REDHAWK FOR VITA 49 DEVELOPMENT IN OPEN RADIO ACCESS NETWORKS

by

Theodore Banaszak

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Engineering

Department of Electrical and Computer Engineering

Fort Wayne, Indiana

December 2020

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Todor Cooklev

Department of Electrical and Computer Engineering

Dr. Chao Chen

Department of Electrical and Computer Engineering

Dr. David Cochran

Department of Electrical and Computer Engineering

Approved by:

Dr. Hosni Abu-Mulaweh

For my father, I could not have asked for a better guide and mentor, who was always there for me throughout my early life and academic career, and without whom I am sure I would not have had the will to get beyond my first years of college.

ACKNOWLEDGMENTS

I would like to first thank Dr. Todor Cooklev for assisting and advising me throughout the thesis process, and for giving me this opportunity for expansion and learning in the field of wireless communications. I would also like to thank the rest of my thesis advisory board, Dr. Chao Chen and Dr David Cochran for assisting me in the thesis writing process. I would next like to thank Dr. Elizabeth Thompson and again Dr. Chao Chen who were my undergraduate and graduate advisors for their assistance throughout my academic career at Purdue. I would next like to thank Dr James Isaacs and again Dr. Elizabeth Thompson for allowing me the opportunity to assist and teach the labs sections of their digital circuits courses. I would also like to thank my parents for their support throughout my college career and thesis process. I would lastly like to thank those individuals who were there for me (even if only through the internet) and helped me continue working during the coronavirus pandemic lockdown during early 2020, some of whom I've never even met in person, and without whom it would have taken me significantly longer to finish my work, if I'd finished at all. These individuals wish to be known only by their online usernames: GL6095, HoneyDroplets, Mavellion, and Miikachu.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

RAN	**Radio Access Network**	**OBSAI**	**Open Base Station Architecture Initiative protocol**
O-RAN	Open RAN	**ORI**	Open Radio equipment Interface
SDR	Software Defined Radio	**DL**	Downlink
RAT	Radio Access Technology	**UL**	Uplink
C-RAN	Cloud RAN	**FFT**	Fast Fourier Transform
CPRI	Common Public Radio Interface	**ICIC**	Inter-Cell Interference Coordination
VITA	VME bus International Trade Association	**PDU**	Protocol Data Units
VRT	VITA Radio Transport	**SDU**	Service Data Units
RRU	Remote Radio Unit	**eNB**	evolved NodeB
BBU	Base Band Unit	**RANaaS**	RAN-as-a-Service
CoMP	Co-ordinated Multipoint	**FHGW**	Fronthaul Gateway
3GPP	3rd Generation Partnership Project	**TSI**	Timestamp Integer
RIC	RAN Intelligent Controller	**OUI**	Unique Identifier
Non-RT	Non-Real Time	**IDE**	Integrated Development Environment
Near-RT	Near Real Time	**SRI**	Signal Related Information
RRM	Radio Resource Management	**CF**	Core Framework (For The REDHAWK IDE)
CU	Centralized Unit	**DCD**	Device Configuration Descriptor
DU	Distributed Unit	**SAD**	Software Assembly Descriptor
R-NIB	Radio-Network Information Base	**FEI**	FrontEnd Interfaces

ABSTRACT

This thesis establishes the need for a standardized, interoperable, front end interface to support the development of open RAN technologies, and establishes the viability and desirability of the VITA 49 interface standard as the alternative to other interface technologies. The purpose of this work is to propose a testbed platform for the further development for VITA 49 as a standard front-end interface as other current testbeds are not designed not as well suited to the VITA 49 standard or open RAN architecture. The VITA 49 interface standard provides a packetized interface between the front-end and the digital back-end of a split architecture system in a way that enables hardware interoperability between and within vendor supplies. The VITA 49 Radio Transport standard is ideally appropriate for integration into SDRs [12] due to its flexibility and metadata support. The REDHAWK platform is an integrated development environment which is used to develop a radio system that utilizes a remote radio unit to send and receive signals which transmits it using the VITA 49 protocol to the base band unit for processing. It was found that REDHAWK is better than GNURadio for this purpose, and that VRT technology is a much better than the current CPRI Standard as it provides an open standard, that enables a flexible, scalable interface that enables long-term growth.

1. INTRODUCTION

It is well known that the data rate and performance demands of wireless communication systems is constantly increasing. Wireless systems have been able to meet this requirement but at the same time have become extremely complex, with proprietary and tightly coupled hardware and software. This traditional proprietary nature of wireless systems has led to a lack of interoperability between components or subsystems originating from different suppliers. This is not just inconvenient, but has become an impediment to technology innovation. Open-RAN provides an alternative solution to this traditional structure in virtualization and standardization of component interfaces to allow for modularity and thus greater optimization of wireless systems [1]. Open-RAN also provides an interface that is conducive to the integration of software defined radio which allows for more ready integration of multiple radio access technologies, and the use of cloud radio access networks due to modularity. One well-known option for the main interface in Open-RAN is the Common Public Radio Interface, or CPRI.

Many development platforms for the implementation of SDRs exist. The REDHAWK platform was chosen for this work due to its capabilities and recent (January 2019) shift into becoming an open source platform, which generated an increased interest. The shift of REDHAWK to an open-source platform is also of interest to the O-RAN community, as its goals include the use of open source software in its radio standards. This work will describe the development of a radio platform programmed with REDHAWK that utilizes a remote radio unit to receive a signal and transmit it to the base band unit using VITA 49 for processing.

The purpose of this work is to explore the use of VITA 49 in O-RAN systems as a standard, interoperable front end interface, and to propose REDHAWK as a testbed for the further development. First, the topics of software defined radio and Cloud RAN (C-RAN) are discussed, followed by background information on Open-RAN (O-RAN), and VITA 49, including some comparisons of VITA 49 to another popular alternative known as CPRI. Next the use of VITA 49 in the context of O-RAN will be discussed and use cases will be presented. Next, an introduction of the REDHAWK platform, including a brief comparison to the popular alternative: GNURadio. This will be followed by a description of an implementation of VITA 49 as an O-RAN interface. To validate this approach we use the REDHAWK platform for the purpose of exploring the capabilities of VITA 49 as a modular, interoperable front-end interface for use in an O-RAN

architecture. The usage of VITA 49 metadata in an O-RAN system will also be discussed in contexts including metadata necessary for O-RAN, data helpful to O-RAN, and data useful for the implementation of AI/Machine learning controlled radio resources in an O-RAN system.

Chapter 9 contains the conclusions and suggestions for future work.

2. SOFTWARE DEFINED RADIO

Software Defined Radio (SDR) has been a topic of discussion within the wireless communications industry for some time. The goal of software defined radio system is wireless communications systems where all components, protocols, and signal processing are defined entirely within software. The purpose of defining all parts of a radio system with software, is that software can easily be changed and upgraded with deployment of new software to the system, thus eliminating the need for the costly hardware changes required in traditional radio systems when new RATs are deployed.

The main technology trends in modern SDR systems include hardware/software independence and virtualization of components. On the wireless supplier end, this removes the costly operation of implementing and maintaining RAT specific hardware. On the user end, the virtualization of RAT implementation allows a user device to bypass the usual size constraint on containing multiple different connection protocols, thereby helping a user to ensure that they can always get a connection to the network [6],[12].

3. CLOUD RADIO ACCESS NETWORKS

Cloud radio access networks, or C-RANs, focus on the separation of the front-end radio components from the radio signal processing. In traditional radio architecture, the eNBs are standalone – they contain all necessary hardware and software to communicate with UEs. The goal of C-RAN is to change this traditional radio architecture to one where the eNB is split into two components: the Remote Radio Unit (RRU) containing mainly front-end hardware, and the Base Band Unit (BBU), where all other processing is done. This allows the BBU to be at a separate physical location and to connect to multiple RRUs simultaneously [12]. This centralization of radio processing allows for more cost-efficient upgrading and maintaining of networks, while at the same time achieving performance advantages. One advantage is the improved coordination of the transmission and/or reception from multiple RRUs using a technique known as Coordinated Multipoint (CoMP). Other performance advantages include better use of network resources, more efficient spectrum usage, and interference avoidance [13].

3.1 Dual Connectivity

One feature in particular that is enabled by the existence of a backhaul network is dual connectivity. In 4G LTE, dual connectivity was introduced to assist in handling the ever-increasing wireless resource load demand. It functions through utilization of a Master evolved NodeB (MeNB), which is a traditional macro-cell evolved NodeB (eNB), and Secondary eNB (SeNB) pico or fempto cell inside the coverage area of the MeNB. Through connection with both the MeNB and a SeNB at the same time, as shown in figure 3-1, the throughput of a UE could be increased.

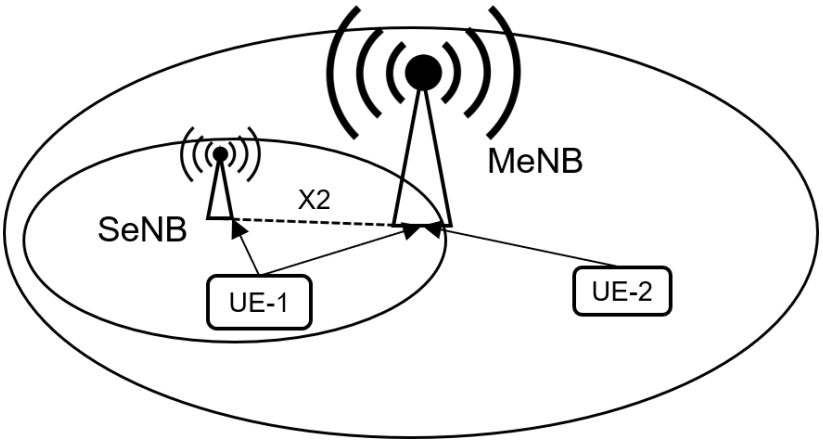

Fig. 3-1: UE-1 connected to a MeNB and a SeNB through dual connectivity, and UE-2 connected traditionally only to MeNB.

This connectivity system results in increased throughput by allowing radio spectrum resources to be reused more often due to the smaller sub-cell sizes, while allowing the MeNB to handle travel reconnection and the more frequent handovers that result by communicating over the X2 backhaul network.

4. OPEN RAN

4.1 Intro to O-RAN

Modern RAN systems typically follow an architecture that consists of proprietary front-end hardware, proprietary middleware, proprietary back end processing and proprietary software, all of which is incompatible with nearly any alternate hardware or software outside of the company that produces it. O-RAN systems are instead designed to be as modular and inter-interoperable as possible, as opposed to the described traditional RAN systems. The open nature of O-RAN also calls for open interfaces, and increased software content, leading to AI and machine learning algorithms to be put into usage, especially for the purpose of AI-enabled RAN controllers [1]. The addition of machine learning further increases the potential for optimization through the use of dynamic resource allocation. The purpose of O-RAN is to create systems based on open interfaces. It is expected that this will stimulate further technology innovation and competition amongst suppliers [1].

4.2 O-RAN Architecture

Currently O-RAN architectures are being investigated at the O-RAN Alliance – the organization that has taken leading role in the development of O-RAN standards. The O-RAN architectures are intended to complement but not replace the 4th and 5th generation of cellular standards developed by the Third Generation Partnership Project (3GPP). Fig. 4-1 shows an O-RAN architecture [1]. It is appropriate for the implementation of 5G gNBs. Although other O-RAN architectures are possible, in this thesis we will focus on the architecture developed by the O-RAN Alliance.

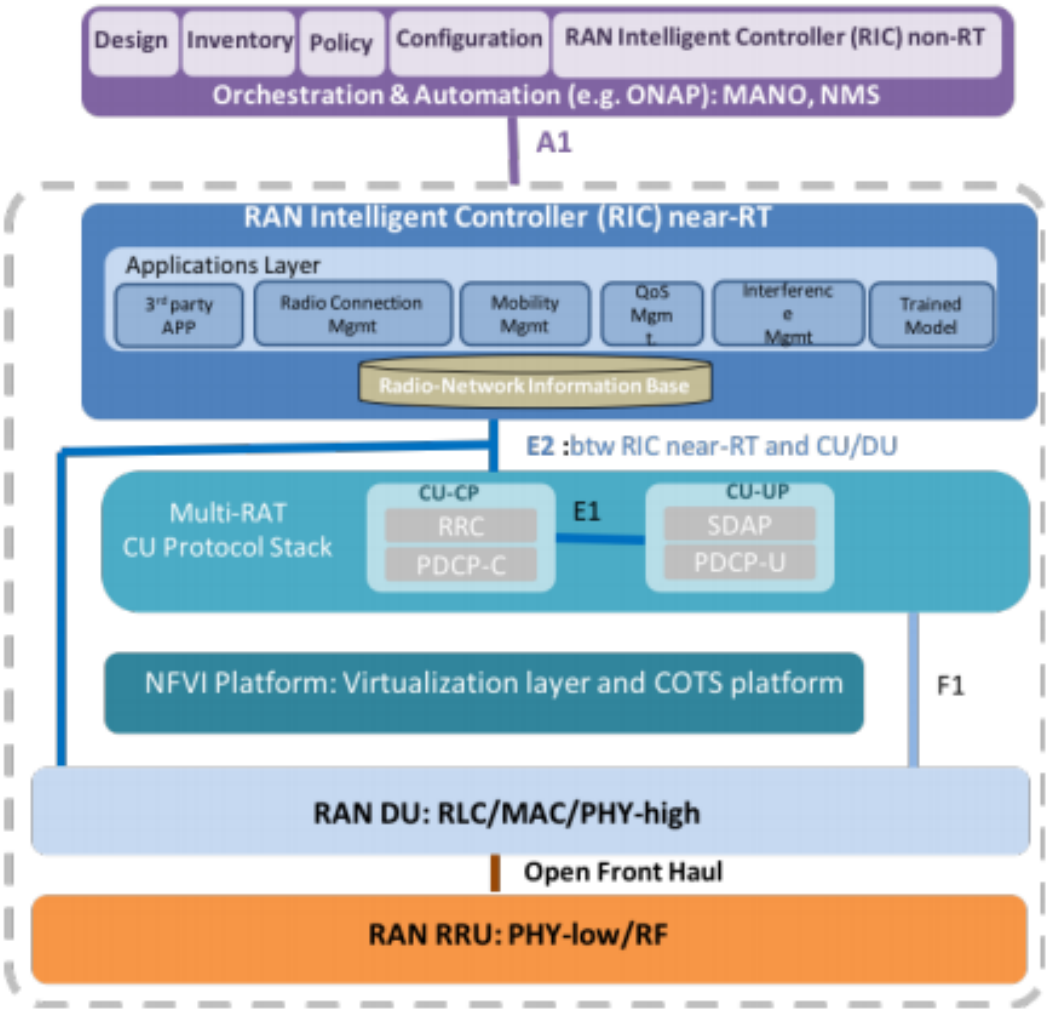

Fig. 4-1: The O-RAN Architecture. From [1].

O-RAN architecture utilizes what is known as a RAN intelligent controller (RIC). The RIC utilizes two planes of control within the architecture, one being non-real-time (Non-RT) and one being near-real-time (Near-RT). One important feature is the decoupling of the control-plane from the user-plane. In decoupling the control plane from the user-plane, the RIC also allows for a split of the centralized processing unit and the distributed processing unit for the purpose of implementing cloud processing and C-RAN. The decoupling of the user-plane also allows for it to become more effectively standardized, and to allow for use of data-driven allocation of radio resources in a closed-loop manner, utilizing the RIC Non-RT layer to allow for a wide range of methods including AI and machine learning [1].

4.2.1 The RIC Non-RT layer

The RIC Non-RT layer is the layer that handles all non-real-time (> 1s latency) control functionality. The Non-RT layer handles all service/policy management, and analytics and model-training algorithms for the near-RT RAN control and radio resource management (RRM) functionality. These trained models, intended for real-time RRM are distributed to the RIC near-RT layer for runtime execution.

The RIC Non-RT layer interface, called interface A1, connects the RIC Non-RT layer to the eNB/gNB layer that contains the RIC near-RT. Interface A1 is to be a standardized interface for reliable communication of data between the centralized unit (CU) to the network management applications of the RIC Non-RT. As the RIC Non-RT contains the model training algorithms and deploys them the to the RIC Near-RT, the RAN behavior can be modified through deployment of new models with various optimizations objectives and operator policies.

4.2.2 The RIC Near-RT layer

The RIC Near-RT layer utilizes the models trained by the RIC Non-RT layer with the intention of providing for the utilization of machine learning/AI controlled RRM algorithms. However the two layer split of the RIC also allows for implementation of traditional/legacy RRM through RRM model deployment. The RIC Near-RT layer also handles RB management interference detection and mitigation and is intended for handling of per-UE controlled load balancing. This layer also includes embedded intelligence algorithms including those handling QoS management, connectivity management, and seamless handover control algorithms. The RIC Near-RT layer also function as a platform for on-boarding of third-party control applications. The RIC Near-RT layer includes a database called the Radio-Network Information Base (R-NIB) for its functions by allowing for near real-time state capture of the underlying network via the E2 interface and the commands originating from the RIC Non-RT over the A1 interface.

The interfaces involved with the RIC-Near-RT layer are the A1 and E2 interfaces. The A1 interface, as described above, connects the DU RIC Non-RT layer to the modular CU containing the RIC Near-RT layer. The E2 interface is the connection between the RIC near-RT and the Multi-RAT CU protocol stack and the underlying RAN DU. This interface is an equivalent to the legacy RRM to RRC traditional connection interface, except as a standard interface between the RIC

Near-RT, and the CU/DU under the context of the architecture of an O-RAN system. The E2 interface is utilized by the RIC Near-RT for communication of configuration commands directly to the CU/DU, and is used for feeding of the required data to the RIC Near-RT for facilitation of RRM functions.

4.2.3 Multi-RAT CU protocol stack and platform

The Multi-RAT CU protocol stack processes all required RATs, and handles the connection, handover and communication protocols of each RAT to each user. The Multi-RAT CU protocol stack functions are executed according to control commands from the RIC near-RT.

In addition to the described E1, E2, and F1 interfaces, the Multi-RAT CU protocol stack contains several interfaces that require standardization that are not shown in Figure 4-1. These interfaces include the W1, X2, and Xn interfaces, and the component interfaces of F1, F1u and F1c. The layout of these interfaces is shown in Figure 4-2 below.

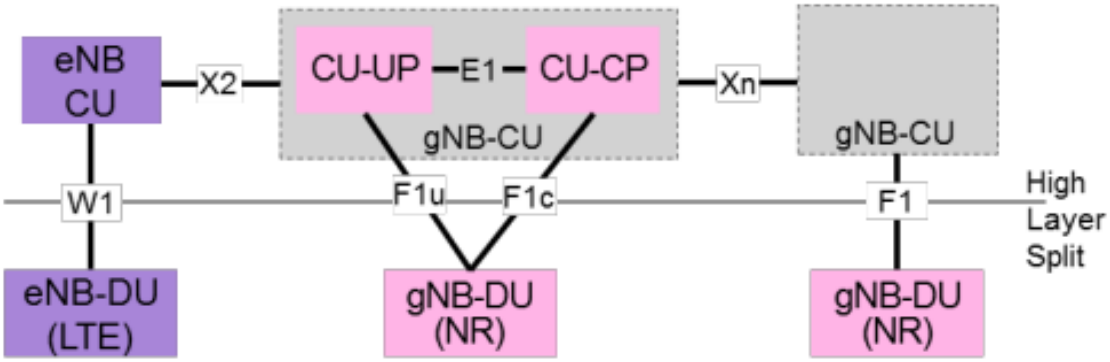

Fig. 4-2: The F1, W1, E1, X2, Xn and F1c/F1u interface layout. From [1].

4.2.4 RAN DU and RAN RRU

The RAN DU and RRU constitute the full front-end, including the RLC, MAC, and PHY control of the RF signal to a digital baseband signal that can be processed. The RAN RRU is similar to the discussed remote radio head (RRH), which collects and downconverts the RF signals into IF signals that can be processed by the BBU/DU.

The open front-haul interface between these two components must then be capable of conveying all signal data to the BBU/DU. The RRU must also send all required information needed for any intelligent RRM functions controlled by the RIC.

The open front-haul interface will be the interface that receives the main focus of this paper.

4.3 The O-RAN Open Front-Haul Interface

The O-RAN open front-haul interface is a physical interface that connects the DU and CU and must deliver the raw IQ samples of the analog RF front end RRH to the BBU pool. These IQ samples must therefore be encapsulated in a protocol and transmitted typically over a point-to-point connection. Multiple protocols for this transport exist, including the Open Base Station Architecture Initiative protocol (OBSAI), and Open Radio equipment Interface (ORI). The most common RRH to BBU interface currently in use is the Common Public Radio Interface (CPRI). Another interface, that has not received as much, attention is the VITA Radio Transport (VRT) interface. The VRT is the interface of focus for this thesis and will be discussed in detail.

4.4 O-RAN Split Options

The architecture of O-RAN systems is built upon the idea of a physical partition, where the base station functions are divided into categories. In such an architecture, the functional location of the front-haul interface is the determining factor of which functions are handled by the gNB/DU, and which are handled by the CU. The split options are visualized in figure 4-3; wherein the functions above the split are handled by the CU and the functions below the split are handled by the DU.

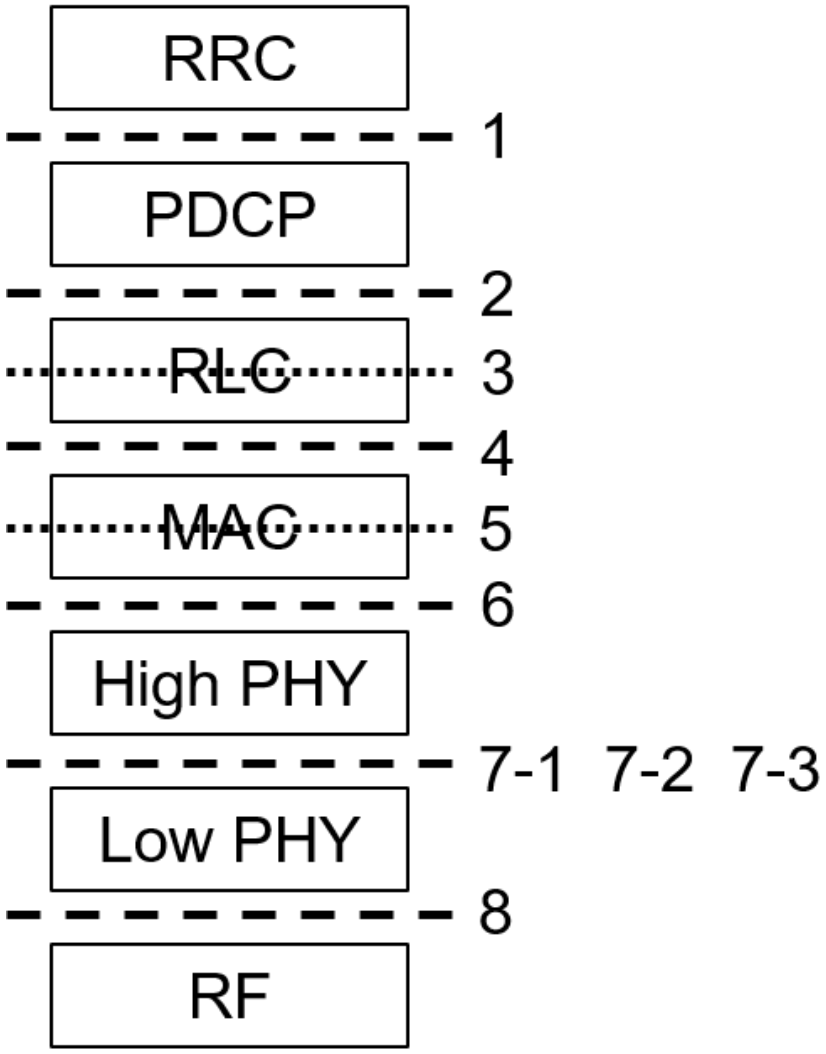

Fig. 4-3: The split options with respect to the functional stack.

