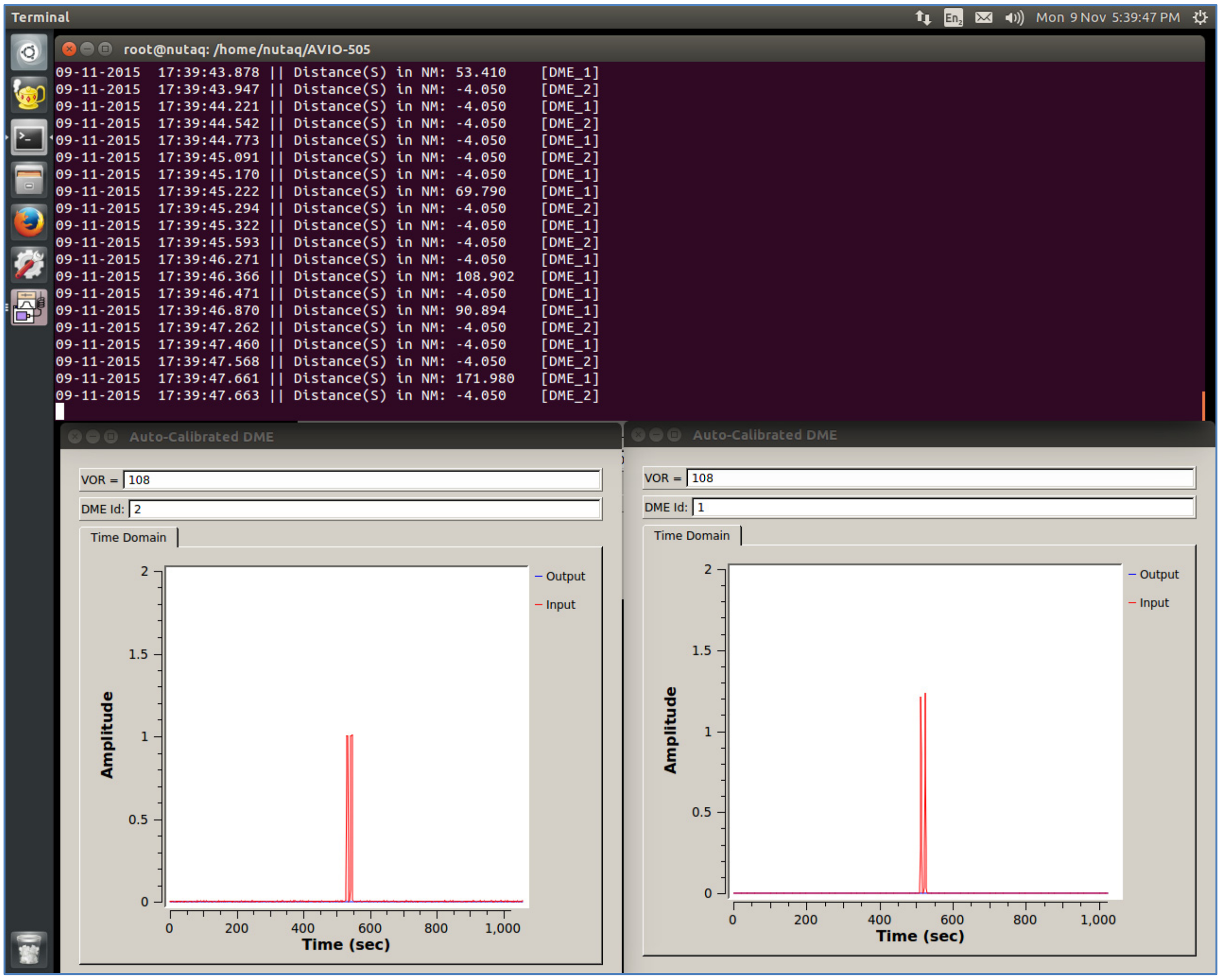
VOR = 108
DME Id: 2

Features:
Speed : 0 Heading : 0
Latitude : 0 Longitude : 0
Flight_ID : 0 ICAO_ID : 0
BDS_5 : -1.86779G BDS_6 : 1.47676G BDS_8 : 88 BDS_9 : 1.17506G

Figure 6.2 Multi-SDA System Operation

Figure 6.3 depicts two DME's SDA application-windows and the terminal window where 1 of every 10 DME measured distances is displayed[11]. In this example 'DME 1' makes use of Radio1 while 'DME 2' takes Radio2 to operate concurrently. Notice that both of the DME's are tuned to VOR channel 108, which was selected as the calibration channel for DME SDA.

[11] Resulting in a data update frequency (on screen) of 1.6Hz



```
09-11-2015  17:39:43.878 || Distance(S) in NM: 53.410    [DME_1]
09-11-2015  17:39:43.947 || Distance(S) in NM: -4.050    [DME_2]
09-11-2015  17:39:44.221 || Distance(S) in NM: -4.050    [DME_1]
09-11-2015  17:39:44.542 || Distance(S) in NM: -4.050    [DME_2]
09-11-2015  17:39:44.773 || Distance(S) in NM: -4.050    [DME_1]
09-11-2015  17:39:45.091 || Distance(S) in NM: -4.050    [DME_2]
09-11-2015  17:39:45.170 || Distance(S) in NM: -4.050    [DME_1]
09-11-2015  17:39:45.222 || Distance(S) in NM: 69.790    [DME_1]
09-11-2015  17:39:45.294 || Distance(S) in NM: -4.050    [DME_2]
09-11-2015  17:39:45.322 || Distance(S) in NM: -4.050    [DME_1]
09-11-2015  17:39:45.593 || Distance(S) in NM: -4.050    [DME_2]
09-11-2015  17:39:46.271 || Distance(S) in NM: -4.050    [DME_1]
09-11-2015  17:39:46.366 || Distance(S) in NM: 108.902   [DME_1]
09-11-2015  17:39:46.471 || Distance(S) in NM: -4.050    [DME_1]
09-11-2015  17:39:46.870 || Distance(S) in NM: 90.894    [DME_1]
09-11-2015  17:39:47.262 || Distance(S) in NM: -4.050    [DME_2]
09-11-2015  17:39:47.460 || Distance(S) in NM: -4.050    [DME_1]
09-11-2015  17:39:47.568 || Distance(S) in NM: -4.050    [DME_2]
09-11-2015  17:39:47.661 || Distance(S) in NM: 171.980   [DME_1]
09-11-2015  17:39:47.663 || Distance(S) in NM: -4.050    [DME_2]
```

VOR = 108
DME Id: 2

VOR = 108
DME Id: 1

Figure 6.3 Two DME SDAs in Operation (Non-tracking)

6.2 Multi-SDA Functional Testing

To simulate DME ground stations and for ADS-B data transmission, “IFR-6000” (Cobham AvComm, 2015) was used. The physical setup used for DME and ADS-B functional tests is shown in Figure 6.4 and Figure 6.5.

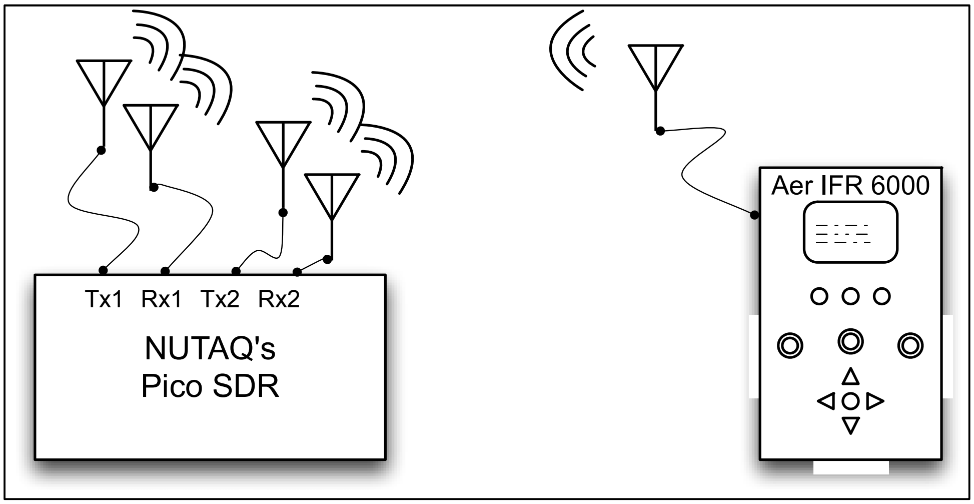


Figure 6.4 Functional SDAs setup

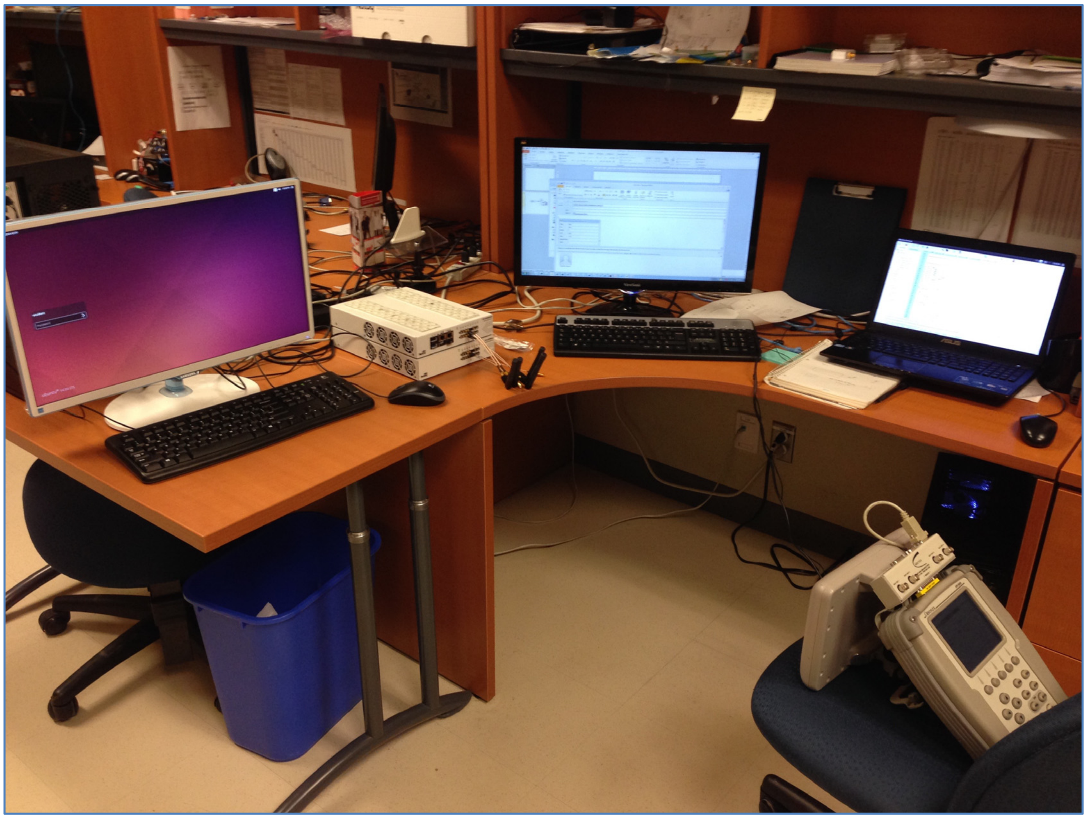
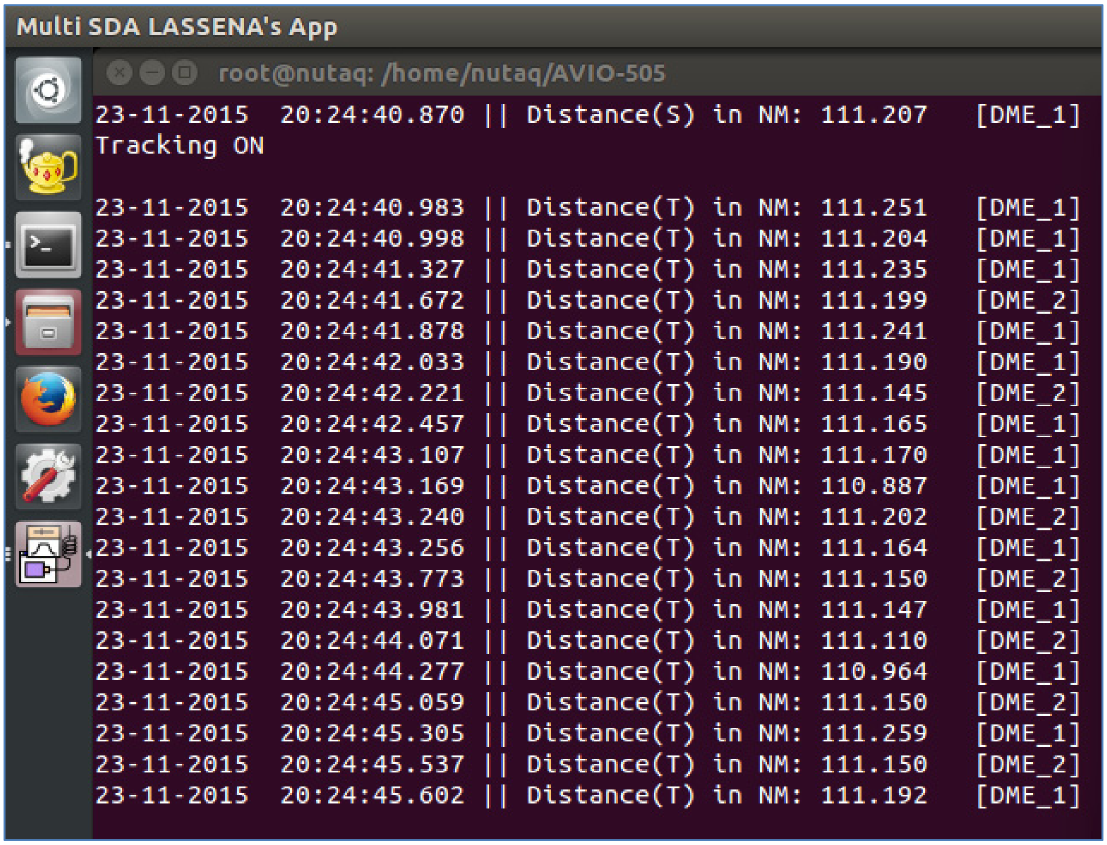


Figure 6.5 Functional tests equipment setup

Figure 6.6 shows two DME SDA’s tuned to the same ‘Ground Station’ (VOR 116.7) and tracking an IFR-6000 simulated distance of 111.0 NM, notice how both measurements are ± 0.226 % of the real simulated distance.



```
23-11-2015  20:24:40.870 || Distance(S) in NM: 111.207   [DME_1]
Tracking ON

23-11-2015  20:24:40.983 || Distance(T) in NM: 111.251   [DME_1]
23-11-2015  20:24:40.998 || Distance(T) in NM: 111.204   [DME_1]
23-11-2015  20:24:41.327 || Distance(T) in NM: 111.235   [DME_1]
23-11-2015  20:24:41.672 || Distance(T) in NM: 111.199   [DME_2]
23-11-2015  20:24:41.878 || Distance(T) in NM: 111.241   [DME_1]
23-11-2015  20:24:42.033 || Distance(T) in NM: 111.190   [DME_1]
23-11-2015  20:24:42.221 || Distance(T) in NM: 111.145   [DME_2]
23-11-2015  20:24:42.457 || Distance(T) in NM: 111.165   [DME_1]
23-11-2015  20:24:43.107 || Distance(T) in NM: 111.170   [DME_1]
23-11-2015  20:24:43.169 || Distance(T) in NM: 110.887   [DME_1]
23-11-2015  20:24:43.240 || Distance(T) in NM: 111.202   [DME_2]
23-11-2015  20:24:43.256 || Distance(T) in NM: 111.164   [DME_1]
23-11-2015  20:24:43.773 || Distance(T) in NM: 111.150   [DME_2]
23-11-2015  20:24:43.981 || Distance(T) in NM: 111.147   [DME_1]
23-11-2015  20:24:44.071 || Distance(T) in NM: 111.110   [DME_2]
23-11-2015  20:24:44.277 || Distance(T) in NM: 110.964   [DME_1]
23-11-2015  20:24:45.059 || Distance(T) in NM: 111.150   [DME_2]
23-11-2015  20:24:45.305 || Distance(T) in NM: 111.259   [DME_1]
23-11-2015  20:24:45.537 || Distance(T) in NM: 111.150   [DME_2]
23-11-2015  20:24:45.602 || Distance(T) in NM: 111.192   [DME_1]
```

Figure 6.6 Two DME SDAs tuned to the same ground station (Tracking ON)

As specified in (DO-189) the accepted error in the distance measurement is “±0.17 NM, or ±0.25% of the distance”, considering that this are raw measured data and the maximum error was withn the accepted margin, the results were considered as acceptable for the prototype that appears in Figure 6.6.

6.2.1 Multi-SDA Processing System´s Load

The processor’s mean load was measured with the SDAR (concurrently) executing all of the designed systems (2 DMEs + 1 ADS-B) to obtain the performance footprint and to verify that the SDAs were instantiated in the cores where they were assigned. The characteristic load of the SDAR in this scenario is explained in Tables 0.7 and 0.8. Table 6.3 shows the resources allocated per SDA.

Table 6.1 Total CPU Resources Allocation

Total CPU Resources Allocation		
	CPU mean utilization	
Total Available	100%	
	Min (%)	Max (%)
Total Taken (OS and 3 SDAs)	17	28.32

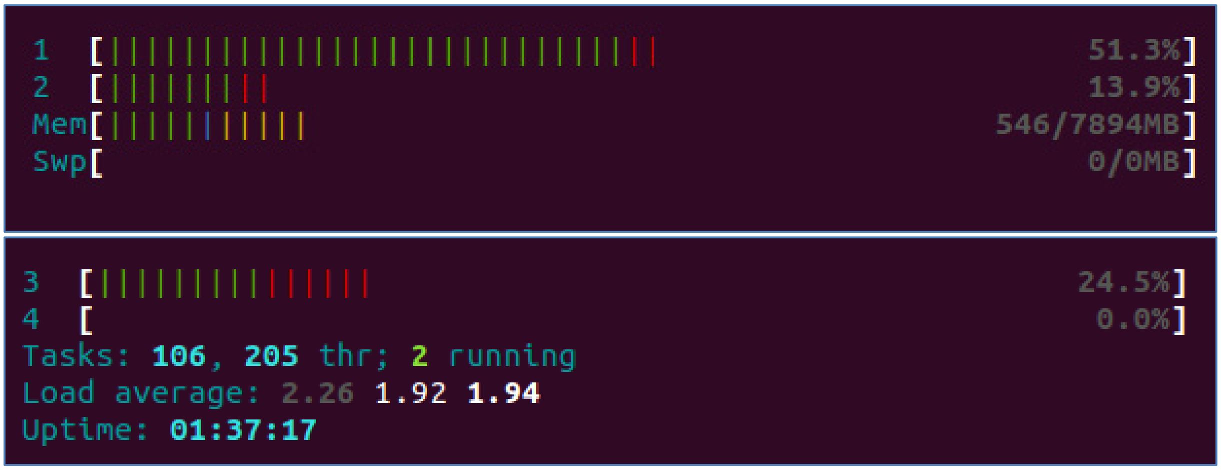
Table 6.2 Total Memory Allocation

Total Memory Allocation		
	RAM	
	MB	(%)
Total Available	8192	100
Total Taken (OS and 3 SDAs)	546	6.67%

Table 6.3 Resources Allocation per SDA

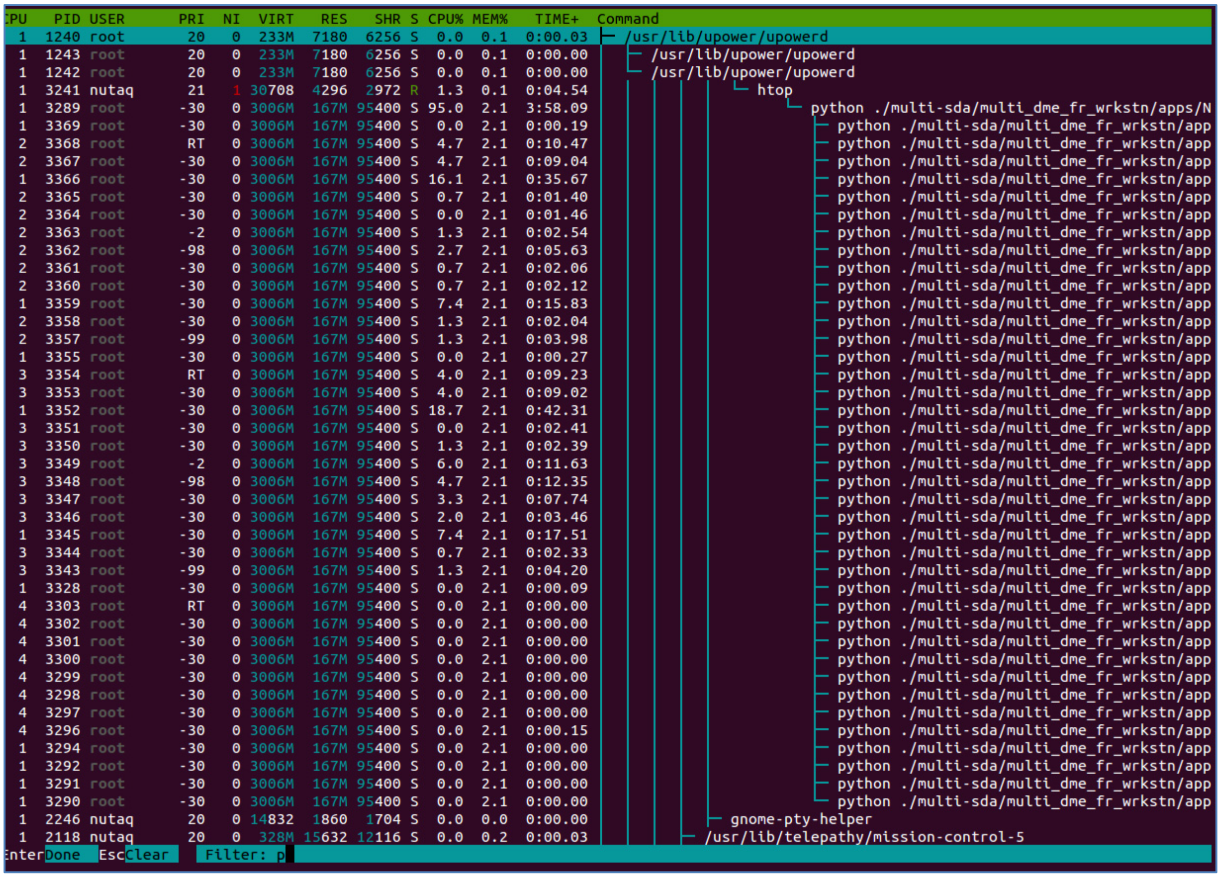
Resources Allocation per SDA					
Process	**Mean utilization % (1 Core)**		**RAM**		
	Min	**Max**	**(MB)**	**% (of Used)**	**% (of Total)**
OS (Non-SDAR tasks)	**20**	**51.3**	**413**	**75.64**	**5.04**
ADSB	**9**	**12**	**66**	**12.09**	**0.81**
DME	**11**	**25**	**67**	**12.27**	**0.82**

