

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой №14

д-р.т.н., проф.
должность, уч. степень, звание

[подпись]
подпись, дата
19.06.2019

инициалы, фамилия

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему Исследование программного интерфейса радиочастотной части мобильного
мультирадиотерминала

выполнена Шатуновым Леонидом Владимировичем

фамилия, имя, отчество студента в творительном падеже

по направлению подготовки

09.04.01

код

Информатика и вычислительная техника

наименование направления

направленности

наименование направления

31

код

Встроенные системы

наименование направленности

обработки информации и управления

наименование направленности

Студент группы №

1740М

[подпись] 17.06.19

подпись, дата

Л. В. Шатунов

инициалы, фамилия

Руководитель

к.т.н., доцент

должность, уч. степень, звание

[подпись] 17.06.19

подпись, дата

В.Н. Иванов

инициалы, фамилия

Санкт-Петербург 2019

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

УТВЕРЖДАЮ

Заведующий кафедрой № 14

д.т.н., проф. _____ _____ _____
должность, уч. степень, звание подпись, дата инициалы, фамилия

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ МАГИСТЕРСКОЙ ДИССЕРТАЦИИ

студенту группы № 1740М Шатунову Леониду Владимировичу
(фамилия, имя, отчество)

на тему Исследование программного интерфейса радиочастотной части мобильного
мультирадиотерминала

утвержденную приказом ГУАП от 05.04.2019 № 07-296/19

Цель исследования: Изучение концепции реконфигурируемых радиосистем; исследование,
имплементация и оптимизация программного интерфейса радиочастотной части
реконфигурируемых мобильных устройств.

Задачи исследования: 1) изучить возможность реализации программного интерфейса
радиочастотной части мобильных терминалов; 2) имплементировать и оптимизировать
информационную модель интерфейса RRFL.

Содержание диссертации (основные разделы): обзор существующих систем программного
радио, исследование реконфигурируемой системы, программная реализация

Срок сдачи диссертации « 21 » июня 2019

Руководитель

к.т.н., доцент

должность, уч. степень, звание

Задание принял к исполнению

студент группы №

1740М

_____ 6.06.19
подпись, дата

_____ 7.06.19
подпись, дата

В.Н. Иванов

инициалы, фамилия

Л. В. Шатунов

инициалы, фамилия

РЕФЕРАТ

Пояснительная записка к выпускной квалификационной работе выполнена на 95 страницах, содержит 27 рисунков, 14 источников и включает в себя 7 приложений.

Тема выпускной квалификационной работы – «Исследование программного интерфейса радиочастотной части мобильного мультимедийного терминала».

Цель выполнения работы – Изучение концепции реконфигурируемых радиосистем, имплементация и оптимизация программного интерфейса радиочастотной части реконфигурируемых мобильных устройств.

Пояснительная записка включает в себя:

- обзор существующих систем программного радио;
- исследование реконфигурируемой радиосистемы;
- описание информационной модели разрабатываемого интерфейса;
- программную реализацию на языке C++;
- описание результатов работы программы.

Разработан и оптимизирован программный интерфейс для связи компонент радиокomпьютера с реконфигурируемым трансивером.

Ключевые слова: реконфигурируемое радио, программное радио, радиокomпьютер, мобильное устройство.

СОДЕРЖАНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ СОКРАЩЕНИЙ	6
ВВЕДЕНИЕ.....	7
ГЛАВА 1. ОБЗОР СУЩЕСТВУЮЩИХ СИСТЕМ ПРОГРАММНОГО РАДИО	8
1.1 Концепция Software Defined Radio.....	8
1.2 Программа Joint Tactical Radio System	11
1.3 Software Communication Architecture	12
1.4 Использование беспроводных реконфигурируемых технологий в космосе	15
1.5 Интерфейсы для управления радиочастотной частью устройств.....	17
ГЛАВА 2. ИССЛЕДОВАНИЕ РЕКОНФИГУРИРУЕМОЙ РАДИОСИСТЕМЫ	20
2.1 Реконфигурируемое мобильное устройство	21
2.1.1 Уровень коммуникационных услуг (CSL)	23
2.1.2 Среда для управления радио (RCF)	24
2.1.3 Унифицированные радиоприложения (URA).....	25
2.1.4 Радиоплатформа (Radio Platform)	25
2.2 Реконфигурируемый радиочастотный интерфейс (RRFI)	26
ГЛАВА 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ.....	28
3.1 Постановка задачи	28
3.2 Информационная модель радиокomпьютера.....	28
3.3 Функции интерфейса RRFI	32
3.3.1 SpectrumControlServices	33
3.3.2 PowerControlServices.....	36
3.3.3 Antenna Management Services	37

3.3.4 Tx/Rx Chain Control Services	39
3.4 Используемые инструменты разработки.....	41
3.4.1 CLion IDE (Integrated development environment).....	41
3.4.2 Библиотека BOOST.....	42
3.4.3 Оборудование USRP B210	42
3.4.4 Драйвер UHD (USRP Hardware Driver)	43
3.5 Программная реализация	44
3.5.1 Функции класса SpectrumControlServices	44
3.5.2 Функции класса PowerControlServices.....	46
3.5.3 Функции класса AntennaManagementServices.....	48
3.5.4 Функции класса TxRxChainControlServices	50
3.5.5 Функции класса RRFI.....	53
3.5.6 Класс USRP_Device	53
3.5.7 Файл RRFI_Tests	54
3.6 Результаты работы программы.....	58
3.8 Руководство пользователя.....	63
3.9 Дальнейшая разработка и применение интерфейса	63
ЗАКЛЮЧЕНИЕ	64
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ	65
ПРИЛОЖЕНИЕ А	67
ПРИЛОЖЕНИЕ Б.....	71
ПРИЛОЖЕНИЕ В	73
ПРИЛОЖЕНИЕ Г	76
ПРИЛОЖЕНИЕ Д	80
ПРИЛОЖЕНИЕ Е.....	84
ПРИЛОЖЕНИЕ Ж	88