The physical architecture of an O-RAN system relies on the functional split option chosen. For this reason, the choice of functional split is considered the central design question of O-RAN. The option chosen determines how many functions remain local to the BS, remaining closer to the user and relaxing the fronthaul network bitrate and delay requirements, and how many functions will be moved to the CU, where greater processing capabilities are possible. As can be seen in the figure above, the split has eight possible locations within the functional stack and includes sub-options in some locations.

Each functional split will be analyzed and compared in terms of the functions that are handled locally by the DU and which are centralized to the CU, and the impact of this split to the

network behavior. Split advantages, disadvantages, and use cases will also be presented. A method for fronthaul bitrate calculation will also be included.

The eight functional split locations shown in figure 4-3 have been proposed by 3GPP. In figure 4-3 the functional location of each split is marked by a number and a dashed or dotted line indicating the split option number. These options state that all lower-level functions be implemented in the DU, and all higher level functions be implemented at the CU. Functions left in the DU are intended to have close proximity to the users, and are to be located physically at the antenna mast. Those functions to be located at the CU will have access to the centralized high-processing-power datacenter called a CU-pool. The required data rate of the fronthaul network is directly dependent on this split, as more functions, and therefore more processing being done at the DU means that less processing must be done at the CU, and therefore requires a lower fronthaul bitrate.

There are three layers in the 3GPP protocol stack, the lowest being the physical layer, followed by the data link layer with the network layer on top. Within this stack, the physical layer handles conversion between digital bits and waveforms; from bits to a waveform in the Downlink (DL) and the reverse for the Uplink (UL). If the functional split resides in the physical layer, the bitrate for the fronthaul becomes dependent on the maximum number of concurrent users in the current cell, as each subframe contains a certain number of symbols per user.

In general, the per-user data rate is equal to: No of subcarriers * No of symbols per frame * No of antennas.

Several terms are defined here for the bitrate calculations:

- Sample rate (SR) - the number of samples per second,
- Number of I and Q bits (BTW)
- Number of antenna ports (AP), which is the number of antennas connected to the DU
- Number of subcarriers (SC)
- Number of symbols (SY)
- Number of layers (LA)
- Number of layers for control signaling (CLA)
- Peak rate (PR) measured in Mbps
- Schedule/control signaling rate (CR) measured in Mbps

- Bandwidth (BW)
- Bandwidth for control signals (CBW)

Some Assumptions must also be made in order to calculate the required bitrate for each split. The assumptions are summarized in table 4-1 From [20], and are based on the requirements of 5G.

Table 4-1: The assumptions made for split bandwidth calculations.

Items	Assumption	Applicability
Channel Bandwidth	100MHz(DL/UL) (5x20MHz aggregation)	All options
Modulation	256QAM(DL/UL)	
Number of MIMO layer	8(DL/UL)	
IQ bitwidth	2*7bit(DL), 2*10bit(UL)	Option 7-1 Option 7-2 Option 7-3
	2*16bit(DL/UL)	Option 8
Number of antenna port	32(DL/UL)	Option 7b Option 7c(UL) Option 8

4.4.1 Split Option 8

Split option 8, located just after the RF block, requires the lowest number of functions to be implemented in the DU. Above this split is the link the MAC layer through the data link layer which transmits transport blocks between the physical and data link layers.

This 3GPP split option is the most common as it is the traditional split of RRH-BBU that has been around for several years. This split leaves only the RF sampler and upconverter within the DU. This split leaves the DU as a highly simplistic architecture that can easily support different RATs. All other functions are centralized to the CU, allowing the greatest potential for processing resource sharing for a split. This split allows for efficient implementation of functions such as CoMP and mobility and efficient resource management due to the centralization of the protocol stack. This can also offer greater robustness in non-ideal transmission conditions, as the ARQ is centralized to the CU.

Split option 8 requires a bitrate that is constant, but very high and scales with the number of antennas. This makes split option 8 difficult to scale in massive MIMO scenarios.

As defined by 3GPP in [20] the DL fronthaul bitrate for split option 8 is:

*FH bitrate = SR*BTW*AP*(100/20)* = 30.72*32*32*5 = 157.3Gb/s eq. 4-1

As defined by 3GPP the UL fronthaul bitrate for split option 8 is:

*FH bitrate = SR*BTW*AP*(100/20)* = 30.72*32*32*5 = 157.3Gb/s eq. 4-2

In both equations, the signal rate is 30.72Mbps, the number of I and Q bits is 32, the number of antenna ports is 32, across 100MHz/20Mhz = 5 channels.

This split option has the highest fronthaul bitrate among the 3GPP split options and is mainly possible for operators with cheap fronthaul network access. Split option 8 is capable of achieving the greatest multiplexing gain to the CU resources and has the highest potential energy efficiency.

In current systems, this split is implemented using CPRI. CPRI is a constant bitrate fronthaul interface that utilizes a time division multiplexing protocol with a regular framing interval. CPRI is specifically designed for use in transporting sampled radio waveforms, and defines several different line bitrates in order to meet the flexibility and cost efficiency requirements. CPRI however, leaves some parts of its protocol to be vendor proprietary, which complicates interoperability between vendors.

4.4.2 Split Option 7-1

Split option 7-1 differs from split option 8 in that the Fast Fourier Transform (FFT) is localized to the DU. This change of the FFT to the DU means that the fronthaul interface transmission data is represented by subcarriers. This split allows the DU to remove the guard subcarriers from the signal by removing the cyclic prefix and using the FFT to transfer the signal into frequency-domain. Removal of the guard subcarriers reduces the fronthaul bitrate as compared to option 8. In split option 7-1, the fronthaul transmission is still constant bitrate, as unused subcarriers detection, which is required to achieve a variable fronthaul bitrate, requires resource element mapping, which is still centralized to the CU. CoMP functions are supported in this split option, using JR for DL and JT for UL. The ARQ is localized to the CU in this option, resulting

in greater robustness. In split 7-1, the DL MAC info is 713.9Mbps, and the UL MAC info is 120Mbps.

As defined by 3GPP in [20] the DL fronthaul bitrate for split option 7-1 is:

*FH bitrate = SC*SY*AP*(BTW*2)*1000 +MAC info* eq. 4-3

= (1200*5)*14*8*(7*2)*1000+713.9M = 9.8Gb/s

As defined by 3GPP in [20] the UL fronthaul bitrate for split option 7-1 is:

*FH bitrate = SC*SY*AP*BTW *2*1000 +MAC info* eq. 4-4

= (1200*5)*14*8*(7*2)*1000+120M = 15.2Gb/s

These fronthaul bitrates are significantly smaller than those of split option 8 (22.2Gbps/21.6Gbps as compared to 157.3Gbps), are constant-bitrate, and are largely independent of UE traffic.

4.4.3 Split Option 7-2

Split option 7-2 includes precoding and resource element mapping in the DU, in addition to those functions included in the split options described above. This inclusion of the precoding and resource element mapping means that the DU will increase in complexity, however it will also result in a lower bitrate, as the fronthaul link will mainly transport subframe symbols. This split option will also lower the potential for processing resource sharing, as some of the digital processing is removed from the CU. This split, and all splits proceeding place the FFT and resource element mapper within the DU, and thus will have a variable bitrate for the fronthaul link. CoMP functions are supported in this split option, using JR for DL and JT for UL. The ARQ is localized to the CU in this option, resulting in greater robustness. In split option 7-2, the DL MAC info is 121Mbps, and the UL MAC info is 80Mbps [20].

As defined by 3GPP in [20] the DL fronthaul bitrate for split option 7-2 is:

*FH bitrate = SC*SY*LA*BTW*2*1000 +MAC info* eq. 4-5

= (1200*5)*14*8*7*2*1000+121M = 9.2Gb/s

As defined by 3GPP in [20] the UL fronthaul bitrate for split option 7-2 is:

*FH bitrate = SC*SY*LA*BTW *2*1000 +MAC info* eq. 4-6

= (1200*5)*14*32*16*2*1000+80M = 60.4Gb/s

The main factors in the fronthaul bitrate for split option 7-2 are the number of symbols, the quantized bits per symbol, and the control information required to carry out PHY processing. This split options gains extra overhead from scheduling, synchronization, and data frames. This split option offers lower fronthaul bitrate requirements and potential for reduction of synchronization requirements, however it makes latency constraints more difficult to meet. With the increase of functions in the DU, split option 7-2 and beyond are capable of variable fronthaul data rates and enable more time sensitive networking through technologies such as a packet-based fronthaul network.

The O-RAN alliance architecture presented above uses a modified version of this split option known as 7-2x. in this modified split, the resource element mapping is not local to the DU, however digital beamforming operations are local.

4.4.4 Split Option 7-3

This split option includes scrambling, modulation, and layer mapping in the DU. This is expected to lower the bitrate requirement for the fronthaul link, as the modulation, which maps several bits per symbol, is included in the DU. This option includes the FEC close to the MAC, in the CU-pool. This split option requires an in-band protocol for modulation, multi-antenna processing, and PRB allocation support, due to the high-level split within the physical layer. This split option may limit support for CoMP functionality due to latency but is still potentially capable of coordinated scheduling. The ARQ is localized to the CU in this option, resulting in greater robustness. This split option also has additional processing delay at the DU, as compared to split option 8, due to the implementation of modulation at the DU.

Split option 7-3 includes additional overhead due to scheduling control, synchronization, and data framing. The fronthaul bitrate estimate for this split option is the same as for option 7-2. This split option is considered by 3GPP only for the DL.

4.4.5 Split Option 6

Split option 6 places the functional split directly between the data link layer and the physical layer by splitting between the MAC and PHY functional layers. In this split option, all physical processing is localized to the DU, while the MAC scheduler remains centralized to the CU. This split option reduces the CU pooling gain from option 8 by removing all physical processing from the CU, allowing only data link and network layer functions to benefit from processing resource sharing, which may only make up about 20% of baseband processing, depending on the implementation. In this split option, the fronthaul link transmits transport blocks, making the fronthaul load dependent on the load at the 3GPP S1 interface.

Extra overhead in this split option comes from the scheduling control, synchronization, and data frame. This split option has a greater overhead in regard to scheduling control. Split option 6 DL has a peak rate of 196Mbps, and a control signaling rate of 5Mbps [20]. The split option 6 UL has a peak rate of 75Mbps, and a control signaling rate of 44Mbps [20].

As defined by 3GPP in [20] the DL fronthaul bitrate for split option 6 is:

$$FH\ bitrate = (PR+CR)*(BW/CBW)*(LA/CLA)*(8/6) \quad \text{eq. 4-7}$$

(196M+5M)*(100/20)*(8/2)*(8/6) = 5626.7Mb/s

As defined by 3GPP in [20] the UL fronthaul bitrate for split option 6 is:

$$FH\ bitrate = (PR+CR)*\ (BW/CBW)*\ (LA/CLA)*(6/4) \quad \text{eq. 4-8}$$

(75M+45M)*(100/2)*(8/1)*(6/4) = 7140Mb/s

In this split option, time critical processes are still centralized in the CU, meaning that it has very strict requirements in regards to delay, like the splits in the physical layer. This split option makes implementation of CoMP challenging. The ARQ is localized to the CU in this option, resulting in greater robustness. This split option requires an in-band protocol for modulation, multi-antenna processing, and PRB allocation support, due to the high-level split of the physical and MAC layers. This split option has a significantly lower fronthaul bitrate requirement compared to split option 8.

4.4.6 Split Option 5

This split option moves a MAC sublayer to the local processing of each DU to handle time critical processing, and retains the overall scheduler central to the CU. This split options and all split options moving forward handle all time critical procedures in the HARQ, and all functions where performance and latency are proportional locally in the DU. In this split option, the CU-pool and DUs communicate via scheduling commands and HARQ reports. Split option 5 has reduced delay requirements due to time sensitive processing being localized to the DU, which increases the maximum distance between the CU-pool and the DUs, however this localizes much of the processing to the DU, and significantly limits the benefit of shared processing. The low MAC sublayer is controlled by the high MAC sublayer, which also manages Inter-Cell Interference Coordination (ICIC). It is possible that the increased distance between the CU and the DUs will lead to greater latency, limiting the capability to implement CoMP UL JR.

The Fronthaul bitrate requirements for this option are the same as for option split 6.

The fronthaul transmissions for this split option consist of pre-multiplexed higher-layer datagrams and scheduling commands. This split option allows the MAC scheduler in the CU to create bundles of multiple low speed subframes while simultaneously operating the MAC scheduler and HARQ at high speed. The ARQ is localized to the CU in this option, resulting in greater robustness. This split option benefits from decentralized HARQ and relaxation of latency requirements. This split is limited in it’s inter-cell interference reduction capability relative to options 7 and 8.

4.4.7 Split Option 4

This split option is set between the RLC and MAC layers, localizing the MAC layer to the DU while the RLC is centralized to the CU. The fronthaul transmission between the CU and DU consists of RLC Protocol Data Units (PDUs) in the DL, and MAC Service Data Units (SDUs) in the UL. The virtualized RLC processing allows for storage and processor utilization sharing. The ARQ is localized to the CU in this option, resulting in greater robustness. With split option 4, the peak rate on DL is196Mbps, and 75 on UL.

As defined by 3GPP in [20] the DL fronthaul bitrate for split option 4 is:

FH bitrate = PR(BW/CBW)*(LA/CLA)*(8/6)* eq. 4-11

= 196M*(100/20)*(8/2)*(8/6) = 5226.7Mb/s

As defined by 3GPP in [20] the UL fronthaul bitrate for split option 4 is:

FH bitrate = PR(BW/CBW)*(LA/CLA)*(6/4)* eq. 4-12

= 75M*(100/20)*(8/1)*(6/4) = 4500Mb/s

The RLC and MAC are closely linked in the DL, due to scheduling decisions being made by the MAC every TTI, and preparation of data by the RLC being an on-request operation. Because of this link, the fronthaul link must either have a very tight latency requirement, or a flow control scheme with a data buffer must be implemented at the DU in order to support the functional split.

The benefits offered by this functional split are significantly lower than those offered by other split option, and some consider this option as non-beneficial in regard to protocols such as LTE. The RLC/MAC are closely linked, making this split impractical, particularly in the shorter subframe sizes in 5G.

4.4.8 Split Option 3

Split option 3 separates the RLC into a high RLC and a low RLC. Segmentation functions are contained in the low RLC at the DU, and all other RLC functions, including ARQ are in the high RLC centralized to the CU. In this split, the CU handles all UP processing of PDCP and asynchronous RLC processing. All other RLC functions, including synchronous RLC network functions, are handled by the DU. This split option allows the possibility of a single RLC entity being associated with multiple MAC entities. In split option 3, all real-time operations and scheduling are local to the DU, reducing the constraints on fronthaul latency. The ARQ is localized to the CU in this option, resulting in greater robustness. The fronthaul bitrate of this split option is roughly the same, although lower than split option 2.

4.4.9 Split Option 2

This split option localizes all functions to the DU except for the PDCP and RRC functions. In this split, PDCP SDUs are transmitted over the fronthaul in the DL direction, and RLC PDUs are transmitted over the fronthaul in the UL direction. This split has a standardized interface that is similar to the 3C architecture in LTE dual connectivity. This standardized interface simplifies the interoperation of the constituent elements. The dual connectivity allows for multi-connectivity support through transmission of some of the PDCP PDUs to the RLC. This splits the traffic into multiple flows that can then be directed to access nodes. This split option places all real-time operation locally in the DU, thus having the most relaxed fronthaul latency requirements of all the splits. To ensure packets are kept in the correct order, both the DU and CU require re-sequencing buffers. In this split the potential for coordinated scheduling is limited, but may use of beamforming may be able to compensate. With the PDCP centralized, header compression protocols leading to statistical multiplexing gain in aggregation points is possible. The signaling bits apply to an estimated 10% of UEs [20].

As defined by 3GPP in [20] the DL fronthaul bitrate for split option 2 is:

FH bitrate = PR(BW/CBW)*(LA/CLA)*(8/6) + signaling* eq. 4-13

150M*(100/20)*(8/2)*(8/6)+signaling = 4016Mb/s (bandwidth)

As defined by 3GPP in [20] the UL fronthaul bitrate for split option 2 is:

FH bitrate = PR(BW/CBW)*(LA/CLA)*(6/4) + signaling* eq. 4-14

50M*(100/20)*(8/1)*(6/4)+signaling = 3024 (bandwidth)

Some believe that split option 2 has only minimal gain over current technologies such as a fully integrated evolved NodeB (eNB), due to most of the functions being localized to the DU, moving only the PDCP and RRC to the CU, resulting in minimal pooling gain and cost reduction. Split option 2 supports centralized over-the-air encryption and handover procedure and coordination of mobility potential. The potential for coordinated scheduling between DUs is limited.

4.4.10 Split Option 1

Split option 1 places the entirety of the UP locally in the DU. This full localization of the UP is beneficial regarding caching by keeping user data near the point of transmission. The lack of centralized PDCP processing in this split option means that it will not support many inter-cell coordination functions. While most functions are handled locally in this split, the centralization of RRC adds potential for faster mobility management, and removes the management and maintenance requirements of the X2 interface.

As defined by 3GPP in [20] the D fronthaul bitrate for split option 1 is:

FH bitrate = PR(BW/CBW)*(LA/CLA)*(8/6)* eq. 4.15

150M*(100/20)*(8/2)*(8/6) = 4Gb/s

As defined by 3GPP in [20] the UL fronthaul bitrate for split option 1 is:

FH bitrate = PR(BW/CBW)*(LA/CLA)*(6/4)* eq. 4.16

50M*(100/20)*(8/1)*(6/4) = 3Gb/s

Due to the fact that all CP functions are contained in the RRC and all UP functions operated in the PDCP and above, this split is sometimes called a CP/UP split. The CP and UP are tightly coupled, and this split attempts to separate them physically which can present a challenge. In this split, the CU is effectively a multi-DU CP and has limited application in wide area deployment.

4.4.11 A Flexible split

A more recent idea is the suggestion that the functional split does not have to be static, but instead can have functions localized or distributed for certain situations, and change which functions are handled by the DU or CU in other situations. This flexible split option is known as RAN-as-a-Service (RANaaS). In RANaaS, the functions that are centralized or distributed can be chosen and changed to whatever happens to be the optimal operating point for a system at any point in time. This partial functionality, however, is clearly not possible for a system with a front-end that has limited capabilities, and requires that any part of the front-end processing that should be available for transference to the DU.

Each split option, having a different set of functions centralized to the CU or localized to the DU, has its own pros and cons. The ability to switch between different split options allows a user to take full advantage of the benefits of whichever split option best suits the transmission situation and protocol.

One possible way of implementing a flexible split is to have a standard split option set for each protocol and data type, manually setting the most likely option to be optimal to a given scenario. It is true however, that different split options will be optimal based on a very large number of variables, which makes hardcoding which option to use incredibly complicated.

Another option is the use of AI in deciding which option is optimal for a given situation. This is one possible use of the embedded AI/Machine learning capabilities that O-RAN is intended to support in the RIC. The RIC may be capable of training models that can decide which split option is best for a given transmission scenario, allowing the algorithm to develop as needed over time. This method of split flexibility however, requires that the front-end systems be capable of transmitting all information pertaining to the models being build by the RIC to the back end processing. This method also requires that the front-end is capable of collecting the necessary information for transmission to the back end.

4.5 The Fronthaul Gateway

In split architecture, the DU and CU must communicate over some fronthaul link, this link has the option of utilizing a device known as a Fronthaul Gateway (FHGW). A FHGW device aggregates multiple radio units together at the functional split level as shown in figure 4-4. The connection between he FHGW and the DU is defined as the Back End fronthaul interface, and the connection between he FHGW and the CU is defined as the Front End fronthaul interface. These interfaces have the option of being co-located or having a shared hardware configuration.

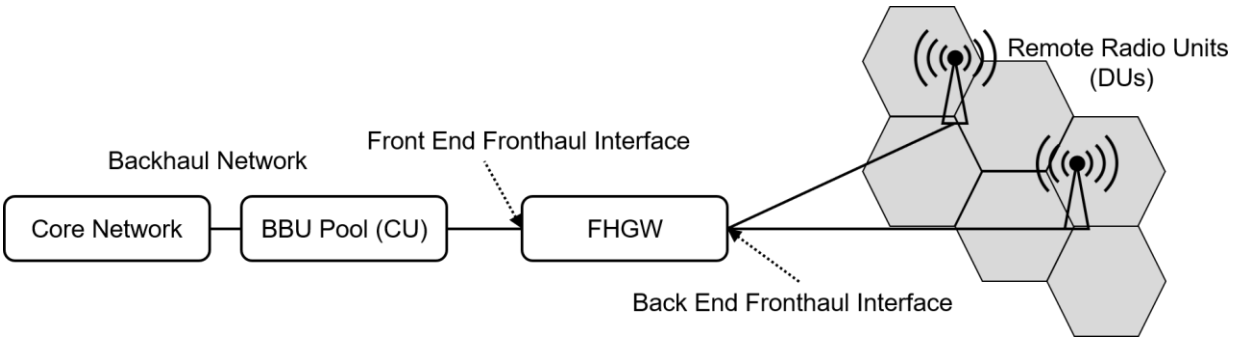

Fig. 4-4: The FHGW interfaces and aggregation.

Use of a fronthaul gateway is a cost-effective enabler of shared cell architecture. The FHGW enables a large number of DU radio units to be grouped together as seen in figure 4-4 above. This allows the DUs to share radio resources within a single cell.

This thesis will give an overlook description of split architecture O-RAN systems that use VRT framework in the FHGW before proposing a testbed platform for such systems.

5. THE VITA 49.2 STANDARD

5.1 Intro to VITA 49

In the implementation of SDR systems, there must be a point between the antenna and signal processing where the signal is converted from the analog RF signal to a digital signal that a computer can process. Due to the high-frequency nature of radio signals being higher frequency than a typical processor can keep up with, these signals are often converted to or from an IF frequency or baseband, between the ADC/DAC and the system transceivers. In a typical wireless architecture, this leads to a network of proprietary IF distributers that must connect each of the High, Mid, and Low band transceivers to each signal processor's ADC, as in the traditional "stovepipe" software radio shown in figure 5-1.