Figure 6.7 shows the active cores utilization for a system where two DME SDAs are concurrently operating. In Figure 6.8 the python thread structure for the same case can be appreciated.



```
1  [||||||||||||||||||||||||||||||               51.3%]
2  [|||||||||                                    13.9%]
Mem[|||||||||||                             546/7894MB]
Swp[                                              0/0MB]

3  [|||||||||||||||                              24.5%]
4  [                                              0.0%]
Tasks: 106, 205 thr; 2 running
Load average: 2.26 1.92 1.94
Uptime: 01:37:17
```

Figure 6.7 Measured Cores Mean Load



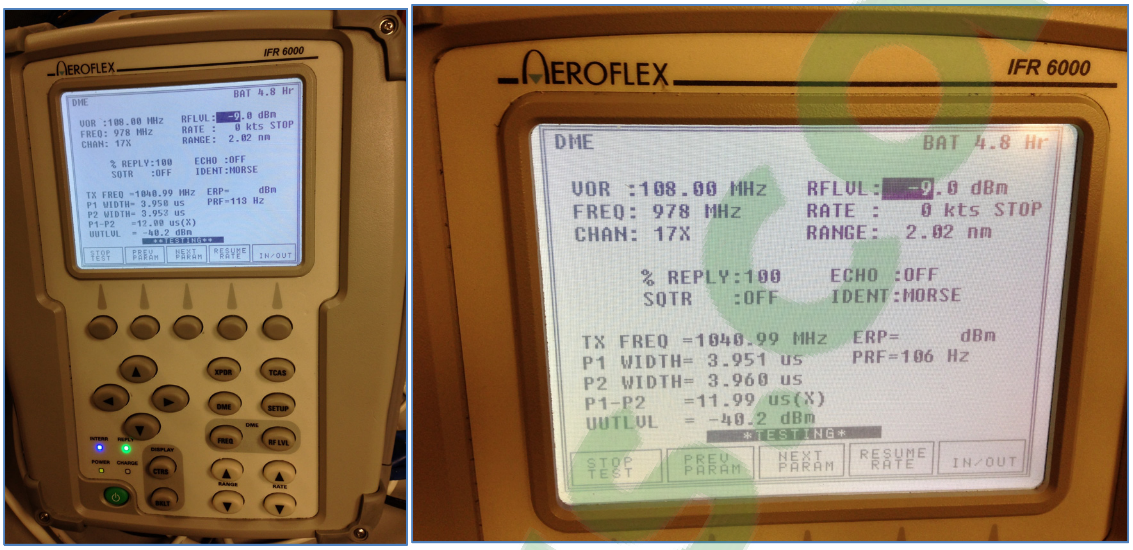
```
CPU   PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
  1  1240 root       20   0  233M  7180  6256 S  0.0  0.1  0:00.03 ├─ /usr/lib/upower/upowerd
  1  1243 root       20   0  233M  7180  6256 S  0.0  0.1  0:00.00 │  ├─ /usr/lib/upower/upowerd
  1  1242 root       20   0  233M  7180  6256 S  0.0  0.1  0:00.00 │  └─ /usr/lib/upower/upowerd
  1  3241 nutaq      21   1 30708  4296  2972 R  1.3  0.1  0:04.54 │  │  │  │  │     └─ htop
  1  3289 root      -30   0 3006M  167M 95400 S 95.0  2.1  3:58.09 │  │  │  │  │        └─ python ./multi-sda/multi_dme_fr_wrkstn/apps/N
  1  3369 root      -30   0 3006M  167M 95400 S  0.0  2.1  0:00.19 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  2  3368 root       RT   0 3006M  167M 95400 S  4.7  2.1  0:10.47 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  2  3367 root      -30   0 3006M  167M 95400 S  4.7  2.1  0:09.04 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  1  3366 root      -30   0 3006M  167M 95400 S 16.1  2.1  0:35.67 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  2  3365 root      -30   0 3006M  167M 95400 S  0.7  2.1  0:01.40 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  2  3364 root      -30   0 3006M  167M 95400 S  0.0  2.1  0:01.46 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  2  3363 root       -2   0 3006M  167M 95400 S  1.3  2.1  0:02.54 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  2  3362 root      -98   0 3006M  167M 95400 S  2.7  2.1  0:05.63 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  2  3361 root      -30   0 3006M  167M 95400 S  0.7  2.1  0:02.06 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  2  3360 root      -30   0 3006M  167M 95400 S  0.7  2.1  0:02.12 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  1  3359 root      -30   0 3006M  167M 95400 S  7.4  2.1  0:15.83 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  2  3358 root      -30   0 3006M  167M 95400 S  1.3  2.1  0:02.04 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  2  3357 root      -99   0 3006M  167M 95400 S  1.3  2.1  0:03.98 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  1  3355 root      -30   0 3006M  167M 95400 S  0.0  2.1  0:00.27 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  3  3354 root       RT   0 3006M  167M 95400 S  4.0  2.1  0:09.23 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  3  3353 root      -30   0 3006M  167M 95400 S  4.0  2.1  0:09.02 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  1  3352 root      -30   0 3006M  167M 95400 S 18.7  2.1  0:42.31 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  3  3351 root      -30   0 3006M  167M 95400 S  0.0  2.1  0:02.41 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  3  3350 root      -30   0 3006M  167M 95400 S  1.3  2.1  0:02.39 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  3  3349 root       -2   0 3006M  167M 95400 S  6.0  2.1  0:11.63 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  3  3348 root      -98   0 3006M  167M 95400 S  4.7  2.1  0:12.35 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  3  3347 root      -30   0 3006M  167M 95400 S  3.3  2.1  0:07.74 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  3  3346 root      -30   0 3006M  167M 95400 S  2.0  2.1  0:03.46 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  1  3345 root      -30   0 3006M  167M 95400 S  7.4  2.1  0:17.51 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  3  3344 root      -30   0 3006M  167M 95400 S  0.7  2.1  0:02.33 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  3  3343 root      -99   0 3006M  167M 95400 S  1.3  2.1  0:04.20 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  1  3328 root      -30   0 3006M  167M 95400 S  0.0  2.1  0:00.09 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  4  3303 root       RT   0 3006M  167M 95400 S  0.0  2.1  0:00.00 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  4  3302 root      -30   0 3006M  167M 95400 S  0.0  2.1  0:00.00 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  4  3301 root      -30   0 3006M  167M 95400 S  0.0  2.1  0:00.00 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  4  3300 root      -30   0 3006M  167M 95400 S  0.0  2.1  0:00.00 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  4  3299 root      -30   0 3006M  167M 95400 S  0.0  2.1  0:00.00 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  4  3298 root      -30   0 3006M  167M 95400 S  0.0  2.1  0:00.00 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  4  3297 root      -30   0 3006M  167M 95400 S  0.0  2.1  0:00.00 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  4  3296 root      -30   0 3006M  167M 95400 S  0.0  2.1  0:00.15 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  1  3294 root      -30   0 3006M  167M 95400 S  0.0  2.1  0:00.00 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  1  3292 root      -30   0 3006M  167M 95400 S  0.0  2.1  0:00.00 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  1  3291 root      -30   0 3006M  167M 95400 S  0.0  2.1  0:00.00 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  1  3290 root      -30   0 3006M  167M 95400 S  0.0  2.1  0:00.00 │  │  │  │  │           ├─ python ./multi-sda/multi_dme_fr_wrkstn/app
  1  2246 nutaq      20   0 14832  1860  1704 S  0.0  0.0  0:00.00 │  │  │  │  ├─ gnome-pty-helper
  1  2118 nutaq      20   0  328M 15632 12116 S  0.0  0.2  0:00.03 │  │  │  ├─ /usr/lib/telepathy/mission-control-5
EnterDone  EscClear   Filter: p
```

Figure 6.8 Python Threads Tree

6.2.2 System Testing with IFR 6000

Figure 6.9 shows IFR 6000 while being interrogated by the developed DME SDA; it can be observed the pulses width and spacing of the received interrogations.

- P1 width = 3.951µs,
- P2 width = 3.960µs,
- P1 – P2 pulse separation = 11.99µs.



```
DME                                BAT 4.8 Hr

 VOR :108.00 MHz    RFLVL:  -9.0 dBm
 FREQ: 978 MHz      RATE :    0 kts STOP
 CHAN: 17X          RANGE:  2.02 nm

     % REPLY:100        ECHO :OFF
     SQTR   :OFF        IDENT:MORSE

 TX FREQ =1040.99 MHz  ERP=      dBm
 P1 WIDTH= 3.950 us    PRF=113 Hz
 P2 WIDTH= 3.953 us
 P1-P2    =12.00 us(X)
 UUTLVL   = -40.2 dBm
          **TESTING**
```

```
DME                                BAT 4.8 Hr

 VOR :108.00 MHz    RFLVL:  -9.0 dBm
 FREQ: 978 MHz      RATE :    0 kts STOP
 CHAN: 17X          RANGE:  2.02 nm

     % REPLY:100        ECHO :OFF
     SQTR   :OFF        IDENT:MORSE

 TX FREQ =1040.99 MHz  ERP=      dBm
 P1 WIDTH= 3.951 us    PRF=106 Hz
 P2 WIDTH= 3.960 us
 P1-P2    =11.99 us(X)
 UUTLVL   = -40.2 dBm
          *TESTING*
```

Figure 6.9 IFR 6000 DME SDA test

The expected pulse separation for DME according to (DO-189) is 12µs. The measured value is just 10ns off which is acceptable for the prototype.

6.2.3 Switching Times of the SDAR.

The switching time of the system was measured by adding a timestamp to the samples output. It was achieved by modifying the ‘work function’ of the DME code block (in GNU Radio). In that way the last received sample of the SDA to be turned off could be compared with the timestamp of the first received sample of the system to be set in operation.

For every test the switching time was always either 980.5 ms when switching from the DME and 940.1 ms for the ADS-B.

6.2.4 DME Latency Calibration Results.

Due to the use of RTDEx protocol, which is non-deterministic (since it is based on Ethernet) and the unknown initial state of the RX/TX Ethernet buffers, a constant predictable latency is not achievable for applications that need to transmit data between the CPU and FPGA such as the DME SDA.

For this a calibration routine was added to the DME SDA, which allows to measure the latency added by these phenomena and to correct the value so that the distance measurement is exclusively based on the propagation time of the signal plus the intentionally added 50µs delay.

Table 6.4 presents the measured latency values in multiple executions of the developed application. It can be observed that the latency values differ between different execution sessions of the program but are closely coupled during single execution sessions. This proves the need and usefulness of the developed calibration routine.

Table 6.4 Latency Calibration results

Session Number	DME ID	Repetition				
		1	2	3	4	5
		Time in milliseconds				
1	DME 1	23.964	26.198	26.318	26.191	26.318
	DME 2	73.481	73.614	73.613	73.487	73.486
2	DME 1	12.452	14.829	12.471	12.962	12.384
	DME 2	56.358	56.539	56.314	56.532	56.315
3	DME 1	25.183	25.962	25.827	25.290	25.816
	DME 2	86.340	86.246	86.902	87.012	86.324
4	DME 1	21.382	21.894	21.756	21.774	21.328
	DME 2	76.835	77.013	77.312	77.229	77.187

The system operation resutls were presented in this chapter; an demonstration of the multi-sda working prototype is shown in section 6.1, which focuses on the usability aspect of the user interface. Section 6.2 focuses on the operational results taking into consideration the allowed operation limits for each avionics system that was integrated into the target system.

Validation of these operational aspect of each SDA is performed making use of IFR-6000 and the switching and calibration times are also measured and its impact is presented in the last section of the chapter.

CONCLUSION

The stages of the work developed during this project have been carefully introduced in this document. A proof of concept of a scalable platform aligned with the research problematic presented in section 1.2 was conceived, implemented and tested. It features three independent SDA avionics sytems that can be operated concurrently allowing the user to select and alter their configuration at runtime. The entire architecture design takes into account future expandability of the system.

An appropiate task definition and separation was performed in order to make the most out of the available hardware platform where the project was implemented. Signal conditioning as well as the required decimation and interpolation were implemented where necessary to obtain the required behavior of the system.

Possible improvements and interesting optimizations were identified throughout the development of the prototype they appear in the “Recommendations” section of this document and are left for future work.

The final contribution of this work is a reutilizable and scalable SDA integration architecture, which was demonstrated on a fully functional SDAR. The developed SDAR is capable of simultaneous operation of up to 3 independent and interchangeable SDAs with characteristic switching times of 940 ms and 980 ms. With a data rate requirement of 128 Mbps per channel, the SDAR makes use of 34.71% of the measured total thoughput (553*2 Mbps) provided by the 1Gbps RTDEx link in the proposed configuration.

A peak power AGC was implemented entirely in the FPGA, it offloads the CPU from power measuring tasks allowing for more processor time to be allocated to the SDA processing. In the same way, the switching and routing logic was implemented in the FPGA providing deterministic switching times.

Incremental tests were performed for the logic defined in the FPGA and the accompanying software applications developed for each SDA by making use of different tools such as offline simulation and hardware in the loop where applicable.

The final result was tested by professional aviation certified equipment such as IFR-6000 to validate the correct operation of the prototype. The error of the DME SDA was found to be 0.226 % of the measured distance, which is compliant with the performance standards for airborne distance measuring equipment (DO-189).

The original objective of designing an expandable architecture which allowed the concurrent interaction of multiple avionics systems was achieved and its functionallity proved to be useful for future developments. Work based on the contributions here presented is currently (at the date of submittal of this document) under progress at LASSENA.

RECOMENDATIONS

The identified recommendations for potential improvements and optimizations regarding the work presented in this project are listed in this section.

TMS Hardware Defined Implementation

As mentioned in section 4.1, the available TMS implementation at LASSENA does not respect the timing constraints required for the proper operation of TMS specification. A purely hardware defined version of this SDA might be a solution. Since this was out of the scope of the project it was left out as recommended for future work.

Signal Multiplexing possible

The implementation here presented makes use of one RTDEx channel for each SDAs that is made available to the architecture, this results in a maximum of 8 SDAs being able to be instanciated at a given time. This limitation may be overcome by multiplexing data streams coming from different SDAs and making use of a single RTDEx channel so that its maximum throughput is used.

Decimation/Interpolation in the FPGA

An improvement in performance should be obtained by offloading as much as possible GNU Radio from signal processing stages. It results of particular interest to migrate the Decimation/Interpolation stages from the CPU to the FPGA so that the current CPU load used by performing these tasks could be freed to be used for other purposes.

PCI Communication Protocol

The PCI-RTDEx variant of RTDEx should be able to provide higher total throughput for data transfer in the system. A Linux compliant driver is made available by NUTAQ and supports a maximum of 5 TX channels and 8 RX channels simultaneously. However care should be taken since the use of a DMA engine and the use of translation windows to map the memory between the FPGA and Host may add undesired overhead that might negatively affect the overall performance of the system.

APPENDIX I

CPU Core Isolation Techniques

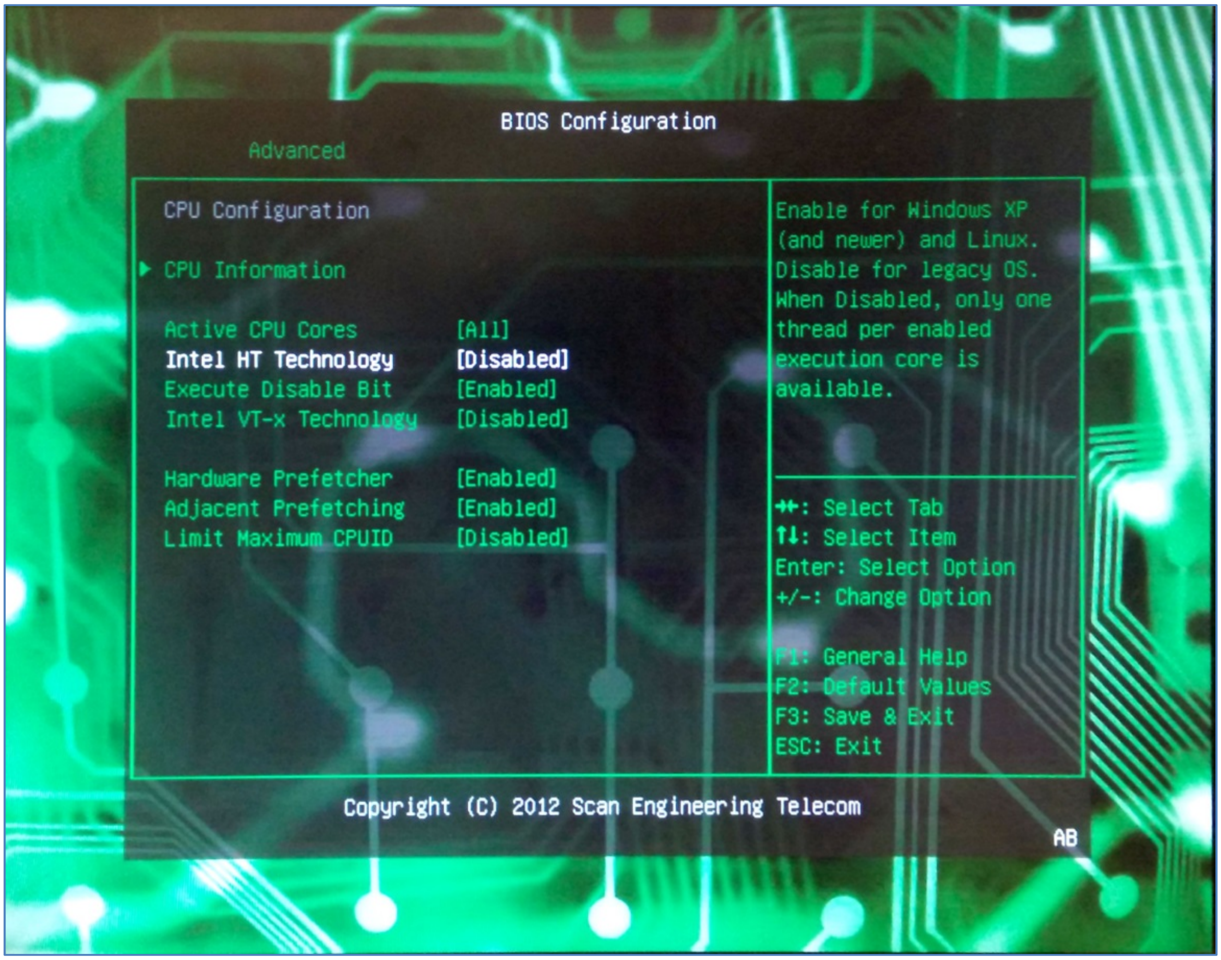
The following steps were taken to achieve the core isolation feature required by the application. This techniques are valid for the Ubuntu 14.04 Linux Operative System.

A group of operations were performed on the system to achive the desired behavior. They include: BIOS configuration, bootargs modification and python scripts developed for the Linux OS.

The following lines briefly explain each of the taken actions:

BIOS Configuration

The Hyperthreading® technology was disabled in the BIOS configuration since when enabled, a decline of the overall stability of the system was observed.

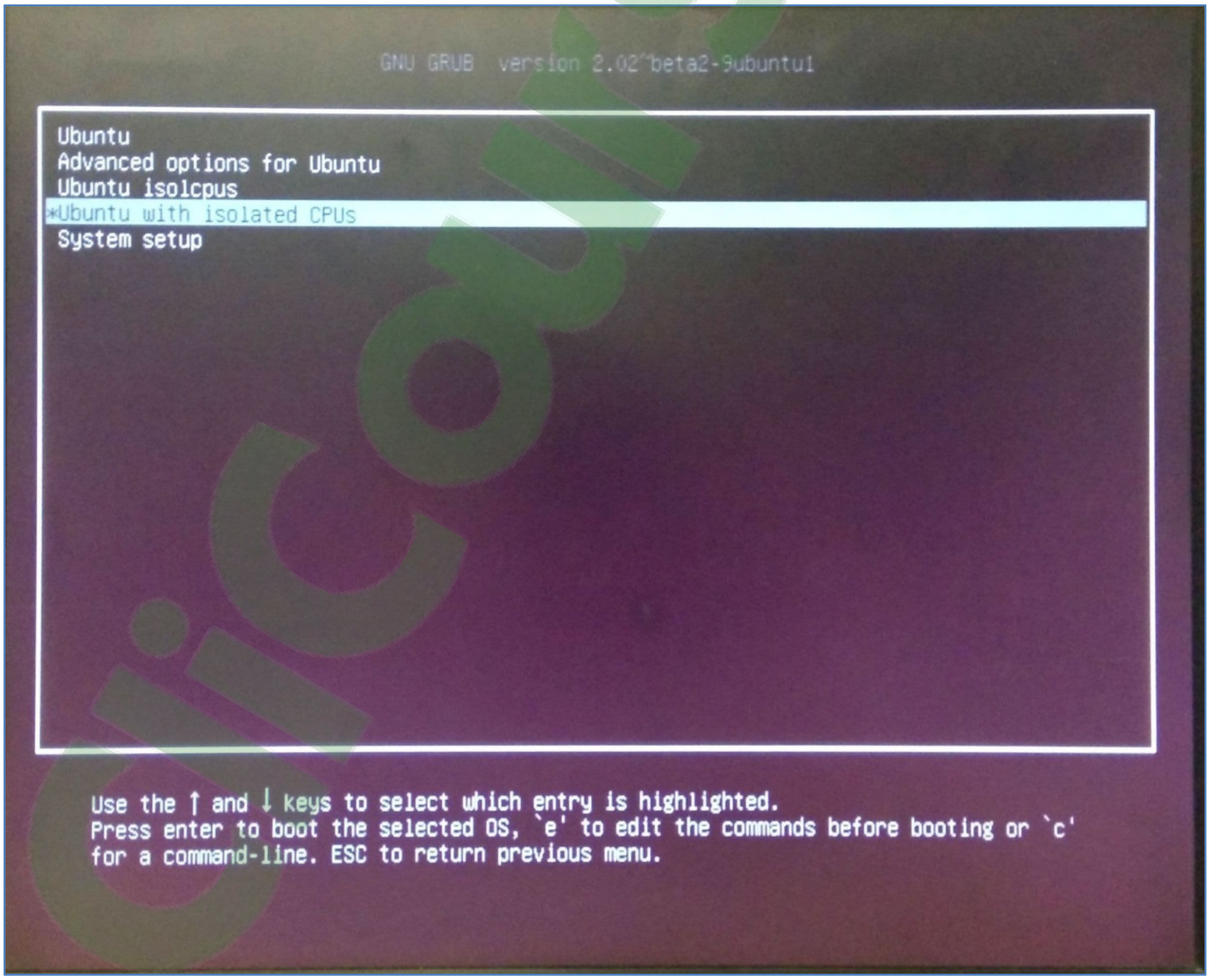


```
                        BIOS Configuration
          Advanced

  CPU Configuration                   Enable for Windows XP
                                      (and newer) and Linux.
▸ CPU Information                     Disable for legacy OS.
                                      When Disabled, only one
  Active CPU Cores        [All]       thread per enabled
  Intel HT Technology     [Disabled]  execution core is
  Execute Disable Bit     [Enabled]   available.
  Intel VT-x Technology   [Disabled]

  Hardware Prefetcher     [Enabled]   ←→: Select Tab
  Adjacent Prefetching    [Enabled]   ↑↓: Select Item
  Limit Maximum CPUID     [Disabled]  Enter: Select Option
                                      +/-: Change Option

                                      F1: General Help
                                      F2: Default Values
                                      F3: Save & Exit
                                      ESC: Exit

             Copyright (C) 2012 Scan Engineering Telecom
                                                        AB
```