СПИСОК ИСПОЛЬЗОВАННЫХ СОКРАЩЕНИЙ

SDR – Software Defined Radio

FPGA – Field-Programmable Gate Array

JTRS – Joint Tactical Radio System

SCA – Software Communication Architecture

API – Application Programming Interface

POSIX – Portable Operating System Interface

ПО – Программное обеспечение

NASA – National Aeronautics and Space Administration

STRS – Space Telecommunications Radio System

RFIC – Radio Frequency Integrated Circuit

HAL – Hardware abstraction layer

RRS – Reconfigurable Radio System

CSL – Communication Services Layer

RCF – Radio Control Framework

URA – Unified Radio Applications

MURI – Multiradio Interface

RRFI – Reconfigurable Radio Frequency Interface

Tx – Transmit

Rx – Receive

UML – Unified Modeling Language

MD – Mobile Device

IDE – Integrated development environment

USRP – Universal Software Radio Peripheral

UHD – USRP Hardware Driver

ВВЕДЕНИЕ

В настоящий момент в мире существует множество устройств, которые не используются сами по себе, а тесно связаны между собой сетью. Эти устройства позволяют обществу обмениваться текстом, голосом, видеопотоком и помогают собирать различный контент. С каждым днем устройства эволюционируют: разрабатываются новые процессоры, экраны, блоки памяти, создаются новые программы, протоколы, сервисы, приложения. Также довольно активно развиваются протоколы связи. Например, в 2019 году началось внедрение мобильной связи пятого поколения (5G).

Но существует большая проблема, которая не позволяет быстро перейти на новые протоколы — это необходимость в обновлении оборудования в соответствии с обновленными коммуникационными стандартами.

В связи с этой проблемой появилась тенденция увеличения возможностей аппаратной части коммуникационного оборудования, которая даст возможность быстрого развития и внедрения новых технологий в общество.

Для достижения этой цели была начата разработка реконфигурируемых устройств, имеющих большой спектр возможностей радиооборудования и способность использовать набор различных протоколов на одной платформе.

В данной работе будет проведено исследование программного интерфейса, работающего с радиоспектром на реконфигурируемом устройстве, а также будет реализована его имплементация с использованием подходящего оборудования.

ГЛАВА 1. ОБЗОР СУЩЕСТВУЮЩИХ СИСТЕМ ПРОГРАММНОГО РАДИО

1.1 Концепция Software Defined Radio

Software Defined Radio — программно-определяемая радиосистема, разрабатываемая с 1970 годов, реализуемая на аппаратном уровне в виде радиоприемника/передатчика, позволяющая с помощью ПО устанавливать или изменять радио параметры, такие как, диапазон частот, тип модуляции или выходную мощность. Хотя концепция программно-определяемой радиосистемы не нова, быстро развивающиеся возможности цифровой электроники сделали выполнимыми на практике многие процессы, которые раньше были только теоретически возможными.

В совсем недавнем прошлом беспроводные радиосистемы имели такую конструкцию, что устройство поддерживало один или два типа сигнала и между собой могли связываться только однотипные устройства. Это являлось и является сильным ограничением, а также усложняет организацию связи между разнотипными устройствами. В связи с этим постоянно ощущалась потребность в радио с гибкой архитектурой, которая могла бы изменяться при помощи программного обеспечения. Так появилось словосочетание Software Defined Radio, в которых существенные параметры радио, такие как вид модуляции сигнала передатчика или кодирование сигнала управляется встраиваемым микроконтроллером. Очевидно, что и приёмник для демодуляции сигнала также использует программные средства. [1]

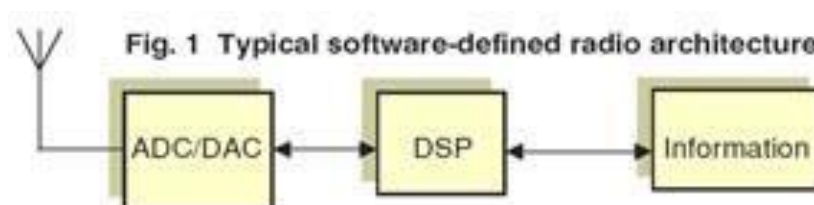


Рисунок 1 – Простая архитектура типового SDR

На рисунке 1 показана упрощенная архитектура для SDR (Software Defined Radio). Она содержит блоки аналого-цифрового, цифро-аналогового преобразования, антенну и цифровой сигнальный процессор. Именно процессор отвечает за гибкость всей системы и используется для алгоритма обработки сигнала. Затем информация попадает на цифро-аналоговый преобразователь и далее на антенну.

SDR определяет набор аппаратных и программных технологий, в которых некоторые или все операционные функции радио (обрабатываемые на физическом уровне) реализуются с помощью модифицируемого программного обеспечения. Такая технология может работать при помощи различных видов программируемых устройств: программируемые пользователем вентильные матрицы (FPGA), цифровые сигнальные процессоры (DSP), процессоры общего назначения (GPP), программируемые системы на кристалле (SoC), а также другие программируемые процессоры, используемые в особых случаях. Подобные устройства позволяют добавлять новые функции и возможности в существующие радиосистемы, не требуя нового или обновленного оборудования. [2]

SDR может выступать в качестве ключевой технологии для различных других реконфигурируемых радиосистем. Хотя SDR не требуется для реализации каждого из типов радио представленных ниже, но технологии SDR могут обеспечить этим типам систем гибкость и преимущества, которые необходимы для достижения их полного потенциала и могут помочь снизить затраты и повысить эффективность системы [3]:

1. Адаптивное радио

Это радио, в котором устройства связи имеют средства контроля своих собственных характеристик и для их улучшения изменяют рабочие параметры. Использование технологий SDR в адаптивном радио обеспечивает большую степень свободы и, следовательно, более высокий уровень производительности и более высокое качество обслуживания на линиях связи.

2. Когнитивное радио

Это радио, в котором устройства связи определяют свое внутреннее состояние и окружающую радиосреду, например, использование радиочастотного спектра в определенном месте. Они могут принимать решения о своем поведении работы, сопоставляя информацию, которую они определяют, с заранее определенными целями. Когнитивное радио во многом использует технологию SDR, чтобы автоматически регулировать свое поведение и операции для достижения желаемых целей. Её использование очень важно, так как позволяет конечным пользователям оптимально использовать имеющийся частотный спектр и другие параметры беспроводных сетей. Это позволяет сократить расходы конечного пользователя, предоставляя возможность ему общаться с тем, кто им нужен, когда это необходимо и каким образом это целесообразно.