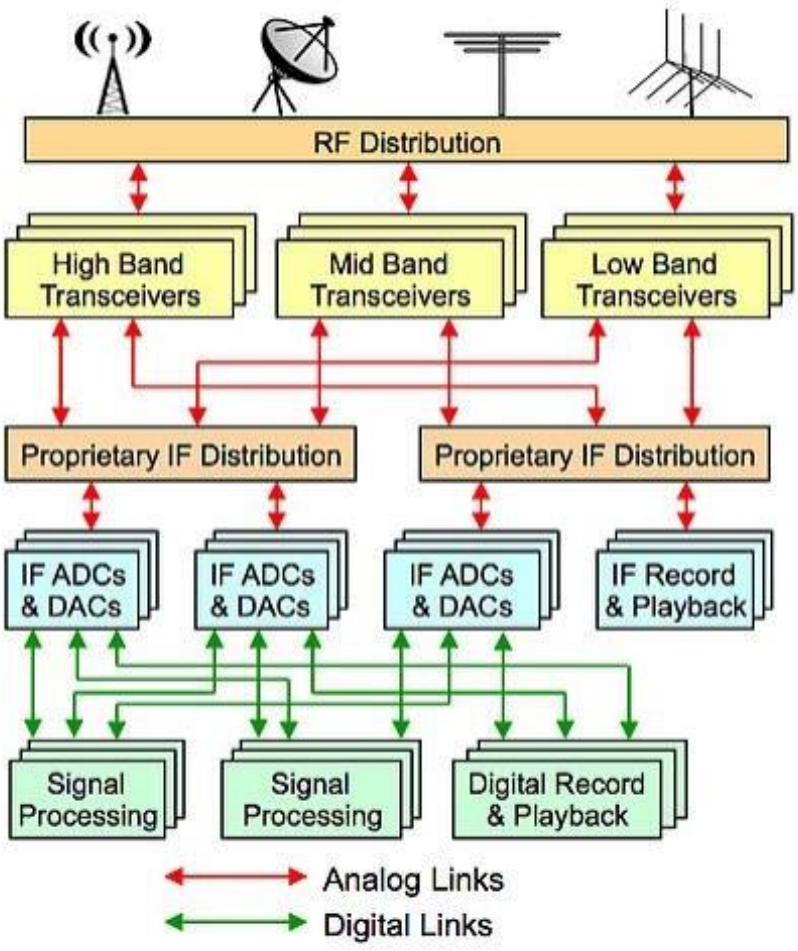

Fig. 5-1: The "Stovepipe" Radio Architecture. From [9].

The VRT protocol, known as the VMEBUS International Trade Association (VITA) 49 standard functions as a layer standard that provides for analog to digital conversion that can be significantly closer to the antenna. This layer removes the need for proprietary IF distribution by combining the transceivers with the ADC/DAC, and relegating the signal distribution to a digital network layer that can directly connect to each to each signal processor. The VRT protocol also provides a stream of context (or metadata) packets that are multiplexed with the payload packets to the digital hardware, providing the digital hardware with context information about incoming signals. This metadata provides the necessary information for the proper handling of signals, and for the use of intelligent radio control and radio resource management.

It is worth noting that despite the name VITA radio transport, VRT is not a transport protocol, but rather a standard for packet definition. It is also worth noting that VITA 49 is not a networking technology and can be used on top of any networking technology such as Ethernet, etc. VITA 49 instead is an interoperable interface framework, from which specifications can be derived, intended for allowing modularity of components, and interoperability of parts made by different manufacturers.

5.2 VRT Receiver Architecture

The VITA 49 receiver is capable of gathering and relaying a set of metadata to the VRT encoder, allowing the encoder to pack the data into a context packet with a stream identifier to be sent through a packet multiplexer to the digital hardware, as shown in figure 5-2.

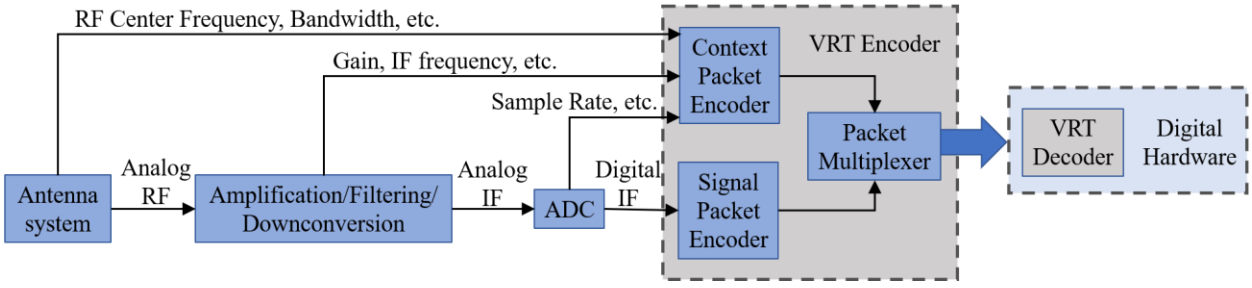

Fig. 5-2: VRT Radio Receiver Architecture

The VRT Radio Receiver Architecture starts with an analog front end that includes the antenna system, the front-end amplifiers and filters, the IF downconverters, and the analog to digital converters. The next part of the architecture is the VRT encoder, which consists of a signal

packet encoder, a context packet encoder, and a multiplexer for serial delivery of signal payload and context packets to the VRT decoder within the digital hardware. In this architecture, the data is collected by the context packet encoder within the VRT encoder from the analog front-end components and encoded into a packet with a stream identifier. The stream identifier tells the digital hardware which context packets contain metadata pertaining to each signal.

The delivery of context information of each signal from the front-end hardware to the digital hardware assists a digital radio controller in employing the use of intelligent radio control techniques, as the context information contains data regarding the timing and spectral usage of incoming signals. Knowledge of spectrum usage assists in the implementation of many advanced RRM algorithms, including AI and machine learning controlled RRM.

The architecture of the VITA 49 exciter is similar to that of the receiver, replacing the context packet encoder with a control packet decoder, and reversing the flow of data, as shown in figure 5.3.

5.3 VRT Packets

5.3.1 VRT Packet Overview

VRT functions on a packet-based system, using different packet types for signal data, signal context data, and control. These packet types are further separated into six categories; Signal Data, Context, Command, Extension Signal Data, Extension Context, and Extension Command. Table 5-1 shows the content of these types.

Table 5-1: The contents of each of the six VRT packet types.

Contents	Standard Formats	Custom Formats
Data	Signal Data Packets Convey a digitized signal (Signal Data) • Real/complex data • Fixed/floating-point formats • Flexible packing schemes	Extension Data Packet Conveys any signal or any data derived from a signal • Any type of data • Custom packet format
Context	Context Packet • Conveys common Context for Signal Data • Spatial • Spectral • Signal • Waveform • Identifiers • Discrete IO • Array of Records • etc.	Extension Context Packet Conveys additional Context for Signal Data or Extension Data • Any kind of metadata not available in Context Packets • Custom packet format
Command	Command Packets Control mechanism to set the context field types shown in the context section	Extension Command Packet Control mechanism to set attributes not defined in the V49.2 control packets

The full list of VRT packet types includes eight total packet types:

- Signal Data with Stream ID
- Signal Data Extension with Stream ID
- Signal Data without Stream ID
- Signal Data Extension without Stream ID
- Signal Context
- Signal Context Extension
- Command
- Command Extension

The optional Stream identifier is used for the association of packets into information streams, as well as a reference to the source or destination of the information of a packet within a component, system, or system of systems [14].

5.3.2 VRT Packet Structure

The VRT standard contains a generic common template for VRT packets that includes several mandatory and optional parts. All VRT packet types follow this template, which allows for variability of fields, subject to some rules, from [14]:

- The packet shall be in big-endian byte order.
- The order of the fields in a VRT packet shall be organized as shown in [Figure 5-3].
- When an optional field is not present in a VRT Packet Class, the remaining words in the packet shall “move up” toward the header, with no padding.

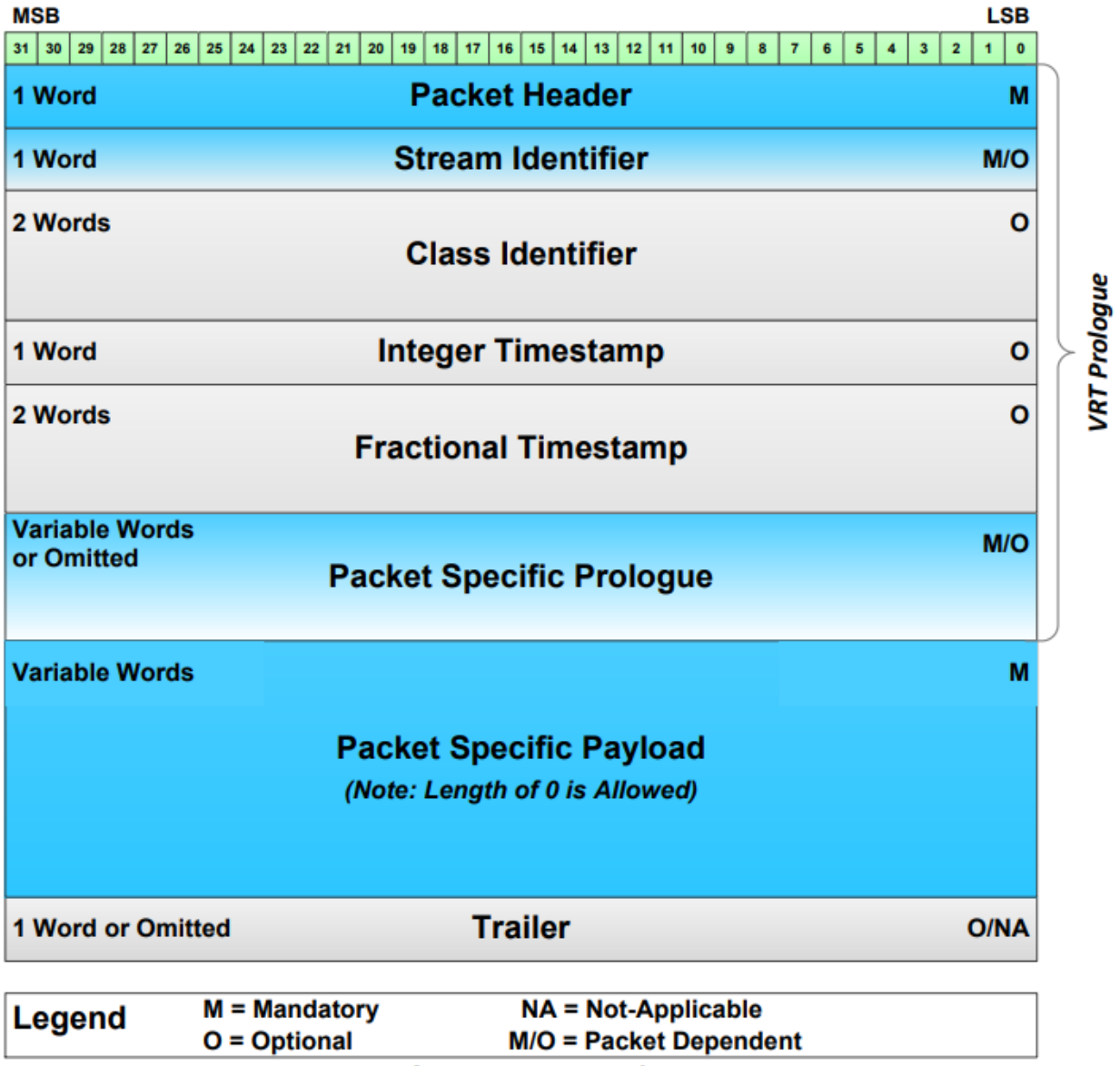

Fig. 5-3: The common VRT packet template [14].

- The requirements for prologue and trailer fields for each packet type shall comply with [Table 5-2].

Table 5-2: VRT Packet Field vs Packet type inclusion Matrix. From [14].

	Signal Data	Extension Data	Context	Extension Context	Control / Ack	Extension Cntrl / Ack
Header	M	M	M	M	M	M
Stream Identifier	O	O	M	M	M	M
Class Identifier	O	O	O	O	O	O
Integer-seconds Timestamp	O	O	O	O	O	O
Fractional-seconds Timestamp	O	O	O	O	O	O
Packet-Specific Prologue fields	NA	NA	NA	NA	M/O	M/O
Payload	M	M	M	M	M	M
Trailer	O	O	NA	NA	NA	NA

Legend
M = Mandatory
O = Optional
NA = Not-Applicable

For each packet, the packet header first indicates the packet type, then has a series of indicators that convey which optional parts of the packet are or are not present. Next is the timestamp integer (TSI) code, which indicates which, if any type of timestamp is present, followed by the timestamp fractional code, which indicates the type, if any of the fractional-second timestamps are included. Next is the packet count field, which indicates the modulo-16 packet count of a VRT stream. At the end of the header is the packet size field, which contains 16 bits indicating the number of 32-bit words contained in the VRT packet, including the header, payload, and all optional fields. The template of the packet header is shown in figure 5.3.2-2.

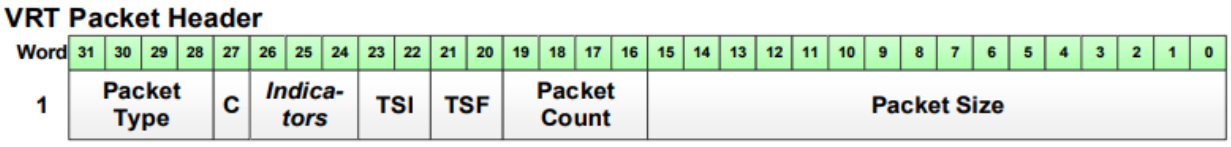

Fig. 5-4: The VRT packet header. From [14].

The stream identifier is a 32-bit number that is assigned to a VRT packet stream. VRT requires that all packets of the same stream use the same stream ID, and that all packets in a stream

either include or omit the stream ID consistently. It is also required that a stream ID be used when context packets are paired with data packets.

The class identifier field contains three important subfields; the Organizationally Unique Identifier (OUI), the information class code, and the packet class code.

- Organizationally unique identifier: A 24-bit number for identifying the company or VRT profile that created the information class and the packet class generating the IF data packet stream.
- Information Class Code: Indicator for which of the company's is used to define the information stream containing the packet stream.
- Packet Class Code: Identifier for which of the company's packet classes was used the make the packet.

The structure of the Class ID field is shown in figure 5-5.

Fig. 5-5: Contents of the Class ID Field. From [14].

The integer timestamp and fractional timestamp together constitute the packet timestamp. Timestamps are generated relative to a device or system reference time point. The integer timestamp is a 32-bit word measured in integer-seconds from the reference time point, and the fractional timestamp is a 64-bit unsigned integer, measured in fractional-seconds, included for adding increased resolution to the timestamp.

The packet specific prologue and packet specific payload contain the payload data of the VRT stream packet, and their contents depend on the packet type. The contents of the payload field were shown above in table 5-1.

The signal data and extension data packets contain an optional trailer field. The trailer field contains a set of indicators for data events to be conveyed from a VRT emitter to a VRT receiver. The indicators in this field are for conveying any event that affects any portion of the signal data packet payload. The individual packet structures for each packet type are shown in figure 5-6.

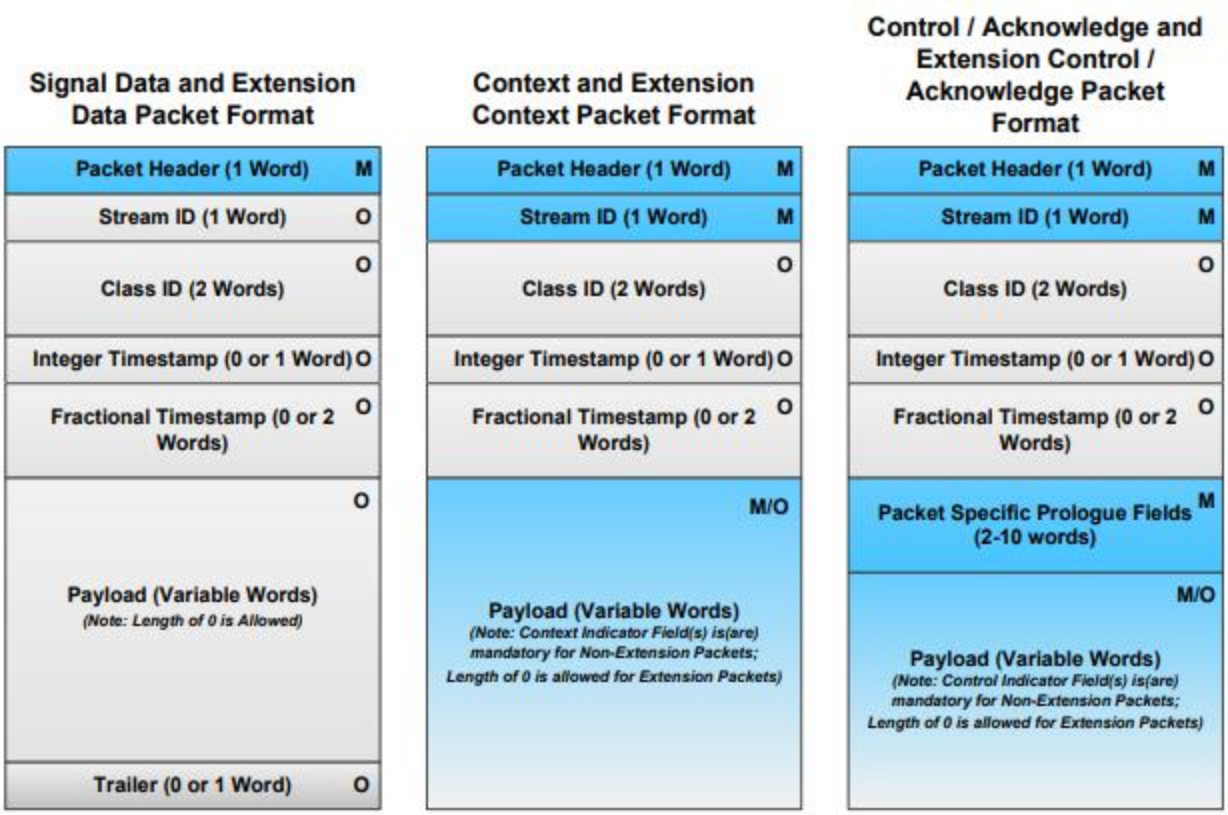

Fig. 5-6: The Packet Structure for Each of the VRT Packet Types. From [14].

5.4 VITA 49 Context Information

The VITA 49.2 standard, as described above is capable of transmitting what are known as "context packets". These packets have the capability of conveying to the back end any and all necessary information about the front-end hardware, software, and processes, as well as all information regarding the signal and any transformations made to it. The standard VITA 49 context packet contains a standard list of potential context information to be sent.

Context packets have use, but only in the UL direction from the RF/PHY to the back end. In the VITA 49 standard, when transmitting in the DL direction, the stream may contain packets known as "command packets". The command packets follow the same bit layout as the context packet, and are used in a similar fashion, however rather than containing information on what has been done to a packet, the command packets contain control information for the front-end to utilize in transmitting data. The standard VITA 49 command packet also has a list of possible information fields. The list of possible information fields for the standard VITA 49 context and command

packets is in Table 5-3. This table also shows on the left the bit each context item uses in the context indicator field within the packet.

Table 5-3: The possible information fields for VITA 49 context and command packets. From [14].

Bit	CIF 0 (V49.0) *Legacy Fields, CIF enables*	CIF 1 (V49.2) *Spatial, Signal, Spectral, I/O, Ctl*	CIF 2 (V49.2) *Identifiers (tags)*	CIF 3 (V49.2) *Temporal, Environmental*	CIF 7 (V49.2) *Attributes*
31	Context Field Change Indicator	Phase Offset	Bind	Timestamp Details	Current Value
30	Reference Point Identifier	Polarization	Cited SID	Timestamp Skew	Average Value
29	Bandwidth	3-D Pointing vector	Sibling(s) SID	*Reserved*	Median Value
28	IF Reference Frequency	3-D Pointing Vector Structure	Parent(s) SID	*Reserved*	Standard Deviation
27	RF Reference Frequency	Spatial Scan Type	Child(ren) SID	Rise Time	Max Value
26	RF Reference Frequency Offset	Spatial Reference Type	Cited Message ID	Fall Time	Min Value
25	IF Band Offset	Beam width	Controllee ID	Offset Time	Precision
24	Reference Level	Range (Distance)	Controllee UUID	Pulse Width	Accuracy
23	Gain	*Reserved*	Controller ID	Period	1st Derivative (Velocity)
22	Over-range Count	*Reserved*	Controller UUID	Duration	2nd Derivative (Acceleration)
21	Sample Rate	*Reserved*	Information Source	Dwell	3rd Derivative
20	Timestamp Adjustment	E_b/N_o BER	Track ID	Jitter	Probability
19	Timestamp Calibration Time	Threshold	Country Code	*Reserved*	Belief
18	Temperature	Compression Point	Operator	*Reserved*	*Reserved*
17	Device Identifier	2nd and Third-Order Intercept Points	Platform Class	Age	*Reserved*
16	State/Event Indicators	SNR/Noise Figure	Platform Instance	Shelf Life	*Reserved*
15	Signal Data Packet Payload Format	Aux Frequency	Platform Display	*Reserved*	*Reserved*
14	Formatted GPS	Aux Gain	EMS Device Class	*Reserved*	*Reserved*
13	Formatted INS	Aux Bandwidth	EMS Device Type	*Reserved*	*Reserved*
12	ECEF Ephemeris	*Reserved*	EMS Device Instance	*Reserved*	*Reserved*
11	Relative Ephemeris	Array of CIFS	Modulation Class	*Reserved*	*Reserved*
10	Ephemeris Ref ID	Spectrum	Modulation Type	*Reserved*	*Reserved*
9	GPS ASCII	Sector Scan/Step	Function ID	*Reserved*	*Reserved*
8	Context Association Lists	Reserved	Mode ID	*Reserved*	*Reserved*
7	Field Attributes Enable	Index List	Event ID	Air Temperature	*Reserved*
6	*Reserved for CIF expansion*	Discrete I/O (32 bit)	Function Priority ID	Sea/Ground Temperature	*Reserved*
5	*Reserved for CIF expansion*	Discrete I/O (64 bit)	Communication Priority ID	Humidity	*Reserved*
4	*Reserved for CIF expansion*	Health Status	RF Footprint	Barometric Pressure	*Reserved*
3	CIF 3 Enable	V49 Spec Compliance	RF Footprint Range	Sea and Swell State	*Reserved*
2	CIF 2 Enable	Version and Build Code	Reserved	Tropospheric State	*Reserved*
1	CIF 1 Enable	Buffer Size	Reserved	Network ID	*Reserved*
0	*Reserved*	*Reserved*	*Reserved*	*Reserved*	*Reserved*

The VITA 49 standard also includes a standard format for context and command packets known as “extended context packets” and “extended command packets”. These packet types allow a user to define and transmit additional information as needed when the standard command and context packets do not contain the information fields desired by the user.

5.5 VRT Conclusions

The VITA 49 based fronthaul interface adds packetization to the fronthaul interface of a system. This enables several features including variable fronthaul bitrates and multiplexing of multiple DU systems to a single interface. The VITA 49 standard also provides standardized context and command interfaces as built into the data interface, which is again enabled by the packetization structure of the standard.

VRT is also capable of acting as a fronthaul interface standard, enabling interoperability of equipment from different vendors. This ability for equipment interoperation also allows for equipment to be changed independently, reducing the cost of maintenance, and allowing for greater flexibility in testing and operation.

VRT is not a true front-end architecture standard, however it does require certain things of the front-end architecture, such as the ability to communicate to the VRT encoder the information found in the context information fields in use, and the ability to operate on commands from the front-end VRT decoder. Instead of being a front-end architecture, VRT instead sets standards for the format of all signals going into and out of a VRT unit, allowing the specific hardware for implementing the standard to remain arbitrary, so long as it is capable of meeting the standard’s requirements.