Figure 0.1 PicoSDR BIOS Configuration

Specific Menuentry with Special Bootargs Modification

```
menuentry 'Ubuntu isolcpus' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id_option 'gnulinux-simple-05daca0b-d063-453e-ab2e-43fafb0028de' {
    recordfail
    load_video
    gfxmode $linux_gfx_mode
    insmod gzio
    insmod part_msdos
    insmod ext2
    set root='hd0,msdos1'
    if [ x$feature_platform_search_hint = xy ]; then
      search  --no-floppy  --fs-uuid  --set=root  --hint-bios=hd0,msdos1  --
hint-efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1  05daca0b-d063-453e-ab2e-
43fafb0028de
    else
      search  --no-floppy  --fs-uuid  --set=root  05daca0b-d063-453e-ab2e-
43fafb0028de
    fi
    linux     /boot/vmlinuz-3.16.0-30-generic  root=UUID=05daca0b-d063-453e-
ab2e-43fafb0028de ro  quiet splash isolcpus=1,2,3 $vt_handoff
    initrd  /boot/initrd.img-3.16.0-30-generic
}
### END /etc/grub.d/11_custom ###
```



```
GNU GRUB  version 2.02~beta2-9ubuntu1

Ubuntu
Advanced options for Ubuntu
Ubuntu isolcpus
*Ubuntu with isolated CPUs
System setup

Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, `e' to edit the commands before booting or `c'
for a command-line. ESC to return previous menu.
```

Figure 0.2	Created Menuentry in GRUB

Interrupts Disabling from Linux

```
def disabling_IRQ ():
    blocked_f=0
    IRQ_files=glob.glob("/proc/irq/*/smp_affinity") #2 & 0 can't be opened
    for IRQ_f in IRQ_files:
        f = open(IRQ_f,"w")
        try:
            f.write("0f")
            f.close()
        except:
            blocked_f = blocked_f + 1
    if blocked_f > 2 :
        return -1 # TODO could isolate cpus --> Log
    #check if cpus re isolated
    f= open("/proc/cmdline",'r')
    if not "isolcpus" in f.read():
        return -2 # TODO isolcpu not in grub -->Log
    return 0
```

SWAP Disabling from Linux

```
def disabling_swap ():
    ret = os.system("swapoff -a")
    return ret
```

APPENDIX II

Main Multi_SDA Phython Script

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: Multi SDA Plattform
# Author: Rodolfo Solis
# Description: LASSENA
# Generated: Tue Jun  13 12:24:33 2015
##################################################

from PyQt4 import Qt
from PyQt4.QtCore import QObject, pyqtSlot
from gnuradio import analog
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import filter
from gnuradio import gr
from gnuradio import qtgui
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from optparse import OptionParser
# import DME
import time
import sys
import threading
import math
# import adsb
import nutaq
import PyQt4.Qwt5 as Qwt
import sip

############## MOD 26/03/2015
import os
import subprocess

from lib_adsb import ADSB_PERSEUS
from lib_dme import ACDME


# import save

os.environ['LD_LIBRARY_PATH'] =
"$LD_LIBRARY_PATH:/opt/Nutaq/ADP6/ADP_MicroTCA/sdk/lib64
/:/usr/local/lib64/:/usr/local/lib/"
#subprocess.check_call(['sqsub', '-np', sys.arg[1], ])
############## MOD 26/03/2015

# TOD0: Add TMS
# TODO: Add ADS-B
# TODO: Show outputs

########### RADIOS' config variables
cal_flag = True #Calibrate at start

## ADSB
adsb_bandwidth = 0 # 0 MHz
adsb_start_fc = 1090e6
adsb_start_band = 0
adsb_start_tx_gain = -13
adsb_start_rx_gain = -8
adsb_start_rx_filter = 2

## DME
dme_bandwidth = 15 # 1.5 MHz
dme_start_fc = 963e6 #1041e6 # Calibration frequency
# dme_calib_fc = 1201e6 #1041e6 # Calibration frequency
Mirabel's freq
dme_start_band = 0
dme_start_tx_gain = -13
dme_start_rx_gain = 18
dme_start_rx_filter = 2

###########

# Receiver amplifier #2 gain limits
Max_RG2 = 30
Min_RG2 = 0
# Receiver amplifier #3 gain limits
Min_RG3 = -13
Max_RG3 = 18

# dme_count = 0         #Number of DME's alredy running

# Threads initialization
tb_adsb_radio=None
tb_dme1_radio=None
tb_dme2_radio=None
AGC1=None
AGC2=None
CAL1=None
CAL2=None

# ID's
const_carrier_address = "192.168.0.101"
const_radio1_id = "radio 1"
const_radio2_id = "radio 2"
const_ADSB_id = "ADSB"
const_DME1_id = "DME1"
const_DME2_id = "DME2"
const_none_id = "none"

##### Fonctions start and stop ######

#### ADS-B threads #####
def adsb_radio():
		global tb_adsb_radio
		tb_adsb_radio= ADSB_PERSEUS(multi_tb,
multi_tb.nutaq_radio420_tx_1, multi_tb.nutaq_radio420_rx_1, "0",
options)
		# (tb_adsb_radio).set_processor_affinity([6])
		(tb_adsb_radio).set_processor_affinity([3])

#### DME, AGC and CAL threads #####
def dme1_radio():
		global tb_dme1_radio
		tb_dme1_radio = ACDME(multi_tb,
multi_tb.nutaq_radio420_tx_1, multi_tb.nutaq_radio420_rx_1, "2",
options)
		# (tb_dme1_radio).set_processor_affinity([5])
		(tb_dme1_radio).set_processor_affinity([2])

def dme2_radio():
		global tb_dme2_radio
		tb_dme2_radio = ACDME(multi_tb,
multi_tb.nutaq_radio420_tx_2, multi_tb.nutaq_radio420_rx_2, "1",
options)
		# (tb_dme2_radio).set_processor_affinity([4])
		(tb_dme2_radio).set_processor_affinity([1])


#### Radio Parameters ####
```

```
def set_radio_params(system, radioTx, radioRx):

    if system == const_ADSB_id:

        radioTx.enable_path()  #Radio needs to be
enabled before changing its params

        radioRx.set_band(adsb_start_band)
        radioRx.set_lpf(adsb_bandwidth)
        radioRx.set_filter(adsb_start_rx_filter)
        radioRx.set_rx_gain2(15)
        radioRx.set_rx_gain3(18)
        radioRx.set_rx_freq(adsb_start_fc)

    else:
        radioTx.enable_path()  #Radio needs to be
enable in order to change its params
        radioRx.enable_path() #Radio needs to be
enable in order to change its params

        # radioRx.set_band(dme_start_band)
        radioRx.set_lpf(dme_bandwidth)
        # radioRx.set_filter(dme_start_rx_filter)
        # radioRx.set_rx_gain2(15)
        radioRx.set_rx_gain3(dme_start_rx_gain)
        radioRx.set_rx_freq(dme_start_fc)

        radioTx.set_lpf(dme_bandwidth)
        radioTx.set_band(dme_start_band)
        radioTx.set_tx_gain(dme_start_tx_gain)
        radioTx.set_tx_freq(dme_start_fc)
    # else:
    #       print "----------------------------Error... System
"+ system + " Non existant"


#### start functions ####
def start_adsb(radio):
    # Radio 1
    if radio == const_radio1_id:
        tb_adsb_radio.radioTx=
    multi_tb.nutaq_radio420_tx_1
    # Radio 2
    elif radio == const_radio2_id:
        tb_adsb_radio.radioTx=
    multi_tb.nutaq_radio420_tx_2

    tb_adsb_radio.set_max_noutput_items(5120)
    tb_adsb_radio.unlock()
    tb_adsb_radio.show()


def start_dme1(radio):
    global tb_dme1_radio, AGC1, CAL1
    # Radio 1
    if radio == const_radio1_id:
        print "selected option Radio1 DME1"

    tb_dme1_radio.set_radio(multi_tb.nutaq_radio420_rx_
1, multi_tb.nutaq_radio420_tx_1)
        # tb_dme1_radio.radioTx=
    multi_tb.nutaq_radio420_tx_1
        # tb_dme1_radio.radioRx=
    multi_tb.nutaq_radio420_rx_1

        print "++ DME Enable status: %s "
%(tb_dme1_radio.DME_interrogator.get_Enable())

    tb_dme1_radio.DME_interrogator.set_Enable(True)
        print "++ DME Enable status: %s "
%(tb_dme1_radio.DME_interrogator.get_Enable())

        AGC1 = threading.Thread(
target=tb_dme1_radio.agc_impl,
args=(multi_tb.nutaq_custom_register_30, .250, 1) )
```

```
        CAL1 = threading.Thread(
target=tb_dme1_radio.BiasCal )

    # Radio 2
    elif radio == const_radio2_id:
        print "selected option Radio2 DME1"

    tb_dme1_radio.set_radio(multi_tb.nutaq_radio420_rx_
2, multi_tb.nutaq_radio420_tx_2)
        # tb_dme1_radio.radioTx=
    multi_tb.nutaq_radio420_tx_2
        # tb_dme1_radio.radioRx=
    multi_tb.nutaq_radio420_rx_2

        print "++ DME Enable status: %s "
%(tb_dme1_radio.DME_interrogator.get_Enable())

    tb_dme1_radio.DME_interrogator.set_Enable(True)
        print "++ DME Enable status: %s "
%(tb_dme1_radio.DME_interrogator.get_Enable())

        AGC1 = threading.Thread(
target=tb_dme1_radio.agc_impl,
args=(multi_tb.nutaq_custom_register_31, .250, 1) )
        CAL1 = threading.Thread(
target=tb_dme1_radio.BiasCal )

    # AGC and CAL start
    tb_dme1_radio.AGC_stop.clear()
    tb_dme1_radio.CAL_stop.clear()
    tb_dme1_radio.DME_interrogator.set_Calibration_Mod
e(True) #Calibration mode
    print "Calibration Mode: %s"
%(tb_dme1_radio.DME_interrogator.get_Calibration_Mode())

    tb_dme1_radio.set_max_noutput_items(5120)
    tb_dme1_radio.unlock()
    tb_dme1_radio.show()
    AGC1.start()
    CAL1.start()


def start_dme2(radio):
    global tb_dme2_radio, AGC2, CAL2
    # Radio 1
    if radio == const_radio1_id:
        print "selected option Radio1 DME2"
        print " selected Radio prev: %s"
%(tb_dme2_radio.radioRx)
        print " selected Radio prev : %s"
%(tb_dme2_radio.radioTx)

    tb_dme2_radio.set_radio(multi_tb.nutaq_radio420_rx_
1, multi_tb.nutaq_radio420_tx_1)
        print " selected Radio : %s"
%(tb_dme2_radio.radioRx)
        print " selected Radio : %s"
%(tb_dme2_radio.radioTx)
        # tb_dme2_radio.radioTx=
    multi_tb.nutaq_radio420_tx_1
        # tb_dme2_radio.radioRx=
    multi_tb.nutaq_radio420_rx_1

        print "++ DME Enable status: %s "
%(tb_dme2_radio.DME_interrogator.get_Enable())

    tb_dme2_radio.DME_interrogator.set_Enable(True)
        print "++ DME Enable status: %s "
%(tb_dme2_radio.DME_interrogator.get_Enable())

        AGC2 = threading.Thread(
target=tb_dme2_radio.agc_impl,
args=(multi_tb.nutaq_custom_register_30, .250, 1) )
        CAL2 = threading.Thread(
target=tb_dme2_radio.BiasCal )
```

```
        # Radio 2
        elif radio == const_radio2_id:
                print "selected option Radio2 DME2"
                print " selected Radio prev: %s"
%(tb_dme2_radio.radioRx)
                print " selected Radio prev : %s"
%(tb_dme2_radio.radioTx)

        tb_dme2_radio.set_radio(multi_tb.nutaq_radio420_rx_
2, multi_tb.nutaq_radio420_tx_2)
                print " selected Radio : %s"
%(tb_dme2_radio.radioRx)
                print " selected Radio : %s"
%(tb_dme2_radio.radioTx)
                # tb_dme2_radio.radioTx=
        multi_tb.nutaq_radio420_tx_2
                # tb_dme2_radio.radioRx=
        multi_tb.nutaq_radio420_rx_2

                print "++ DME Enable status: %s "
%(tb_dme2_radio.DME_interrogator.get_Enable())

        tb_dme2_radio.DME_interrogator.set_Enable(True)
                print "++ DME Enable status: %s "
%(tb_dme2_radio.DME_interrogator.get_Enable())

                AGC2 = threading.Thread(
target=tb_dme2_radio.agc_impl,
args=(multi_tb.nutaq_custom_register_31, .250, 1) )
                CAL2 = threading.Thread(
target=tb_dme2_radio.BiasCal )

        # AGC and CAL start
        tb_dme2_radio.AGC_stop.clear()
        tb_dme2_radio.CAL_stop.clear()
        tb_dme2_radio.DME_interrogator.set_Calibration_Mod
e(True) #Calibration mode
        print "Calibration Mode: %s"
%(tb_dme2_radio.DME_interrogator.get_Calibration_Mode())

        tb_dme2_radio.set_max_noutput_items(5120)
        tb_dme2_radio.unlock()
        tb_dme2_radio.show()
        AGC2.start()
        CAL2.start()



#### Stop functions ####

def stop(System):
        if System == const_ADSB_id:
                stop_adsb()
        elif System == const_DME1_id:
                stop_dme1()
        elif System == const_DME2_id:
                stop_dme2()
        else:
                print "Nothing to Stop... Swtching system"
# def stop_adsb(radio):
def stop_adsb():
        global tb_adsb_radio

        tb_adsb_radio.set_max_noutput_items(1024)
        tb_adsb_radio.lock()
        tb_adsb_radio.hide()
        print "tb_adsb_radio: %s " %{tb_adsb_radio}

# def stop_dme1(radio):
def stop_dme1():
        global tb_dme1_radio, tb_dme1_radio, AGC1, CAL1

        tb_dme1_radio.set_max_noutput_items(1024)
        tb_dme1_radio.lock()
        tb_dme1_radio.DME_interrogator.set_Enable(False)
        tb_dme1_radio.hide()

        tb_dme1_radio.AGC_stop.set()
        tb_dme1_radio.CAL_stop.set()
        AGC1.join(None)
        CAL1.join(None)
        del AGC1
        del CAL1

# def stop_dme2(radio):
def stop_dme2():
        global tb_dme2_radio, AGC2, CAL2

        tb_dme2_radio.set_max_noutput_items(1024)
        tb_dme2_radio.lock()
        tb_dme2_radio.DME_interrogator.set_Enable(False)
        tb_dme2_radio.hide()

        tb_dme2_radio.AGC_stop.set()
        tb_dme2_radio.CAL_stop.set()
        AGC2.join(None)
        CAL2.join(None)
        del AGC2
        del CAL2


#### Updating the register #####
def wr_Register(self, mask_0, mask_1, conf):
        self.nutaq_custom_register_28.set_value(self.nutaq_c
ustom_register_28.get_value() & 15)
        self.nutaq_custom_register_28.set_value(self.nutaq_c
ustom_register_28.get_value() | mask_1)
        self.nutaq_custom_register_28.set_value(self.nutaq_c
ustom_register_28.get_value() & mask_0)
        self.nutaq_custom_register_28.set_value(self.nutaq_c
ustom_register_28.get_value() | conf)
        print "Register 28: %s "
%{self.nutaq_custom_register_28.get_value()}

def change_Register(self, system_1, system_2):
# Taking the old register and updating with the corrects bits
        mask_0 = 255
        mask_1 = 255
        conf = 255

    if system_1 == const_none_id and system_2 ==
const_none_id:
                        mask_0 = 255
                        mask_1 = 0
                        conf = 0

    elif system_1 == const_none_id and system_2 ==
const_ADSB_id:
                        mask_0 = 255
                        mask_1 = 4
                        conf = 16

    elif system_1 == const_none_id and system_2 ==
const_DME2_id:
                        mask_0 = 255
                        mask_1 = 2
                        conf = 32

    elif system_1 == const_none_id and system_2 ==
const_DME1_id:
                        mask_0 = 255
                        mask_1 = 1
                        conf = 48

    elif system_1 == const_ADSB_id and system_2 ==
const_none_id:
                        mask_0 = 251
                        mask_1 = 0
                        conf = 64
    elif system_1 == const_ADSB_id and system_2 ==
const_DME2_id:
                        mask_0 = 251
                        mask_1 = 2
                        conf = 96

    elif system_1 == const_ADSB_id and system_2 ==
const_DME1_id:
                        mask_0 = 251
                        mask_1 = 1
                        conf = 114

    elif system_1 == const_DME2_id and system_2 ==
const_none_id:
                        mask_0 = 253
                        mask_1 = 0
                        conf = 128

    elif system_1 == const_DME2_id and system_2 ==
const_ADSB_id:
                        mask_0 = 253
                        mask_1 = 4
                        conf = 144

    elif system_1 == const_DME2_id and system_2 ==
const_DME1_id:
                        mask_0 = 253
                        mask_1 = 1
                        conf = 176

    elif system_1 == const_DME1_id and system_2 ==
const_none_id:
                        mask_0 = 254
                        mask_1 = 0
                        conf = 192

    elif system_1 == const_DME1_id and system_2 ==
const_ADSB_id:
                        mask_0 = 254
                        mask_1 = 4
                        conf = 208

    elif system_1 == const_DME1_id and system_2 ==
const_DME2_id:
                        mask_0 = 254
                        mask_1 = 2
                        conf = 224

        wr_Register(self, mask_0, mask_1, conf)


#### Radio System ####
def set_Sys_State(previous_Sys, System, radio):
        # Taking the the System and the radio and starting the
correct thread

        # System = ADSB
        if System == const_ADSB_id:
                if previous_Sys== const_DME1_id:
                        stop_dme1()
                        start_adsb(radio)
                elif previous_Sys== const_DME2_id:
                        stop_dme2()
                        start_adsb(radio)
                else:
                        start_adsb(radio)

                previous_system = const_ADSB_id

        #System = DME1
        elif System == const_DME1_id:
                if previous_Sys == const_ADSB_id:
                        stop_adsb()
                        start_dme1(radio)
                elif previous_Sys == const_DME2_id:
                        stop_dme2()
                        start_dme1(radio)
                else:
                        start_dme1(radio)

                previous_system= const_DME1_id

        # System = DME2
        elif System == const_DME2_id:
                if previous_Sys == const_ADSB_id:
                        stop_adsb()
                        start_dme2(radio)
                elif previous_Sys == const_DME1_id:
                        stop_dme1()
                        start_dme2(radio)
                else:
                        start_dme2(radio)

                previous_system = const_DME2_id

        # System = none
        elif System == const_none_id:
                if previous_Sys == const_ADSB_id:
                        stop_adsb()
                elif previous_Sys == const_DME1_id:

                        stop_dme1()
                elif previous_Sys == const_DME2_id:

                        stop_dme2()

                previous_system = const_none_id

        return previous_system

class MULTI(gr.top_block, Qt.QWidget):
# class MULTI(gr.hier_block2, Qt.QWidget):

  def __init__(self):
    gr.top_block.__init__(self, "MULTI")
    Qt.QWidget.__init__(self)
    self.setWindowTitle("Multi SDA LASSENA's App")
    try:
      self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
    except:
      pass
    self.top_scroll_layout = Qt.QVBoxLayout()
    self.setLayout(self.top_scroll_layout)
    self.top_scroll = Qt.QScrollArea()
    self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
    self.top_scroll_layout.addWidget(self.top_scroll)
    self.top_scroll.setWidgetResizable(True)
    self.top_widget = Qt.QWidget()
    self.top_scroll.setWidget(self.top_widget)
    self.top_layout = Qt.QVBoxLayout(self.top_widget)
    self.top_grid_layout = Qt.QGridLayout()
    self.top_layout.addLayout(self.top_grid_layout)



    self.settings = Qt.QSettings("GNU Radio", "adsb_perseus")

self.restoreGeometry(self.settings.value("geometry").toByteArray()
)

    self.UDF_stop = threading.Event()


####################################################
    # Variables (Radios)

####################################################
    self.samp_rate = samp_rate = options.samp_rate #4e6
#default 4e6
```

```
        self.enable = enable = 1
        self.enable2 = enable2 = 1
        self.TXGain = TXGain = options.tx_gain3_cal #-13
        self.TXGain2 = TXGain2 = options.tx_gain3_cal #-13

##################################################
        # Variables (FPGA Selector)

##################################################
        # self.variable_qtgui_label_0 = variable_qtgui_label_0 = 0
        # self.variable_qtgui_entry_0 = variable_qtgui_entry_0 = 0
        self.bit7_check_box = bit7_check_box = False
        self.bit6_check_box = bit6_check_box = False
        self.bit5_check_box = bit5_check_box = False
        self.bit4_check_box = bit4_check_box = False
        self.bit3_check_box = bit3_check_box = False
        self.bit2_check_box = bit2_check_box = False
        self.bit1_check_box = bit1_check_box = False
        self.bit0_check_box = bit0_check_box = False

        # Blocks variables initialization
        self.Radio_1_current = const_none_id
        self.Radio_2_current = const_none_id
        self.Radio_1_previous = const_none_id
        self.Radio_2_previous = const_none_id
        self.variable = variable = ""
        self.variable2 = variable2 = ""
        self.Rx_label = Rx_label = ""
        self.Rx_label2 = Rx_label2 = ""
        self.Tx_Freq = Tx_Freq = 0
        self.Rx_Freq = Rx_Freq = 0
        self.Gain_3 = Gain_3= 0
        self.Gain_2 = Gain_2 = 0
        self.Tx_Freq2 = Tx_Freq2 = 0
        self.Tx_Freq2 = Tx_Freq2 = 0
        self.Rx_Freq2 = Rx_Freq2 = 0
        self.Gain_32 = Gain_32= 0
        self.Gain_22 = Gain_22 = 0


##################################################
        # Blocks

##################################################

        ### First block ###

        #Chooser
        self._System_options = (const_ADSB_id, const_DME1_id,
const_DME2_id, const_none_id, )
        self._System_labels = (str(self._System_options[0]),
str(self._System_options[1]), str(self._System_options[2]),
str(self._System_options[3]) )
        self._System_tool_bar = Qt.QToolBar(self)

self._System_tool_bar.addWidget(Qt.QLabel(const_radio1_id+":
"))
        self._System_combo_box = Qt.QComboBox()
        self._System_tool_bar.addWidget(self._System_combo_box)
        for label in self._System_labels:
self._System_combo_box.addItem(label)
        self._System_callback = lambda i:
self._System_combo_box.setCurrentIndex(self._System_options.i
ndex(i))
        self._System_callback(self.Radio_1_current)
        self._System_combo_box.currentIndexChanged.connect(
                lambda i:
self.set_selected_radio(self._System_options[i],const_radio1_id))
        self.top_grid_layout.addWidget(self._System_tool_bar,
0,0,1,3)