3. Интеллектуальное радио

Это когнитивное радио, которое способно к машинному обучению. Это позволяет улучшить способы адаптации к изменениям в окружающей радиосреде, для того чтобы лучше обслуживать потребности конечного пользователя.

Основываясь на программной ориентированности SDR можно отметить множество преимуществ для различных сфер.

Для производителей радиооборудования:

- «Семейства» радиоустройств проектируются с использованием общей платформы, позволяющей быстрее внедрять новые продукты на рынок
- Программное обеспечение повторно используется в радиоустройствах, резко сокращая затраты на разработку.
- Упрощение исправления ошибок при разработке и изготовлении радиоустройств, уменьшение денежных затрат на поддержку и обслуживание.

Для поставщиков радиослужб:

- Новые функции и возможности, которые добавляются к существующей инфраструктуре, не требуя новых и значительных затрат
- Дистанционное обновление программного обеспечения, что позволяет гораздо проще держать полностью обновленными радиоустройства

Для конечных пользователей:

- Сокращение расходов на предоставление конечным пользователям беспроводной связи удобным им способом и как следствие уменьшение нагрузки на бюджет пользователя.

Исходя из этих преимуществ началась активная стадия разработки SDR систем, но для коммерческого пользования это было дорого, поэтому за основную разработку взялись военные и огромный вклад внесла программа Joint Tactical Radio System.

1.2 Программа Joint Tactical Radio System

JTRS (Joint Tactical Radio System) – это перспективная военная радиосистема связи американской армии, программа по разработке которой финансировалась с середины 90-х по 2012 год. Изначально система была предназначена для замены 25-30 разных типов военных радиосистем (многие из которых не могли связываться друг с другом) на одну программную радиосистему, которая могла бы работать во всех диапазонах частот. JTRS использовала широкодиапазонные радиостанции, которые использовали принципы программно-определяемых радиосистем (SDR), что давало возможность работать с множеством существующих военных и гражданских радиосредств.

JTRS включала в себя несколько направлений разработки [4]:

- Ground Mobile Radio (GMR).

Разрабатывается для армейских формирований для комплектования наземных транспортных средств. В направление включена разработка

протокола связи WNW. Основной разработчик — фирма Boeing. Изначально стоимость программы составляла \$370 миллионов, однако в настоящее время стоимость уже превысила \$1.4 миллиарда.

- HMS (Handheld, Manpack and Small form factor).

Портативные и носимые радиостанции

- Радиостанции для BBC, ВМФ и стационарные (Airborne, Maritime, Fixed Station — AMF).

Объединяет разработку радиостанций для BMC, BBC, армейских вертолетов. Первоначальная оценка стоимости разработки составляла — \$500 миллионов, однако в начале 2008 года на программу было выделено дополнительные \$700 миллионов. 28 марта 2008 Lockheed Martin объявила, что она была выбрана JTRS Joint Program Executive Office (учреждением, ведущим общее управление программой JTRS) в качестве основного разработчика программы. Стоимость первоначального контракта на дизайн и опытного демонстрационного образца — \$766 миллионов.

- Радиостанции специального назначения (Special Radios).

Разработка радиостанций для сил специальных операций (USSOCOM)

- Сетевой домен (Networking Enterprise Domain).

Разработка протоколов связи (waveforms) и программного обеспечения управления сетями для всех JTRS радиостанций.

Основной элемент системы JTRS – открытая архитектура SCA (Software Communication Architecture), которая определяет структуру приложений и военных протоколов связи (waveforms). Совместимость различных радиостанций увеличивается вследствие того, что программные компоненты протоколов связи легко переносятся на любые радиостанции, поддерживающие архитектуру SCA.

1.3 Software Communication Architecture

SCA – это открытая архитектура, которая представляет собой воплощение концепции SDR, предлагающая разработчикам

коммуникационного оборудования стандартный способ создания, настройки и управления сигналами приложений, которые выполняются на платформе.

Фундаментальной особенностью SCA является отделение waveforms от основной аппаратной платформы, тем самым облегчая их переносимость и дальнейшее повторное использование, а также для избежания затрат на очередную разработку новых waveforms. SCA следует парадигме разработки на основе компонент (Component Based Development), где каждое waveform приложение является отдельной компонентой. Пример архитектуры радиоплатформы на основе SCA проиллюстрирован на рисунке 2. ПО waveforms изолировано различными унифицированными API (Application Programming Interface). [5]

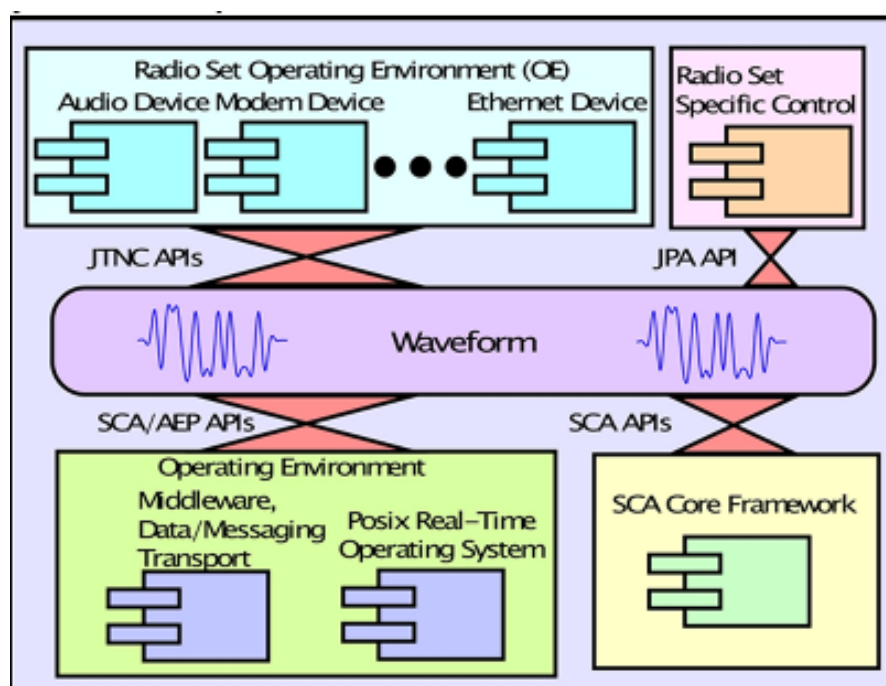


Рисунок 2 – Пример архитектуры SCA

Рассмотрим эту схему. Основная часть – это SCA Core Framework, которая определяет необходимый "базовый" набор открытых программных интерфейсов и профилей, которые обеспечивают развертывание, управление, взаимосвязь и взаимодействие компонентов программных приложений во встроенной системе.