6. VITA 49 FRONTHAUL FOR O-RAN SYSTEMS

As discussed, in a VRT O-RAN system, the interface standard that VITA 49 will be investigated for use in is the fronthaul interface that connects the RAN CU to the RAN DU. This is the interface described under the split options as potentially controlled by the Fronthaul Gateway (FHGW). The common interface used for this purpose is CPRI, which unlike the packet-based system of VITA 49 functions using a constant-bitrate TDMA frame encapsulation. This fronthaul analysis is not based on any particular split option, but instead is based on the standpoint of a standard fronthaul interface that may be applied to any split option.

The constant bitrate nature of CPRI is useful in situations where a constant bitrate is required, such as split options 8 and 7.3, or the option used by the OpenRAN alliance: interface 7.2x. If a system is designed using split 7.2 or lower however, a CPRI fronthaul system loses the ability to benefit from a variable fronthaul bitrate requirement. VRT's packetized nature allows it to both accommodate constant-bitrate and variable-bitrate frontend applications.

6.1 Front-Haul System Requirements

For split option 7-1 applied to 5G, according to equations 4-1 through 4-4, the required fronthaul bitrate is 157.3Gb/s (UL/DL) for split option 8 and 9.8(DL)/15.2(UL) for split option 7.1, as the main advantage that split option 7-1 has over split option 8 is to remove the guard subcarriers, which are 40% in LTE. Clearly these fronthaul links have a very high bitrate requirement. The bitrate requirement can be reduced by using a split option such as 7-2, 7-3, 6 or lower. Changing to a lower split option however, as described above, removes the benefit of using a constant bitrate fronthaul interface. The VRT standard is capable of transporting bits for any of the split options, with the benefit of a variable bitrate for the frontend. VRT is also potentially capable of application to the high bitrate split options, as it is capable of very low overhead due to its flexibility in packet framing and size.

If a system is designed with the flexible split mentioned previously as the intended split option, then the system front-end has to meet the requirements for implementing such a flexible split. The VRT standard meets the virtualization requirements, and also meets the requirements of variable fronthaul bitrate and being capable of transporting different split option data packets with

the same transmission protocol. VITA 49.2 also has the capability built in to transport the command and context information necessary for implementing AI control to the flexible split, as VRT context and extended context packets are capable of carrying any required context data to the back end, and the VRT command and extended command packets can contain any necessary control information to the front end. The command and control packets also include the stream IDs, allowing all context and command information to be easily categorized and matched with the appropriate data stream.

7. THE REDHAWK PLATFORM

The developers of the REDHAWK software describe it as: “REDHAWK is a software-defined radio (SDR) framework designed to support the development, deployment, and management of real-time software radio applications.” REDHAWK contains a set of tools for developing software modules called “Components”, to be developed into “Waveform Applications”, and further supports deployment of waveform applications onto a single computer, or a network of computers [3]. REDHAWK furthermore includes an Integrated Development Environment (IDE) that allows for the systems to be developed, deployed, monitored, and managed on a personal computer for testing.

Applications in REDHAWK are known as “Waveforms”, waveforms consist of a set of components, their configurations, and the interconnections of the components, compiled as a Software Assemble Descriptor file. These applications function on an extension of the SCA v2.2.2 standard.

7.1 REDHAWK Waveform Applications

7.1.1 REDHAWK Components

Components in REDHAWK are modular blocks meant to be applicable to any signal processing applications for the purpose of performing any specific, reusable function. The full definition of a component includes all component interfaces, properties and functionality [3]. The properties and interfaces of a waveform are described by .xml files, and the functionality of a component is described by code contained within the component.

The three major parts of a REDHAWK Component are the Ports, Properties, and functionality. A port is either a “provides” (input) or a “uses” (output), as an input port provides functionality that can be used by an output port [3]. Properties are found via an ID or optionally a name. There are also “enumerations” for associating values with symbolic names, allowing the symbolic names to be used in place of a literal value within the component properties. The functionality part of a component is described by the associated code which can be written in Java, Python, or C++.

Within the REDHAWK framework, code is encapsulated into a REDHAWK Container which includes a set in I/O Ports and Control elements. This container and encapsulated code together constitute a REDHAWK Component. This is illustrated in figure 7-1.

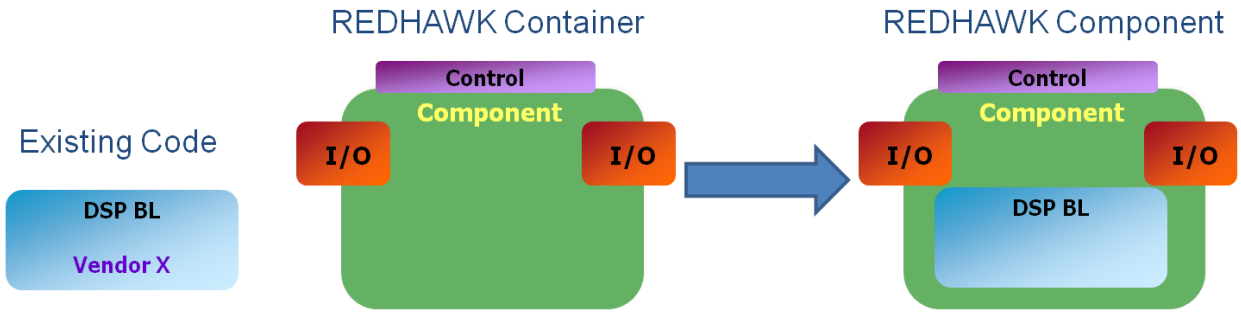

Fig. 7-1: The REDHAWK Component Structure. From [14].

7.1.2 Component Connections

Within the REDHAWK platform, a user can interconnect components within a waveform to create a complete signal processing application. Because REDHAWK is an extension of SCA v2.2.2, REDHAWK components and waveforms created using one system can be redeployed on any other SCA v2.2.2 compliant or compatible system.

The connection interface associated with a component is known as a Port. The connections between components all have a client-server connection pattern where the sender is the server and the recipient is the client. The client must have knowledge of what interfaces the server provides and how to use them. The server then must provide the set of functions that can be called by the client.

Connection data flow is managed through two sets of standard interfaces: Bulk Input/Output (BulkIO) and Burst Input/Output (BurstIO). Maximum efficiency for a bulk data transfer stream is achieved with BulkIO, and BurstIO is applied to smaller data transfers. Content processing, metadata association, and Signal Related Information (SRI) transfer, including a precision time stamp are supported by both of these interfaces. The list of supported SRI data is shown in table 7-1.

Table 7-1: The Supported SRI Data Structure Fields for REDHAWK [3].

NAME	TYPE	DESCRIPTION
hversion	long	Version of the Stream SRI header. Set to 1.
xstart	double	Specifies the start of the primary axis. (Refer to SRI Fields for Contiguous Data or SRI Fields for Framed Data)
xdelta	double	Specifies the interval along the primary axis. (Refer to SRI Fields for Contiguous Data or SRI Fields for Framed Data
xunits	short	Specifies the units associated with the xstart and xdelta values. Refer to the REDHAWK Interface Control Document (ICD) for definitions.
subsize	long	For contiguous data, 0. For framed data, specifies the number of data elements in each frame (i.e., the row length).
ystart	double	Specifies the start of the secondary axis. (Refer to SRI Fields for Framed Data)
ydelta	double	Specifies the interval along the secondary axis. Refer to (SRI Fields for Framed Data)
yunits	short	Specifies the units associated with the ystart and ydelta values. Refer to the REDHAWK ICD for definitions.
mode	short	0-Scalar, 1-Complex. Complex data is passed as interleaved I/Q values in the sequence. The type for the sequence remains the same for both real and complex data.
streamID	string	Stream ID. Unique streams can be delivered over the same port, where each stream is identified by a unique string (generated or passed along by the provides side). The generation of this Stream ID is Application-specific and not controlled by the REDHAWK Core Framework (CF).
blocking	boolean	Flag to determine whether the receiving port exhibits back pressure. If this is false and the provides-side queue is full, the data is dumped. If this is true and the provides-side queue is full, the pushPacket() call blocks.
keywords	sequence <CF::DataType>	User defined keywords. This is a sequence of structures that contain an ID of type string and a value of type CORBA Any. The content of the CORBA Any can be any type

This SRI data is also split into two modes of operation for BulkIO, with one mode for contiguous data and one for framed data, utilizing the SRI subsize field for frame size, and a size of 0 for contiguous data. These modes are described in table 7-2.

Table 7-2: The SRI Data Mode Descriptions for Contiguous data and for Framed data [3].

SRI FIELDS FOR CONTIGUOUS DATA	
NAME	**DESCRIPTION**
xstart	Specifies, in units identifed by xunits, the start time of the first sample, relative to the Unix epoch (January 1, 1970).
xdelta	Specifies the interval between consecutive samples.
xunits	Specifies the units associated with the xstart and xdelta values.
subsize	Set to 0.
ystart	Not used.
ydelta	Not used.
yunits	Not used.
SRI FIELDS FOR FRAMED DATA	
NAME	**DESCRIPTION**
xstart	Specifies an abscissa-style value (i.e., relative to xunits) associated with the first element in each frame. For example, in streams containing a series of one-dimensional FFT results, each frame represents a frequency interval with xstart specifying the frequency associated with the lower end of the interval. For real-valued samples, xstart is typically zero, while for complex-valued samples, xstart is typically bw/2.
xdelta	Specifies the interval between consecutive samples in a frame.
xunits	Specifies the units associated with the xstart and xdelta values.
subsize	Specifies the number of data elements in each frame (i.e., the row length).
ystart	Interpreted the same way as the xstart field in contiguous data (Refer to SRI Fields for Contiguous Data), except that it refers to the start time of the first frame.
ydelta	Specifies the interval between consecutive frames.
yunits	Specifies the units associated with the ystart and ydelta values

The use of SRI data is important in maintaining knowledge of signal metadata for a given data stream. The inclusion of SRI data allows information about received signals to be transported

from the system front-end to the back-end processing, allowing the back end to have access to all information about the received signal.

7.2 REDHAWK Devices

Within a waveform, a REDHAWK component can use a REDHAWK Device to interact with hardware. REDHAWK devices can be linked to a component through a usesdevice relationship, this allows a component to interact with a receiver or digitizer and use the hardware in a REDHAWK waveform. Devices are usable within the REDHAWK Sandbox, or within a domain. At startup, devices are deployed by a Device Manager and have a lifecycle linked to the lifecycle of the scripting environment, shutting down along with the scripting environment. Devices are launched upon device manager startup and released upon device manager shutdown.

Each waveform includes a Device Configuration Descriptor (.DCD) files. A DCD file is written in XML and contains device configuration information. When associated with a device manager instance, a DCD file is referred to as a Node.

When a waveform is launched by the domain, the domain reads the Software Assembly Descriptor (SAD) file, an XML file which contains information on application device hardware requirements, such as the usesdevice relationships. The domain then searches the deployed devices for one that satisfies the given dependencies and may set the device as unusable for other applications to ensure availability. The device is returned to the pool of available devices upon release of the related application.

REDHAWK contains a standardized API for interaction with RF devices, known as FrontEnd Interfaces (FEI). The application of the FEI standardizes the modeling and interaction of tuner devices within the REDHAWK Core Framework to remove the tie between the application and hardware, and to provide added flexibility. Generic tuner types supported by the FEI module include:

- Receiver (RX) tuner
- Receiver Digitizer (RX_DIGITIZER) tuner
- Channelizer (CHANNELIZER) tuner
- Digital Down Converter (DDC) tuner
- Receiver Digitizer Channelizer (RX_DIGITIZER_CHANNELIZER) tuner
- Transmitter (TX) tuner

- Receiver With Scanning Capability (RX_SCANNER_DIGITIZER) tuner

Additional FEI devices can be added via the FEI Wizard within the REDHAWK IDE.

The FEI contains a library of standard functions and data structures that it provides, shown in table 7-3. These items are included to add flexibility to the customization of FEI devices.

Table 7-3: The Standard Function and Data Structure Table for the FEI Module [3].

FUNCTION/DATA STRUCTURE	DESCRIPTION
setNumChannels	Used to size various FrontendTunerDevice class data structures.
frontend_tuner_status	This is the FrontEnd tuner status property, which is a vector of structs. The indices match the tuner_id or index of the tuner used by the FrontEnd Tuner device. The developer is responsible for maintaining all fields with the sole exception of the allocation_id_csv, which is managed internally by the FrontendTunerDevice class.
getControlAllocationId	Returns the control Allocation ID for the tuner specified, or an empty string if not allocated.
getTunerMapping	Returns the tuner ID or tuner index of the tuner associated with the Allocation ID, or -1 if the Allocation ID is not associated with any tuner.
create	Returns a StreamSRI object constructed using the frontend_tuner_status for a tuner, including the required Signal Related Information (SRI) keywords. Only required FrontEnd tuner status fields are used in constructing the StreamSRI, and any additional information that affects StreamSRI must be manually modified. In the case of Digital Down Converter (DDC) tuners, there is an optional parameter accepted by create for specifying the collector frequency since this information cannot be gathered from the frontend_tuner_status struct.
printSRI	Used for debug purposes to print the values of a StreamSRI object to stdout.
addModifyKeyword	Used to add a keyword to a StreamSRI object, or modify an existing keyword.
uuidGenerator	Used to generate a new UUID string.
floatingPointCompare	Used to handle potential errors introduced by floating-point math. Default precision is to the tenths place, and there is an optional parameter that can be used to specify a different precision.
matchAllocationIdToStreamId	Only available when multi-out ports are specified. Multi-out capability of a Bulk Input/Output (BulkIO) port only pushes stream data with a particular Stream ID to connections that have a Connection ID that matches the Allocation ID. It is recommended that this function be called in deviceSetTuning.
validateRequest	Used to verify that a value falls within the specified range. This function is overloaded to accept a range as well, to verify that it falls within a second range.

Table 7-3 continued

FUNCTION/DATA STRUCTURE	DESCRIPTION
validateRequestVsSRI	Used to check that the input data stream can support the allocation request. The output mode (True if complex output) is used when determining the necessary sample rate required to satisfy the request. The entire frequency band of the request must be available for True to be returned, not just the center frequency. True is returned upon success, otherwise FRONTEND::BadParameterException is thrown. If the CHAN_RF and FRONTEND::BANDWIDTH keywords are not found in the SRI, FRONTEND::BadParameterException is thrown.
validateRequestVsRFInfo	Used to check that the analog capabilities can support the allocation request. The mode (True if complex) is used when determining the necessary sample rate required to satisfy the request. The entire frequency band of the request must be available for True to be returned, not just the center frequency. True is returned upon success, otherwise FRONTEND::BadParameterException is thrown.
validateRequestVsDevice	Used to check that the input data stream and the device can support an allocation request. The mode (True if complex output) is used when determining the necessary sample rate required to satisfy the request. The entire frequency band of the request must be available for True to be returned, not just the center frequency. True is returned upon success, otherwise FRONTEND::BadParameterException is thrown. This function is overloaded to accept RFInfoPkt for an analog input data stream, and StreamSRI for a digital input data stream. For StreamSRI, if the CHAN_RF and FRONTEND::BANDWIDTH keywords are not found in the SRI, FRONTEND::BadParameterException is thrown.

REDHAWK devices can also be used to interface with other hardware (FPGAs, data acquisition units, microprocessors, etc.). Devices are what connects the software designed in REDHAWK to the hardware that the software is designed to interact with.

7.3 The REDHAWK Runtime Environment and the Domain Manager

The REDHAWK Runtime Environment is designed for the support of the infrastructure necessary for the deployment and management of the interconnected components running as an application. This includes providing the mechanism for management of the life cycle of the components, including the creation, tear down, initialization, and interconnection of deployed components.

The personification of the runtime environment is the Domain Manager program and Device Manager program binaries. The purpose of the device manager program is to host an instance of a

DeviceManager object, as well as an instance of a file system. The purpose of the domain manager program is to host an instance of a DomainManager Class, and several supporting objects, including the ApplicationFactory, the Application, and a FileManager. To allow users to interact with objects arbitrarily, the API is made remotely available, because of this, the major classes making up these programs must be imported. These objects always reside exclusively within either the Domain Manager program or the Device Manager program, irrespective of their API.

Only one Domain Manager, but an arbitrary number of Device Managers exist within a single instance of a REDHAWK system. This is because within a network deployment of REDHAWK, an instance of the Device Manager exists on each host within the REDHAWK network area that acts as a proxy to describe the microprocessor system, while the Domain Manager acts as a central bookkeeper, and a universal point where system applications can be created or torn down.

7.3.1 The Domain Manager Program

The Domain Manager controls and configures the entire systems domain. The system domain includes the full set of hardware devices and available application within a single-unit or network-deployed instance of REDHAWK. The three main categories of Domain Manager responsibilities are: Registration, Core Framework (CF) administration, and Human Computer Interfacing, as described below:

- **Registration:** The creation and destruction of all Device Managers, devices, and applications are handled through the Domain Manager. When such a module is created, it is registered to the Domain Manager, and is unregistered when it is destroyed.
- **CF administration:** The Domain Manager CF administration exists to allow changes to be made to a domain from outside a running domain. The CF administration makes the API of the Device Managers, ApplicationFactories, applications, and the FileManager available to external software through registration to the domain.
- **Human Computer Interfacing:** The Domain Manager is responsible for providing functionality to allow for simple interaction between the user and the running system. The domain manager provides the functionality to retrieve stored

and current information about the domain, as well as allow for configuration and launching of maintenance functions.

The Domain Manager program also handles the following:

- receives an XML file describing a waveform that is to be deployed.
- scans all running devices on all Device Managers for a suitable place to deploy the components making up the waveform.
- uses the File Manager / File System to copy whatever files are necessary to run the components to the target Device Managers.
- remotely invokes the component processes.
- interconnects components over the network.
- tears down applications appropriately.

A given system domain contains a single Domain Manager to keep track of a File Manager, the set of Device Managers, and a set of Application Factories. The Domain Manager also maintains all information regarding waveform implementations within the system.

The Domain Manager configuration is contained within the Domain Manager Configuration Descriptor (DMD) file. This XML file contains the name, ID, and a description of the domain.

7.3.2 The File System and File Manager

The Redhawk File System interface gives an OS-independent interface for reading and writing individual files on a system. It does this by defining CORBA operations that abstract files from an OS's real file system.

The REDHAWK File Manager interacts with all distributed file systems of a REDHAWK implementation, allowing them to be interacted with as a single entity. This allows the Domain Manager and Device Managers to function as though under a single file system within the domain. The File Manager is responsible for proper delegation of tasks from the CF to the correct mounted file systems based off of the path names. As applications are installed and launched, the File Manager is also responsible for the appropriate copying of component files into the proper Device Manager's file system.

7.3.3 REDHAWK Applications

Within REDHAWK, waveforms are represented by software objects called applications. Applications are used as a means of organization for their constituent linked components to facilitate the execution of a useful computational task. They also serve as a convenient tool for testing and modifying components to accomplish these tasks by allowing for easy interchanging of components.

The application object also monitors all aspects of its execution through various data structures. It does this to provide control, configuration, and status of the applications instantiated within the domain. When an application is completed, all executable devices are stopped, all allocated memory is released, all system resource allocation is deallocated, all component object references are released, all connected ports are disconnected, all consumers/producers connected to CORBA event channels are removed, and all component naming contexts are unbound from the naming service.

7.4 Comparison of REDHAWK to GNURadio

The major SDR IDE that is widely used is known as GNURadio, a program which functions similarly to REDHAWK, and has a much larger following, and therefore more resources, tutorials, and instructions available by users. Other platforms do exist, but nearly all are not designed for SDR, lack a significant development environment, or have so little following that there is no support. GNURadio and REDHAWK are similar enough to have some rough equivalencies between them, listed in table 7-4 that helps to outline their similarities.

Table 7-4: Similar Parts Between GNURadio and REDHAWK. From [23].

GNURadio	REDHAWK	Description
Block	Component	Individual piece of an SDR algorithm
Block (RTL, UHD, etc.)	Device	Hardware available to the algorithm
Flow Graph	Waveform	An SDR algorithm
Port	Port	Conveys the signal/data stream between algorithm elements
Variable	Property	A tuning or configuration parameter

This table shows that much of the surface-level functionality between these two platforms is similar, using a similar categorization method, and both having an overarching program to contain an individual algorithm. The differences mainly lie in how each platform carries out processing the algorithm, and how they attach to hardware.

In GNURadio, Flow Graphs are designed to be executed on a single host system, and all receiver hardware is defined as an explicit block within the flow graph. To enable an algorithm to use another vendor's hardware, the flow graph blocks must be changed to that vendor's blocks. This system keeps the hardware and the algorithm co-existent, and therefore keeps algorithms hardware dependent [23].

In REDHAWK, Waveforms are designed to function in a distributed computing network. The REDHAWK manual refers to a waveform that utilizes this type of processing as a Network Deployed Waveform. This allows a single deployment of a REDHAWK waveform to distribute computing processes across multiple processors within its network. REDHAWK waveform processes are also hardware independent, due to the SCA compatibility it contains. This SCA compatibility allows waveforms made in REDHAWK to be deployed to any systems that meet the hardware capability requirements of the waveform, so long as the hardware is also SCA v2.2.2 compliant [23].

Between GNURadio and REDHAWK, GNURadio has around it a much larger following, and is therefore much more common in practice however REDHAWK's distributed computing framework also allows for much greater performance in computing [23]. Additionally, for the purpose of SDR O-RAN front end interface development however, REDHAWK's distributed computing and hardware independence are much more desirable than GNURadio's single host, hardware dependent deployment [23]. This stems from the goals of O-RAN to create interoperable standardized interfaces and software that are vendor-independent.

8. REDHAWK TESTBED FOR VITA 49 IN ORAN SYSTEMS

The purpose of this thesis is to investigate REDHAWK as a testbed for the development and testing of VRT in O-RAN type systems. The motivation for the research of VITA 49 as an O-RAN fronthaul was covered in section 6. This section is intended to provide sufficient motivation and information such as to propose REDHAWK as an appropriate testbed for the future research and testing of VITA 49 as a fronthaul standard in O-RAN type systems.

For this purpose, a testbed was created that implements O-RAN based on a split architecture that uses VITA 49 as the fronthaul interface. Appendix A serves as a tutorial-based description and introduction of REDHAWK systems, and describes the system built for testing. The system setup is described in section A.3. Section A.4 describes waveform deployment and execution, and Section A.5 contains results from testing, and describes some methods of obtaining tangible results.

8.1 Basic Compatibility and Requirements

If REDHAWK is to be used as a testbed for VITA 49 in any system, it must be compatible with the standard. The VRT standard is a data format standard that acts as a fronthaul framework, as described in sections 5.1 and 5.2, because this framework only requires that data to and from the fronthaul to match a specific format, VRT is capable of being fully implemented in software, provided the front-end processor systems can access all information required for filling out context packets. If the front-end is software defined, then the front-end processing systems will have access to the necessary information for the VITA 49 context packets.

The REDHAWK program is designed to function as an SDR IDE, thereby giving it access to the information described above, thereby meeting the data access requirement of VITA 49. In addition to this, REDHAWK's core assets includes VITA 49 encoders and decoders.

Another factor in compatibility is the ability to function on a packet-based input and output method, REDHAWK again meets this requirement through the use of the BulkIO and BurstIO ports.