        #Txfreq entry
        self._Tx_Freq_tool_bar = Qt.QToolBar(self)
        self._Tx_Freq_tool_bar.addWidget(Qt.QLabel("Tx Freq"+": "))
        self._Tx_Freq_line_edit = Qt.QLineEdit(str(self.Tx_Freq))
        self._Tx_Freq_tool_bar.addWidget(self._Tx_Freq_line_edit)
        self._Tx_Freq_line_edit.returnPressed.connect(
                lambda:
self.set_Tx_Freq(int(self._Tx_Freq_line_edit.text().toAscii())))
        self.top_grid_layout.addWidget(self._Tx_Freq_tool_bar,
1,2,1,2)

        #Rxfreq entry
        self._Rx_Freq_tool_bar = Qt.QToolBar(self)
        self._Rx_Freq_tool_bar.addWidget(Qt.QLabel("Rx Freq"+":
"))
        self._Rx_Freq_line_edit = Qt.QLineEdit(str(self.Rx_Freq))
        self._Rx_Freq_tool_bar.addWidget(self._Rx_Freq_line_edit)
        self._Rx_Freq_line_edit.returnPressed.connect(
                lambda:
self.set_Rx_Freq(int(self._Rx_Freq_line_edit.text().toAscii())))
        self.top_grid_layout.addWidget(self._Rx_Freq_tool_bar,
1,4,1,2)

        #Check box
        _enable_check_box = Qt.QCheckBox("ON/OFF")
        self._enable_choices = {True: 1, False: 0}
        self._enable_choices_inv = dict((v,k) for k,v in
self._enable_choices.iteritems())
        self._enable_callback = lambda i:
Qt.QMetaObject.invokeMethod(_enable_check_box,
"setChecked", Qt.Q_ARG("bool", self._enable_choices_inv[i]))
        self._enable_callback(self.enable)
        _enable_check_box.stateChanged.connect(lambda i:
self.set_enable(self._enable_choices[bool(i)]))
        self.top_grid_layout.addWidget(_enable_check_box,0,3,1,3)

        #Calibration button
        _Calibration_push_button = Qt.QPushButton("Calibration")
        self._Calibration_choices = {'Pressed': 1, 'Released': 0}
        _Calibration_push_button.pressed.connect(lambda:
self.set_Calibration(self._Calibration_choices['Pressed']))
        # _Calibration_push_button.released.connect(lambda:
self.set_Calibration(self._Calibration_choices['Released']))

self.top_grid_layout.addWidget(_Calibration_push_button,0,5,1,3)

        #Slider
        self._TXGain_layout = Qt.QVBoxLayout()
        self._TXGain_tool_bar = Qt.QToolBar(self)
        self._TXGain_layout.addWidget(self._TXGain_tool_bar)
        self._TXGain_tool_bar.addWidget(Qt.QLabel("TX Gain"+": "))
        class qwt_counter_pyslot(Qwt.QwtCounter):
            def __init__(self, parent=None):
                Qwt.QwtCounter.__init__(self, parent)
            @pyqtSlot('double')
            def setValue(self, value):
                super(Qwt.QwtCounter, self).setValue(value)
        self._TXGain_counter = qwt_counter_pyslot()
        self._TXGain_counter.setRange(-13, 18, 1)
        self._TXGain_counter.setNumButtons(1)
        self._TXGain_counter.setValue(self.TXGain)
        self._TXGain_tool_bar.addWidget(self._TXGain_counter)

self._TXGain_counter.valueChanged.connect(self.set_TXGain)
        self._TXGain_slider = Qwt.QwtSlider(None, Qt.Qt.Horizontal,
Qwt.QwtSlider.BottomScale, Qwt.QwtSlider.BgSlot)
        self._TXGain_slider.setRange(-13, 18, 1)
        self._TXGain_slider.setValue(self.TXGain)
        self._TXGain_slider.setMinimumWidth(10)
        self._TXGain_slider.valueChanged.connect(self.set_TXGain)
        self._TXGain_layout.addWidget(self._TXGain_slider)
        self.top_grid_layout.addLayout(self._TXGain_layout,2,0,2,8)

        #Gain 3 entry
        self._Gain_3_tool_bar = Qt.QToolBar(self)
        self._Gain_3_tool_bar.addWidget(Qt.QLabel("Rx :        Gain
3"+": "))
        self._Gain_3_line_edit = Qt.QLineEdit(str(self.Gain_3))
        self._Gain_3_tool_bar.addWidget(self._Gain_3_line_edit)
        self._Gain_3_line_edit.returnPressed.connect(
                lambda:
self.set_Gain_3(int(self._Gain_3_line_edit.text().toAscii())))
        self.top_grid_layout.addWidget(self._Gain_3_tool_bar,
3,1,2,2)

        #Gain 2 entry
        self._Gain_2_tool_bar = Qt.QToolBar(self)
        self._Gain_2_tool_bar.addWidget(Qt.QLabel("Gain 2"+": "))
        self._Gain_2_line_edit = Qt.QLineEdit(str(self.Gain_2))
        self._Gain_2_tool_bar.addWidget(self._Gain_2_line_edit)
        self._Gain_2_line_edit.returnPressed.connect(
                lambda:
self.set_Gain_2(int(self._Gain_2_line_edit.text().toAscii())))

self.top_grid_layout.addWidget(self._Gain_2_tool_bar,3,3,2,2)

        #Label
        self._variable_tool_bar = Qt.QToolBar(self)
        self._variable_tool_bar.addWidget(Qt.QLabel("Sampling
Freq"+": "))
        self._variable_label = Qt.QLabel(str(self.variable))
        self._variable_tool_bar.addWidget(self._variable_label)

self.top_grid_layout.addWidget(self._variable_tool_bar,3,5,2,2)


        ### Second block ###

        #Chooser
        self._System2_options = (const_ADSB_id, const_DME1_id,
const_DME2_id, const_none_id, )
        self._System2_labels = (str(self._System2_options[0]),
str(self._System2_options[1]), str(self._System2_options[2]),
str(self._System2_options[3]) )
        self._System2_tool_bar = Qt.QToolBar(self)
        self._System2_tool_bar.addWidget(Qt.QLabel("Radio 2"+":
"))
        self._System2_combo_box = Qt.QComboBox()

self._System2_tool_bar.addWidget(self._System2_combo_box)
        for label in self._System2_labels:
self._System2_combo_box.addItem(label)
        self._System2_callback = lambda i:
self._System2_combo_box.setCurrentIndex(self._System2_option
s.index(i))
        self._System2_callback(self.Radio_2_current)
        self._System2_combo_box.currentIndexChanged.connect(
                lambda i:
self.set_selected_radio(self._System2_options[i],
const_radio2_id))

self.top_grid_layout.addWidget(self._System2_tool_bar,5,0,1,3)

        #Txfreq entry
        self._Tx_Freq2_tool_bar = Qt.QToolBar(self)
        self._Tx_Freq2_tool_bar.addWidget(Qt.QLabel("Tx Freq"+":
"))
        self._Tx_Freq2_line_edit = Qt.QLineEdit(str(self.Tx_Freq2))

self._Tx_Freq2_tool_bar.addWidget(self._Tx_Freq2_line_edit)
        self._Tx_Freq2_line_edit.returnPressed.connect(
                lambda:
self.set_Tx_Freq2(int(self._Tx_Freq2_line_edit.text().toAscii()))

self.top_grid_layout.addWidget(self._Tx_Freq2_tool_bar,6,2,1,2)

        #Rxfreq entry
        self._Rx_Freq2_tool_bar = Qt.QToolBar(self)
        self._Rx_Freq2_tool_bar.addWidget(Qt.QLabel("Rx Freq"+":
"))
        self._Rx_Freq2_line_edit = Qt.QLineEdit(str(self.Rx_Freq2))

self._Rx_Freq2_tool_bar.addWidget(self._Rx_Freq2_line_edit)
```

```
        self._Rx_Freq2_line_edit.returnPressed.connect(
                lambda:
self.set_Rx_Freq2(int(self._Rx_Freq2_line_edit.text().toAscii())))

self.top_grid_layout.addWidget(self._Rx_Freq2_tool_bar,6,4,1,2)

        #Checkbox
        _enable2_check_box = Qt.QCheckBox("ON/OFF")
        self._enable2_choices = {True: 1, False: 0}
        self._enable2_choices_inv = dict((v,k) for k,v in
self._enable2_choices.iteritems())
        self._enable2_callback = lambda i:
Qt.QMetaObject.invokeMethod(_enable2_check_box,
"setChecked", Qt.Q_ARG("bool", self._enable2_choices_inv[i]))
        self._enable2_callback(self.enable2)
        _enable2_check_box.stateChanged.connect(lambda i:
self.set_enable2(self._enable2_choices[bool(i)]))
        self.top_grid_layout.addWidget(_enable2_check_box,5,3,1,3)

        #Calibration button
        _Calibration2_push_button = Qt.QPushButton("Calibration")
        self._Calibration2_choices = {'Pressed': 1, 'Released': 0}
        _Calibration2_push_button.pressed.connect(lambda:
self.set_Calibration2(self._Calibration2_choices['Pressed']))
        # _Calibration2_push_button.released.connect(lambda:
self.set_Calibration2(self._Calibration2_choices['Released']))

self.top_grid_layout.addWidget(_Calibration2_push_button,5,5,1,3
)

        #QT slider
        self._TXGain2_layout = Qt.QVBoxLayout()
        self._TXGain2_tool_bar = Qt.QToolBar(self)
        self._TXGain2_layout.addWidget(self._TXGain2_tool_bar)
        self._TXGain2_tool_bar.addWidget(Qt.QLabel("TX Gain"+":
"))
        class qwt_counter_pyslot(Qwt.QwtCounter):
            def __init__(self, parent=None):
                Qwt.QwtCounter.__init__(self, parent)
            @pyqtSlot('double')
            def setValue(self, value):
                super(Qwt.QwtCounter, self).setValue(value)
        self._TXGain2_counter = qwt_counter_pyslot()
        self._TXGain2_counter.setRange(-13, 18, 1)
        self._TXGain2_counter.setNumButtons(1)
        self._TXGain2_counter.setValue(self.TXGain)
        self._TXGain2_tool_bar.addWidget(self._TXGain2_counter)

self._TXGain2_counter.valueChanged.connect(self.set_TXGain2)
        self._TXGain2_slider = Qwt.QwtSlider(None,
Qt.Qt.Horizontal, Qwt.QwtSlider.BottomScale,
Qwt.QwtSlider.BgSlot)
        self._TXGain2_slider.setRange(-13, 18, 1)
        self._TXGain2_slider.setValue(self.TXGain2)
        self._TXGain2_slider.setMinimumWidth(10)

self._TXGain2_slider.valueChanged.connect(self.set_TXGain2)
        self._TXGain2_layout.addWidget(self._TXGain2_slider)
        self.top_grid_layout.addLayout(self._TXGain2_layout,7,0,2,8)

        #Gain 3 entry
        self._Gain_32_tool_bar = Qt.QToolBar(self)
        self._Gain_32_tool_bar.addWidget(Qt.QLabel("Rx :    Gain
3"+": "))
        self._Gain_32_line_edit = Qt.QLineEdit(str(self.Gain_32))
        self._Gain_32_tool_bar.addWidget(self._Gain_32_line_edit)
        self._Gain_32_line_edit.returnPressed.connect(
                lambda:
self.set_Gain_32(int(self._Gain_32_line_edit.text().toAscii())))

self.top_grid_layout.addWidget(self._Gain_32_tool_bar,8,1,2,2)

        #Gain 2 entry
        self._Gain_22_tool_bar = Qt.QToolBar(self)
        self._Gain_22_tool_bar.addWidget(Qt.QLabel("Gain 2"+": "))
        self._Gain_22_line_edit = Qt.QLineEdit(str(self.Gain_22))
        self._Gain_22_tool_bar.addWidget(self._Gain_22_line_edit)
        self._Gain_22_line_edit.returnPressed.connect(
                lambda:
self.set_Gain_22(int(self._Gain_22_line_edit.text().toAscii())))

self.top_grid_layout.addWidget(self._Gain_22_tool_bar,8,3,2,2)

        #Label
        self._variable2_tool_bar = Qt.QToolBar(self)
        self._variable2_tool_bar.addWidget(Qt.QLabel("Sampling
Freq"+": "))
        self._variable2_label = Qt.QLabel(str(self.variable2))
        self._variable2_tool_bar.addWidget(self._variable2_label)

self.top_grid_layout.addWidget(self._variable2_tool_bar,8,5,2,2)

        #Label
        self._Rx_label2_tool_bar = Qt.QToolBar(self)
        self._Rx_label2_tool_bar.addWidget(Qt.QLabel("FPGA's
selector"))
        self._Rx_label2_label = Qt.QLabel(str(self.Rx_label2))
        self._Rx_label2_tool_bar.addWidget(self._Rx_label2_label)

self.top_grid_layout.addWidget(self._Rx_label2_tool_bar,9,1,2,5)


        ## Register 28 Bits manual ctrl
        # Check box
        bit7_check_box = Qt.QCheckBox("Bit7")
        self.bit7_check_box_choices = {True: 1, False: 0}
        self.bit7_check_box_choices_inv = dict((v,k) for k,v in
self.bit7_check_box_choices.iteritems())
        self.bit7_check_box_callback = lambda i:
Qt.QMetaObject.invokeMethod(bit7_check_box, "setChecked",
Qt.Q_ARG("bool", self.bit7_check_box_choices_inv[i]))
        self.bit7_check_box_callback(self.enable)
        bit7_check_box.stateChanged.connect(lambda i:
self.push_bit(self.bit7_check_box_choices[bool(i)],7,bit7_check_bo
x))
        self.top_grid_layout.addWidget(bit7_check_box,10,0,2,1)

        #Check box
        bit6_check_box = Qt.QCheckBox("Bit6")
        self.bit6_check_box_choices = {True: 1, False: 0}
        self.bit6_check_box_choices_inv = dict((v,k) for k,v in
self.bit6_check_box_choices.iteritems())
        self.bit6_check_box_callback = lambda i:
Qt.QMetaObject.invokeMethod(bit6_check_box, "setChecked",
Qt.Q_ARG("bool", self.bit6_check_box_choices_inv[i]))
        self.bit6_check_box_callback(self.enable)
        bit6_check_box.stateChanged.connect(lambda i:
self.push_bit(self.bit6_check_box_choices[bool(i)],6,bit6_check_bo
x))
        self.top_grid_layout.addWidget(bit6_check_box,10,1,2,1)

        #Check box
        bit5_check_box = Qt.QCheckBox("Bit5")
        self.bit5_check_box_choices = {True: 1, False: 0}
        self.bit5_check_box_choices_inv = dict((v,k) for k,v in
self.bit5_check_box_choices.iteritems())
        self.bit5_check_box_callback = lambda i:
Qt.QMetaObject.invokeMethod(bit5_check_box, "setChecked",
Qt.Q_ARG("bool", self.bit5_check_box_choices_inv[i]))
        self.bit5_check_box_callback(self.enable)
        bit5_check_box.stateChanged.connect(lambda i:
self.push_bit(self.bit5_check_box_choices[bool(i)],5,bit5_check_bo
x))
        self.top_grid_layout.addWidget(bit5_check_box,10,2,2,1)

        #Check box
        bit4_check_box = Qt.QCheckBox("Bit4")
        self.bit4_check_box_choices = {True: 1, False: 0}
        self.bit4_check_box_choices_inv = dict((v,k) for k,v in
self.bit4_check_box_choices.iteritems())
```

```
        self.bit4_check_box_callback = lambda i:
Qt.QMetaObject.invokeMethod(bit4_check_box, "setChecked",
Qt.Q_ARG("bool", self.bit4_check_box_choices_inv[i]))
        self.bit4_check_box_callback(self.enable)
        bit4_check_box.stateChanged.connect(lambda i:
self.push_bit(self.bit4_check_box_choices[bool(i)],4,bit4_check_bo
x))
        self.top_grid_layout.addWidget(bit4_check_box,10,3,2,1)

        #Check box
        bit3_check_box = Qt.QCheckBox("Bit3")
        self.bit3_check_box_choices = {True: 1, False: 0}
        self.bit3_check_box_choices_inv = dict((v,k) for k,v in
self.bit3_check_box_choices.iteritems())
        self.bit3_check_box_callback = lambda i:
Qt.QMetaObject.invokeMethod(bit3_check_box, "setChecked",
Qt.Q_ARG("bool", self.bit3_check_box_choices_inv[i]))
        self.bit3_check_box_callback(self.enable)
        bit3_check_box.stateChanged.connect(lambda i:
self.push_bit(self.bit3_check_box_choices[bool(i)],3,bit3_check_bo
x))
        self.top_grid_layout.addWidget(bit3_check_box,10,4,2,1)

        #Check box
        bit2_check_box = Qt.QCheckBox("Bit2")
        self.bit2_check_box_choices = {True: 1, False: 0}
        self.bit2_check_box_choices_inv = dict((v,k) for k,v in
self.bit2_check_box_choices.iteritems())
        self.bit2_check_box_callback = lambda i:
Qt.QMetaObject.invokeMethod(bit2_check_box, "setChecked",
Qt.Q_ARG("bool", self.bit2_check_box_choices_inv[i]))
        self.bit2_check_box_callback(self.enable)
        bit2_check_box.stateChanged.connect(lambda i:
self.push_bit(self.bit2_check_box_choices[bool(i)],2,bit2_check_bo
x))
        self.top_grid_layout.addWidget(bit2_check_box,10,5,2,1)

        #Check box
        bit1_check_box = Qt.QCheckBox("Bit1")
        self.bit1_check_box_choices = {True: 1, False: 0}
        self.bit1_check_box_choices_inv = dict((v,k) for k,v in
self.bit1_check_box_choices.iteritems())
        self.bit1_check_box_callback = lambda i:
Qt.QMetaObject.invokeMethod(bit1_check_box, "setChecked",
Qt.Q_ARG("bool", self.bit1_check_box_choices_inv[i]))
        self.bit1_check_box_callback(self.enable)
        bit1_check_box.stateChanged.connect(lambda i:
self.push_bit(self.bit1_check_box_choices[bool(i)],1,bit1_check_bo
x))
        self.top_grid_layout.addWidget(bit1_check_box,10,6,2,1)

        #Check box
        bit0_check_box = Qt.QCheckBox("Bit0")
        self.bit0_check_box_choices = {True: 1, False: 0}
        self.bit0_check_box_choices_inv = dict((v,k) for k,v in
self.bit0_check_box_choices.iteritems())
        self.bit0_check_box_callback = lambda i:
Qt.QMetaObject.invokeMethod(bit0_check_box, "setChecked",
Qt.Q_ARG("bool", self.bit0_check_box_choices_inv[i]))
        self.bit0_check_box_callback(self.enable)
        bit0_check_box.stateChanged.connect(lambda i:
self.push_bit(self.bit0_check_box_choices[bool(i)],0,bit0_check_bo
x))
        self.top_grid_layout.addWidget(bit0_check_box,10,7,2,1)


##################################################
        # Nutaq

##################################################
        # Nutaq's Carrier
        self.nutaq_carrier_perseus_0 =
nutaq.carrier(0,"nutaq_carrier_perseus_0", const_carrier_address)
```

```
        #
##################################################
        # # Connections
        #
##################################################

        ##### DME Radios
        self.nutaq_radio420_tx_1 =
nutaq.radio420_tx("nutaq_carrier_perseus_0", 1, 2)
        #~ self.nutaq_radio420_tx_1.calibrate_pll()

        self.nutaq_radio420_tx_1.set_default_enable(0)
        # self.nutaq_radio420_tx_1.set_default_tx_freq(fc_Tx)
        self.nutaq_radio420_tx_1.set_default_reference(0)
        self.nutaq_radio420_tx_1.set_default_datarate(samp_rate*2)
        self.nutaq_radio420_tx_1.set_default_calibrate(cal_flag)
        # self.nutaq_radio420_tx_1.set_default_band(0)
        self.nutaq_radio420_tx_1.set_default_update_rate(10000)
        self.nutaq_radio420_tx_1.set_default_tx_vga1_gain(-10) #-35
-4
        self.nutaq_radio420_tx_1.set_default_tx_vga2_gain(3) # 0 25
        self.nutaq_radio420_tx_1.set_default_tx_gain3(TXGain) #-13
18
        #
self.nutaq_radio420_tx_1.set_default_tx_lpf_bandwidth(bandwidth
)
        # (self.nutaq_radio420_tx_1).set_processor_affinity([4,5])
        # (self.nutaq_radio420_tx_1).set_processor_affinity([4, 5, 6,
7])
        # (self.nutaq_radio420_tx_1).set_thread_priority(20)

        self.nutaq_radio420_rx_1 =
nutaq.radio420_rx("nutaq_carrier_perseus_0", 1,2)
        self.nutaq_radio420_rx_1.set_default_enable(0)
        # self.nutaq_radio420_rx_1.set_default_rx_freq(fc_Rx)
        self.nutaq_radio420_rx_1.set_default_reference(0)
        self.nutaq_radio420_rx_1.set_default_datarate(samp_rate*2)
        self.nutaq_radio420_rx_1.set_default_calibrate(cal_flag)
        # self.nutaq_radio420_rx_1.set_default_band(0)
        self.nutaq_radio420_rx_1.set_default_update_rate(10000)
        self.nutaq_radio420_rx_1.set_default_rx_lna_gain(3)
        self.nutaq_radio420_rx_1.set_default_rx_vga1_gain(3)
        self.nutaq_radio420_rx_1.set_default_rx_gain2(15)
        self.nutaq_radio420_rx_1.set_default_rx_gain3(18)
        self.nutaq_radio420_rx_1.set_default_rx_rf_filter(2)
        #
self.nutaq_radio420_rx_1.set_default_rx_lpf_bandwidth(bandwidth
)
        # self.nutaq_radio420_rx_1.set_default_ref_clk_ctrl(0)
        # self.nutaq_radio420_rx_1.set_default_rf_ctrl(0)
        # self.nutaq_radio420_rx_1.set_default_rx_gain_ctrl(0)
#ASSIGN GAIN CONTROL TO HOST OR FPGA
        # self.nutaq_radio420_rx_1.set_default_pll_cpld_ctrl(0)
        # (self.nutaq_radio420_rx_0).set_processor_affinity([4,5])
        # (self.nutaq_radio420_rx_1).set_processor_affinity([4, 5, 6,
7])
        # (self.nutaq_radio420_rx_1).set_thread_priority(20)