Также немаловажная часть – Operating environment (рабочее окружение), которое содержит в себе:

1. Real-time operating system – это операционная система реального времени, которая выполняет операции, определенные в POSIX Application Environment Profile.
2. POSIX AEP – это подмножество спецификации POSIX (Portable Operating System Interface), которое определяет операции, которые могут быть вызваны из ПО (программного обеспечения) waveforms в базовую операционную систему
3. CORBA (Common Object Request Broker Architecture) Middleware – это механизм связи между процессами, который обеспечивает независимость языка программирования и операционной системы для компонентов SCA.

В верхней части схемы расположены Devices (устройства) – они обеспечивают связь между протоколами связи и физическим радиооборудованием, которое в свою очередь работает с радиоспектром.

Приложения SCA – многоазовые и портативные компоненты SCA, которые развернуты и запущены при помощи SCA Core Framework и взаимодействуют с физическими устройствами через SCA Devices.

Как показано на рисунке 3 аппаратная платформа осуществляет работу через набор Joint Tactical Networking Center API (JTNC API), которые определяют функциональные возможности каждого устройства, независимо от их исполнения. Программное обеспечение приложений SCA защищено от изменений путем вызова только четко определенных API для различных типов оборудования. Для этого платформа SCA предоставляет программные proxies (доверенности), проще называемыми устройствами SCA, которые взаимодействуют с приложением SCA, используя функции API, перенаправляющие вызовы приложения SCA на физическое оборудование, которое использует уже собственные драйверы [6]. Таким образом обновление аппаратной части платформы не требует каких-либо изменений программного

обеспечения приложения SCA, требуется только адаптация проху (устройства SCA) для использования нового оборудования через собственные драйверы.

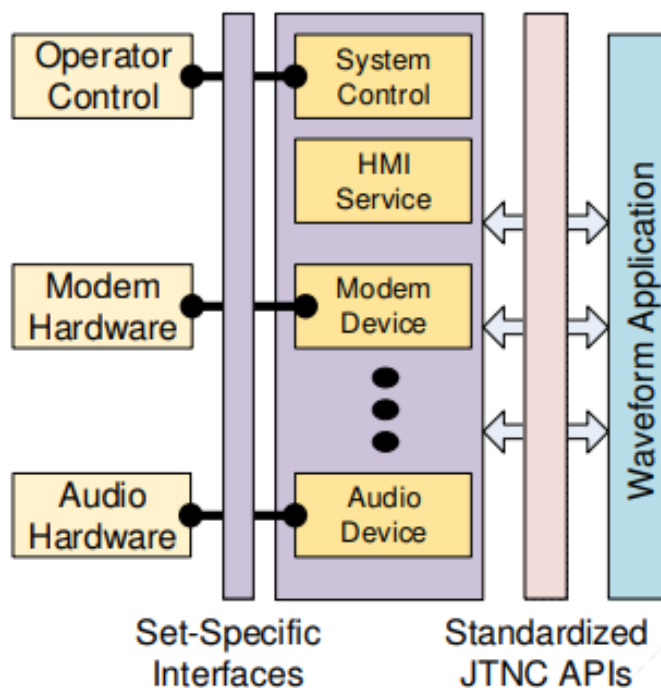


Рисунок 3 – Аппаратная часть SCA

Четкое разделение аппаратных устройств и waveform приложений поддерживает целостность структуры SCA и сохраняет ее применимость в достаточно широком диапазоне областей, а также позволяет каждому семейству спецификаций развиваться в соответствии с его собственным графиком разработки.

1.4 Использование беспроводных реконфигурируемых технологий в космосе

На рынке растет число реконфигурируемых устройств и это дает возможность улучшить процесс разработки и эксплуатации космических трансиверов для связи и навигации в космических миссиях. Программно-определяемые радиостанции с функциями связи и навигации, реализованными в программном обеспечении, обеспечивают возможность изменения функциональных возможностей радиостанции во время подготовки к миссии

или после запуска в космос. Способность изменять эксплуатационные характеристики радиоприемника с помощью программного обеспечения после его полного развертывания в космосе обеспечивает возможность использовать новые научные возможности и потенциально снижает затраты на разработку путем адаптации типовых космических платформ к конкретным требованиям каждой миссии.

NASA (National Aeronautics and Space Administration) разрабатывает архитектуру космической телекоммуникационной радиосистемы (Space Telecommunications Radio System – STRS) в качестве стандарта открытой архитектуры. Такая архитектура уменьшит зависимость от «разовых» реализаций радиосистем, которые разрабатываются под каждую миссию отдельно. Эти «разовые» реализации оставляют NASA в зависимости от собственных разработок, которые, как правило, не являются масштабируемыми и не позволяют повторно использовать проект и использовать другие методы открытого стандарта архитектуры.

Открытый стандарт архитектуры способствует использованию уже разработанных и четко определенных интерфейсов, которые позволяют различным поставщикам предоставлять радиостанции, соответствующие одному стандарту интерфейса, обеспечивая тем самым совместимость между поставщиками различных аппаратных и программных элементов. Стандартизированные интерфейсы предоставляют возможность для замены компонентов, внедрения технологий и снижения рисков за счет повторного использования аппаратных и программных компонентов. Цель NASA в разработке стандарта STRS состоит в том, чтобы одновременно использовать экономию и преимущества технологии SDR и открытого стандарта архитектуры. [7]

В ходе разработки системы агентство NASA адаптировало архитектуру к различным ситуациям, которые возникают в космосе:

1. Космическая среда

Компоненты, которые используются в системе сконструированы таким образом, чтобы выдерживать космическое влияние, например, радиацию или свет от солнца.