In order to function as a viable testbed, clearly REDHAWK must be capable of testing more than a single system. As described in section 7.1.1, the REDHAWK IDE is capable of

implementing each part of a system as a component, in as “fine-grain” a method a user may desire and allows those components to be created and modified in whatever method the user desires. The only requirement to support of any specific function is that it be programmed in any one of the Python, C++, or Java programming languages.

It is also described to the purpose of supporting a variety of systems that REDHAWK is built with an extension of JTRS SCA v2.2.2. This means that any waveform built in REDHAWK can be deployed on any other SCA 2.2.2 compatible system that has the required hardware devices to execute the waveform. This is a significant advantage over similar software, such as GNU Radio Companion (GRC), which promises similar functionality, but lacks SCA compatibility.

8.2 The O-RAN RAN Intelligent Controller

As Described in section – O-RAN is intended to employ the use of a “RAN Intelligent Controller” (RIC) which is split into the Near-RT RIC and the Non-RT RIC in order to split the CP and UP more readily. In order to function properly, the RIC is to be implemented in software, and it is desired that the Non-RT RIC uses AI/Machine Learning algorithms to facilitate the creation of models to be used by the Near-RT RIC. The goal of the Near-RT RIC is to utilize these models, generated by the Non-RT RIC, to facilitate the implementation of RB management interference detection and mitigation, per-UE load balancing QoS management, connectivity management, and handover control algorithms.

In order to implement these functions, it is desirable that the entire system control be digital, as this allows control to be easily implemented based on the models trained be the RIC Non-RT. It is also necessary that the back end is based on digital processing to allow new models to be instantiated as needed. The REDHAWK environment fulfills these requirements, as it is an SDR environment. An additional benefit to this architecture is that the AI/ML algorithms can be directly programmed into control components in REDHAWK, requiring only that these algorithms be implementable in Python, C++, or Java.

One downside to the software-based nature of the REDHAWK SDR environment is that it has been designed to function on a GPP based architecture, on top of an OS on a device. This reduces the potential for processing large amounts of data quickly, although GPPs have become increasingly capable.

8.3 O-RAN Split Architecture

For REDHAWK to be used as an O-RAN testbed in general, it is desirable that it is capable of implementing each of the split options. It is clear that REDHAWK is capable of this, as components are implemented in software and can easily be implemented or removed on the CU and DU. Due to the virtual nature of REDHAWK.

8.3.1 The flexible split

To test VITA 49 Fronthaul in a flexible-split architecture scenario, then it is necessary that the both the CU and DU have access to all of the functions that they might be required to implement. For a software based system, this is simple, as each of the functions can be implemented or skipped depending on incoming control commands. In REDHAWK, this means the development of components that will implement variable sets of functions depending on commands from control components that make decisions based on any factors that can be coded in.

The use of VITA 49 further facilitates this via the capability to facilitate the communication of context information about all of the incoming and outgoing signals from any function or device to any other in the system using context and control packets.

Furthermore, both REDHAWK and VITA 49 assist in the facilitation of an AI/ML controlled flexible split via the context transmission capabilities of VITA 49 described above, and the software function nature of REDHAWK components.

8.4 Home Implementation of a VRT Receiver Testbed in REDHAWK

For the exploration of this research, an implementation of VITA 49 as the fronthaul interface standard for a split architecture system running on REDHAWK was built. This system, shown in figure 8-1, and the same system described in Appendix A, is an FM receiver based on split option 7-2, built on a pair of home PCs running REDHAWK on the Linux CentOS 7 operating system. As shown in the figure, the system on the left is the DU and the system on the right is the CU.

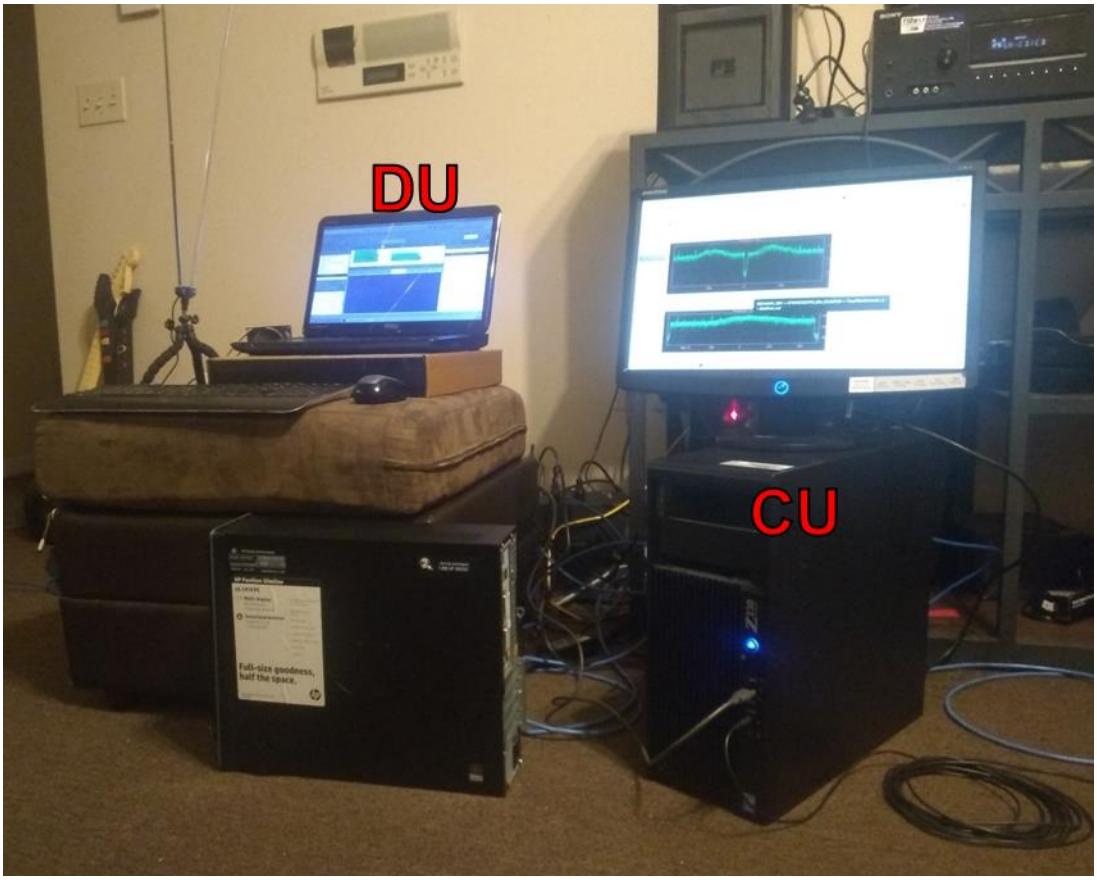

Fig. 8-1: The home implementation of the VRT FM receiver.

Shown in figure 8-2 is the REDHAWK waveform for the DU. The DU for this receiver, as in the 7-2 split receives a portion of the spectrum, filters it out, and sends it to the CU for processing. In this system, the interface between the DU and the CU is REDHAWK over an ethernet cable.

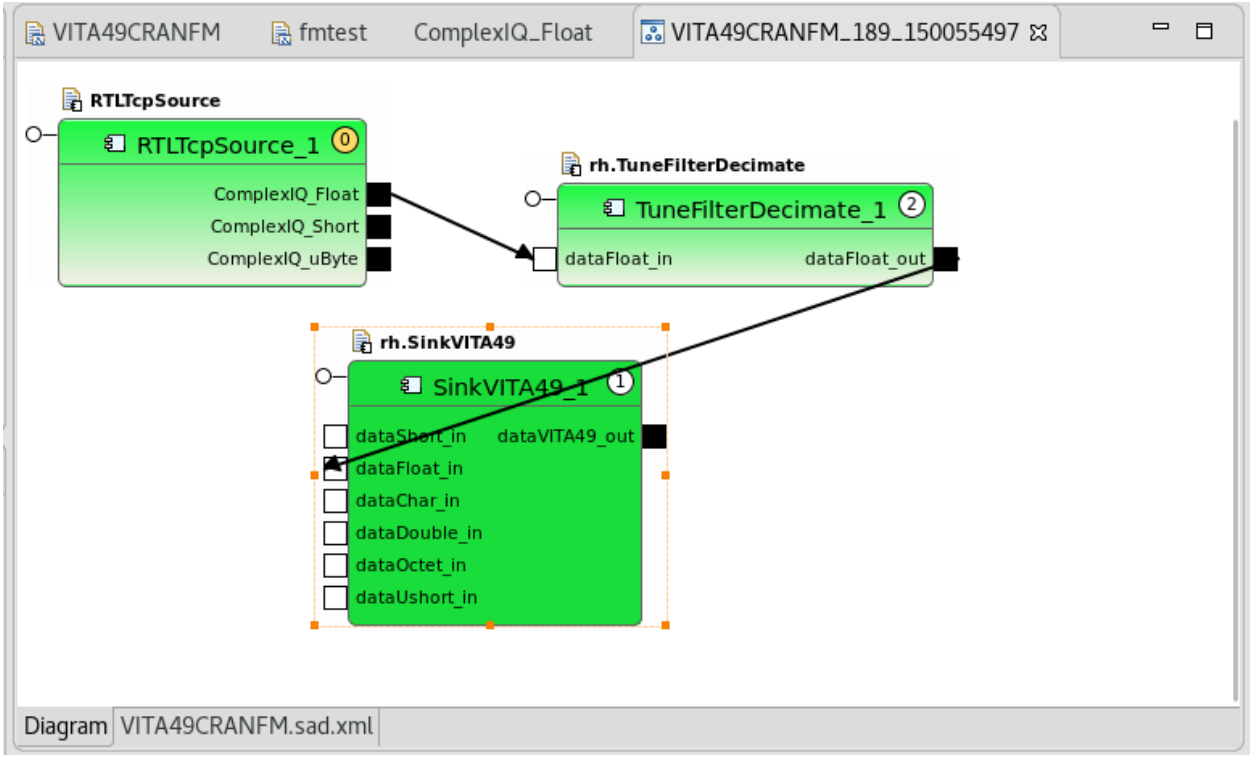

Fig. 8-2: The VITA 49 DU receiver waveform.

It is worth noting here that the waveform shown was developed on another system and was transferred to this system without need for modification. This seamless transfer was possible due to the hardware independent nature of REDHAWK waveforms.

The CU waveform for this system, shown in figure 8-3, shows a system that is designed to receive one or more FM signals, tune to one, demodulate it, and output the audio signal to a speaker. Figure 8-4 shows the signal received by the CU in this system on top, and the baseband signal just before demodulation on bottom.

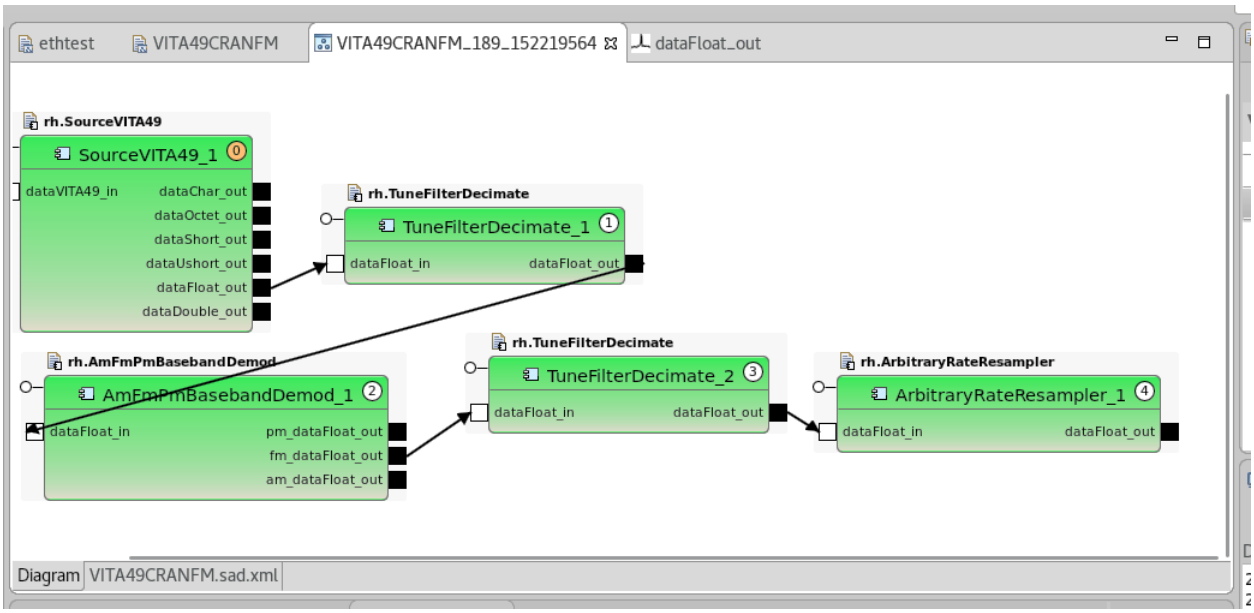

Fig. 8-3: The VITA 49 CU receiver waveform.

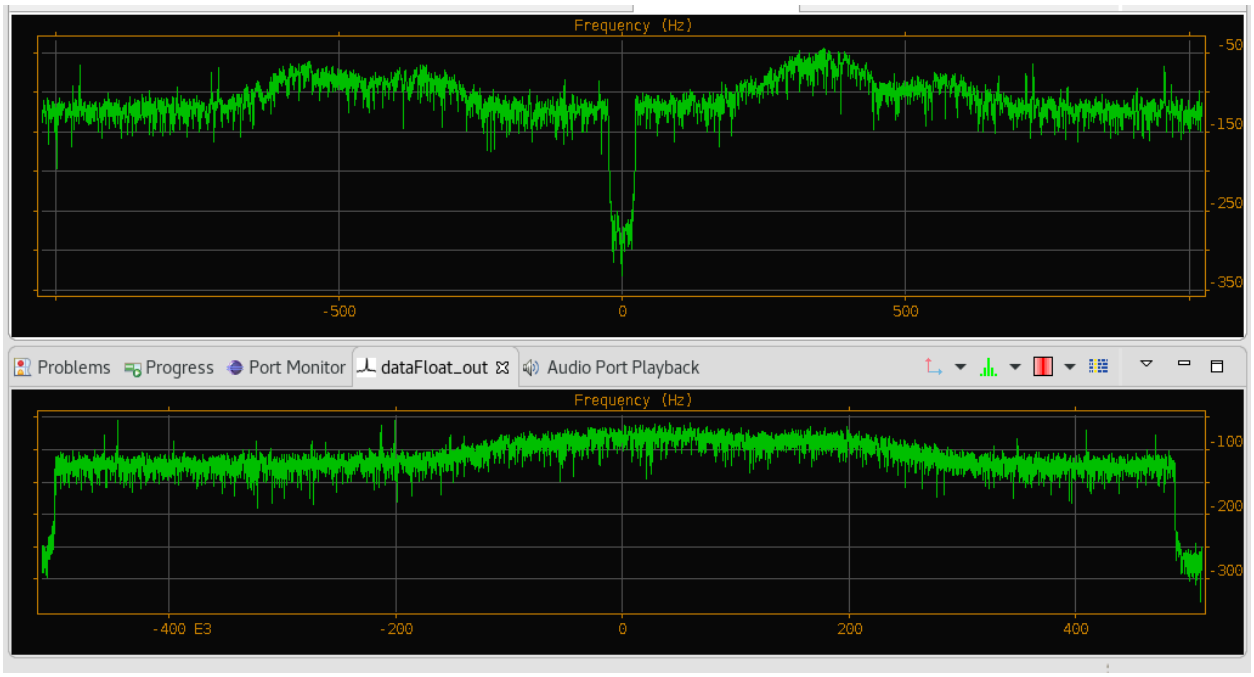

Fig. 8-4: The signal received by the CU (top) and the baseband FM signal (bottom).

On the CU end, several packets were captured using the Wireshark Network Analyzer tool, figure 8-5 shows several captured VITA 49 packets, and shows selected a VITA 49 context packet, this shows that the receiver metadata was properly received and maintained by the CU.

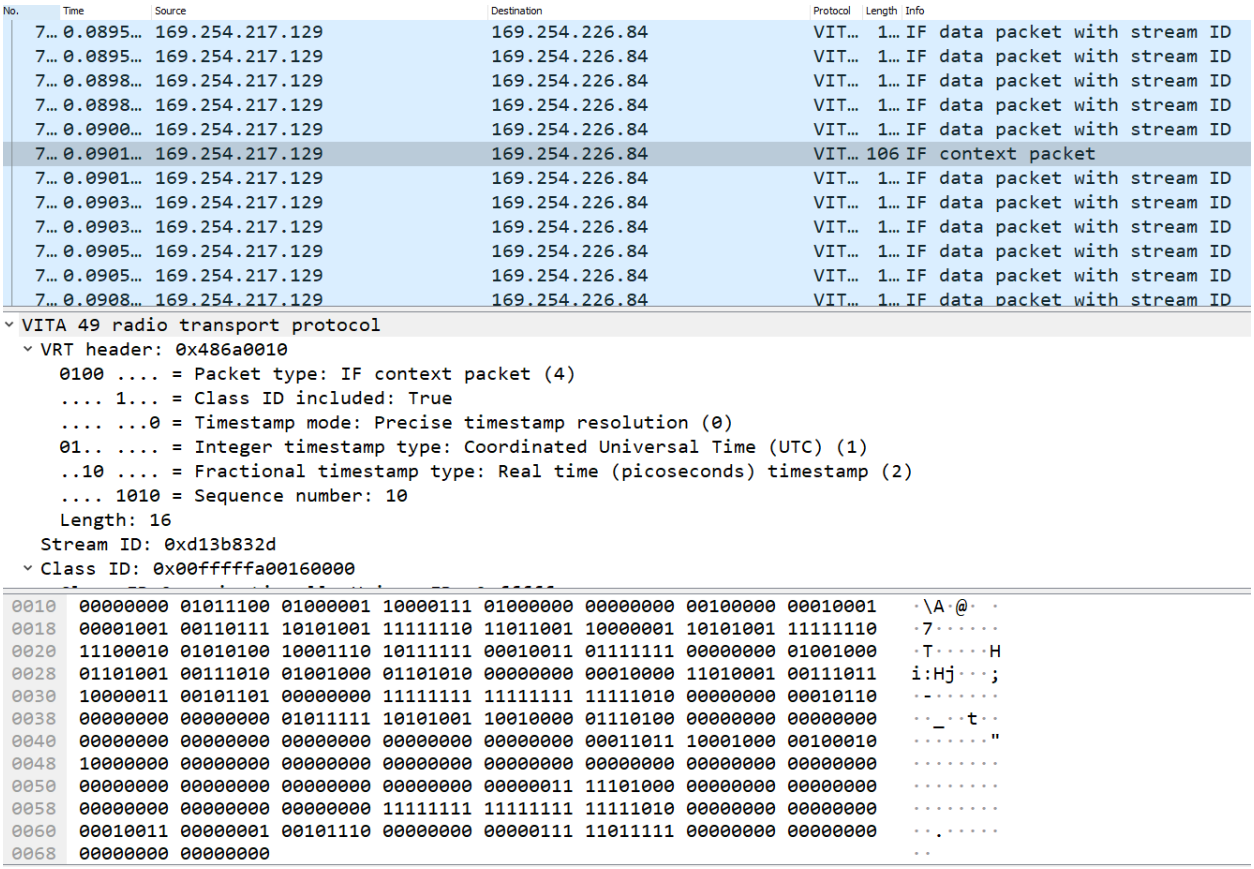

Fig. 8-5: VITA 49 packets in the Wireshark Network Analyzer.

9. CONCLUSIONS AND FUTURE WORK

9.1 Thesis Sections in Overview

Chapters 1, 2, and 3 introduce the issue of modern cellular development and cover the basic concepts of software defined radio and cloud radio access networks.

In Chapter 4, the concept of open RAN is described, starting with an example of O-RAN architecture developed by the O-RAN alliance. This discussion of O-RAN is used to bring forth the issue of where a network should be split. A split is defined as the interface that functionally connects the DU to the CU. Next is a discussion of the possible split options, as defined by the Third Generation Partnership Project (3GPP). In this section, particular attention is given to the required bitrates and the complexity of the DU and CU for each split option. This discussion leads to another discussion on the possibility of the "flexible split", and the need for a fronthaul gateway with standardized interfaces that has the flexibility to support selecting and processing for each included split option.

Chapter 5 introduces the VITA 49 standard, a communications interface standard to address the issues of metadata, digitization, and interoperability. In Chapter 5, VITA 49 is presented mainly from the standpoint of its data stream packets and packet structure and contents, as VITA 49 packets are the main enabler of its flexibility and interoperability.

Chapter 6 discusses VITA 49 from the standpoint of implementation in an O-RAN system to address the issues of applicability and disrability. The potential advantages of VITA 49 are covered in the areas of interoperability between vendors, data stream packetization, and flexibility through virtualization, and packet content options, which allow VITA 49 to function effectively as a fronthaul interface for different split options, and for a flexible split. This chapter establishes the testbed for the study of VITA 49 as an interface for O-RAN systems.

Chapter 7 discusses the REDHAWK platform, an eclipse-based platform that generates waveforms based on interconnected components. This chapter discusses the systems and functionality of the REDHAWK platform and describes the 'components' which are the central piece around which the platform is oriented. This chapter concludes by describing the applications that are built from the interconnected components. This chapter is finished by a comparison to the popular alternative, GNURadio. The findings from this research then provide the motivation for

the use of REDHAWK over other similar potential testbeds from the standpoint of compatibility with different split options, flexible split systems, and the RAN Intelligent Controller. This comparison implies that the REDHAWK platform is much more suited to the distributed network architecture and high processing environment needs that are required for O-RAN development.

Chapter 8 discusses the basic requirements of REDHAWK to be a feasible and useful testbed from the standpoint of REDHAWK as an SDR IDE and the compatibility with the VRT framework. The findings in this chapter imply that the REDHAWK platform is well suited to the implementation of VITA 49 through its digital environment, metadata support, and distributed network deployments. Further findings in this section indicate that the REDHAWK platform is very well-suited O-RAN and the achievement of its goals, including open development environments, interoperability and modularity in hardware and software, and implementation of AI/ML enabled RAN Intelligent Controllers, whereas other platforms, such as GNURadio are only well suited to some of these goals. This discussion leads to the conclusion that the REDHAWK platform is the best platform for the purpose of this O-RAN development.

9.2 Future Work

This work describes the need for a standard and interoperable interface, and the importance of the front-end interface in particular, and the desirability of the VRT standard for that interface. The purpose of this work was to establish that the REDHAWK platform is sufficient and desirable for use as a testbed for the development of use of VITA 49 in O-RAN systems as a front-end interface. To this end, much of the future work beyond this thesis should be in the use of the REDHAWK platform

9.2.1 Exploration of Systems and Standards Interactions

The example of REDHAWK as a fronthaul testbed presented in appendix A uses VITA 49 as a fronthaul for an FM radio system. In order to better simulate the functionality of REDHAWK as a testbed for VITA 49, a frontend system should be developed that utilizes these VITA 49 on REDHAWK to handle simulated cellular inputs and outputs. This experimentation should be done for practical verification of REDHAWK and VITA 49 in supporting any desired RATs.