        ## Radio 2
        self.nutaq_radio420_tx_2 =
nutaq.radio420_tx("nutaq_carrier_perseus_0", 2, 2)
        self.nutaq_radio420_tx_2.set_default_enable(0)
        # self.nutaq_radio420_tx_2.set_default_tx_freq(fc_Tx)
        self.nutaq_radio420_tx_2.set_default_reference(0)
        self.nutaq_radio420_tx_2.set_default_datarate(samp_rate*2)
        self.nutaq_radio420_tx_2.set_default_calibrate(cal_flag)
        # self.nutaq_radio420_tx_2.set_default_band(0)
        self.nutaq_radio420_tx_2.set_default_update_rate(10000)
        self.nutaq_radio420_tx_2.set_default_tx_vga1_gain(-10) #-35
-4
        self.nutaq_radio420_tx_2.set_default_tx_vga2_gain(3) # 0 25
        self.nutaq_radio420_tx_2.set_default_tx_gain3(TXGain2) #-
13 18
        #
self.nutaq_radio420_tx_2.set_default_tx_lpf_bandwidth(bandwidth
)
       # (self.nutaq_radio420_tx_2).set_processor_affinity([4,5])
       # (self.nutaq_radio420_tx_2).set_processor_affinity([4, 5, 6,
7])
                # (self.nutaq_radio420_tx_2).set_thread_priority(20)

        self.nutaq_radio420_rx_2 =
nutaq.radio420_rx("nutaq_carrier_perseus_0", 2,2)
        self.nutaq_radio420_rx_2.set_default_enable(0)
        # self.nutaq_radio420_rx_2.set_default_rx_freq(fc_Rx)
        self.nutaq_radio420_rx_2.set_default_reference(0)
        self.nutaq_radio420_rx_2.set_default_datarate(samp_rate*2)
        self.nutaq_radio420_rx_2.set_default_calibrate(cal_flag)
        # self.nutaq_radio420_rx_2.set_default_band(0)
        self.nutaq_radio420_rx_2.set_default_update_rate(10000)
        self.nutaq_radio420_rx_2.set_default_rx_lna_gain(3)
        self.nutaq_radio420_rx_2.set_default_rx_vga1_gain(3)
        self.nutaq_radio420_rx_2.set_default_rx_gain2(15)
        self.nutaq_radio420_rx_2.set_default_rx_gain3(18)
        self.nutaq_radio420_rx_2.set_default_rx_rf_filter(2)
        #
self.nutaq_radio420_rx_2.set_default_rx_lpf_bandwidth(bandwidth
)
        # self.nutaq_radio420_rx_2.set_default_ref_clk_ctrl(0)
        # self.nutaq_radio420_rx_2.set_default_rf_ctrl(0)
        # self.nutaq_radio420_rx_2.set_default_rx_gain_ctrl(0)
#ASSIGN GAIN CONTROL TO HOST OR FPGA
        # self.nutaq_radio420_rx_2.set_default_pll_cpld_ctrl(0)
       # (self.nutaq_radio420_rx_2).set_processor_affinity([4,5])
       # (self.nutaq_radio420_rx_2).set_processor_affinity([4, 5, 6,
7])
                # (self.nutaq_radio420_rx_2).set_thread_priority(20)


##################################################
        # Custom Registers

##################################################




        #SRC_Select
        self.nutaq_custom_register_1 =
nutaq.custom_register("nutaq_carrier_perseus_0",6)
        self.nutaq_custom_register_1.set_index(1)
        self.nutaq_custom_register_1.set_default_value(6)
        self.nutaq_custom_register_1.set_update_rate(100000)
#
(self.nutaq_custom_register_1).set_processor_affinity([4,5])
        # (self.nutaq_custom_register_1).set_processor_affinity([4, 5,
6, 7])
        # (self.nutaq_custom_register_1).set_thread_priority(20)

        #MIMO Write
        self.nutaq_custom_register_3 =
nutaq.custom_register("nutaq_carrier_perseus_0",6)
        self.nutaq_custom_register_3.set_index(3)
        self.nutaq_custom_register_3.set_default_value(1)
        self.nutaq_custom_register_3.set_update_rate(100000)
#
(self.nutaq_custom_register_3).set_processor_affinity([4,5])
        # (self.nutaq_custom_register_3).set_processor_affinity([4, 5,
6, 7])
        # (self.nutaq_custom_register_3).set_thread_priority(20)

        #MIMO Sync Select
        self.nutaq_custom_register_4 =
nutaq.custom_register("nutaq_carrier_perseus_0",6)
        self.nutaq_custom_register_4.set_index(4)
        self.nutaq_custom_register_4.set_default_value(0)
        self.nutaq_custom_register_4.set_update_rate(100000)
#
(self.nutaq_custom_register_4).set_processor_affinity([4,5])
        # (self.nutaq_custom_register_4).set_processor_affinity([4, 5,
6, 7])
      # (self.nutaq_custom_register_4).set_thread_priority(20)

#           #Master_Reset
#           self.nutaq_custom_register_10 =
nutaq.custom_register("nutaq_carrier_perseus_0",6)
#           self.nutaq_custom_register_10.set_index(10)
#           self.nutaq_custom_register_10.set_default_value(0)
#           self.nutaq_custom_register_10.set_update_rate(10000)
# #
(self.nutaq_custom_register_10).set_processor_affinity([4,5])
#
(self.nutaq_custom_register_10).set_processor_affinity([4, 5, 6, 7])
#           (self.nutaq_custom_register_10).set_thread_priority(20)

      #AGC_Threshold
      self.nutaq_custom_register_16 =
nutaq.custom_register("nutaq_carrier_perseus_0",6)
      self.nutaq_custom_register_16.set_index(16)
      self.nutaq_custom_register_16.set_default_value(2500000)
      self.nutaq_custom_register_16.set_update_rate(1000000)
#
(self.nutaq_custom_register_16).set_processor_affinity([4,5])
      # (self.nutaq_custom_register_16).set_processor_affinity([4,
5, 6, 7])
      # (self.nutaq_custom_register_16).set_thread_priority(20)

      #Gain RX Selector
#           self.nutaq_custom_register_17 =
nutaq.custom_register("nutaq_carrier_perseus_0",6)
#           self.nutaq_custom_register_17.set_index(17)
#           self.nutaq_custom_register_17.set_default_value(25000)
#           self.nutaq_custom_register_17.set_update_rate(10000)
# #
(self.nutaq_custom_register_17).set_processor_affinity([4,5])
#
(self.nutaq_custom_register_17).set_processor_affinity([4, 5, 6, 7])
#           (self.nutaq_custom_register_17).set_thread_priority(20)

#           # Gain RX Selector
#           self.nutaq_custom_register_17 =
nutaq.custom_register("nutaq_carrier_perseus_0",6)
#           self.nutaq_custom_register_17.set_index(19)
#           self.nutaq_custom_register_17.set_default_value(4)
#           self.nutaq_custom_register_17.set_update_rate(10000)
# #
(self.nutaq_custom_register_17).set_processor_affinity([4,5])
#
(self.nutaq_custom_register_17).set_processor_affinity([4, 5, 6, 7])
#           (self.nutaq_custom_register_17).set_thread_priority(20)

      #AGC_Enable
      self.nutaq_custom_register_18 =
nutaq.custom_register("nutaq_carrier_perseus_0",6)
      self.nutaq_custom_register_18.set_index(18)
      self.nutaq_custom_register_18.set_default_value(1)
      self.nutaq_custom_register_18.set_update_rate(100000)
#
(self.nutaq_custom_register_18).set_processor_affinity([4,5])
      # (self.nutaq_custom_register_18).set_processor_affinity([4,
5, 6, 7])
      # (self.nutaq_custom_register_18).set_thread_priority(20)

      #AGC_Update_rate_multiplier
      self.nutaq_custom_register_19 =
nutaq.custom_register("nutaq_carrier_perseus_0",6)
      self.nutaq_custom_register_19.set_index(19)

self.nutaq_custom_register_19.set_default_value(1*4)#decimation
_factor)
      self.nutaq_custom_register_19.set_update_rate(1000000)
#
(self.nutaq_custom_register_19).set_processor_affinity([4,5])
      # (self.nutaq_custom_register_19).set_processor_affinity([4,
5, 6, 7])
      # (self.nutaq_custom_register_19).set_thread_priority(20)


      # R28
      self.nutaq_custom_register_28 =
nutaq.custom_register("nutaq_carrier_perseus_0",6)
      self.nutaq_custom_register_28.set_index(28)
      self.nutaq_custom_register_28.set_default_value(255)
      self.nutaq_custom_register_28.set_update_rate(100000)

      #AGC_MaxPw
      self.nutaq_custom_register_30 =
nutaq.custom_register("nutaq_carrier_perseus_0",6)
      self.nutaq_custom_register_30.set_index(30)
      # self.nutaq_custom_register_19.set_default_value(1)
      self.nutaq_custom_register_30.set_update_rate(10000)
#
(self.nutaq_custom_register_19).set_processor_affinity([4,5])
      # (self.nutaq_custom_register_30).set_processor_affinity([4,
5, 6, 7])
      # (self.nutaq_custom_register_30).set_thread_priority(20)

      #AGC_Gain_Ctrl_Val
      self.nutaq_custom_register_31 =
nutaq.custom_register("nutaq_carrier_perseus_0",6)
      self.nutaq_custom_register_31.set_index(31)
      # self.nutaq_custom_register_19.set_default_value(1)
      self.nutaq_custom_register_31.set_update_rate(10000)
#
(self.nutaq_custom_register_19).set_processor_affinity([4,5])
      # (self.nutaq_custom_register_31).set_processor_affinity([4,
5, 6, 7])
      # (self.nutaq_custom_register_31).set_thread_priority(20)



              #############################################
#############
              # Def
              #############################################
#############

   ### System Chooser boxes ###
   def set_selected_radio(self, System, radio):
              self.lock()

              if radio == const_radio1_id :
                            if self.Radio_2_current == System:
                                          stop(System)
                                          self.Radio_2_current =
const_none_id
                                          self.Radio_2_previous =
const_none_id

                            ## DEBUG ##
                     print "PREVIOUS_R1:" + self.Radio_1_previous
                     print "CURRENT_R1:" + self.Radio_1_current
                     ###########

              set_radio_params(System, self.nutaq_radio420_tx_1,
self.nutaq_radio420_rx_1)
              self.Radio_1_previous =
set_Sys_State(self.Radio_1_previous,System,radio)
              self.Radio_1_current = System

      elif radio == const_radio2_id:
              if self.Radio_1_current == System:
                            stop(System)
                            self.Radio_1_current = const_none_id
                            self.Radio_1_previous = const_none_id

              ## DEBUG ##
```

```
                        print "PREVIOUS_R2:" + self.Radio_2_previous
                        print "CURRENT_R2:" + self.Radio_2_current
                        ###########

                set_radio_params(System, self.nutaq_radio420_tx_2,
self.nutaq_radio420_rx_2)
                # previous_system = self.Radio_2_previous
                self.Radio_2_previous =
set_Sys_State(self.Radio_2_previous,System,radio)
                self.Radio_2_current = System

        else:
                print "ERROR: invalid radio ID"

        change_Register(self,
self.Radio_1_current,self.Radio_2_current)

        self.unlock()

                ### TxGain sliders ###

    def get_TXGain(self):
        return self.TXGain

    def set_TXGain(self, TXGain):
        self.TXGain = TXGain
        Qt.QMetaObject.invokeMethod(self._TXGain_counter,
"setValue", Qt.Q_ARG("double", self.TXGain))
        Qt.QMetaObject.invokeMethod(self._TXGain_slider,
"setValue", Qt.Q_ARG("double", self.TXGain))
        self.nutaq_radio420_tx_1.set_tx_gain(int(self.TXGain))


    def get_TXGain2(self):
        return self.TXGain2

    def set_TXGain2(self, TXGain2):
        self.TXGain2 = TXGain2
        Qt.QMetaObject.invokeMethod(self._TXGain2_counter,
"setValue", Qt.Q_ARG("double", self.TXGain2))
        Qt.QMetaObject.invokeMethod(self._TXGain2_slider,
"setValue", Qt.Q_ARG("double", self.TXGain2))
        self.nutaq_radio420_tx_2.set_tx_gain(int(self.TXGain2))

                ### Reg28 bit checkboxes
    def push_bit(self,val,pos,cb):
                                self.cb = val
                                bit= 1 << pos
                                print
self.nutaq_custom_register_28.get_value()

                                if val:
                                                print "val 1"

                self.nutaq_custom_register_28.set_value(self.nutaq_c
ustom_register_28.get_value() | bit)
                                                #
self._Tx_Freq_line_edit.setText(str("ENEMIGO"))

                                else:
                                                print "val 0"

                self.nutaq_custom_register_28.set_value(self.nutaq_c
ustom_register_28.get_value() & ~bit)
                                                #
self._Tx_Freq_line_edit.setText(str("AMIGO"))

                                print
self.nutaq_custom_register_28.get_value()

                ### Enable checkboxes ###
    def set_enable(self, enable):
        self.enable = enable

        if enable:
            self.nutaq_radio420_tx_1.enable_path()
            print("Radio 1 (ON)")

        else:
            self.nutaq_radio420_tx_1.disable_path()
            print("Radio 1 (OFF)")

        self._enable_callback(self.enable)


    def set_enable2(self, enable2):
        self.enable2 = enable2

        if enable2:
            self.nutaq_radio420_tx_2.enable_path()
            print("Radio 2 (ON)")

        else:
            self.nutaq_radio420_tx_2.disable_path()
            print("Radio 2 (OFF)")

        self._enable2_callback(self.enable2)



                ### Gain 3 entries ###
    def get_Gain_3(self):
        return self.Gain3

    def set_Gain_3(self, Gain_3):
        self.Gain_3= Gain_3

    def get_Gain_32(self):
        return self.Gain32

    def set_Gain_32(self, Gain_32):
        self.Gain_32= Gain_32


    ### Gain 2 entries ###
    def get_Gain_2(self):
        return self.Gain2

    def set_Gain_2(self, Gain_2):
        self.Gain_2= Gain_2

    def get_Gain_22(self):
        return self.Gain22

    def set_Gain_22(self, Gain_22):
        self.Gain_22= Gain_22


                #TxFreq entries
    def get_Tx_Freq(self):
        return self.Tx_Freq

    def set_Tx_Freq(self, Tx_Freq):
        self.Tx_Freq= Tx_Freq

    def get_Tx_Freq2(self):
        return self.Tx_Freq2

    def set_Tx_Freq2(self, Tx_Freq2):
        self.Tx_Freq2= Tx_Freq2


                #RxFreq entries
    def get_Rx_Freq(self):
        return self.Rx_Freq

    def set_Rx_Freq(self, Rx_Freq):
        self.Rx_Freq= Rx_Freq

    def get_Rx_Freq2(self):
        return self.Rx_Freq2

    def set_Rx_Freq2(self, Rx_Freq2):
        self.Rx_Freq2= Rx_Freq2

                ### Calibration Buttons
    def get_Calibration(self):
        return self.Calibration

    def set_Calibration(self, Calibration):
        self.Calibration = Calibration
        self.nutaq_radio420_rx_1.set_default_calibrate(True)
        print "entro"
        #~ self.nutaq_radio420_tx_1.init()
        self.nutaq_radio420_rx_1.calibrate_pll()

        #~ self.nutaq_radio420_tx_1.calibrate_vga()
        #~ int calibrate_lpf(double ref_freq);                //Leave this
commented.

    def get_Calibration2(self):
        return self.Calibration2

    def set_Calibration2(self, Calibration2):
        self.Calibration2 = Calibration2
        self.nutaq_radio420_tx_2.calibrate_pll()

                ### Updating the gains and freqs values
    def update_fields(self):
                        while (not self.UDF_stop.is_set() ):

                                if self.Radio_1_current ==
const_DME1_id:

                self._Gain_3_line_edit.setText(eng_notation.num_to_s
tr(tb_dme1_radio.get_rx_gain3()))

                self._Gain_2_line_edit.setText(eng_notation.num_to_s
tr(tb_dme1_radio.get_rx_gain2()))
                                                #
self._Gain_32_line_edit.setText(eng_notation.num_to_str(tb_dme
2_radio.get_rx_gain3()))
                                                #
self._Gain_22_line_edit.setText(eng_notation.num_to_str(tb_dme
2_radio.get_rx_gain2()))
                                if self.Radio_2_current ==
const_DME1_id:

                self._Gain_32_line_edit.setText(eng_notation.num_to_
str(tb_dme1_radio.get_rx_gain3()))

                self._Gain_22_line_edit.setText(eng_notation.num_to_
str(tb_dme1_radio.get_rx_gain2()))
                                                #
self._Gain_3_line_edit.setText(eng_notation.num_to_str(tb_dme2
_radio.get_rx_gain3()))
                                                #
self._Gain_2_line_edit.setText(eng_notation.num_to_str(tb_dme2
_radio.get_rx_gain2()))
                                if self.Radio_1_current ==
const_DME2_id:

                self._Gain_3_line_edit.setText(eng_notation.num_to_s
tr(tb_dme2_radio.get_rx_gain3()))

                self._Gain_2_line_edit.setText(eng_notation.num_to_s
tr(tb_dme2_radio.get_rx_gain2()))
                                                #
self._Gain_32_line_edit.setText(eng_notation.num_to_str(tb_dme
1_radio.get_rx_gain3()))
                                                #
self._Gain_22_line_edit.setText(eng_notation.num_to_str(tb_dme
1_radio.get_rx_gain2()))
                                if self.Radio_2_current ==
const_DME2_id:
```

```
                self._Gain_32_line_edit.setText(eng_notation.num_to_
str(tb_dme2_radio.get_rx_gain3()))

                self._Gain_22_line_edit.setText(eng_notation.num_to_
str(tb_dme2_radio.get_rx_gain2()))
                                                #
self._Gain_3_line_edit.setText(eng_notation.num_to_str(tb_dme1
_radio.get_rx_gain3()))
                                                #
self._Gain_2_line_edit.setText(eng_notation.num_to_str(tb_dme1
_radio.get_rx_gain2()))


                self._Tx_Freq_line_edit.setText(eng_notation.num_to_
str(self.nutaq_radio420_tx_1.get_tx_freq()))

                self._Tx_Freq2_line_edit.setText(eng_notation.num_to
_str(self.nutaq_radio420_tx_2.get_tx_freq()))

                self._Rx_Freq_line_edit.setText(eng_notation.num_to_
str(self.nutaq_radio420_rx_1.get_rx_freq()))

                self._Rx_Freq2_line_edit.setText(eng_notation.num_to
_str(self.nutaq_radio420_rx_2.get_rx_freq()))

                                        time.sleep(.9)


#########################################################
#########################################################
#####
# Main definition
#########################################################
#########################################################
#####

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"

    parser = OptionParser(option_class=eng_option,
usage="%prog: [options]")

    parser.add_option("-V", "--VOR", type="float",
                help="Select VOR frequency. Default is 108.0",
default="108.0")
    parser.add_option("-s", "--samp_rate", type="float",
                help="Select sampling rate in Hz. Default is 4 MHz",
default="4e6")
    parser.add_option("-T", "--tx_gain3_cal", type="int",
                help="Select gain3 of the transmitter during
calibration in dB. Default is -13 dB", default="-13")
    parser.add_option("-t", "--tx_gain3", type="int",
                help="Select gain3 of the transmitter after calibration
in dB. Default is -13 dB", default="-13")
    parser.add_option("-R", "--rx_gain3", type="int",
                help="Select initial rx_gain3 of the receiver in dB.
Default is -13 dB", default="-13")
    parser.add_option("-r", "--rx_gain2", type="int",
                help="Select initial rx_gain2 of the receiver in dB.
Default is 0 dB", default="0")
    parser.add_option("-v", action="store_true", dest="scopes",
                help="Show time domain I/O signals. Default is not
to show", default=False)
    parser.add_option("-f", action="store_true", dest="fft",
                help="Show frequency domain input signal. Default
is not to show", default=False)
    parser.add_option("-H", action="store_true", dest="hist",
                    help="Show histogram of range measured. Default is
not to show", default=False)
    parser.add_option("-o", "--rangefile", type="string",
                    help="Filename where you want to log the measured
ranges. Default is /dev/null", default="/dev/null")
    parser.add_option("-p", "--platform", type="string",
                    help="Platform to be run in. Options are: pico, zepto,
or null. Default is pico", default="pico")

    (options, args) = parser.parse_args()

    if gr.enable_realtime_scheduling() != gr.RT_OK:
        print "Error: failed to enable realtime scheduling."