2. Космические ресурсы

Размер, вес и электропитание являются очень ценными ресурсами, когда объект находится в космосе. Они сильно ограничивают радиоустройства, которые разрабатываются для космических целей.

3. Надежность и доступность

Радиосистемы на орбите должны иметь высокую степень надежности и должны быть всегда доступны, особенно в случаях применения на космических аппаратах с экипажем.

4. Сокращение времени подготовки к миссиям

Время разработки космического радиооборудования гораздо дольше, чем время, которое отводится на подготовку миссии, поэтому должны использоваться существующие радиоустройства с программными изменениями.

Подводя итог можно сделать вывод, что архитектура STRS, учитывающая уникальность космической среды, позволяет NASA существенно экономить время и денежные средства при подготовке к космическим миссиям, а также иметь ряд стандартизированных интерфейсов благодаря открытому стандарту архитектуры.

1.5 Интерфейсы для управления радиочастотной частью устройств

Существуют различные интерфейсы для управления радиочастотной частью на многих устройствах.

Один из таких интерфейсов – DigRF. Это стандарт цифрового интерфейса, разрабатываемый и поддерживаемый альянсом MIPI (Mobile Industry Processor Interface), который позволяет соединить радиопередатчик мобильного устройства с радиоспектром.

DigRF специально был разработан для быстро развивающегося спектра новых мобильных технологий. Это высокоскоростная технология последовательного интерфейса, поддерживающая различные режимы, которые обеспечивают хорошую энергоэффективность и высокую скорость передачи сигнала. Широкий диапазон моделей разработки и их применения — неотъемлемая часть этого стандарта, в связи с чем он предлагает различные возможности для системных разработчиков.

При разработке интерфейса MIPI руководствовались следующими критериями:

- минимизация количества контактов интерфейса;
- минимизация общего энергопотребления интерфейса;
- обеспечение надежного физического уровня путём обнаружения и исправления ошибок;
- обеспечение достижения совместимости интерфейса между RFIC (Radio Frequency Integrated Circuit) разных производителей и радиоплатформой.

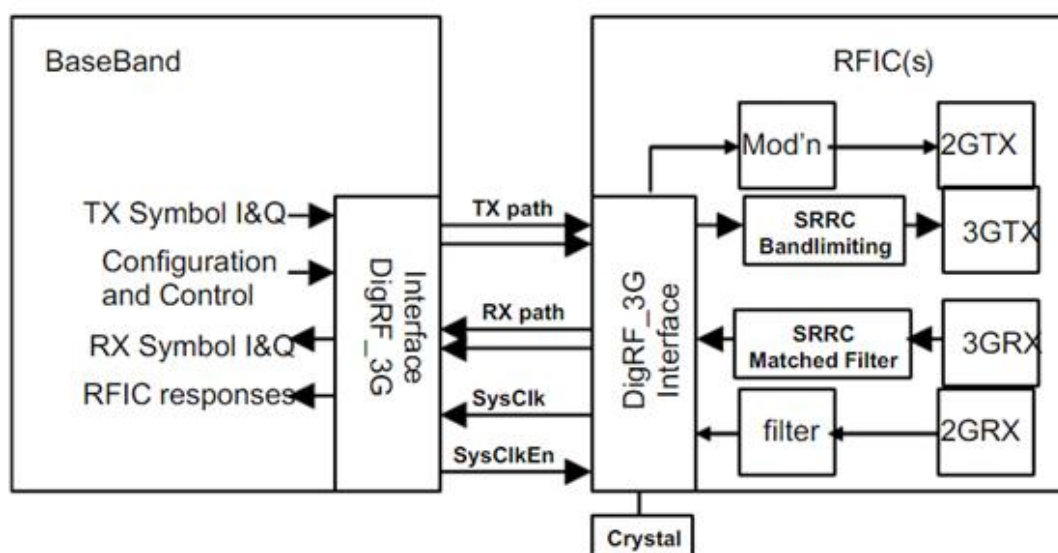


Рисунок 4 – Схема DigRF

На рисунке 4 представлено схематичное изображение DigRF. Интерфейс встраивается между RFIC и радиоплатформой, реализован и

специфицирован на физическом уровне. DigRF работает на основе деления сигнала приема и передачи информации на синфазную (I) и квадратурную (Q) составляющую.

Интерфейс представляет собой последовательную шину данных с шестью каналами: два для Rx (приема) I/Q данных, два для Tx (передачи) I/Q данных, один для тактового сигнала, и один для синхронизации. Каналы приема и передачи работают на технологии LVDS - низковольтной дифференциальной передачи сигналов. Также сигналы, в зависимости от их типа, при приеме или передаче проходят различную обработку, например, через SSRC(square-root-raised-cosine) фильтр. [8]

Подводя итог обзора интерфейса DigRF можно отметить следующие особенности:

- Интерфейс специфицируется на нижнем уровне;
- Реализованы только функции приема и передачи и их синхронизации;
- Позволяет добиться высокой скорости передачи и приема.

ГЛАВА 2. ИССЛЕДОВАНИЕ РЕКОНФИГУРИРУЕМОЙ РАДИОСИСТЕМЫ

Существует 2 подхода в программировании:

1. Программа предназначена для «стандартного» компьютера (типа машины фон Неймана). Это означает, что таком компьютере есть универсальный процессор, имеется операционная система, а также специальный слой HAL (Hardware abstraction layer), который абстрагирует аппаратную платформу от программной среды. Например, можно писать программу под Windows или Linux.
2. Неважно какая аппаратная платформа, но имеется некоторая виртуальная машина и программный код пишется для этой конкретной виртуальной машины. Затем компилятор отображает эту выбранную виртуальную машину в целевую аппаратную платформу.

SDR — это реализация первого подхода. Однако в случае встраиваемых приложений этот подход не позволяет добиться оптимального использования аппаратных ресурсов. Во встраиваемой системе имеются разнородные вычислительные ресурсы с разным уровнем программируемости, поэтому само понятие программного кода, как кода для центрального процессора становится бессмысленным. Таким образом первый подход налагает определенные ограничения на аппаратную архитектуру. Кроме того, наличие слоя аппаратных абстракций также предполагает некоторое ограничение архитектуры. Еще одна проблема подхода, который использует SDR – это переносимость программного кода. Она не достигается в общем случае даже с использованием POSIX. И наконец SDR основывается на программной шине и модели «клиент-сервер». Эта модель не формализуется и поэтому формально не верифицируется.