9.2.2 Split Comparison

Beyond exploration of the interactions that REDHAWK and VITA 49 may have when implemented in a modern communication system, it may be useful to explore the exact effects of using such a system in different split configurations. The results should give particularly attention to the effects that the packetization overhead may cause, and the significance that that may have on any bitrate requirements. Because while VITA 49 is designed to be very low-overhead, when using different split options the metadata and control information requirements increase as the split option decreases. This increased need will cause an increase in control and context packet usage, and may cause significant overhead. This increase is even more likely to be significant if any split options require the implementation of the VITA 49 extension data packet types, for use of the included custom data fields.

9.2.3 Intelligent RAN Control

One feature enabled by REDHAWK that should be explored in greater detail is the implementation of RAN Intelligent Control, meaning the use of AI and/or machine learning in controlling radio resources for optimization. This should be done to assess the full extent to which these technologies can be supported by VRT and REDHAWK. Criterium should include efficiency and efficacy of different algorithms, models, and model generating, and maximum complexity of algorithms without causing system delay or significant performance impacts. It may also be explored here the impact that intelligent control can have on enabling the flexible split. This may be done either in the O-RAN alliance suggested Non-Real-time model generation, or in a real time scheme.

LIST OF REFERENCES

[1]C. I and S. Katti, O-RAN: Towards an Open and Smart RAN. O-RAN Alliance, 2018, pp. 6-19.

[2]"RedHawk SDR software", Dangerous Prototypes, 2020. .

[3]"REDHAWK Manuals and release notes", Redhawksdr.org, 2020. [Online]. Available: https://redhawksdr.org/. [Accessed: 18- Jun- 2020].

[4]A. de la Oliva, J. Hernandez, D. Larrabeiti and A. Azcorra, "An overview of the CPRI specification and its application to C-RAN-based LTE scenarios", IEEE Communications Magazine, vol. 54, no. 2, pp. 152-159, 2016. Available: 10.1109/mcom.2016.7402275.

[5]Rui Wang, Honglin Hu and Xiumei Yang, "Potentials and Challenges of C-RAN Supporting Multi-RATs Toward 5G Mobile Networks", IEEE Access, vol. 2, pp. 1187-1195, 2014. Available: 10.1109/access.2014.2360555.

[6]V. Jungnickel et al., "Software-Defined Open Architecture for Front- and Backhaul in 5G Mobile Networks", IEEE ACCESS, pp. 1-4, 2014. [Accessed 18 June 2020].

[7]N. Gomes, P. Chanclou, P. Turnbull, A. Magee and V. Jungnickel, "Fronthaul evolution: From CPRI to Ethernet", Optical Fiber Technology, vol. 26, pp. 50-58, 2015. Available: 10.1016/j.yofte.2015.07.009.

[8]R. Normoyle, "VITA 49 enhances capabilities and interoperability for transporting SDR data", Vita.com, 2008. [Online]. Available: https://www.vita.com/. [Accessed: 18- Jun- 2020].

[9]R. Hosking, "VITA 49 Radio Transport: The new software radio protocol", VITA TECHNOLOGIES, 2020. .

[10]R. Normoyle, "VITA 49 VITA Radio Transport (VRT)A Spectrum Language for Software Defined Radios", Johns Hopkins Applied Physics Laboratory, 2014.

[11]"The NSA’s Software Defined Radio application “RedHawk” is now open source", SWling post, 2019. .

[12]T. Cooklev, R. Normoyle and D. Clendenen, "The VITA 49 Analog RF-Digital Interface", IEEE Circuits and Systems Magazine, vol. 12, no. 4, pp. 21-32, 2012. Available: 10.1109/mcas.2012.2221520.

[13]Hsin-Hung Cho, Chin-Feng Lai, T. Shih and Han-Chieh Chao, "Integration of SDR and SDN for 5G", IEEE Access, vol. 2, pp. 1196-1204, 2014. Available: 10.1109/access.2014.2357435.

[14] VITA Radio Transport (VRT) Standard for Electromagnetic Spectrum: Signals and Applications, ANSI/VITA 49.2-2017, VITA, American National Standards Institute, Inc, Aug 3, 2017

[15] T. de Laurea, "Implementation of a suite of components for Software Defined Radio using an SCA-compliant framework", Undergraduate, UNIVERSITA DI PISA, 2014.

[16]E. Englund, "REDHAWK Eclipse as a Software Defined Radio Development Environment", Axios Inc., 2013.

[17] SOFTWARE COMMUNICATIONS ARCHITECTURE SPECIFICATION, SCA 2.2.2, JTRS Standards, Joint Program Executive Office (JPEO) Joint Tactical Radio System (JTRS), May 2006

[18] L. Larsen, A. Checko and H. Christiansen, "A Survey of the Functional Splits Proposed for 5G Mobile Crosshaul Networks", IEEE Communications Surveys & Tutorials, vol. 21, no. 1, pp. 146-172, 2019. Available: 10.1109/comst.2018.2868805.

[19]A. Umesh, T. Yajima, T. Uchino and S. Okuyama, "Overview of O-RAN Fronhaul Specifications", NTT DOCOMO Technical Journal, vol. 21, no. 1, pp. 46-59, 2019. Available: https://www.nttdocomo.co.jp/english/binary/pdf/corporate/technology/rd/technical_journal/bn/vol21_1/vol21_1_007en.pdf. [Accessed 9 July 2020].

[20] NTT DOCOMO, “R3-162102: CU-DU split: Refinement for Annex A.” 3GPP.

[21] “3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Study on new radio access technology: Radio access architecture and interfaces (Release 14),” 3GPP, Valbonne, France, 3GPP TR 38.801 V14.0.0 (2017-03) [online], Available: https://panel.castle.cloud/view_spec/38801-e00/pdf/ Oct 23 2020

[22] R. Antonioli, G. Parente, T. Maciel, F. Cavalcanti, C. Silva, and E. Rodrigues, “Dual Connectivity for LTE-NR Cellular Networks,” *Anais de XXXV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*, Sep. 2017.

[23] T. Goodwin, "GNURadio REDHAWK Integration", REDHAWK, 2017. https://geontech.com/gnuradio-redhawk-integration/.

A. TUTORIAL-BASED DESCRITPION OF REDHAWK

A.1 Introduction

The REDHAWK IDE is a free and open source framework designed for SDR systems. REDHAWK is designed to support the development, deployment, and management of real-time software radio applications. The REDHAWK IDE provides the required tools for creating, developing, and testing the software modules it uses, called **components** for connection and composition into **Waveform Applications**. These waveforms can then be deployed to a single computer domain or a network distributed domain of multiple computers.

This tutorial is based on deploying a REDHAWK waveform application to create a split architecture radio receiver using two computers, one as the DU and one as the CU, and an **RTL-2832U USB** receiver for the front-end receiver and analog-to-digital conversion. This deployment scenario will be used as it doubles as an example of a split architecture VITA 49 fronthaul system developed in REDHAWK, as it is proposed in the body of this thesis that REDHAWK is a suitable testing and development environment for such a system. The system built in this tutorial will not be based on a network deployment of REDHAWK and will instead serve as an example of the ability of REDHAWK systems to interoperate needing only a suitable interface for communication. This tutorial is also meant only to serve as a "bare bones" guide, meant to illustrate the description of REDHAWK presented in the body of this paper. Furthermore, this tutorial will not cover installation of either REDHAWK, or the RTL_SDR library utilized for implementing the RTLTCPSource Device that controls the RTL-2832U USB device.

A.2 The REDHAWK IDE

The REDHAWK IDE, an example of which is shown in figure A-1, provides a UI for interaction that has several main parts: the **RedHawk Explorer**, **CORBA Name Browser**, **Project Explorer**, **Sandbox** and several editors and viewers.

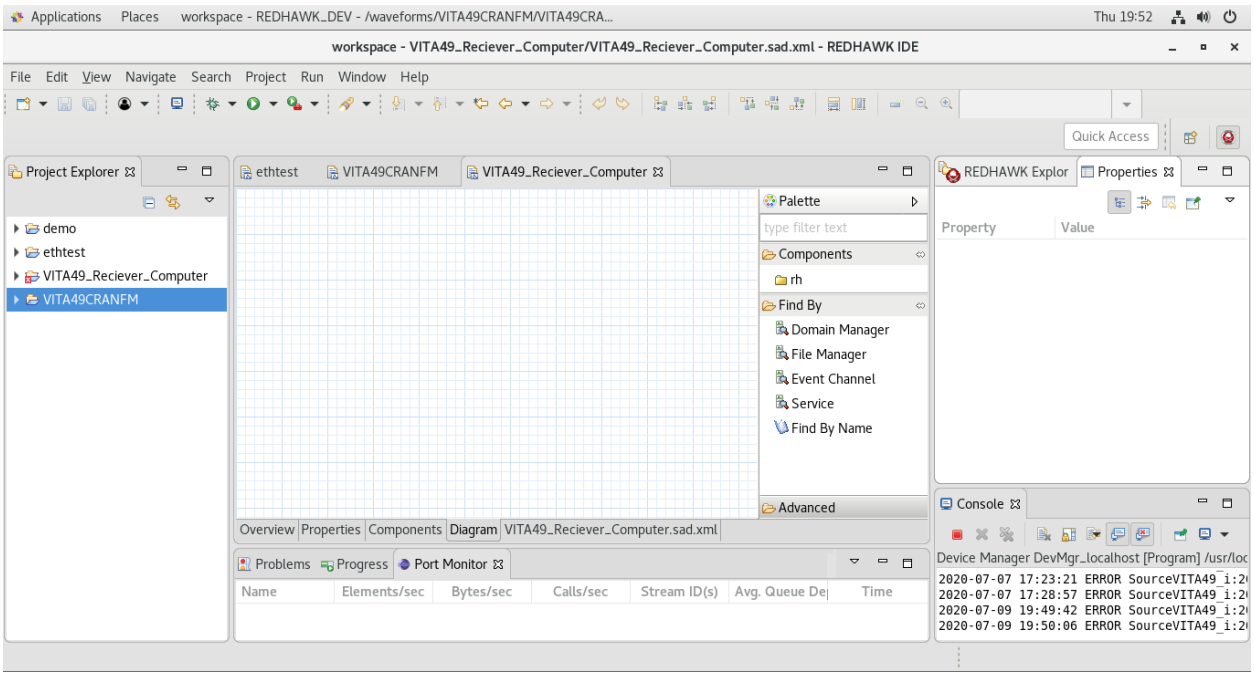

Figure A-1: The REDHAWK IDE UI.

REDHAWK Explorer:

The REDHAWK Explorer, shown in figure A-2, allows a user to view, configure instantiate resources in, and launch applications in the domain. The REDHAWK Explorer also acts as an interface for navigating the contents of a domain and Provides access to the **Sandbox**.

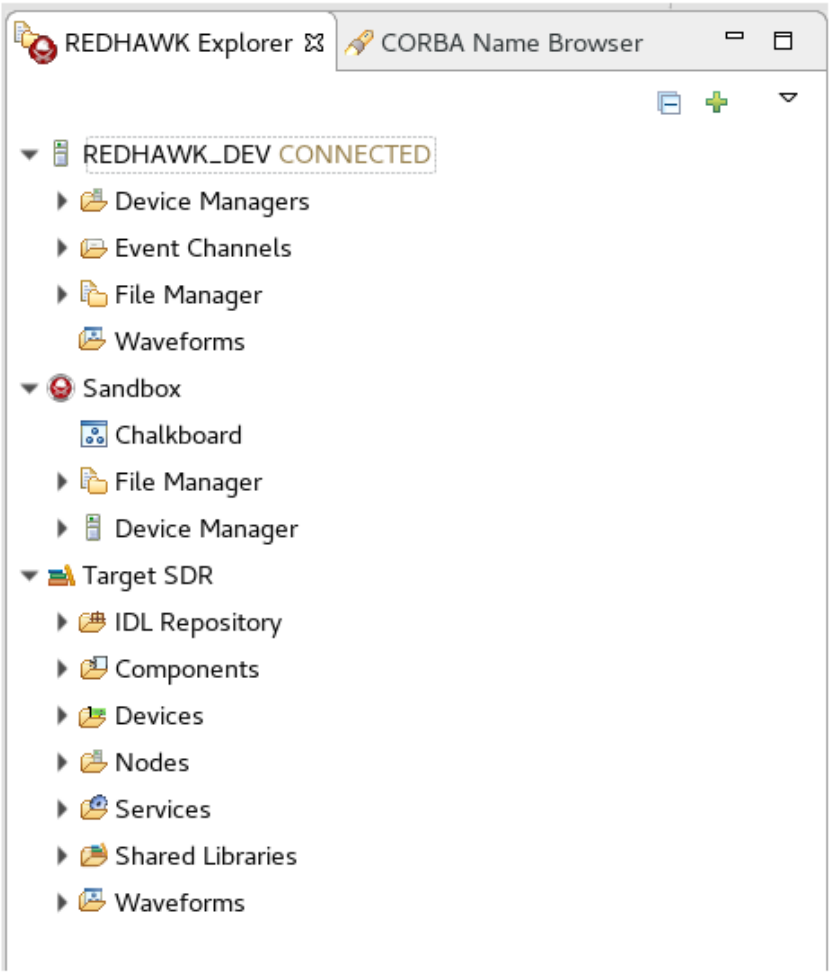

Figure A-2: The REDHAWK Explorer.

CORBA Name Browser:

The CORBA Name Browser, shown in figure A-3, allows the user to explore the contents of the Naming Service, and allows for basic manipulation. The CORBA Name Browser shows all currently bound naming contexts and objects.

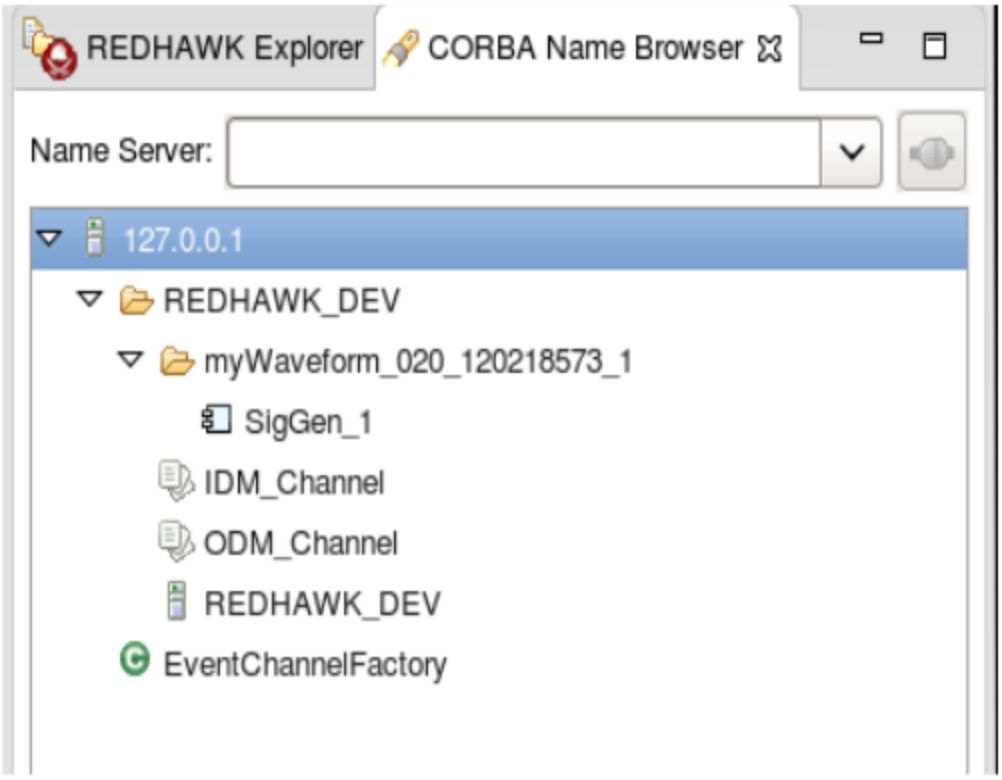

Figure A-3: The CORBA Name Browser.

Project Explorer:

The Project Explorer, shown in figure A-4, allows the user to view, open, create, and access REDHAWK projects.

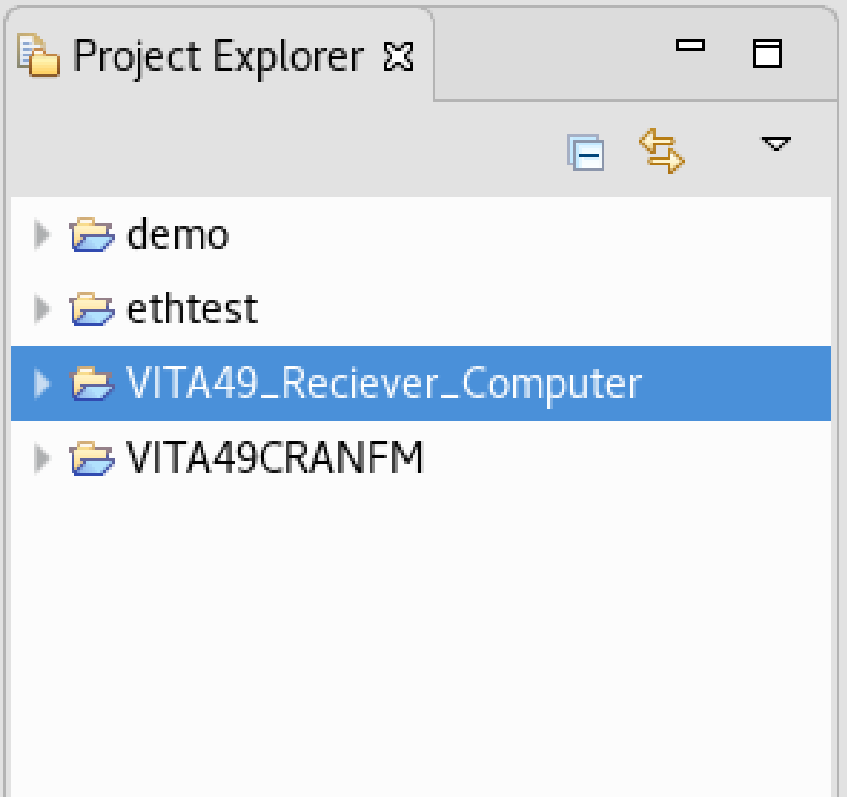

Figure A-4: The Project Explorer.

Sandbox:

In REDHAWK, components can be connected into a waveform and deployed in the domain, however, a testing environment is also included called the **Sandbox**. The Sandbox allows an environment to run a component or device without the need to use the Domain Manager, Device Manager, Name Service, or event service. REDHAWK has two sandbox environments: The Python Sandbox, and the IDE Sandbox. The Python Sandbox is capable of running on any python session on any computer that has an instance of REDHAWK installed. The IDE Sandbox gives a graphical environment for running components, and can host any instance of the Python Sandbox, allowing elements from each to interact. Components launched in the IDE Sandbox are interfaced to the user through the **Chalkboard**, which is similar to the **Diagram** tab, which is the center window in figure A-1. The Diagram tab is described further below in this tutorial.

Editors and Viewers

REDHAWK contains a number of editors and viewers, most of which will not be described here. Those objects described in this tutorial will be described further below. The list of common REDHAWK editors and viewers consists of:

- SoftPkg Editor
- Waveform Editor
- Node Editor
- NeXtMidas Plot Editor
- REDHAWK Explorer View
- REDHAWK Plot View
- Plot Settings Dialog
- Event Viewer View
- Data List and Statistics Views
- Port Monitor View
- SRI View
- Console View
- Properties View

A.3 System Setup Description

The physical system setup used in the creation of this tutorial and the one used in testing REDHAWK as a VITA 49 SDR testbed for the body of this thesis, is shown in figure A-5. The DU radio head is the laptop PC shown on the left, and the CU is the PC shown on the right.

Figure A-5: The Setup Used in Testing REDHAWK as a Testbed.

The two computer systems are connected via ethernet cable, with a simple network switch between to facilitate their connection.

A.3.1 CU System Setup

In this tutorial, the system that receives VITA 49 packets over its network connection and processes the signal is referred to as the CU. The CU setup begins with creating a new REDHAWK waveform project. The REDHAWK new waveform project window is shown in figure A-6. For this project, only the Project Name field is changed from default.

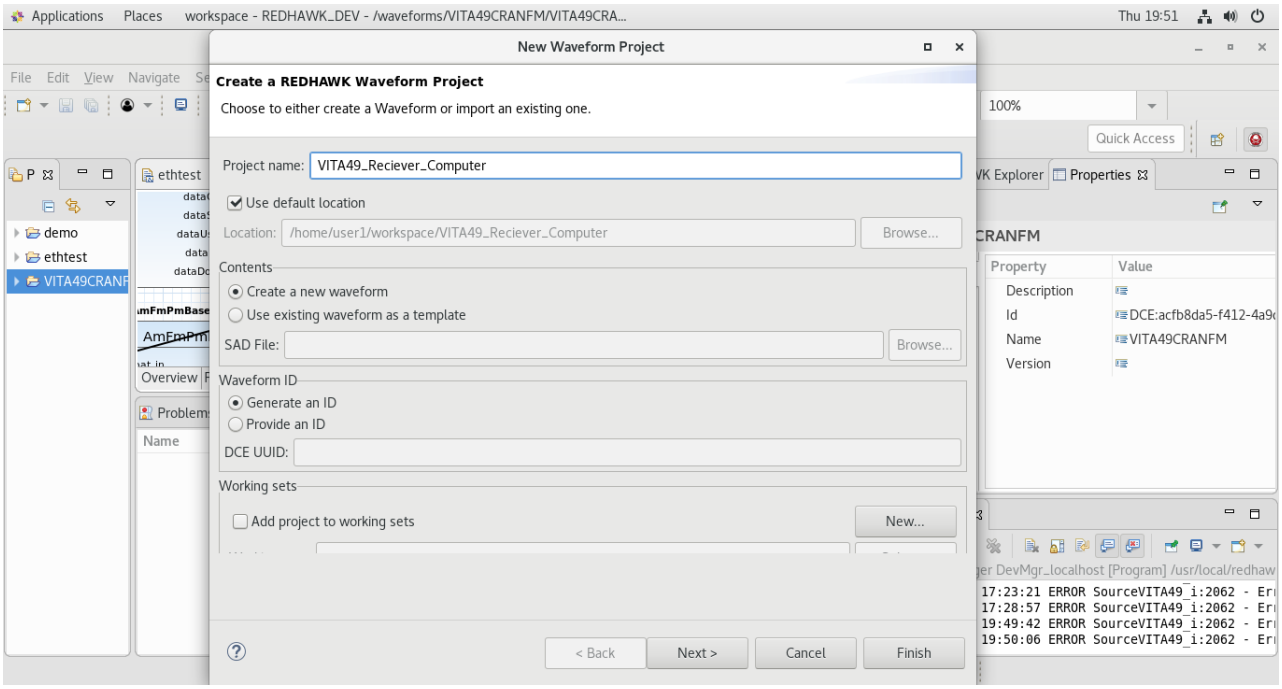

Figure A-6: The REDHAWK New Waveform Project Window.

When a waveform project has been created, the Diagram tab appears in the center of the IDE, and the Project folder appears in the Project Explorer on the left, as shown in figure A-7.