Qt.QApplication.setGraphicsSystem(gr.prefs().get_string('qtgui','st
yle','raster'))
    qapp = Qt.QApplication(sys.argv)


    #Create MULTI tb
    multi_tb = MULTI()
    multi_tb.start()

    # Creating all the threads

    adsb_radio()
    dme1_radio()
    dme2_radio()

    tb_adsb_radio.start(1024)
    tb_adsb_radio.lock()

    tb_dme1_radio.start(1024)
    tb_dme1_radio.lock()

    tb_dme2_radio.start(1024)
    tb_dme2_radio.lock()

    UDF = threading.Thread( target=multi_tb.update_fields,args=() )
    # AUF = threading.Thread(
target=tb_adsb_radio.adsb_update_fields,args=() )

    UDF.start()
    # AUF.start()

    # Stopping AGC and CAL threads
    # tb_dme1_radio.AGC_stop.set()
    # tb_dme1_radio.CAL_stop.set()

    # Checking the reg28 value
    print "Register 28: %s "
%(multi_tb.nutaq_custom_register_28.get_value())
    multi_tb.nutaq_custom_register_28.set_value(255)
    multi_tb.show()
    print "Register 28: %s "
%(multi_tb.nutaq_custom_register_28.get_value())



                  # #########################################
 #    ## Single SDA correct execution order ##

 #    tb_dme2_radio1.start()
 #    time.sleep(2)

 #    multi_tb.lock()

 #    set_radio_params(multi_tb.System,
multi_tb.nutaq_radio420_tx_1, multi_tb.nutaq_radio420_rx_1)

 #    tb_dme2_radio1.lock()
 #    # tb_dme2_radio1.radioTx=    multi_tb.nutaq_radio420_tx_2
 #    # tb_dme2_radio1.radioRx=    multi_tb.nutaq_radio420_rx_2
 #    tb_dme2_radio1.DME_interrogator.set_Enable(True)
```

```
 #    tb_dme2_radio1.unlock()

 #    tb_dme2_radio1.show()
 #    AGC2 = threading.Thread( target=tb_dme2_radio1.agc_impl,
args=(multi_tb.nutaq_custom_register_30, .250, 1) )
 #    CAL2 = threading.Thread( target=tb_dme2_radio1.BiasCal )
 #    AGC2.start()
 #    CAL2.start()

 #    # multi_tb.nutaq_custom_register_28.set_value(128)
 #    multi_tb.nutaq_custom_register_28.set_value(34)

 #    multi_tb.unlock

 #    #####################################
 #    #####################################


  #   while(1):
                                    # print "Register 28: %s "
%{multi_tb.nutaq_custom_register_28.get_value()}

    #Start Nutaq's ADP CLI
    # subprocess.Popen(['xterm'])

    # # Calibration thread
    # CAL = threading.Thread( target=tb1.BiasCal )
    # CAL.start()
    # CAL2 = threading.Thread( target=tb2.BiasCal )
    # CAL2.start()

    # Wait for end
    # tb1.show() # Show QT interface
    # tb2.show()
    def quitting():


        multi_tb.UDF_stop.set()

        tb_adsb_radio.stop()
        tb_dme1_radio.stop()
        tb_dme2_radio.stop()
        multi_tb.stop()
        tb_adsb_radio.wait()
        tb_dme1_radio.wait()
        tb_dme2_radio.wait()
        multi_tb.wait()

    qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
    qapp.exec_()

        # #Start Nutaq's ADP CLI
        # subprocess.Popen(['xterm'])

    # Exit
    # print '++ AGC stop event set'
    # tb1.AGC_stop.set()
    # tb2.AGC_stop.set()
    # AGC.join(None)
    tb_dme2_radio.AGC_stop.set()
    tb_dme2_radio.CAL_stop.set()
    tb_dme1_radio.AGC_stop.set()
    tb_dme1_radio.CAL_stop.set()
    tb_adsb_radio.AUF_stop.set()
    # AGC2.join(None)
    # CAL2.join(None)
    # CAL1.join(None)
    # multi_tb.nutaq_radio420_tx_1.set_default_enable(0)
    # multi_tb.nutaq_radio420_tx_1.set_default_enable(0)
    multi_tb.nutaq_radio420_tx_1.reset()
    multi_tb.nutaq_radio420_tx_2.reset()
    multi_tb.nutaq_radio420_rx_1.reset()
    multi_tb.nutaq_radio420_rx_2.reset()

    # print '++ CLI stop event set'
# tb_dme1_radio.CLI_stop.set()

# tb2.CLI_stop.set()
# print '++ CAL stop event set'
# tb1.CAL_stop.set()
# tb2.CAL_stop.set()
# print '++'
# print '++ Please: Press Enter to close the CLI and exit'
# tb_dme1_radio.CLI.join(None)
# tb2.CLI.join(None)
# CAL.join(None)
# CAL2.join(None)
UDF.join(None)
```

```
print '++'
print '+'*67 + '\n'

tb_adsb_radio = None
tb_dme1_radio = None
tb_dme2_radio = None
multi_tb = None
init_tb = None
AGC1=None
AGC2=None
CAL1=None
CAL2=None
```

APPENDIX III

DME SDA Python’s Library

```
#!/usr/bin/env python
############################################
######
# Gnuradio Python Flow Graph
# Title: DME
# Author: Omar Yeste
# Description: LASSENA
# Generated: Tue Jun  4 09:28:54 2013
############################################
######

from PyQt4 import Qt
from PyQt4.QtCore import QObject, pyqtSlot
from gnuradio import analog
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import filter
from gnuradio import gr
from gnuradio import qtgui
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from optparse import OptionParser
import DME
import time
import sys
import threading
import math
import adsb
import nutaq
import PyQt4.Qwt5 as Qwt
import sip
############## MOD 26/03/2015
import os
import subprocess


bandwidth = 15 # 1.5 MHz
# calib_fc = 1201e6 #1041e6 # Calibration frequency
Mirabel's freq
calib_fc = 963e6 #1041e6 # Calibration frequency

# Receiver amplifier #2 gain limits
Max_RG2 = 30
Min_RG2 = 0
# Receiver amplifier #3 gain limits
Min_RG3 = -13
Max_RG3 = 18
decimation_factor = 4

cal_flag = False #Calibrate at start

def showprogress(progress):
    if progress == 0: sys.stdout.write("/\b")
    if progress == 1: sys.stdout.write("-\b")
    if progress == 2: sys.stdout.write("\\\b")
    if progress == 3: sys.stdout.write("|\b")
    progress += 1
    if progress == 4: progress = 0

    sys.stdout.flush()
    time.sleep(0.2)
    return progress
```

```
def showprogress_2(progress):
    sys.stdout.write( "++ Set Bias: %s\r" %(progress) )
    sys.stdout.flush()
    time.sleep(0.05)

class ACDME(gr.top_block, Qt.QWidget):
    # def __init__(self, radioTx, radioRx, samp_rate,
options):
    def __init__(self, main, radioTx, radioRx, rtdexID,
options):
        gr.top_block.__init__(self, "Auto-Calibrated DME")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Auto-Calibrated DME")
        try:

self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-
grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()

self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout =
Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)
        self.radioTx = radioTx = radioTx
        self.radioRx = radioRx = radioRx
        self.options = options = options
        self.main = main = main
        self.rtdexID = rtdexID = rtdexID

        self.settings = Qt.QSettings("GNU Radio",
"ACDME")

self.restoreGeometry(self.settings.value("geometry").to
ByteArray())

############################################
#########################
############################################
#########################
############################  DME
##################################
############################################
#########################
############################################
#########################


############################################
######
        # Events

############################################
```

```
######
        self.CLI = threading.Thread(
target=self.CommandLineInterface )


############################################
######
        # Events

############################################
######
        self.AGC_stop = threading.Event()
        self.CLI_stop = threading.Event()
        self.CAL_stop = threading.Event()
        self.PTH_stop = threading.Event()


############################################
######
        # Variables

############################################
######
        self.variable_qtgui_label_0 =
variable_qtgui_label_0 = 0
        self.variable_qtgui_entry_0 =
variable_qtgui_entry_0 = 0



        self.samp_rate = samp_rate = main.samp_rate
        self.dme_samp_freq = dme_samp_freq =
samp_rate/decimation_factor
        self.bandwidth = bandwidth
        self.fc_Rx = fc_Rx = calib_fc
        self.fc_Tx = fc_Tx = calib_fc #1041e6
        self.G_Tx = G_Tx = options.tx_gain3_cal
        self.calib_fc = calib_fc
        self.calib_reseted = False
        self.Mode = Mode = 'X'
        self.Channel = Channel = None  #default for
calibration
        self.VOR = VOR = options.VOR
        self.previous_VOR = options.VOR
        self.rx_gain2 = rx_gain2 = options.rx_gain2
        self.rx_gain3 = rx_gain3 = options.rx_gain3
        self.agcwindow = 100e-3 #seconds
        self.AGC_updaterate = self.agcwindow
        self.IOScale = IOScale = 2**11-1
        self.filter_taps =
filter.firdes.low_pass_2(3,samp_rate,dme_samp_freq/4
,dme_samp_freq/4,60,5,6.76)


############################################
######
        # Message Queues

############################################
######
        DMEmessage_out = DMEmessage_in =
gr.msg_queue(2)


############################################
######
        # Knobs

############################################
######


############################################
######
        # Blocks
```

```
############################################
######

        #VOR entry
        self._VOR_tool_bar = Qt.QToolBar(self)
        self._VOR_tool_bar.addWidget(Qt.QLabel("VOR
= "))
        self._VOR_line_edit = Qt.QLineEdit(str(self.VOR))

self._VOR_tool_bar.addWidget(self._VOR_line_edit)
        self._VOR_line_edit.returnPressed.connect(
            lambda:
self.set_channel(float(self._VOR_line_edit.text().toAscii
())))

self.top_grid_layout.addWidget(self._VOR_tool_bar,8,1
,2,2)


        # if options.platform == "pico" :

############################################
######
            # Tx

############################################
######

        self.nutaq_sink =
nutaq.rtdex_sink("nutaq_carrier_perseus_0",gr.sizeof_
short,1,0)
        self.nutaq_sink.set_type(0)

self.nutaq_sink.set_packet_size(8960)#8960#8192
        self.nutaq_sink.set_channels(rtdexID)#"1")
        # (self.nutaq_sink).set_processor_affinity([5])
           # (self.nutaq_sink).set_processor_affinity([4, 5,
6, 7])
        (self.nutaq_sink).set_thread_priority(99)


############################################
######
        # Rx

############################################
######

        self.nutaq_source =
nutaq.rtdex_source("nutaq_carrier_perseus_0",gr.size
of_short,1,0)
        self.nutaq_source.set_type(0)
        self.nutaq_source.set_packet_size(8960)
        self.nutaq_source.set_channels(rtdexID)#"1")
        # (self.nutaq_source).set_processor_affinity([5])
           # (self.nutaq_source).set_processor_affinity([4,
5, 6, 7])
        (self.nutaq_source).set_thread_priority(98)

#           # Capture Data at the ADC
#           self.source_capture =
nutaq.rtdex_source("nutaq_carrier_perseus_0",gr.size
of_short,1,5)
#           self.source_capture.set_type(0)
#           self.source_capture.set_packet_size(8960)
#           self.source_capture.set_channels("0")
            #
(self.source_capture).set_processor_affinity([5])
            #
(self.source_capture).set_processor_affinity([4, 5, 6,
7])
#           (self.source_capture).set_thread_priority(99)

#           # Capture file
#            self.blocks_file_sink_0 =
blocks.file_sink(gr.sizeof_short*1, "/home/rhsg/AVIO-
505/logs/IFR_capture.bit", False)
#            self.blocks_file_sink_0.set_unbuffered(False)
#            self.connect((self.source_capture, 0),
(self.blocks_file_sink_0, 0))



        # elif options.platform == "null" :
        #     self.source =
blocks.null_source(gr.sizeof_short*1)
        #     self.sink = blocks.null_sink(gr.sizeof_short*1)



###############################################
######
        # DME
###############################################
######
        self.DME_interrogator =
DME.dmeint_ff(dme_samp_freq)
        #
(self.DME_interrogator).set_processor_affinity([5])
        #
(self.DME_interrogator).set_processor_affinity([4, 5, 6,
7])
        (self.DME_interrogator).set_thread_priority(97)
        print '++'
        print '+'*67

        ## Output formatting
        self.scaler_Tx =
blocks.multiply_const_vff((IOScale, ))
        # (self.scaler_Tx).set_processor_affinity([5])
       # (self.scaler_Tx).set_processor_affinity([4, 5, 6,
7])
        # (self.scaler_Tx).set_thread_priority(99)
        self.float_to_complex_Tx =
blocks.float_to_complex(1)
        #
(self.float_to_complex_Tx).set_processor_affinity([5])
        #
(self.float_to_complex_Tx).set_processor_affinity([4, 5,
6, 7])
        #
(self.float_to_complex_Tx).set_thread_priority(99)
        self.complex_to_interleaved_short_Tx =
blocks.complex_to_interleaved_short()
        #
(self.complex_to_interleaved_short_Tx).set_processor
_affinity([5])
       #
(self.complex_to_interleaved_short_Tx).set_processor
_affinity([4, 5, 6, 7])
        #
(self.complex_to_interleaved_short_Tx).set_thread_pri
ority(99)

        ## Input formatting
        self.interleaved_short_to_complex_Rx =
blocks.interleaved_short_to_complex(False)
        #
(self.interleaved_short_to_complex_Rx).set_processor
_affinity([5])
       #
(self.interleaved_short_to_complex_Rx).set_processor
_affinity([4, 5, 6, 7])
        #
(self.interleaved_short_to_complex_Rx).set_thread_pri
ority(99)
        # self.scaler_Rx =
blocks.multiply_const_vcc((1.0/IOScale, ))
        self.scaler_Rx =
blocks.multiply_const_vff((1.0/(IOScale**2), ))
        # (self.scaler_Rx).set_processor_affinity([5])
       # (self.scaler_Rx).set_processor_affinity([4, 5, 6,
7])
        # (self.scaler_Rx).set_thread_priority(99)
        self.complex_to_mag_squared_Rx =
blocks.complex_to_mag_squared(1)
        #
(self.complex_to_mag_squared_Rx).set_processor_aff
inity([5])
       #
(self.complex_to_mag_squared_Rx).set_processor_aff
inity([4, 5, 6, 7])
        #
(self.complex_to_mag_squared_Rx).set_thread_priorit
y(99)

        ## Resampler to make it compatible with higher
sampling Freq.  #Decimator
        self.rational_resampler_xxx_0 =
filter.rational_resampler_ccc(
                interpolation=1,
                decimation=decimation_factor,
                taps=None,
                fractional_bw=None,
        )
        # Interpolator
        self.rational_resampler_xxx_0_0 =
filter.rational_resampler_ccc(
                interpolation=decimation_factor,
                decimation=1,
                taps=(self.filter_taps),
                fractional_bw=None,
        )
        #
(self.rational_resampler_xxx_0_0).set_processor_affini
ty([5])

        # AGC
        # self.agc =
nutaq.agc_maxhold_f(int(self.agcwindow*dme_samp_f
req), 0.9, 0.3, 5) # Variability margin of 5 dB
(Up=3*Low)
# #        (self.agc).set_processor_affinity([7])
#        (self.agc).set_processor_affinity([4, 5, 6, 7])
#        (self.agc).set_thread_priority(20)


###############################################
######
        # Connections
###############################################
######
        # Tx
        self.connect((self.DME_interrogator, 0),
(self.scaler_Tx, 0))
        self.connect((self.scaler_Tx, 0),
(self.float_to_complex_Tx, 0))
        self.connect((self.float_to_complex_Tx, 0),
(self.rational_resampler_xxx_0_0, 0))
        self.connect((self.rational_resampler_xxx_0_0, 0),
(self.complex_to_interleaved_short_Tx, 0))

self.connect((self.complex_to_interleaved_short_Tx,
0), (self.nutaq_sink, 0))
        #
self.connect((self.complex_to_interleaved_short_Tx,
0), (self.main.nutaq_sink, 0))

        # Rx
        # self.connect((self.main.nutaq_source, 0),
(self.interleaved_short_to_complex_Rx, 0))
        self.connect((self.nutaq_source, 0),
(self.interleaved_short_to_complex_Rx, 0))

self.connect((self.interleaved_short_to_complex_Rx,
0), (self.rational_resampler_xxx_0 , 0))
        # self.connect((self.rational_resampler_xxx_0, 0),
(self.scaler_Rx, 0))
        self.connect((self.rational_resampler_xxx_0, 0),
(self.complex_to_mag_squared_Rx, 0))
        self.connect((self.complex_to_mag_squared_Rx,
0), (self.scaler_Rx, 0))
        self.connect((self.scaler_Rx, 0),
(self.DME_interrogator, 0))
        #
self.connect((self.complex_to_mag_squared_Rx, 0),
(self.InputRec_file, 0))#Abdu
        # AGC
        #
self.connect((self.complex_to_mag_squared_Rx, 0),
(self.agc, 0))


##############################################
######
        # Instruments

##############################################
######
        self.qtgui_tab_widget = Qt.QTabWidget()

        # self._variable_qtgui_label_0_tool_bar =
Qt.QToolBar(self)
        #
self._variable_qtgui_label_0_tool_bar.addWidget(Qt.Q
Label("dme_samp_freq*2"+": "))
        # self._variable_qtgui_label_0_label =
Qt.QLabel(str(self.variable_qtgui_label_0))
        #
self._variable_qtgui_label_0_tool_bar.addWidget(self._
variable_qtgui_label_0_label)
        #
self.top_layout.addWidget(self._variable_qtgui_label_0
_tool_bar)
        self._variable_qtgui_entry_0_tool_bar =
Qt.QToolBar(self)

self._variable_qtgui_entry_0_tool_bar.addWidget(Qt.Q
Label("DME Id"+": "))
        self._variable_qtgui_entry_0_line_edit =
Qt.QLineEdit(str(self.variable_qtgui_entry_0))

self._variable_qtgui_entry_0_tool_bar.addWidget(self._
variable_qtgui_entry_0_line_edit)

self._variable_qtgui_entry_0_line_edit.setText(eng_not
ation.num_to_str(self.DME_interrogator.get_Id()))
        #
self._variable_qtgui_entry_0_line_edit.returnPressed.c
onnect(
            # lambda:
self.set_variable_qtgui_entry_0(int(self.s_variable_qtgu
i_entry_0_line_edit.text().toAscii())))

self.top_layout.addWidget(self._variable_qtgui_entry_0
_tool_bar)

        if options.hist:
            self.Range_tab_widget = Qt.QWidget()
            self.Range_tab_widget_layout =
Qt.QBoxLayout(Qt.QBoxLayout.TopToBottom,
self.Range_tab_widget)
            self.Range_tab_widget_grid_layout =
Qt.QGridLayout()

self.Range_tab_widget_layout.addLayout(self.Range_t
ab_widget_grid_layout)
```

```
self.qtgui_tab_widget.addTab(self.Range_tab_widget,
"Range")

            self.Range_Hist = qtgui.histogram_sink_f(
                300,   # Frame size (inputs required)
                20,    # Number of bins
                0,     # Initial min. value
                100,   # Initial max. value
                "Histogram",
                1      # Number of inputs
            )
            self.Range_Hist.set_update_time(25)
            self.Range_Hist.set_line_label(0, "Range
Measurements")
            self.Range_Hist.enable_menu(True)
            #
(self.Range_Hist).set_processor_affinity([4,5])
            (self.Range_Hist).set_thread_priority(1)

            self._Range_Hist_win =
sip.wrapinstance(self.Range_Hist.pyqwidget(),
Qt.QWidget)

self.Range_tab_widget_layout.addWidget(self._Range
_Hist_win)

            self.connect((self.DMEmessage_source, 0),
(self.Range_Hist, 0))

            self.Scope_Range = qtgui.time_sink_f(
                80, # Frame size (inputs required)
                1,  # Sampling rate
                "Time Evolution",
                1   # Number of inputs
            )
            self.Scope_Range.set_update_time(5)
            self.Scope_Range.set_y_axis(0, 100)
            self.Scope_Range.set_line_label(0, "Range
Measurements")
#           self.Scope_Range.set_y_label("Range",
"nmi")
            #
(self.Scope_Range).set_processor_affinity([4,5])
            (self.Scope_Range).set_thread_priority(1)

            self._Scope_Range_win =
sip.wrapinstance(self.Scope_Range.pyqwidget(),
Qt.QWidget)

self.Range_tab_widget_layout.addWidget(self._Scope
_Range_win)

            self.connect((self.DMEmessage_source, 0),
(self.Scope_Range, 0))

        if options.scopes:
            self.IO_tab_widget = Qt.QWidget()
            self.IO_tab_widget_layout =
Qt.QBoxLayout(Qt.QBoxLayout.TopToBottom,
self.IO_tab_widget)
            self.IO_tab_widget_grid_layout =
Qt.QGridLayout()

self.IO_tab_widget_layout.addLayout(self.IO_tab_widg
et_grid_layout)

self.qtgui_tab_widget.addTab(self.IO_tab_widget,
"Time Domain")

            self.Scope_Tx_Rx = qtgui.time_sink_f(
                1024,      # Frame size (inputs required)
                dme_samp_freq, # Sampling rate
                "I/O DME", # Title
                2          # Number of inputs
            )
            self.Scope_Tx_Rx.set_update_time(0.5)
            self.Scope_Tx_Rx.set_y_axis(0, 2)

self.Scope_Tx_Rx.set_trigger_mode(qtgui.TRIG_MOD
E_NORM, qtgui.TRIG_SLOPE_POS, 0.45,
512/dme_samp_freq, 1, "")
#           self.Scope_Tx_Rx.set_y_label("Amplitude", "")
#Incompatible with gnuradio3.7.3
            #
(self.Scope_Tx_Rx).set_processor_affinity([4,5])
            (self.Scope_Tx_Rx).set_thread_priority(1)

            self._Scope_Tx_Rx_win =
sip.wrapinstance(self.Scope_Tx_Rx.pyqwidget(),
Qt.QWidget)

self.IO_tab_widget_layout.addWidget(self._Scope_Tx_
Rx_win)

            self.connect((self.DME_interrogator, 0),
(self.Scope_Tx_Rx, 0))
            self.Scope_Tx_Rx.set_line_label(0, "Output")
            self.connect((self.scaler_Rx, 0),
(self.Scope_Tx_Rx, 1))
            self.Scope_Tx_Rx.set_line_label(1, "Input")

        if options.fft:
            self.Freq_tab_widget = Qt.QWidget()
            self.Freq_tab_widget_layout =
Qt.QBoxLayout(Qt.QBoxLayout.TopToBottom,
self.Freq_tab_widget)
            self.Freq_tab_widget_grid_layout =
Qt.QGridLayout()

self.Freq_tab_widget_layout.addLayout(self.Freq_tab_
widget_grid_layout)

self.qtgui_tab_widget.addTab(self.Freq_tab_widget,
"Frequency Domain")

            self.FFT_Rx = qtgui.freq_sink_c(
                1024,                   # Frame size (inputs
required)
                firdes.WIN_BLACKMAN_hARRIS, # FFT
Window
                0,                      # fc
                dme_samp_freq,                # bw
                "Received Signal",         # Title
                1                       # Number of inputs
            )
            self.FFT_Rx.set_update_time(2)
            self.FFT_Rx.set_y_axis(-140, -20)
            self.FFT_Rx.set_thread_priority(20)
            # (self.FFT_Rx).set_processor_affinity([4,5])
            (self.FFT_Rx).set_thread_priority(1)

            self._FFT_Rx_win =
sip.wrapinstance(self.FFT_Rx.pyqwidget(),
Qt.QWidget)

self.Freq_tab_widget_layout.addWidget(self._FFT_Rx_
win)

            self.connect((self.scaler_Rx, 0), (self.FFT_Rx,
0))
            self.FFT_Rx.set_line_label(0, "Input")

        self.top_layout.addWidget(self.qtgui_tab_widget)


    # Getting and setting values
    def get_samp_rate(self):
        return self.samp_rate
```

```
    def get_rx_gain2(self):
        return self.rx_gain2

    def set_rx_gain2(self, gain2):
        self.radioRx.set_rx_gain2(gain2)
        self.rx_gain2 = int(round(gain2/3)*3)

    def get_rx_gain3(self):
        return self.rx_gain3

    def set_rx_gain3(self, gain3):
        self.radioRx.set_rx_gain3(gain3)
        self.rx_gain3 = gain3

    def set_radio(self, radio_rx, radio_tx):
        self.radioRx = radio_rx
        self.radioTx = radio_tx


    # Print datas when the channel changes
    def print_mode(self):

        print '+'*67
        print '++'
        print '++ - R420 Sampling frequency = %0.2f
MHz' % self.samp_rate
        print '++ - DME Sampling frequency = %0.2f
MHz' % self.dme_samp_freq
        print '++ - Mode =', self.Mode, ' Channel =',
self.Channel
        print '++ - VOR frequency = %0.2f MHz' %
float(self.VOR)
        print '++ - Response RF = %(#)0.0f MHz' % {"#":
self.fc_Rx/1e6}
        print '++ - Interrogation RF = %(#)0.0f MHz' %
{"#": self.fc_Tx/1e6}
        print '++'
        print '+'*67


self._VOR_line_edit.setText(eng_notation.num_to_str(
self.VOR))