Реконфигурируемые системы (RRS – Reconfigurable Radio System) это реализация второго подхода. В этом случае используется специфическая

виртуальная машина, которая способна выражать такие важные особенности радиосистем как реактивность (т.е. вычисление начинается, когда имеются данные), параллельность и переменная гранулярность вычислений (т.е. операции могут быть разной гранулярности, например, арифметические и преобразование Фурье). В целом эти свойства помогают генерировать оптимальный код для разных целевых платформ и в то же время хорошо поддерживают переносимость кода. В том числе в первом случае связь функциональных компонент происходит через программную шину, в то время как в RRS это взаимодействие описывается на уровне программного кода и компилятор имеет возможность оптимизировать коммуникацию исходя из наличия аппаратных ресурсов. Таким образом можно сделать вывод, что на данный момент системы RRS имеют больше перспектив, чем системы SDR.

RRS — это концепция, основанная на таких технологиях, как реконфигурация программного обеспечения с помощью радиоприложений, радиовиртуальная машина и когнитивное радио, которое имеет возможность самоадаптации к динамически изменяющейся среде с целью более оптимального использования радиоспектра.

Глобальный интерес к RRS подпитывается быстро растущим спросом на беспроводную связь для большого количества различных целей. Сейчас в мире около 7 триллионов беспроводных устройств, которые обслуживают 7 миллиардов пользователей. Для удовлетворения их потребностей, которые ограничиваются размером радиоспектра, необходимы более гибкие способы совместного использования радиосетей и частот сразу несколькими службами и радиосетями – и технология RRS предлагает решение этой проблемы.

2.1 Реконфигурируемое мобильное устройство

Основным элементом в реконфигурируемой радиосистеме является реконфигурируемое мобильное устройство. С этим устройством взаимодействует пользователь, а также оно способно взаимодействовать с

несколькими радиопротколами одновременно путём загрузки и установки пакетов радиоприложений (RAP).

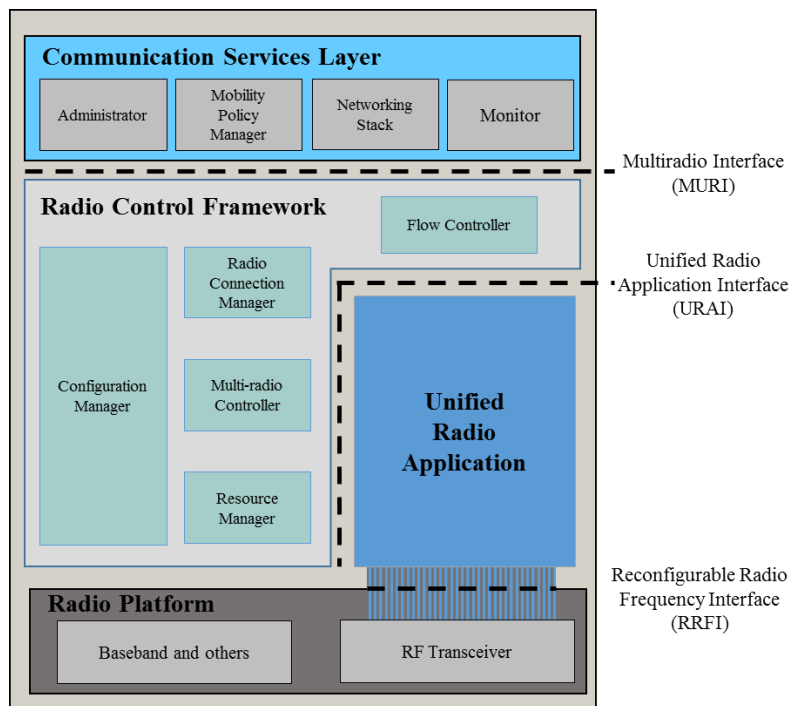


Рисунок 5 – Реконфигурируемое мобильное устройство

На рисунке 5 представлено схематичное изображение архитектуры реконфигурируемого мобильного устройства, которая состоит из четырех частей [9]:

1. **Communication Services Layer (CSL)** – уровень коммуникационных услуг, который включает в себя 4 компонента (Administrator – администрирование, Mobility Policy Manager – управление политикой мобильности, Networking Stack – сетевой стек, Monitor – мониторинг);
2. **Radio Control Framework (RCF)** – среда для управления радио, которая включает в себя 5 компонент (Configuration Manager – управление конфигурациями, Radio Connection Manager – управление радиосоединением, Multiradio Controller – контроллер мультирадио, Resource Manager – управление ресурсами, Flow Controller – контроллер потока);

3. Unified Radio Applications (URA) – унифицированные радиоприложения;
4. Radio Platform – радиоплатформа, состоящая из радиопередатчика и радиоприемника, а также антенны.

Реализация и описание компонент RRS происходит на программном уровне, что позволяет оптимизировать разработку системы. Управление протоколами, представленных в виде приложений, упрощает работу платформы, что открывает возможность обработки любых протоколов беспроводной связи.

Программные компоненты взаимодействуют друг с другом посредством трех интерфейсов:

1. Multiradio Interface (MURI) – мультирадио интерфейс, связывающий CSL и RCF;
2. Unified Radio Applications Interface (URAI) – интерфейс радиоприложений, связывающий RCF и URA;
3. Reconfigurable Radio Frequency Interface (RRFI) – реконфигурируемый радиочастотный интерфейс, связывающий URA и радиоплатформу.

Рассмотрим все компоненты реконфигурируемого мобильного устройства по отдельности.

2.1.1 Уровень коммуникационных услуг (CSL)

Это уровень, связанный со службами связи, которые поддерживают как общие приложения, так и конкретные мультирадио приложения. CSL включает в себя следующие 4 элемента:

- Administrator

Администрирование как правило содержит функции запросов установки и удаления URA, а также функции создания и удаления экземпляров URA. Это включает в себя предоставление информации о URA, их статус и т. д.;

- Mobility Policy Manager

Управление политикой мобильности содержит функции мониторинга радиоокружения и возможностей мобильного устройства, так же оно запрашивает активацию или деактивацию URA и предоставляет информацию о списке доступных URA. Также этот элемент осуществляет выбор среди различных технологий радиодоступа;

- Networking stack

Сетевой стек включает в себя функции отправки и получения пользовательских данных;

- Monitor

Мониторинг содержит функцию передачи информации из URA пользователю или другому объекту назначения в мобильном устройстве.