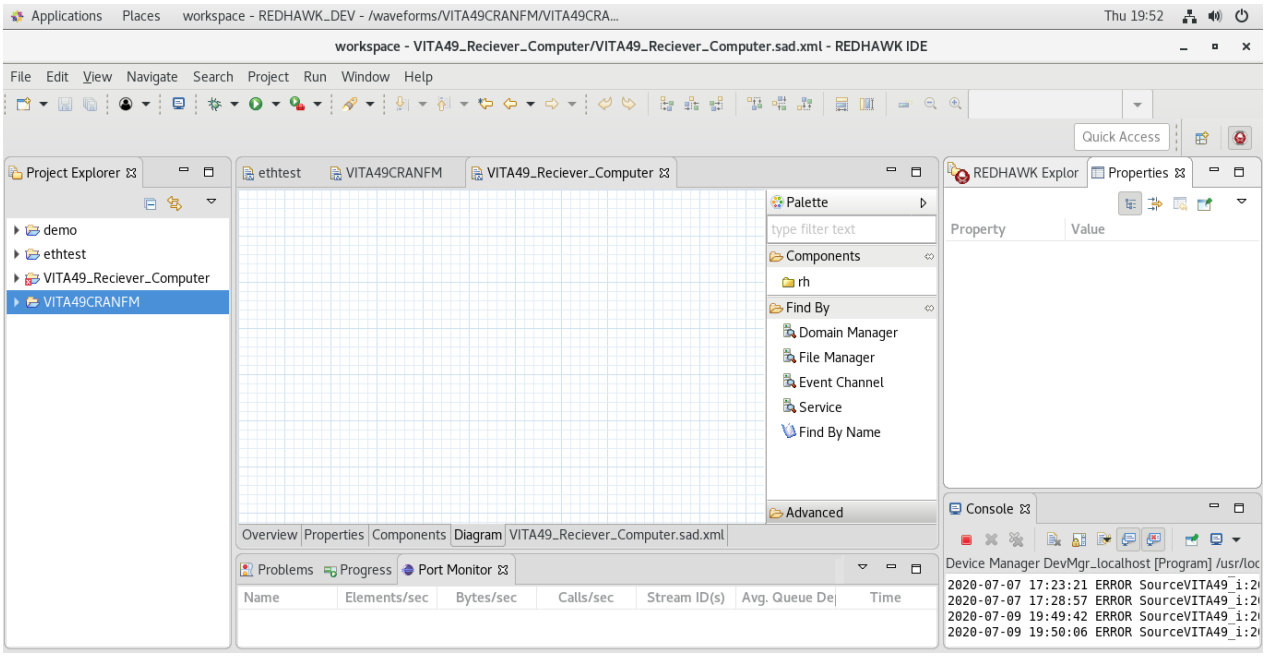

Figure A-7: The REDHAWK view upon creation of a new waveform project.

Next, components can be added to the project. Components in REDHAWK consist of component descriptor files which list all properties of a component and describe the input and output **ports** of the component, and a component function program which is written in Python, C++, or Java.

A SinkVITA49 component is added by clicking and dragging it from the "Components" tab on the right to the Diagram tab. This is depicted in figure A-8. The SinkVITA49 Component allows us to read incoming data from any port within REDHAWK or on the computer in use. In this tutorial, the port used is the ethernet adapter port, referred to as eno1.

The SinkVITA49 component is a special type of component called a **Device**. In REDHAWK, devices are components that have the capability of interacting with hardware on the system. In this case, the SinkVITA49 Device interacts with the computer ethernet socket to receive and transmit data.

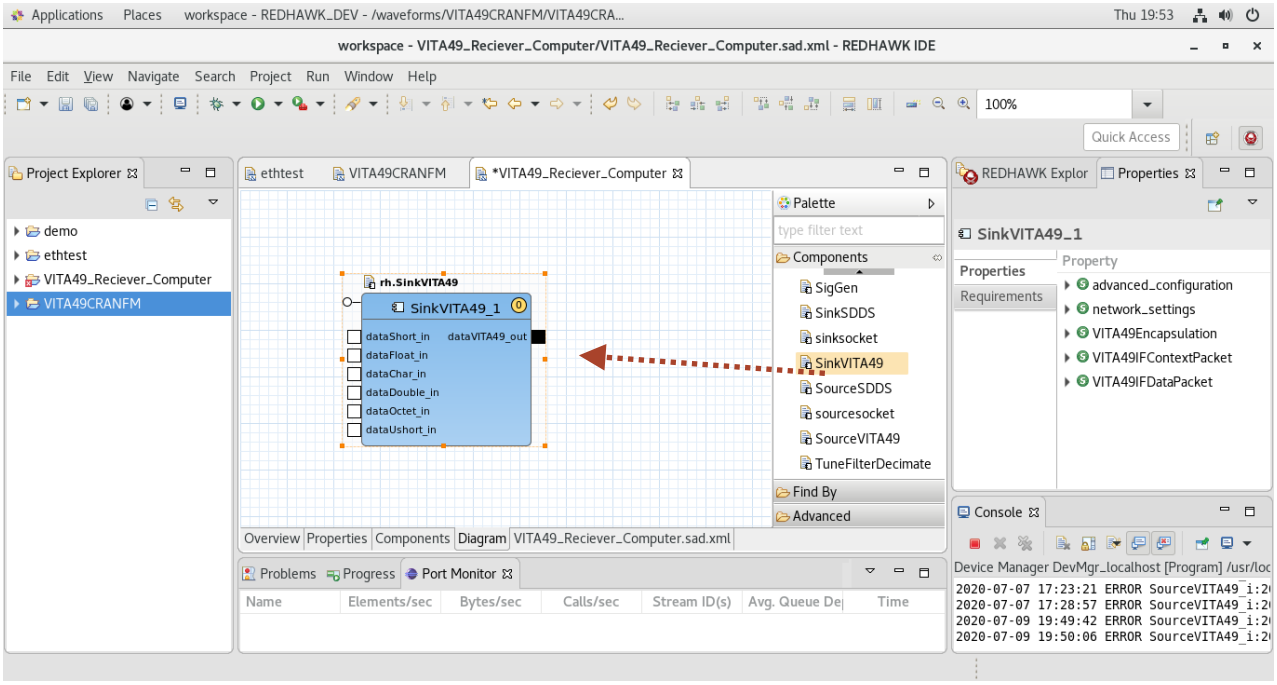

Figure A-8: The added SinkVITA49 component.

Next, The other components are similarly added, the added components are, in order of arrangement after the SinkVITA49 are: TuneFilterDecimate, AmFmPmBasebandDemod, TuneFilterDecimate, and ArbitraryRateResampler, as shown in figure A-9.

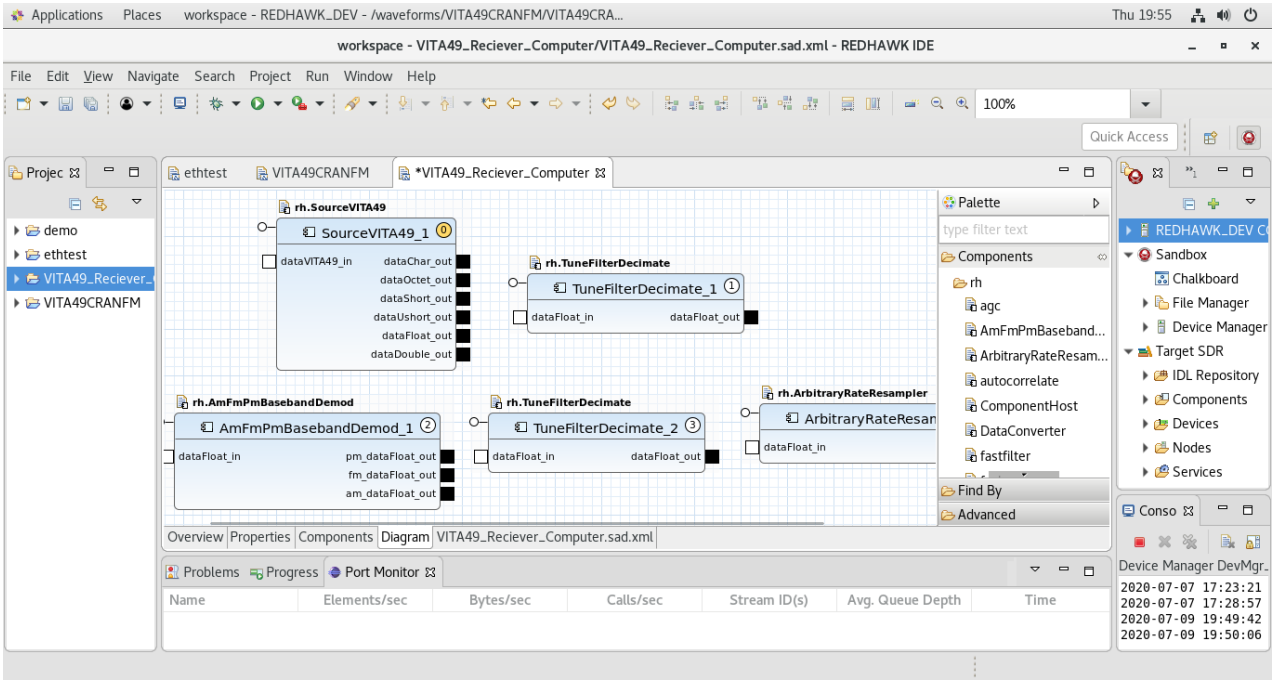

Figure A-9: The REDHAWK window with all components added.

Each REDHAWK component has a set of inputs and outputs called **ports**. Ports in REDHAWK define the connections between component functions by describing the flow of data from each component to the next. Port connections are stored in REDHAWK as an XML file describing each connection in a waveform. Port connection can be made in the REDHAWK IDE simply by clicking on a port, and dragging the cursor to the port it should be connection to. A connection between the SourceVITA49 Device and the first DuneFilterDecimate is shown in figure A-10.

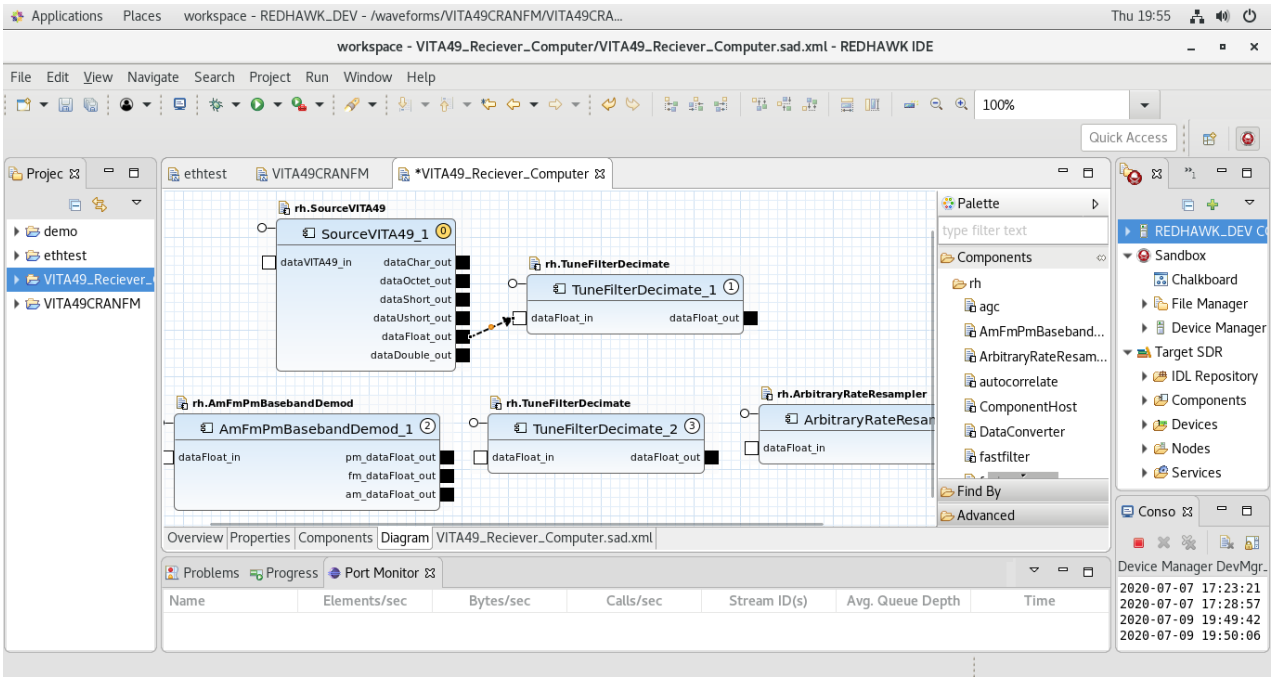

Figure A-10: The REDHAWK window with a connection between SourVITA49_1's dataFloat_out port, and TuneFilterDecimate_1's dataFloat_in port.

Next, more connections are made. The set of connections made in the Cu are described in table A-1, and depicted in figure A-11.

Table A-1: The connections made between the CU components.

Component	Output "Uses" port	- Connects To: -	Component	Input "Provides" port
SourceVITA49_1	dataFloat_out		TuneFilterDecimate_1	dataFloat_in
TuneFilterDecimate_1	dataFloat_out		AmFmPmBaseBandDemod_1	dataFloat_in
AmFmPmBaseBandDemod_1	fm_dataFloat_out		TuneFilterDecimate_2	dataFloat_in
TuneFilterDecimate_2	dataFloat_out		ArbitraryRateResampler	dataFloat_in

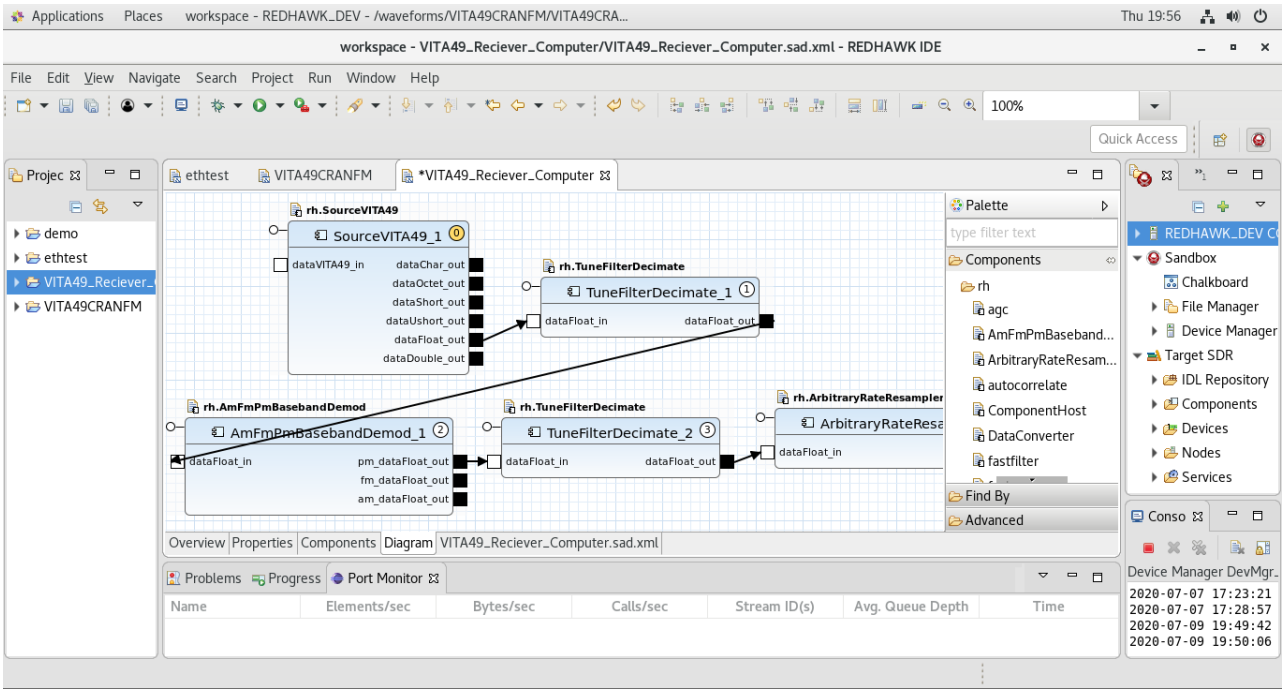

Figure A-11: The connected components in the CU.

The next step is to change the properties of the components such that the waveform will function as intended. The properties of a component can be viewed and edited via the properties window. This window can be brought up by selecting a component, right clicking, and selecting “properties”. An example of the properties window is shown in figure A-12.

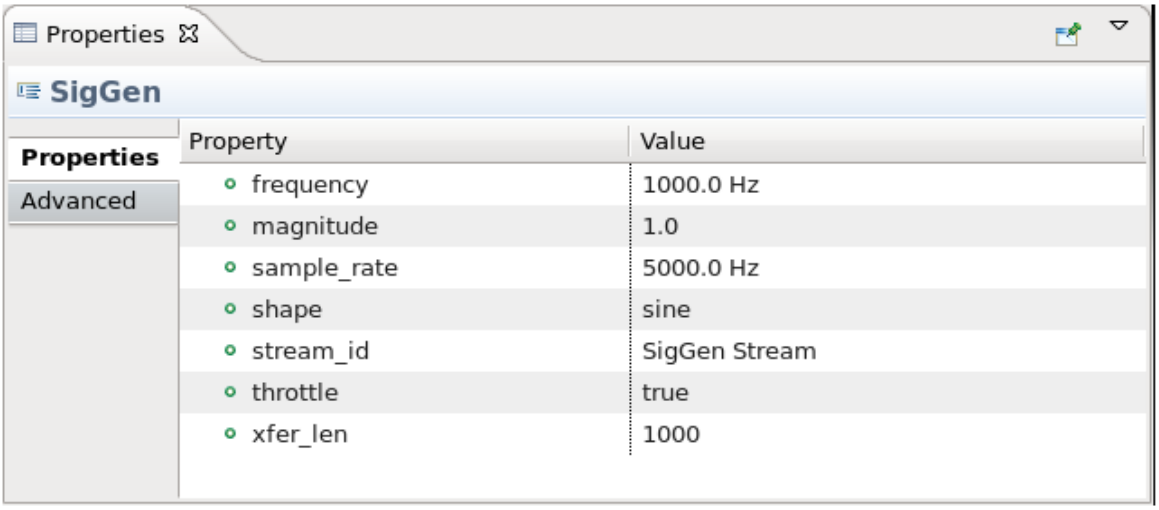

Figure A-12: The properties window.

Within this window, a property can be changed by selecting it and either entering a value or selecting a possible value from a dropdown list of valid property values. The properties set for the components in this waveform are found in table A-2. Any properties not mentioned are left at default values.

Table A-2: The properties set in the CU components.

Component	Property	Value	Notes
Source VITA 49	Buffer size:	16000 bytes	-
	enabled	TRUE	Under: attachment_override
	ip_address	192.168.1.43	-
	port	4991	-
	connection interface	eno1	-
tunefilterdecimate1	desiredoutputrate	1024000 Hz	-
	Filter BW	1000000 Hz	-
	TuneMode	IF	-
	Tuning IF	500000 Hz	-
	Tuning Norm	0.12207…	Automatically set upon starting waveform
AMFMPMbasebanddemod	-	-	All properties default
tunefilterdecimate2	Desiredoutputrate	128000 Hz	-
	FilterBW	48000 Hz	-
ArbitraryRateResampler	OutputRate	48000 Hz	-

With this, the waveform is saved and is ready to be exported to the domain. To export a project to a domain, the project folder can be selected and dragged from the Project Explorer to the Target SDR in the REDHAWK Explorer, as depicted in figure A-13.

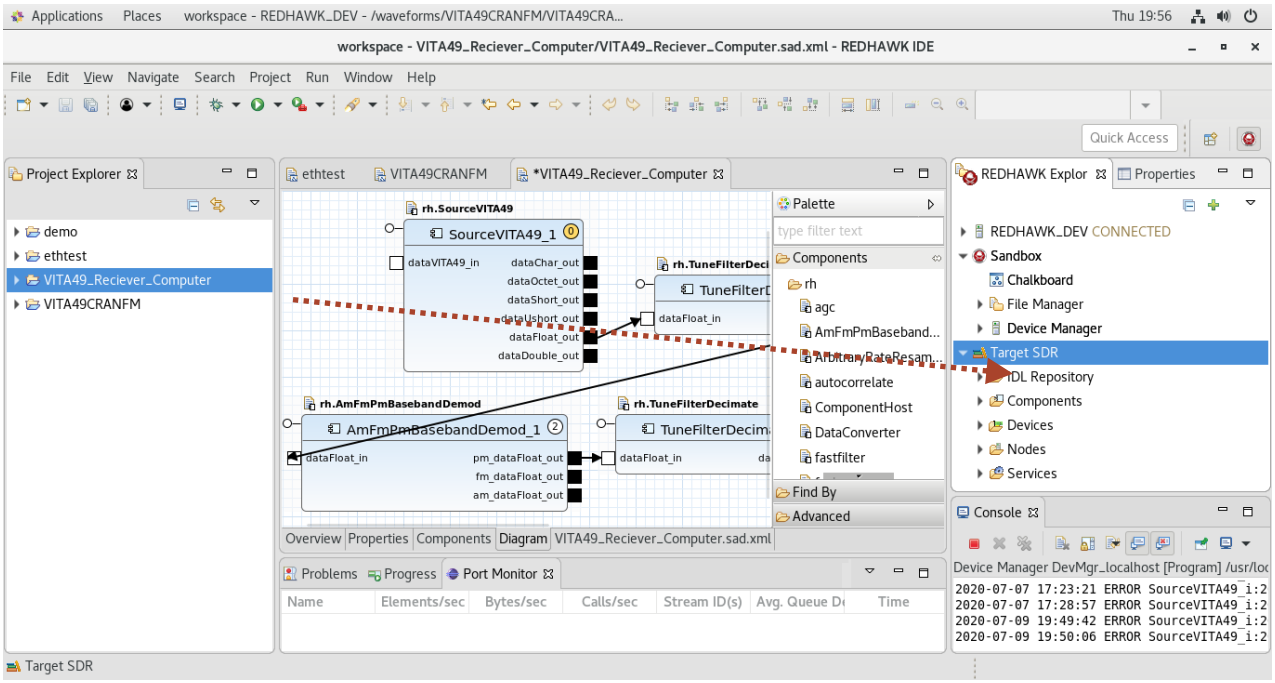

Figure A-13: Exporting the Project to the Target SDR domain.

With this, the CU is ready to launch the waveform and begin receiving and processing VITA 49 packets from the DU.

A.3.2 DU System Setup

The DU in this tutorial is the system that collects and digitizes the waveform using the RTL-2832U device, and sends them to the CU via the ethernet cable connection. Similar to the CU setup, a waveform project is made, and components are added, connected, and configured, and the waveform is then exported to the local Target SDR domain. The process is the same as that used for the DU. The components used in the DU are: RTLTCPSource, TunFilterDecimate, and SinkVITA49; their connections can be found in table A-3. The component properties can be found in table A-4. The final REDHAWK window for this project can be seen in figure A-14.

Table A-3: The connections used for the DU components.

Component	Output “Uses” port	- Connects To:	Component	Input “Provides” port
RTLTcpSource_1	ComplexIQ_Float		TuneFilterDecimate_1	dataFloat_in
TuneFilterDecimate_1	dataFloat_out	-	SinkVITA49_1	dataFloat_in

Table A-4: The properties set in the DU components.