    # Changing the channel asfunction of the VOR
    def set_channel(self, VOR):
        # VOR Frequency
        self.VOR = VOR
        VOR = float(VOR)
        channel = self.Channel


self.main._variable2_label.setText(eng_notation.num_t
o_str(self.samp_rate))

self.main._variable_label.setText(eng_notation.num_to
_str(self.samp_rate))

        if self.calib_reseted:
            print "    ERROR: Change channel is fobidden
during calibration"
        else:
            # Set the channel
            if VOR == 0: self.Channel = 0
            elif 108.0 <= VOR and VOR <= 112.25:
self.Channel = int(math.floor(VOR*10)) - 1063
            elif 112.3 <= VOR and VOR <= 117.95:
self.Channel = int(math.floor(VOR*10)) - 1053
            # Not paired:
            elif 133.3 <= VOR and VOR <= 134.25:
self.Channel = int(math.floor(VOR*10)) - 1273
            elif 134.4 <= VOR and VOR <= 135.85:
self.Channel = int(math.floor(VOR*10)) - 1343
            else: self.Channel = channel

            if self.Channel != channel:
                # Set the mode and thus the frequencies #
VOR = 0 for Calibration
                fc_Tx = (self.Channel + 1024) * 1e6
                if VOR == 0:
                    self.Mode = 'Cal'
                    fc_Rx = self.calib_fc
                    fc_Tx = self.calib_fc
                    self.DME_interrogator.set_Mode(True) #
Only Mode X has equal Tx/Rx Pulse spacing
                elif VOR*10 == math.floor(VOR*10):
                    self.Mode = 'X'
                    if self.Channel <= 63: fc_Rx = fc_Tx -
63e6
                    if self.Channel > 63: fc_Rx = fc_Tx + 63e6
                    if not(self.DME_interrogator.get_Mode()):
                        self.DME_interrogator.set_Mode(True)
                    self.previous_VOR = VOR
                else:
                    self.Mode = 'Y'
                    if self.Channel <= 63: fc_Rx = fc_Tx +
63e6
                    if self.Channel > 63: fc_Rx = fc_Tx - 63e6
                    if self.DME_interrogator.get_Mode():
                        self.DME_interrogator.set_Mode(False)
                    self.previous_VOR = VOR

                # Set the frequencies
                if (self.fc_Tx != fc_Tx):
                    self.fc_Tx = fc_Tx
                    if self.options.platform in {"pico", "zepto"} :
                        self.radioTx.set_tx_freq(self.fc_Tx)
                if (self.fc_Rx != fc_Rx):
                    self.fc_Rx = fc_Rx
                    if self.options.platform in {"pico", "zepto"} :
                        self.radioRx.set_rx_freq(self.fc_Rx)

                # Show summary
                self.print_mode()

            else:
                print "    ERROR: Wrong VOR, please enter
a correct VOR frequency"

    def agc_impl(self, measured_power_cr,
update_rate, mode):
        print "++"
        print "++ Starting AGC ..."
        increase = 0
        error = 0
        integral = 0
        rx_gain2 = 0
        rx_gain3 = 0

        if self.options.platform in {"pico", "zepto"} :
            while
not(self.main.nutaq_carrier_perseus_0.is_peripheral_r
eseted(1)):
                time.sleep(0.5)
            while (not self.AGC_stop.is_set()):
                ###### DEBUG
                # time.sleep(.5)
                # print "Power: %d"
%(measured_power_cr.get_value())
                # print "AGC_MODE: %d" %(mode)
                ######
                ######

                if mode == 1 :        #Direct assing Table

                    past_ctrl_val = ( max(0,min(63,
measured_power_cr.get_value() ) ) )
                    if past_ctrl_val < 32 :
                        rx_gain3 = past_ctrl_val -13
                        rx_gain2 = 0
                    else :
```

```
                        rx_gain3 = Max_RG3
                        rx_gain2 = past_ctrl_val -31

                    #~ self.radioRx.set_rx_gain3(rx_gain3)
                    self.set_rx_gain3(rx_gain3)
                    self.set_rx_gain2(rx_gain2)
                    #~ self.radioRx.set_rx_gain2(rx_gain2)

                    time.sleep(update_rate)

                elif mode == 2 :        #Increase based Gain
calculation

                    self.agc.reset_increase()
                    time.sleep(self.AGC_updaterate)
                    increase = int(0.5*increase +
0.6*self.agc.get_increase())

                    if increase > 0:
                        if rx_gain3 < Max_RG3:
                            rx_gain3 = min(Max_RG3, rx_gain3 +
max(1,increase))
                        elif rx_gain2 < Max_RG2:
                            rx_gain2 = min(Max_RG2, rx_gain2 +
max(1,increase))
                        self.radioRx.set_rx_gain3(rx_gain3)
                        self.radioRx.set_rx_gain2(rx_gain2)
                        time.sleep(self.AGC_updaterate)
                    elif increase < 0:
                        if rx_gain2 > Min_RG2:
                            rx_gain2 = max(Min_RG2, rx_gain2 +
min(-1,increase))
                            # radioID.set_rx_gain2(rx_gain2)
                        elif rx_gain3 > Min_RG3:
                            rx_gain3 = max(Min_RG3, rx_gain3 +
min(-1,increase))
                            # radioID.set_rx_gain3(rx_gain3)
                        #~ self.radioRx.set_rx_gain3(rx_gain3)
                        self.set_rx_gain3(rx_gain3)
                        self.set_rx_gain2(rx_gain2)
                        #~ self.radioRx.set_rx_gain2(rx_gain2)
                        time.sleep(self.AGC_updaterate)

                # print '[{0:5d}] Gain3: {1:5d}, Gain2: {2:5d},
'.format(increase, rx_gain3, rx_gain2)

            print "++"
            print "++ AGC thread ended"

    # QT sink close method reimplementation
    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio",
"ACDME")
        self.settings.setValue("geometry",
self.saveGeometry())
        event.accept()

    def CommandLineInterface(self):
        print "++"
        print "++ Starting Command Line Interface ..."
        while (not self.CLI_stop.is_set()):
            command = raw_input('++ Introduce a
command: \n++\n')
            if command == "" :
                pass
                # print '++ CR18: %d ' %
(self.main.nutaq_custom_register_18.get_value())
                print '++ CR30: %d ' %
(self.main.nutaq_custom_register_30.get_value())
                print '++ CR31: %d ' %
(self.main.nutaq_custom_register_31.get_value())
            elif command == "exit" : break
            else :
                try :
                    self.VOR = float(command)
                    if self.VOR < 136 and self.VOR >= 108 :
self.set_channel(self.VOR)
                except ValueError :
                    print("++ Unknown Command")
        print "++"
        print "++ Exiting Command Line Interface ..."

    def BiasCal(self):
        # Measuring elapsed time

        print "++ Starting CAL ..."
        print (self.filter_taps)

        self.start_time = time.time()

        if self.options.platform in {"pico", "zepto"} :
            print '+'*67
            print "++"
            print "++ Waiting for Nutaq's calibration ..."
            print "++"
            print '+'*67 + '\n'
            waittime = 0
            progress = 0
            # reseted = False
            #Just set the calibration channel once in the
routine
            self.set_channel(0)
            # self.set_channel(self.options.VOR)
            while (not self.CAL_stop.is_set() ):
                if
not(self.main.nutaq_carrier_perseus_0.is_peripheral_r
eseted(1)):
                    waittime += 1
                    time.sleep(0.001)
                elif self.calib_reseted == False:
                    print "Initialization done in %(#)0.1f sec."
% {"#": waittime/10}
                    print '+'*67
                    # Check if calibrated
                    print "++"
                    # sys.stdout.write("++ Calibrating DME...
")
                    print "++ Calibrating DME...  "
                    self.calib_reseted = True
                elif self.calib_reseted:
                    if
self.DME_interrogator.get_Calibration_Mode():
                        # progress = showprogress(progress)
                        progress =
showprogress_2((self.DME_interrogator.get_Bias()/self
.dme_samp_freq*1000))
                    else:
                        sys.stdout.flush()
                        sys.stdout.write("\r++ Done.")
                        # sys.stdout.write("/\n")
                        # Calibrated: Print and go to Normal
Mode
                        print '++ - Bias: %0.3f ms' %
(self.DME_interrogator.get_Bias()/self.dme_samp_freq
*1000)
                        print "++"
                        ## Go to normal Mode
                        self.rx_gain2 = Min_RG2
                        self.set_rx_gain2(Min_RG2)
                        self.rx_gain3 = Min_RG3
                        self.set_rx_gain3(Min_RG3)

                        self.calib_reseted = False
                        self.set_channel(self.previous_VOR)
                        # self.set_G_Tx(self.options.tx_gain3)

                        self.CAL_stop.set()
                            # Prompt for commands
                        #
self.nutaq_radio420_rx_0.set_default_rx_vga1_gain(2)
                        #
self.nutaq_radio420_rx_0.set_default_rx_gain_ctrl(0)

                        # self.CLI.start()
                        # break
            print "++"
            print "++ Calibration thread ended"
```

APPENDIX IV

ADS-B SDA Library's Python Script

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: ADS-B
# Generated: Fri Oct  3 09:46:11 2014
##################################################

from PyQt4 import Qt
from PyQt4.QtCore import QObject, pyqtSlot
from gnuradio import analog
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import filter
from gnuradio import gr
from gnuradio import qtgui
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
import threading
from optparse import OptionParser
import PyQt4.Qwt5 as Qwt
import adsb
import nutaq
import sip
import sys
import time

cal_flag = False #Calibrate at start

class ADSB_PERSEUS(gr.top_block, Qt.QWidget):

    def __init__(self, main, radioTx, radioRx, rtdexID, options):
        gr.top_block.__init__(self, "ADS-B")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("ADS-B")
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)
        self.radioTx = radioTx = radioTx
        self.radioRx = radioRx = radioRx
        self.options = options = options
        self.rtdexID = rtdexID = rtdexID
        self.main = main = main

        self.settings = Qt.QSettings("GNU Radio",
"ADSB_PERSEUS")

self.restoreGeometry(self.settings.value("geometry").toByteArray()
)
        self.radioradio=radioTx

        self.AUF_stop = threading.Event()
```

```
##################################################
        # Variables

##################################################
        self.samp_rate = samp_rate = 4e6 #default 4e6
        self.enable = enable = 1
        self.TXGain = TXGain = -13
        self.IOScale = IOScale = 2**11-1
        self.Speed = Speed = 0
        self.Features = Features = ""
        self.Speed = Speed = 0
        self.Latitude = Latitude = 0
        self.Longitude = Longitude = 0
        self.Heading = Heading = 0
        self.Flight_ID = Flight_ID = 0
        self.ICAO_ID = ICAO_ID = 0
        self.BDS_5 = BDS_5 = 0
        self.BDS_6 = BDS_6 = 0
        self.BDS_8 = BDS_8 = 0
        self.BDS_9 = BDS_9 = 0


##################################################
        # Blocks

##################################################

        #Features
        self._Features_tool_bar = Qt.QToolBar(self)

self._Features_tool_bar.addWidget(Qt.QLabel("Features"+":"))
        self._Features_label= Qt.QLabel(str(self.Features))
        self._Features_tool_bar.addWidget(self._Features_label)

self.top_grid_layout.addWidget(self._Features_tool_bar,1,1,1,6)

        #Speed
        self._Speed_tool_bar = Qt.QToolBar(self)
        self._Speed_tool_bar.addWidget(Qt.QLabel("Speed "+": "))
        self._Speed_line_edit = Qt.QLineEdit(str(self.Speed))
        self._Speed_tool_bar.addWidget(self._Speed_line_edit)
        self._Speed_line_edit.returnPressed.connect(
            lambda:
self.set_Speed(int(self._Speed_line_edit.text().toAscii())))
        self.top_grid_layout.addWidget(self._Speed_tool_bar,2,1,1,2)

        #Latitude
        self._Latitude_tool_bar = Qt.QToolBar(self)
        self._Latitude_tool_bar.addWidget(Qt.QLabel("Latitude "+":
"))
        self._Latitude_line_edit = Qt.QLineEdit(str(self.Latitude))
        self._Latitude_tool_bar.addWidget(self._Latitude_line_edit)
        self._Latitude_line_edit.returnPressed.connect(
            lambda:
self.set_Latitude(int(self._Latitude_line_edit.text().toAscii())))

self.top_grid_layout.addWidget(self._Latitude_tool_bar,3,1,1,2)

        #Longitude
        self._Longitude_tool_bar = Qt.QToolBar(self)
        self._Longitude_tool_bar.addWidget(Qt.QLabel("Longitude
"+": "))
```

```
        self._Longitude_line_edit = Qt.QLineEdit(str(self.Longitude))

self._Longitude_tool_bar.addWidget(self._Longitude_line_edit)
        self._Longitude_line_edit.returnPressed.connect(
            lambda:
self.set_Longitude(int(self._Longitude_line_edit.text().toAscii())))

self.top_grid_layout.addWidget(self._Longitude_tool_bar,3,3,1,2)

        #Heading
        self._Heading_tool_bar = Qt.QToolBar(self)
        self._Heading_tool_bar.addWidget(Qt.QLabel("Heading "+":
"))
        self._Heading_line_edit = Qt.QLineEdit(str(self.Heading))
        self._Heading_tool_bar.addWidget(self._Heading_line_edit)
        self._Heading_line_edit.returnPressed.connect(
            lambda:
self.set_Heading(int(self._Heading_line_edit.text().toAscii())))

self.top_grid_layout.addWidget(self._Heading_tool_bar,2,3,1,2)

        #Flight_ID
        self._Flight_ID_tool_bar = Qt.QToolBar(self)
        self._Flight_ID_tool_bar.addWidget(Qt.QLabel("Flight_ID "+":
"))
        self._Flight_ID_line_edit = Qt.QLineEdit(str(self.Flight_ID))
        self._Flight_ID_tool_bar.addWidget(self._Flight_ID_line_edit)
        self._Flight_ID_line_edit.returnPressed.connect(
            lambda:
self.set_Flight_ID(int(self._Flight_ID_line_edit.text().toAscii())))

self.top_grid_layout.addWidget(self._Flight_ID_tool_bar,4,1,1,2)

        #ICAO_ID
        self._ICAO_ID_tool_bar = Qt.QToolBar(self)
        self._ICAO_ID_tool_bar.addWidget(Qt.QLabel("ICAO_ID "+":
"))
        self._ICAO_ID_line_edit = Qt.QLineEdit(str(self.ICAO_ID))
        self._ICAO_ID_tool_bar.addWidget(self._ICAO_ID_line_edit)
        self._ICAO_ID_line_edit.returnPressed.connect(
            lambda:
self.set_ICAO_ID(int(self._ICAO_ID_line_edit.text().toAscii())))

self.top_grid_layout.addWidget(self._ICAO_ID_tool_bar,4,3,1,2)

        #BDS_5
        self._BDS_5_tool_bar = Qt.QToolBar(self)
        self._BDS_5_tool_bar.addWidget(Qt.QLabel("BDS_5 "+": "))
        self._BDS_5_line_edit = Qt.QLineEdit(str(self.BDS_5))
        self._BDS_5_tool_bar.addWidget(self._BDS_5_line_edit)
        self._BDS_5_line_edit.returnPressed.connect(
            lambda:
self.set_BDS_5(int(self._BDS_5_line_edit.text().toAscii())))

self.top_grid_layout.addWidget(self._BDS_5_tool_bar,5,1,1,1)


        #BDS_6
        self._BDS_6_tool_bar = Qt.QToolBar(self)
        self._BDS_6_tool_bar.addWidget(Qt.QLabel("BDS_6 "+": "))
        self._BDS_6_line_edit = Qt.QLineEdit(str(self.BDS_6))
        self._BDS_6_tool_bar.addWidget(self._BDS_6_line_edit)
        self._BDS_6_line_edit.returnPressed.connect(
            lambda:
self.set_BDS_6(int(self._BDS_6_line_edit.text().toAscii())))

self.top_grid_layout.addWidget(self._BDS_6_tool_bar,5,2,1,1)


        #BDS_8
        self._BDS_8_tool_bar = Qt.QToolBar(self)
        self._BDS_8_tool_bar.addWidget(Qt.QLabel("BDS_8 "+": "))
        self._BDS_8_line_edit = Qt.QLineEdit(str(self.BDS_8))
        self._BDS_8_tool_bar.addWidget(self._BDS_8_line_edit)
        self._BDS_8_line_edit.returnPressed.connect(
            lambda:
self.set_BDS_8(int(self._BDS_8_line_edit.text().toAscii())))

self.top_grid_layout.addWidget(self._BDS_8_tool_bar,5,3,1,1)


        #BDS_9
        self._BDS_9_tool_bar = Qt.QToolBar(self)
        self._BDS_9_tool_bar.addWidget(Qt.QLabel("BDS_9 "+": "))
        self._BDS_9_line_edit = Qt.QLineEdit(str(self.BDS_9))
        self._BDS_9_tool_bar.addWidget(self._BDS_9_line_edit)
        self._BDS_9_line_edit.returnPressed.connect(
            lambda:
self.set_BDS_9(int(self._BDS_9_line_edit.text().toAscii())))

self.top_grid_layout.addWidget(self._BDS_9_tool_bar,5,4,1,1)


        # Sinks
        self.nutaq_rtdex_sink_0 =
nutaq.rtdex_sink("nutaq_carrier_perseus_0",gr.sizeof_short,1,0)
        self.nutaq_rtdex_sink_0.set_type(0)
        self.nutaq_rtdex_sink_0.set_packet_size(8960)#8960)#8192
        self.nutaq_rtdex_sink_0.set_channels(rtdexID) #"2")
        (self.nutaq_rtdex_sink_0).set_thread_priority(99)

        self.nutaq_rtdex_source_0 =
nutaq.rtdex_source("nutaq_carrier_perseus_0",gr.sizeof_short,1,0)
        self.nutaq_rtdex_source_0.set_type(0)
        self.nutaq_rtdex_source_0.set_packet_size(8960)
        self.nutaq_rtdex_source_0.set_channels(rtdexID)#"2")
        (self.nutaq_rtdex_sink_0).set_thread_priority(99)

        self.blocks_null_sink_0 = blocks.null_sink(gr.sizeof_short*1)

# #Abdu...
# # (self.nutaq_rtdex_sink_0).set_processor_affinity(6)
# #       (self.nutaq_rtdex_sink_0).set_thread_priority(99)
# #...Abdu
#        self.adsb_nutaq_radio420_tx_0 =
nutaq.radio420_tx("nutaq_carrier_perseus_0", 2 ,6)
#        self.adsb_nutaq_radio420_tx_0.set_default_enable(1)
#
self.adsb_nutaq_radio420_tx_0.set_default_tx_freq(1090e6)
#        self.adsb_nutaq_radio420_tx_0.set_default_reference(0)
#
self.adsb_nutaq_radio420_tx_0.set_default_datarate(samp_rate*2
)
#
self.adsb_nutaq_radio420_tx_0.set_default_calibrate(cal_flag)
#        self.adsb_nutaq_radio420_tx_0.set_default_band(0)
#        self.adsb_nutaq_radio420_tx_0.set_default_update_rate(1)
#        self.adsb_nutaq_radio420_tx_0.set_default_tx_vga1_gain(-
22)
#
self.adsb_nutaq_radio420_tx_0.set_default_tx_vga2_gain(6)
#        self.adsb_nutaq_radio420_tx_0.set_default_tx_gain3(-13)
#
self.adsb_nutaq_radio420_tx_0.set_default_tx_lpf_bandwidth(0)

#        self.adsb_nutaq_radio420_rx_0 =
nutaq.radio420_rx("nutaq_carrier_perseus_0", 2,4)
#        self.adsb_nutaq_radio420_rx_0.set_default_enable(0)
#
self.adsb_nutaq_radio420_rx_0.set_default_rx_freq(1090e6)
#        self.adsb_nutaq_radio420_rx_0.set_default_reference(0)
#
self.adsb_nutaq_radio420_rx_0.set_default_datarate(2*samp_rate
)
#
self.adsb_nutaq_radio420_rx_0.set_default_calibrate(cal_flag)
#        self.adsb_nutaq_radio420_rx_0.set_default_band(0)
#        self.adsb_nutaq_radio420_rx_0.set_default_update_rate(1)
#        self.adsb_nutaq_radio420_rx_0.set_default_rx_lna_gain(2)
#
```

```
self.adsb_nutaq_radio420_rx_0.set_default_rx_vga1_gain(1)
#        self.adsb_nutaq_radio420_rx_0.set_default_rx_gain2(0)
#        self.adsb_nutaq_radio420_rx_0.set_default_rx_gain3(-8)
#        self.adsb_nutaq_radio420_rx_0.set_default_rx_rf_filter(2)
#
self.adsb_nutaq_radio420_rx_0.set_default_rx_lpf_bandwidth(0)
#Abdu...
#   (self.adsb_nutaq_radio420_tx_0).set_processor_affinity([4,5])
#       (self.adsb_nutaq_radio420_tx_0).set_thread_priority(20)
#...Abdu
        # self.nutaq_custom_register_0 =
nutaq.custom_register("nutaq_carrier",1)
        # self.nutaq_custom_register_0.set_index(1)
        # self.nutaq_custom_register_0.set_default_value(6)
        # self.nutaq_custom_register_0.set_update_rate(1)
#Abdu...
#   (self.nutaq_custom_register_0).set_processor_affinity([4,5])
#       (self.nutaq_custom_register_0).set_thread_priority(20)
#...Abdu
        # self.nutaq_carrier = nutaq.carrier(0,"nutaq_carrier",
"192.168.0.101")
#Abdu...
#       (self.nutaq_carrier_perseus_0).set_processor_affinity([4,5])
#       (self.nutaq_carrier_perseus_0).set_thread_priority(20)
#...Abdu
        self.interp_fir_filter_xxx_0 =
filter.interp_fir_filter_fff(int(samp_rate/2e6),
([1]*int(samp_rate/2e6)))
        self.interp_fir_filter_xxx_0.declare_sample_delay(0)
#Abdu...
#       (self.interp_fir_filter_xxx_0).set_processor_affinity(6)
#       (self.interp_fir_filter_xxx_0).set_thread_priority(99)
#...Abdu
        self.blocks_float_to_complex_0 = blocks.float_to_complex(1)
#Abdu...
#       (self.blocks_float_to_complex_0).set_processor_affinity(6)
#       (self.blocks_float_to_complex_0).set_thread_priority(99)
#...Abdu
#       self.blocks_complex_to_interleaved_short_0 =
blocks.complex_to_interleaved_short(False)
        self.blocks_complex_to_interleaved_short_0 =
blocks.complex_to_interleaved_short()
#Abdu...
#
(self.blocks_complex_to_interleaved_short_0).set_processor_affini
ty(6)
#
(self.blocks_complex_to_interleaved_short_0).set_thread_priority(
99)
#...Abdu
        # self.blocks_char_to_float_0 = blocks.char_to_float(1,
1.0/IOScale)
        self.blocks_short_to_float_0 = blocks.short_to_float(1,
1.0/IOScale)
#Abdu...
#       (self.blocks_char_to_float_0).set_processor_affinity(6)
#       (self.blocks_char_to_float_0).set_thread_priority(99)
#...Abdu
        self.analog_const_source_x_0 = analog.sig_source_s(0,
analog.GR_CONST_WAVE, 0, 0, enable)
#Abdu...
#       (self.analog_const_source_x_0).set_processor_affinity(6)
#       (self.analog_const_source_x_0).set_thread_priority(99)
#...Abdu
        self.adsb_out_0 = adsb.out()
#Abdu...
#       (self.adsb_out_0).set_processor_affinity(7)
        # (self.adsb_out_0).set_thread_priority(99)
#...Abdu


        self.qtgui_time_sink_x_0 = qtgui.time_sink_f(
            int(150e-6*samp_rate)+2, #size
            samp_rate, #samp_rate
            "Message", #name
```

```
            1 #number of inputs
        )
        self.qtgui_time_sink_x_0.set_update_time(0.5)
        self.qtgui_time_sink_x_0.set_y_axis(-0.1, IOScale+1)

#       self.qtgui_time_sink_x_0.set_y_label("Data bits", "")

        self.qtgui_time_sink_x_0.enable_tags(-1, False)

self.qtgui_time_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_N
ORM, qtgui.TRIG_SLOPE_POS, 0.5, 0, 0, "")
        self.qtgui_time_sink_x_0.enable_autoscale(False)
        self.qtgui_time_sink_x_0.enable_grid(False)

        labels = [" ", "", "", "", "",
                  "", "", "", "", ""]
        widths = [1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1]
        colors = ["blue", "red", "green", "black", "cyan",
                  "magenta", "yellow", "dark red", "dark green", "blue"]
        styles = [1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1]
        markers = [-1, -1, -1, -1, -1,
                   -1, -1, -1, -1, -1]
        alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
                  1.0, 1.0, 1.0, 1.0, 1.0]

        for i in xrange(1):
            if len(labels[i]) == 0:
                self.qtgui_time_sink_x_0.set_line_label(i, "Data
{0}".format(i))
            else:
                self.qtgui_time_sink_x_0.set_line_label(i, labels[i])
                self.qtgui_time_sink_x_0.set_line_width(i, widths[i])
                self.qtgui_time_sink_x_0.set_line_color(i, colors[i])
                self.qtgui_time_sink_x_0.set_line_style(i, styles[i])
                self.qtgui_time_sink_x_0.set_line_marker(i, markers[i])
                self.qtgui_time_sink_x_0.set_line_alpha(i, alphas[i])
#Abdu...
        # (self.qtgui_time_sink_x_0 ).set_processor_affinity([4,5])
        # (self.qtgui_time_sink_x_0 ).set_thread_priority(1)
#Abdu...
        self._qtgui_time_sink_x_0_win =
sip.wrapinstance(self.qtgui_time_sink_x_0.pyqwidget(),
Qt.QWidget)
        self.top_layout.addWidget(self._qtgui_time_sink_x_0_win)


##################################################
        # Connections

##################################################
        # self.connect((self.analog_const_source_x_0, 0),
(self.adsb_out_0, 0))
        # self.connect((self.adsb_out_0, 0),
(self.blocks_char_to_float_0, 0))
        # self.connect((self.blocks_complex_to_interleaved_short_0,
0), (self.nutaq_rtdex_sink_0, 0))
        # self.connect((self.blocks_float_to_complex_0, 0),
(self.blocks_complex_to_interleaved_short_0, 0))
        # self.connect((self.blocks_char_to_float_0, 0),
(self.interp_fir_filter_xxx_0, 0))
        # self.connect((self.interp_fir_filter_xxx_0, 0),
(self.qtgui_time_sink_x_0, 0))
        # self.connect((self.interp_fir_filter_xxx_0, 0),
(self.blocks_float_to_complex_0, 0))

        # self.connect((self.analog_const_source_x_0),
(self.adsb_out_0, 0))
        # self.connect((self.main.nutaq_source, 1), (self.adsb_out_0,
0))
        self.connect((self.nutaq_rtdex_source_0, 0),
(self.adsb_out_0, 0))
        self.connect((self.adsb_out_0, 0),
(self.blocks_short_to_float_0, 0))
```

```
        self.connect((self.blocks_short_to_float_0, 0),
(self.interp_fir_filter_xxx_0, 0))
        self.connect((self.interp_fir_filter_xxx_0, 0),
(self.qtgui_time_sink_x_0, 0))
        self.connect((self.interp_fir_filter_xxx_0, 0),
(self.blocks_float_to_complex_0, 0))
        self.connect((self.blocks_float_to_complex_0, 0),
(self.blocks_complex_to_interleaved_short_0, 0))
        self.connect((self.blocks_complex_to_interleaved_short_0,
0), (self.nutaq_rtdex_sink_0, 0))
        # self.connect((self.blocks_complex_to_interleaved_short_0,
0), (self.main.nutaq_sink, 1))

        # self.connect((self.main.nutaq_source, 0),
(self.blocks_null_sink_0, 0))
        # self.connect((self.analog_const_source_x_0, 0),
(self.main.nutaq_sink, 0))


        # self.connect((self.nutaq_rtdex_source_0, 0),
(self.blocks_null_sink_0, 0))


    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio",
"ADSB_PERSEUS")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

    def get_samp_rate(self):
        return self.samp_rate

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

self.interp_fir_filter_xxx_0.set_taps(([1]*int(self.samp_rate/2e6)))
        self.qtgui_time_sink_x_0.set_samp_rate(self.samp_rate)

    def get_enable(self):
        return self.enable

    #~ def set_enable(self, enable):
        #~ # self.enable = enable
        #~ # self.analog_const_source_x_0.set_offset(self.enable)
        #~ # self._enable_callback(self.enable)
#~
        #~ #Test if radioCard Object is accessible from outside the
class (By reference/ is mutable)
        #~ self.enable = enable
        #~ if enable:
            #~ # self.nutaq_radio420_tx_1.enable_path()
            #~ self.radioTx.enable_path()
        #~ else:
            #~ # self.nutaq_radio420_tx_1.disable_path()
            #~ self.radioTx.disable_path()
        #~ self._enable_callback(self.enable)



    def get_Speed(self):
        return self.Speed

    def set_Speed(self,Speed):
        self.Speed= Speed


    def get_Latitude(self):
        return self.Latitude

    def set_Latitude(self,Latitude):
        self.Latitude= Latitude

    def get_Longitude(self):
        return self.Longitude

    def set_Longitude(self,Longitude):
        self.Longitude = Longitude


    def get_Heading(self):
        return self.Heading

    def set_Heading(self,Heading):
        self.Heading = Heading

    def get_Flight_ID(self):
        return self.Flight_ID

    def set_Flight_ID(self,Flight_ID):
        self.Flight_ID = Flight_ID



    def get_ICAO_ID(self):
        return self.ICAO_ID

    def set_ICAO_ID(self,ICAO_ID):
        self.ICAO_ID = ICAO_ID

    def get_BDS_5(self):
        return self.BDS_5

    def set_BDS_5(self,BDS_5):
        self.BDS_5 = BDS_5


    def get_BDS_6(self):
        return self.BDS_6

    def set_BDS_6(self,BDS_6):
        self.BDS_6= BDS_6

    def get_BDS_8(self):
        return self.BDS_8

    def set_BDS_8(self,BDS_8):
        self.BDS_8 = BDS_8


    def get_BDS_9(self):
        return self.BDS_9

    def set_BDS_9(self,BDS_9):
        self.BDS_9= BDS_9

    def get_IOScale(self):
        return self.IOScale

    def set_IOScale(self, IOScale):
        self.IOScale = IOScale
        self.blocks_char_to_float_0.set_scale(1.0/self.IOScale)
        self.qtgui_time_sink_x_0.set_y_axis(-0.1, self.IOScale+1)

    def adsb_update_fields(self):
        while (not self.AUF_stop.is_set() ):


self._Speed_line_edit.setText(eng_notation.num_to_str(self.adsb_
out_0.get_Speed_indicated_air ()))

self._Latitude_line_edit.setText(eng_notation.num_to_str(self.adsb
_out_0.get_Latitude ()))

self._Longitude_line_edit.setText(eng_notation.num_to_str(self.ad
sb_out_0.get_Longitude ()))

self._BDS_5_line_edit.setText(eng_notation.num_to_str(self.adsb
_out_0.get_BDS_5 ()))
```

```
self._BDS_6_line_edit.setText(eng_notation.num_to_str(self.adsb
_out_0.get_BDS_6 ()))

self._BDS_8_line_edit.setText(eng_notation.num_to_str(self.adsb
_out_0.get_BDS_8 ()))

self._BDS_9_line_edit.setText(eng_notation.num_to_str(self.adsb
_out_0.get_BDS_9 ()))
        #~
self._Flight_ID_line_edit.setText(eng_notation.num_to_str(self.ads
b_out_0.get_Flight_ID ()))

self._Heading_line_edit.setText(eng_notation.num_to_str(self.ads
b_out_0.get_Heading ()))
        #~
self._Tx_Freq_line_edit.setText(eng_notation.num_to_str(self.nuta
q_radio420_tx_1.get_tx_freq()))
        #~
self._Tx_Freq2_line_edit.setText(eng_notation.num_to_str(self.nut
aq_radio420_tx_2.get_tx_freq()))
        #~
self._Rx_Freq_line_edit.setText(eng_notation.num_to_str(self.nuta
q_radio420_tx_1.get_rx_freq()))
        #~
self._Rx_Freq2_line_edit.setText(eng_notation.num_to_str(self.nut
aq_radio420_tx_2.get_rx_freq()))

        time.sleep(.5)
```