2.1.2 Среда для управления радио (RCF)

RCF обеспечивает универсальную среду для исполнения URA и единый способ доступа к функционалу радиокomпьютера. RCF работает с элементами CSL через MURI. RCF включает в себя 5 элементов по управлению URA:

- Configuration Manager

Управление конфигурациями как правило включает в себя функции установки/удаления и создания/удаления экземпляров URA, а также функции управления и доступа к радиопараметрам URA.

- Radio Connection Manager

Управление радиосоединением содержит функции активации/деактивации URA в соответствии с запросом пользователя, а также функции управления потоками пользовательских данных, которые могут переключаться между различными RA.

- Flow controller

Контроллер потока включает в себя функции отправки и получения пакетов пользовательских данных, а также функции управления потоком служебных данных.

- Multiradio Controller

Контроллер мультирадио содержит в себе функции для планирования запросов доступа к радиоресурсам и обнаружения и устранения проблем совместимости между одновременно запущенными URA.

- Resource Manager

Управление ресурсами включает в себя функции управления вычислительной мощностью и правильного её распределения между одновременно активными URA для обеспечения их правильного выполнения в режиме реального времени.

2.1.3 Унифицированные радиоприложения (URA)

Как описано в разделе 2.1.2, RCF представляет функциональные возможности радиокомпьютера, который является частью оборудования мобильного устройства, работающий под управлением радио операционной системы на которой выполняются радиоприложения. Также RCF требует, чтобы все радиоприложения подвергались общей конфигурации, имели возможность быть активными параллельно с другими приложениями и грамотно пользоваться вычислительными ресурсами. Поскольку все радиоприложения демонстрируют одинаковое поведение в связи с требованиями RCF, то они получили название – унифицированные радиоприложения. Службы, связанные с активацией/деактивацией приложений, поддержкой потоков пользовательских данных управляются через интерфейс унифицированных радиоприложений (URAI), который связывает RCF и URA.

2.1.4 Радиоплатформа (Radio Platform)

Радиоплатформа содержит в себе оборудование, которое работает с радиоспектром и имеет широкие возможности радиопараметров для их реконфигурации. Все операции по конфигурации радиооборудования производятся через реконфигурируемый радиочастотный интерфейс (RRFI), встраиваемый между радиоплатформой и URA.

2.2 Реконфигурируемый радиочастотный интерфейс (RRFI)

RRFI это цифровой интерфейс, работающий в составе мобильного устройства и обеспечивающий связь URA с радиоплатформой и её настройку.

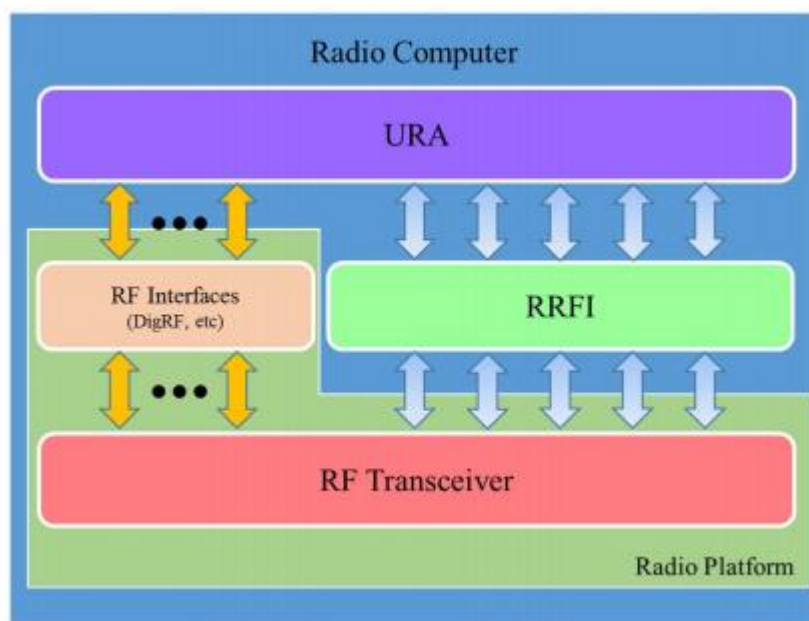


Рисунок 6 – Расположение RRFI в мобильном устройстве

На рисунке 6 представлена часть архитектуры реконфигурируемого мобильного устройства. RRFI встраивается между двумя компонентами системы: радиоплатформой, которая работает с радиоспектром и URA (Unified Radio Application), которое отвечает за настройку радиоплатформы и подготовку данных для передачи. RRFI реализуется на высоком программном уровне и дополняет функциями управления и обслуживания радиоплатформы низкоуровневые интерфейсы, которые имеют только функции приема/передачи данных и синхронизации.

RRFI включает в себя несколько служб для конфигурации радиоплатформы [10]:

1. Spectrum Control Services

Контроль спектра используется для настройки параметров, связанных с радиоспектром, таких как несущая частота, полоса пропускания, частота дискретизации и т.д. Эти параметры определяются в соответствии с URA, которое их задает.

2. Power Control Services

Служба управления мощностью используется для настройки параметров, связанных с мощностью радиопередатчика, например, максимальный уровень передачи (Tx – Transmit), уровень мощности на антенну, коэффициент усиления приема (Rx – Receive) и т.д. Определенные схемы управления мощностью должны контролироваться в соответствии с радиоокружением мобильного устройства. Эта задача также решается этой службой.

3. Antenna Management Services

Служба управления антеннами используется для определения конфигурации антенн. Настраиваются такие параметры как диаграмма направленности антенны, коэффициент усиления, поляризация, диапазон частот и др. В том числе функции этой службы зависят возможности конфигурации самой антенны

4. Tx/Rx Chain Control Services

Служба управления потоками данных на прием и передачу используется для настройки параметров, связанных с контролем этих потоков в режиме реального времени. Она отвечает за параметры синхронизации времени и конца приема или передачи потока данных.