Component	Property	Value	Notes
RTLTcpSource_1	frequency	98500000	FM Station 98.5 MHz
	frequency_correction	47 ppm	-
	gain	40dB	-
	Sample_rate	8192000 Hz	-
TuneFilterDecimate_1	desiredOutputRate	4096000 Hz	-
	FilterBW	4000000 Hz	-
	TuneMode	IF	-
	Tuning IF	1000000 Hz	-
SinkVITA49_1	Force_transmit	TRUE	
	enable	TRUE	Under network_settings
	Interface	p1p1	
	Ip_address	192.168.1.43	
	Enable_vrl_frames	-	-

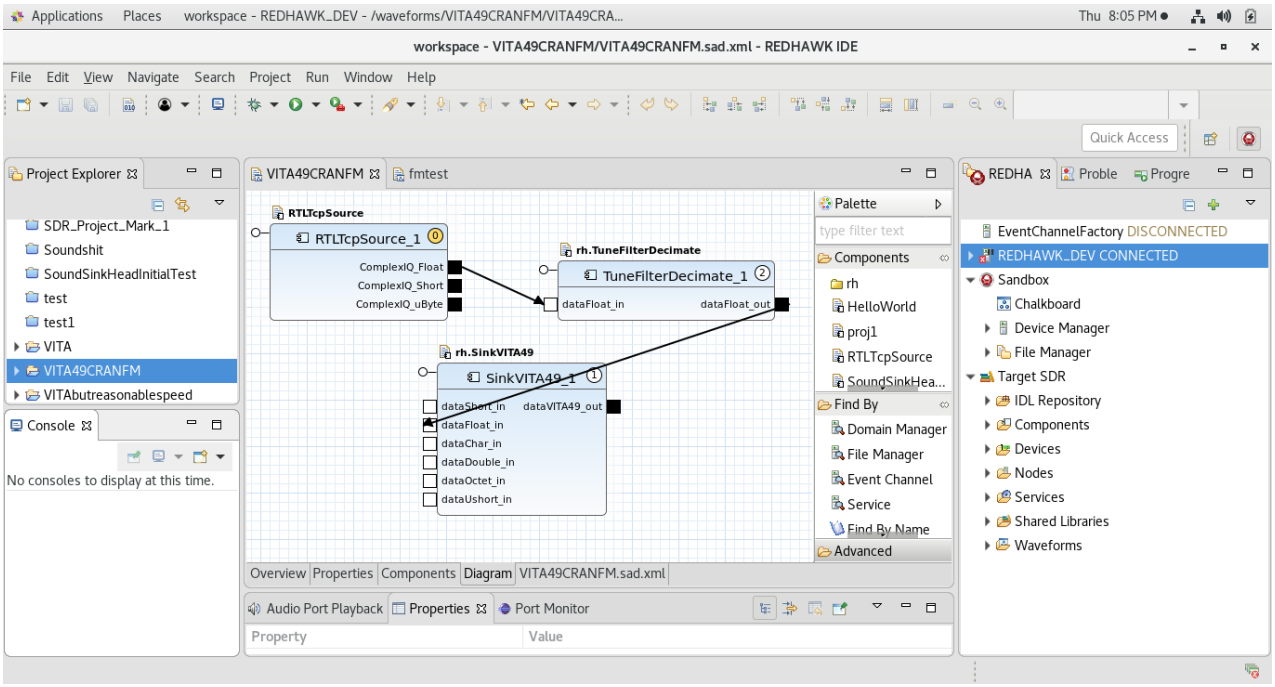

Figure A-14: The final REDHAWK DU window.

A.4 Starting the Waveforms

With the waveform applications constructed and configured on both the CU and the DU, the only step remaining before beginning the initialization process is to start the RTL 2832U device. To start the device, open the Linux command line in the DU PC, and enter the command “rtl_tcp”, this will initialize the device, as shown in figure A-15.

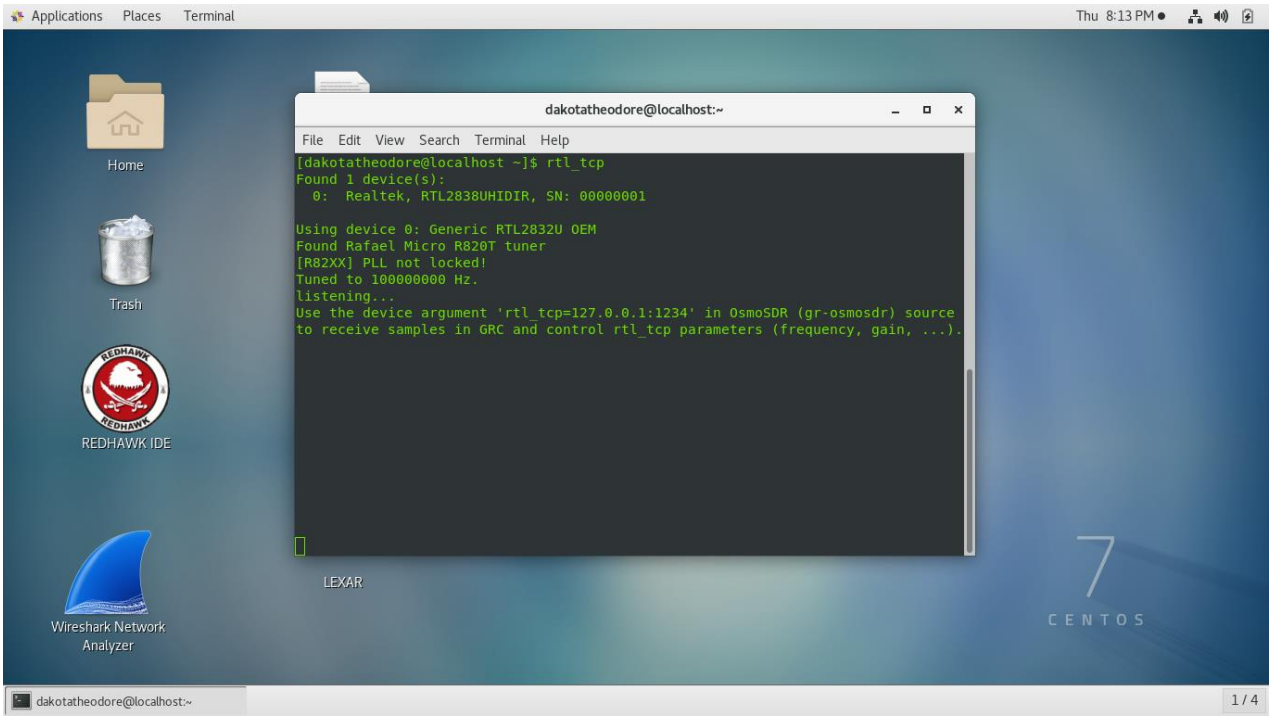

Figure A-15: The Linux command line with the RTL-2832U device initialized.

The first step in initialization of the waveforms is to launch the domain managers for each system. To do this, in each system, right-click on the Target SDR and select “Launch Domain Manager”, this will bring up the Domain Manager Launcher shown in figure A-16. In this window, select the local device manager, which should have a name like “DevMgr_localhost” and click “OK”.

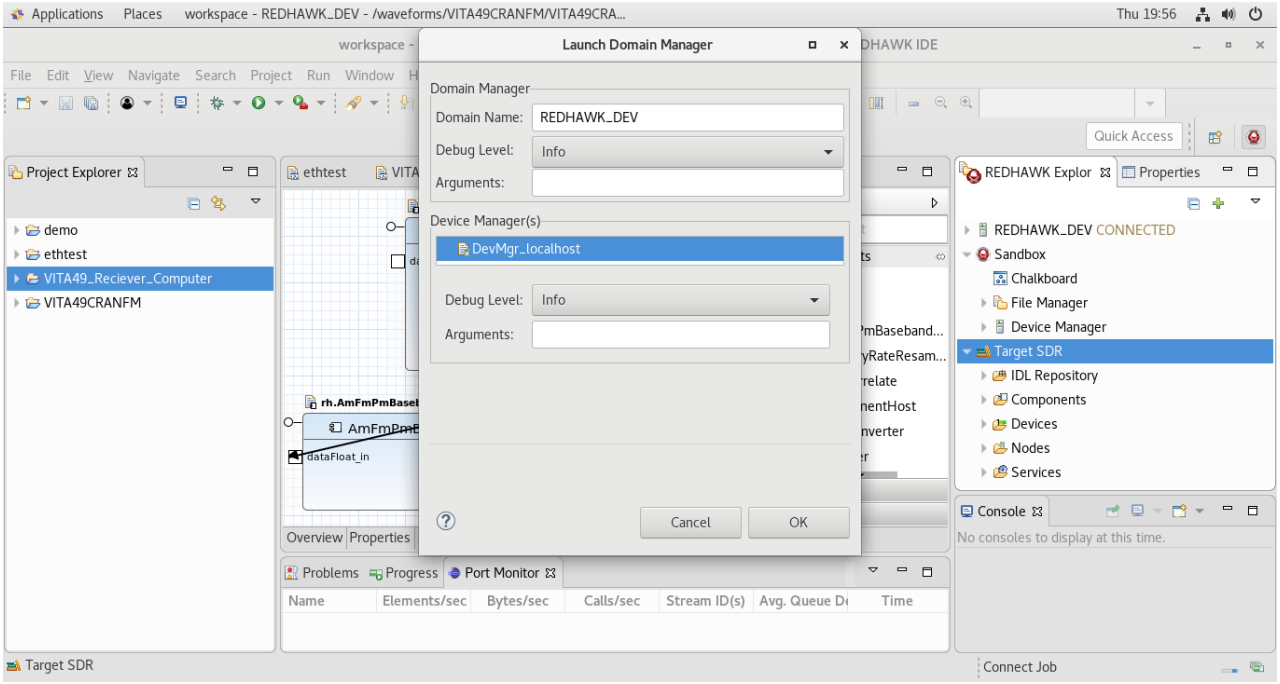

Figure A-16: The REDHAWK Domain Manager Launcher.

Now that the domain managers have been launched, the waveforms can be started from the REDHAWK_DEV domain. This is accomplished by right-clicking on the REDHAWK_DEV domain and selecting “Launch_Waveform”. This will bring up the Waveform Launch window, as shown in figure A-17. In this window, the desired waveform should be selected, before clicking the “OK” button.

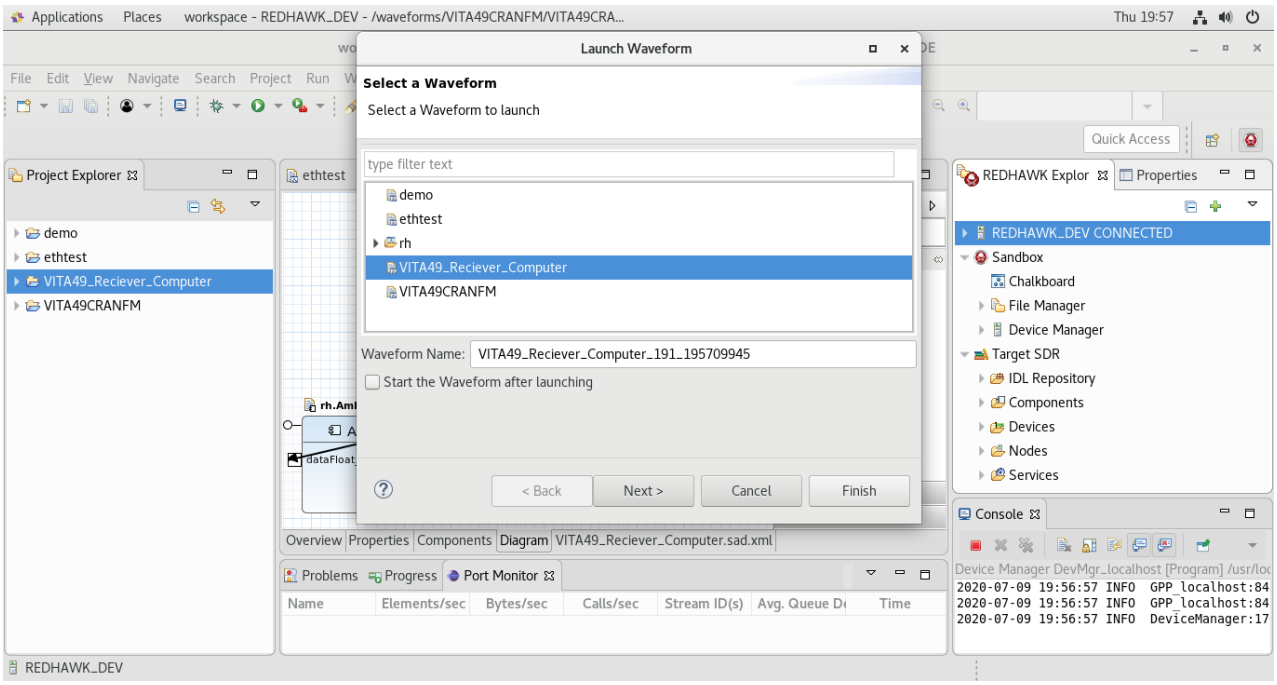

Figure A-17: The REDHAWK Waveform Launcher.

With the waveforms launched on both systems, the components should be started one-by-one. This can be done by selecting each component, right-clicking on it and selecting “Start component”. When all components have been started, the CU and DU REDHAWK windows should resemble those in figures A-18 and A-19 respectively. It is not unusual for the CU to drop some of the incoming VITA 49 packets once both the VITA49Sink and VITA49Source components have been started.

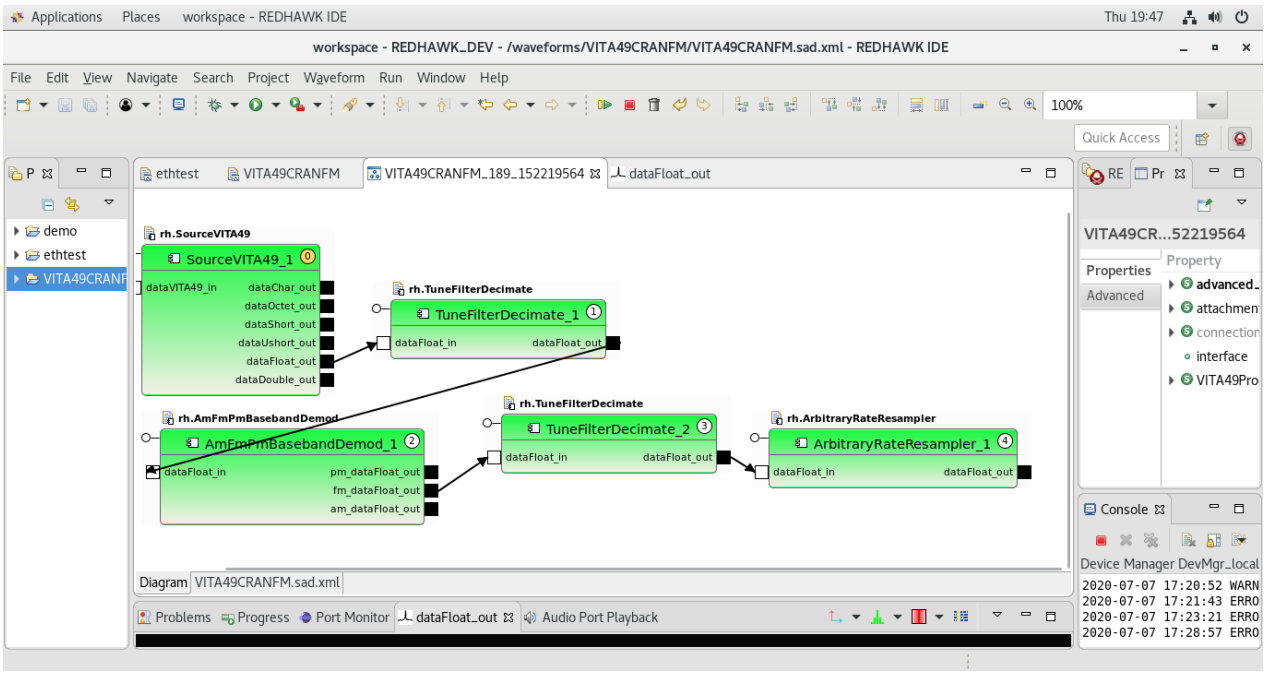

Figure A-18: The CU REDHAWK window after starting all components.

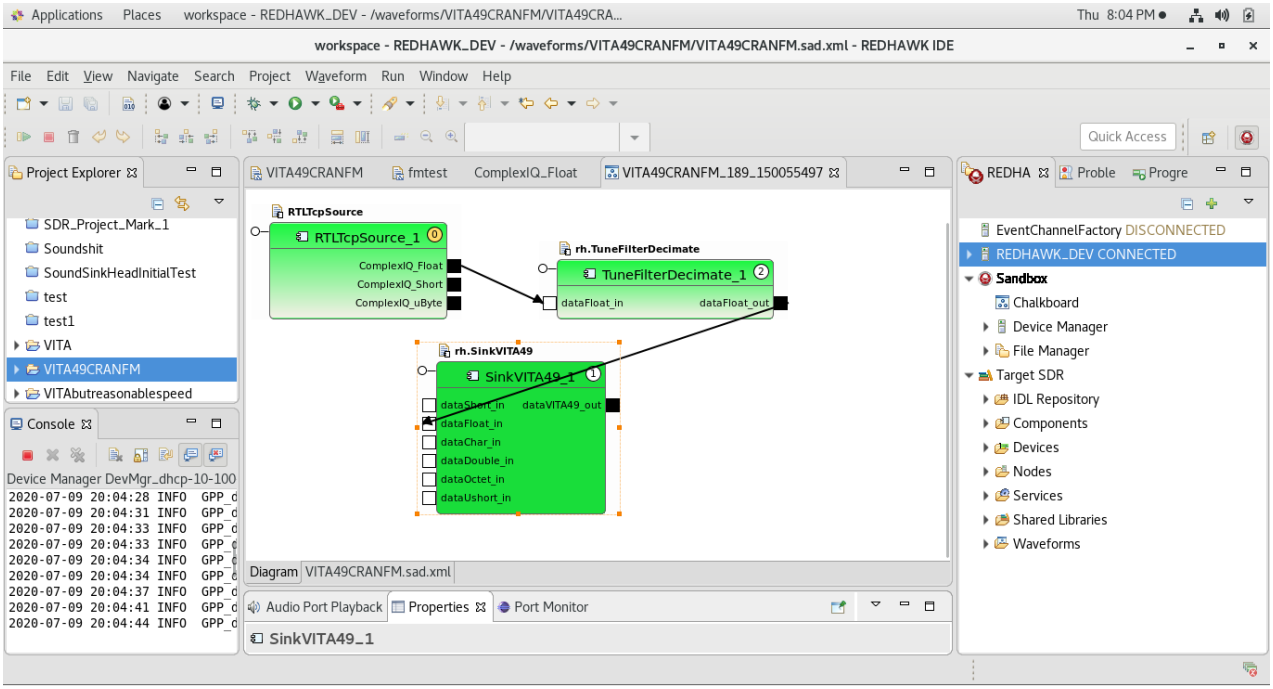

Figure A-19: The DU REDHAWK window after starting all components.

During runtime, the properties of any of the constituent components of the waveforms can be edited in the same manner that they were before deployment. Changes made during runtime however, will not be saved, and properties will revert to their pre-runtime value when the waveforms are released.

A.5 Obtaining Results

There are several functions that can be used to view the results of this implementation. Some built into REDHAWK include the data plot, FFT plot, and port Play. A data plot of any port can be made by selecting a port, right-clicking the port, and selecting “plot port data”, the FFT can be accessed instead by selecting “plot port FFT”. Figure A-20 shows the FFT plots of the outputs of the VITA49Source_1 and TuneFilterDecimate_1 components.

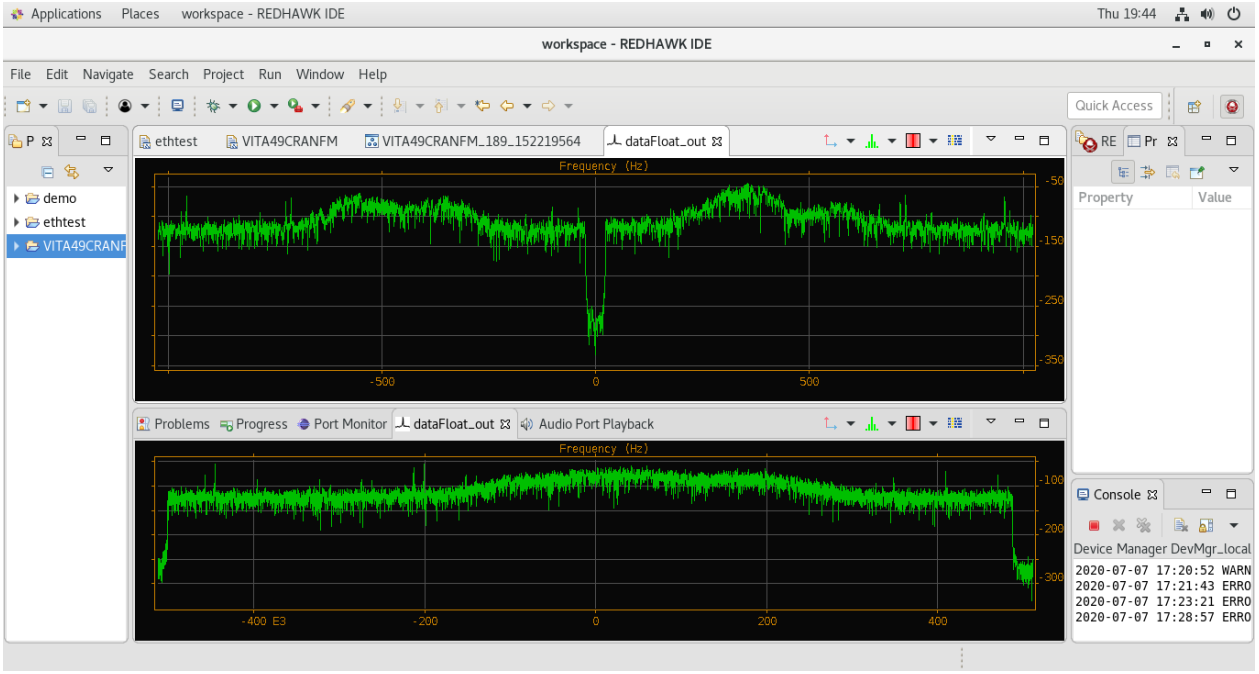

Figure A-20: The FFT plots of the outputs of VITA49Source_1 (top) and TuneFilterDecimate_1 (bottom)

As this implementation is designed as an FM receiver, it may be desirable to listen to the demodulated FM signal. To do this, select the output port of the ArbitraryRateResampler_1

component, right-click, and select “Play port”. This will play the port data through the PC speakers. The gain may have to be increased significantly to hear the signal.

A.6 Conclusions and Other Notes

Using this method, a successful implementation of a split-architecture SDR system was designed, created, and deployed using REDHAWK in all steps of the process. REDHAWK was also used in this deployment to monitor the deployment, and to make changes and perform system administration actively during runtime without the need to stop the process.

It is also worth noting that the DU system in use at the end of testing (the laptop computer on the left in figure A-1) was not the original DU system. The original DU system the original DU system suffered damage outside of testing and was able to be replaced easily by simply moving the REDHAWK waveform files from the old DU system to a working system with REDHAWK installed, and without any changes made to the still-functioning CU system. This shows the importance of the fact that REDHAWK being SCA compatible, as since both the old and new DU systems were SCA compatible, the REDHAWK waveform files could be run on either, and simply requires deployment to the local domain. This extends to all other SCA compatible systems as well, the only caveat being that the receiving system have the required devices for a waveform