APPENDIX V

Xilinx System Generator Implementation by Module

APPENDIX VI

RTDEx Tests

The following tests were executed to verify the RTDEx transmittion and reception data rates. A summary of the results is presented as follows:

1. Gigabit Ethernet (2 channels): 69.151 MB/s = 553*2 Mbps (or 17.28*2 MS/s) (Total throughput);
2. PCI Express: 190.675 MB/s = 1525.4 Mbps (or 47 MS/s).

Note: the tests were performed using jumbo packets to allow for the greates throughput possible.

The test was performed for 1 and 2 RTDEx Channels, however when only one channel was enabled the link seemed unstable. It returned the following message:

```
RTDExReceive() missing 8960 bytes, received 0
```

Test Execution:

Gigabit Ethernet (1 channel):

```
nutaq@nutaq:/opt/Nutaq/ADP6/ADP_MicroTCA/sdk/examples/perseus6010_rtdex_re
cord_playback/host/prj_linux$ sudo ./Launch_rtdex_host_to_perseus.sh
RTDEx functionnal example for the Perseus

Connecting to Perseus at adress 192.168.0.101
Connected!

The test will transfer data
  - On the Gigabit Ethernet medium
  - To and from the Perseus
  - On 1 channel(s)
  - Transfer mode is single transfer
  - Transfer size is 536865280 bytes
  - Packet size is 8960 bytes
  - Frame gap is 10000
  - Burst size is 8960 bytes

Starting data transfer. Please wait!
Testing in free running mode

RTDExReceive() missing 8960 bytes, received 0

Transfer terminated!
Time of transfer: 9.50 second(s)

Statistics:

To FPGA channel 0
  - Byte(s) received by Perseus (To FPGA, Ch 0): 350963200
  - Sample(s) in error (To FPGA, Ch 0):                 0
  - Calculated throughput (To FPGA, Ch 0):      35.243 MB/s

From FPGA channel 0
  - Byte(s) received by host (From FPGA, Ch 0):    350963200
  - Sample(s) in error (From FPGA, Ch 0):              4998
  - Calculated throughput (From FPGA, Ch 0):      53.911 MB/s

Full from FPGA throughput:  53.911 MB/s

Full to FPGA throughput:    35.243 MB/s

The program will terminate. Push any key to close

nutaq@nutaq:/opt/Nutaq/ADP6/ADP_MicroTCA/sdk/examples/perseus6010_rtdex_re
cord_playback/host/prj_linux$
```

Gigabit Ethernet (2 channels):

```
nutaq@nutaq:/opt/Nutaq/ADP6/ADP_MicroTCA/sdk/examples/perseus6010_rtdex_re
cord_playback/host/prj_linux$ sudo ./Launch_rtdex_host_to_perseus.sh
RTDEx functionnal example for the Perseus

Connecting to Perseus at adress 192.168.0.101
Connected!
```

```
The test will transfer data
  - On the Gigabit Ethernet medium
  - To and from the Perseus
  - On 2 channel(s)
  - Transfer mode is single transfer
  - Transfer size is 536865280 bytes
  - Packet size is 8960 bytes
  - Frame gap is 10000
  - Burst size is 8960 bytes

Starting data transfer. Please wait!
Testing in free running mode

Transfer terminated!
Time of transfer: 14.81 second(s)

Statistics:

To FPGA channel 0
  - Byte(s) received by Perseus (To FPGA, Ch 0): 536865280
  - Sample(s) in error (To FPGA, Ch 0):                 0
  - Calculated throughput (To FPGA, Ch 0):      34.576 MB/s

To FPGA channel 1
  - Byte(s) received by Perseus (To FPGA, Ch 1): 536865280
  - Sample(s) in error (To FPGA, Ch 1):                 0
  - Calculated throughput (To FPGA, Ch 1):      34.576 MB/s

From FPGA channel 0
  - Byte(s) received by host (From FPGA, Ch 0):    536865280
  - Sample(s) in error (From FPGA, Ch 0):                 0
  - Calculated throughput (From FPGA, Ch 0):      34.576 MB/s

From FPGA channel 1
  - Byte(s) received by host (From FPGA, Ch 1):    536865280
  - Sample(s) in error (From FPGA, Ch 1):                 0
  - Calculated throughput (From FPGA, Ch 1):      34.576 MB/s

Full from FPGA throughput:  69.151 MB/s

Full to FPGA throughput:    69.151 MB/s

The program will terminate. Push any key to close
nutaq@nutaq:/opt/Nutaq/ADP6/ADP_MicroTCA/sdk/examples/perseus6010_rtdex_re
cord_playback/host/prj_linux$
```

PCI Express:

```
nutaq@nutaq:/opt/Nutaq/ADP6/ADP_MicroTCA/sdk/examples/perseus6010_rtdex_re
cord_playback/host/prj_linux$ sudo ./Launch_rtdex_host_to_perseus.sh
RTDEx functionnal example for the Perseus

Connecting to Perseus at adress 192.168.0.101
Connected!

The test will transfer data
  - On the PCI Express medium
  - To and from the Perseus
  - On 1 channel(s)
  - Transfer mode is single transfer
  - Transfer size is 536832000 bytes
  - Packet size is 128000 bytes
  - Frame gap is 10000
  - Burst size is 128000 bytes

Starting data transfer. Please wait!
Testing in free running mode

Transfer terminated!
Time of transfer: 2.69 second(s)

Statistics:

To FPGA channel 0
  - Byte(s) received by Perseus (To FPGA, Ch 0): 536832000
  - Sample(s) in error (To FPGA, Ch 0):                 0
  - Calculated throughput (To FPGA, Ch 0):     190.675 MB/s

From FPGA channel 0
  - Byte(s) received by host (From FPGA, Ch 0):    536832000
  - Sample(s) in error (From FPGA, Ch 0):                 0
  - Calculated throughput (From FPGA, Ch 0):     190.675 MB/s

Full from FPGA throughput: 190.675 MB/s

Full to FPGA throughput:   190.675 MB/s

The program will terminate. Push any key to close
```

APPENDIX VII

SDAR FPGA Selector’s Truth Table

SDA code		
SDA	BIT 0	BIT 1
NONE	0	0
ADSB	0	1
DME1	1	0
DME2	1	1

Legend	
Un-used codes	▨
Initial condition	░

FPGA's SDA Selector (TRUTH TABLE)					
Currently routed SDA application		Selector bits (FPGA Custom Register 25)			
Radio 1	Radio 2	BIT 0	BIT 1	BIT 2	BIT 3
NONE	NONE	0	0	0	0
NONE	ADSB	0	0	0	1
NONE	DME2	0	0	1	0
NONE	DME1	0	0	1	1
ADSB	NONE	0	1	0	0
ADSB	ADSB	0	1	0	1
ADSB	DME2	0	1	1	0
ADSB	DME1	0	1	1	1
DME2	NONE	1	0	0	0
DME2	ADSB	1	0	0	1
DME2	DME2	1	0	1	0
DME2	DME1	1	0	1	1
DME1	NONE	1	1	0	0
DME1	ADSB	1	1	0	1
DME1	DME2	1	1	1	0
DME1	DME1	1	1	1	1

ANNEXE I – LEVD

(Laboratory Equipment Validation)

ANNEXE II – FTPR

(Flight Test Plan Requirements)

ANNEXE III - FLTD

(Flight Test Document)

ANNEXE IV - AGTD

(Aircraft Ground Tests Document)

ANNEXE V - IECD

(Installed Equipment Configuration)

BIBLIOGRAPHY

Agbeyibor, R. C. (2014). *Secure ADS-B: Towards Airborne Communications Security in the Federal Aviation Administration's Next Generation Air Transportation System.* Master's thesis.

airlinecouncil.ca. (n.d.). Retrieved from http://www.airlinecouncil.ca/en/rules-and-regulations.html

Biçer, S. M. (2002). *A Software Communications Architecture Compliant Software Defined Radio Implementation.* Master's thesis, Northeastern University, Boston.

Bivona, F. (2009). *Reconfigurable Software Defined Radio Platform.* WORCESTER POLYTECHNIC INSTITUTE.

Chávez-Santiago R, M. A. (n.d.). Applications of software-defined radio (SDR) technology in hospital environments.

Cobham AvComm. (2015). *IFR 6000 Flightline Test Set*. Retrieved 01 2016, from Aeroflex.com: http://ats.aeroflex.com/avionics-test-products/identification-products/ifr-6000-flightline-test-set

Collinson, R. (2011). *Introduction to Avionics Systems.* Springer Science & Business Media.

Collinson, R. (2011). *Introduction to Avionics Systems* (3rd ed.).

Collinson, R. P. (2013). *Introduction to avionics systems.* Springer Science & Business Media.

Debatty, T. (2010). Software Defined RADAR a State of the Art. *2nd International Workshop on Cognitive Information Processing.* Brussels.

Di Pu, W. A. (n.d.). *Digital Communication Systems Engineering with Software Defined Radio.*

Di, R. (2013). *A USRP-Based Flexible GNSS Signal Recording and Playback System: Performance Evaluation and Study.* Miami University.

El Salloum, C. E. (2013, 08). The ACROSS MPSoC – A new generation of multi-core processors designed for safety–critical embedded systems. *Microprocessors and Microsystems, 37*(8), 1020-1032.

Eyermann, P. A. (1999). Joint Tactical Radio Systems-A Solution to Avionics Modernization. *Proceedings of 18th Digital Avionics Systems Conference, 2*, 9.A.5-1 - 9.A.5-8.

Freeman, E. (1995). *MIT Lincoln Laboratory: Technology in the National Interest.* Lexinton, Mass.

Gask, T. C. (2015). Future avionic system hybrid processor pooled architectures. *Annual Forum Proceedings - AHS International*, (pp. 915-928).

GNU Radio. (2013). *Blocks Coding Style*. Retrieved 2015, from GNU Radio - gnuradio.org: https://gnuradio.org/redmine/projects/gnuradio/wiki/BlocksCodingGuide

GNU Radio. (2015). *GNU Radio Overview*. Retrieved from https://gnuradio.org/redmine/projects/gnuradio

GNU Radio. (n.d.). *GNU Radio Wiki*. Retrieved from https://gnuradio.org/redmine/projects/gnuradio/wiki

GNURadio. (2015, 10 20). *Gnu Radio Companion*. Retrieved 10 20, 2015, from https://gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioCompanion

Hodson, R. a. (2007). Avionics for exploration. *NASA Technology Exchange Conference.* Galveston, TX, USA.

III, J. M. (2000). *Software Radio Architecture: Object Oriented Approaches to Wireless Systems Engineering.* John Wiley and Sons.

International Telecomunication Union (ITU). (2009). *Definitions of Software Defined Radio (SDR) and Cognitive Radio System (CRS).* ITU-R SM.2152, ITU, Geneva.

J. Chapin, V. B. (2002, 11). The Vanu Software Radio System. *Proceedings of the Software Defined Radio Technical Conference*.

Jalloul, T. (2014). *CONCEPTION D'UN TRANSPONDEUR MODE S ET D'UN SYSTÈME DME/DME EN RADIO LOGICIELLE: IMPLÉMENTATION ET RÉSULTATS DE TESTS.* Master thesis, ETS.

LASSENA. (2013). Retrieved from http://comunite.etsmtl.ca/projects/avio_en.asp

LASSENA. (2013). *AVIO 505 project's Brochure.* Retrieved from LASSENA Intranet: http://lassena.etsmtl.ca/IMG/pdf/-27.pdf

NUTAQ. (2014). *Nutaq PicoSDR FPGA-based, MIMO-Enabled, tunable RF SDR solutions Product Sheet.* Retrieved from http://www.nutaq.com/: http://www.nutaq.com/wp-content/uploads/2015/07/PicoSDR_V1.4_02_14_2014_web.pdf

NUTAQ. (2014). Perseus 601X User´s Guide.

NUTAQ. (2014). Radio420X User's Guide. *Radio420X User's Guide*.

NUTAQ. (2014). RTDEx Programmer's Reference Guide.

NUTAQ. (2015, 11). *An FPGA-based AGC algorithm using System Generator*. Retrieved 11 2015, from www.nutaq.com: http://www.nutaq.com/blog/fpga-based-agc-algorithm-using-system-generator

NUTAQ. (2015). *Perseus 601X Product Sheet.* Quebec, QC: NUTAQ.

Olivier, A. (2015). AVIO 505 project´s documentation.

Platform, R. S. (2009, March 25). *Francesco Bivona.* WORCESTER POLYTECHNIC INSTITUTE.

PR Newswire. (2009, 02). *New Xilinx Virtex-6 FPGA Family Designed to Satisfy Insatiable Demand for Higher...* Retrieved 11 2015, from PR Newswire: http://www.prnewswire.com/news-releases/new-xilinx-virtex-6-fpga-family-designed-to-satisfy-insatiable-demand-for-higher-bandwidth-and-lower-power-systems-65626152.html

RTCA SC-149. (1985). *DO-189 Minimum Operational Performance Standards for Airborne Distance Measuring Equipment (DME) Operating within the Radio Frequency Range of 960-1215 MHz.*

RTCA Special Committee 170. (1992). *Minimum Operational Performance Standards for Airborne Automatic Dependent Surveillance (ADS) Equipment, RTCA/DO-212.* RTCA.

RTCA Special Committee 209. (2011). *Minimum Operational Performance Standards for Air Traffic Control Radar Beacon System/Mode Select (ATCRBS/Mode S) Airborne Equipment, RTCA/DO-181E.* RTCA.

Setdsp. (2014). Retrieved 11 15, 2015, from http://www.setdsp.com/modules/advancedmc/samc-514/?sphrase_id=13369

Spitzer, C. (2014). *Digital Avionics Handbook, Third Edition.* CRC Press.

Strunk, E., Knight, J., & Aiello, M. (2004, 10). Distributed reconfigurable avionics architectures. *in Digital Avionics Systems Conference, 2004. DASC 04*, *2*, pp. 10.B.4-101-10.

Suo, D., An, J., & Jihong Zhu. (2011). A new approach to improve safety of reconfiguration in Integrated Modular Avionics. *Digital Avionics Systems Conference (DASC), 2011* (pp. 1C4-1-1C4-12, 16-20). IEEE/AIAAº.

Upmat, R. I. (1995). Speakeasy: The Military Software Radio. *IEEE Communications Magazine, , 33*(5), 56-61.

Woligroski, B. D. (2009, June 22). *How fast is gigabit ethernet supposed to be, anyway?* Retrieved from Tom's Hardware: http://www.tomshardware.com/reviews/gigabit-ethernet-bandwidth,2321-3.html

Xilinx. (2015). *System Generator User´s guide.*

Xilinx. (2015). *Virtex-6 Family Overview.*