ГЛАВА 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

3.1 Постановка задачи

В данной работе была поставлена задача исследования и имплементации интерфейса RRFI для реконфигурируемого мобильного устройства. Для представления интерфейса в программном виде необходимо исследовать информационную модель радиокomпьютера, включающую компоненты с которыми взаимодействует интерфейс RRFI, изучить функции, которые должен включать в себя этот интерфейс, а также выбрать оборудование и программные средства для реализации.

Для дальнейшего использования и разработки также необходимо добавить руководство пользователя и руководство программиста, в которых будут описаны основные моменты запуска и использования интерфейса в дальнейшей разработке.

3.2 Информационная модель радиокomпьютера

На рисунке 7 представлена UML (Unified Modeling Language) диаграмма радиокomпьютера

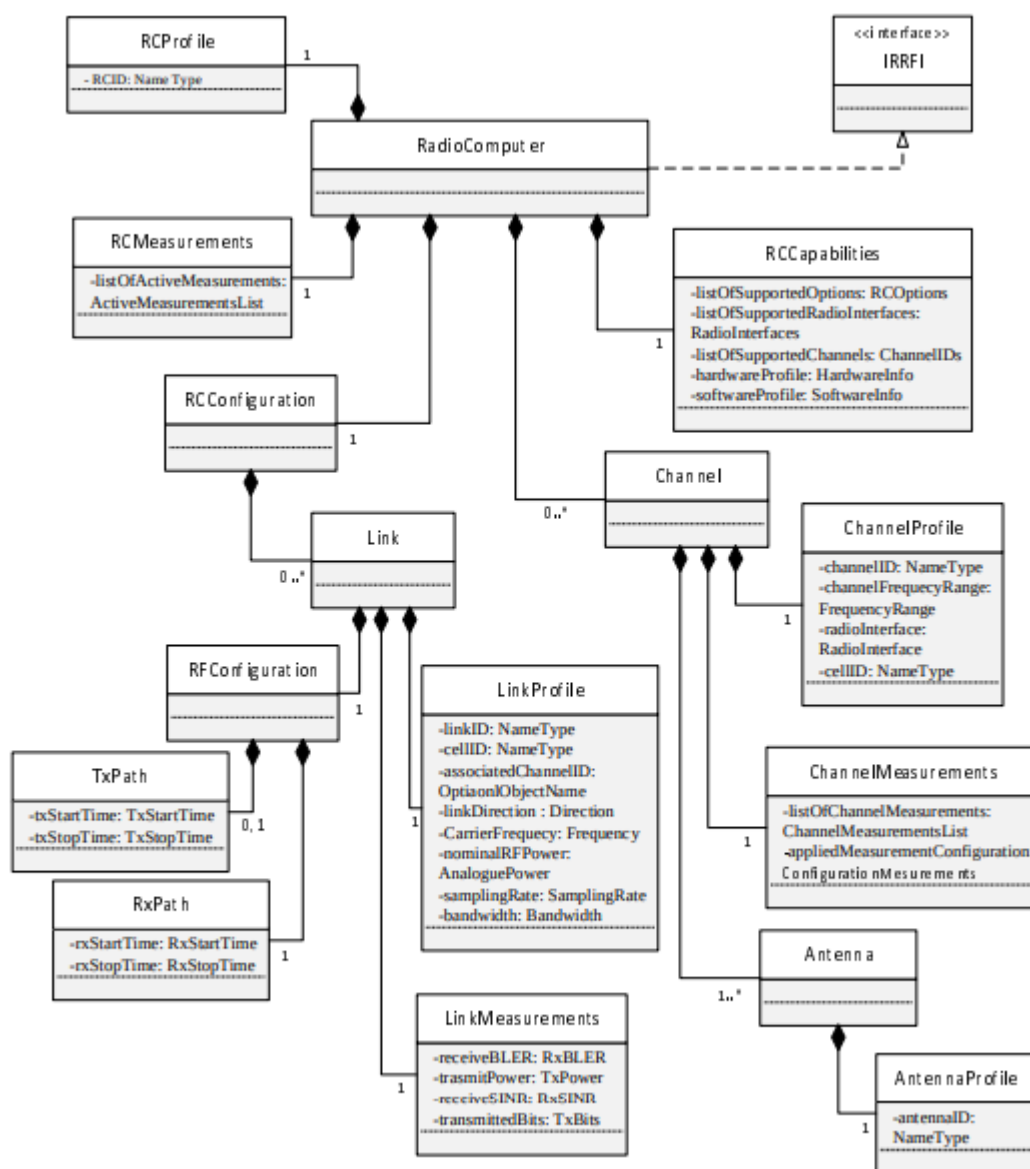


Рисунок 7 – UML диаграмма радиопьютера

Диаграмма содержит описания классов радиопьютера, которые взаимодействуют с интерфейсом RRFI. Рассмотрим каждый из них:

- RadioComputer

Этот класс содержит относящуюся к URA информацию о ресурсах и взаимодействиях, связанных с аппаратным и программным обеспечением мобильного устройства, например, использование вычислительных ресурсов, использование радиоспектра, сбор контекстной информации и т.д.

- RCCapabilities

Этот класс содержит информацию о возможностях радиоустройства, включая аппаратные средства, программное обеспечение, возможности

передачи/приема, а также информацию о поддерживаемых радиопотоколах, максимальной мощности передачи. Каждый экземпляр класса RadioComputer может иметь только один экземпляр класса RC Capabilities в качестве члена.

- Channel

Этот класс содержит один радиоканал, который может использоваться или не использоваться активным радиосоединением. Каждый экземпляр класса RadioComputer может иметь ноль, один или несколько экземпляров класса Channel в качестве членов (0..*). В случае активного радиосоединения доступен хотя бы один экземпляр класса Channel.

- ChannelProfile

Этот класс содержит общие сведения о радиоканале, такие как идентификатор канала, несущая частота, полоса пропускания и используемый радиопотокол. Каждый экземпляр класса Channel может иметь только один экземпляр класса ChannelProfile в качестве члена.

- ChannelMeasurements

Этот класс содержит текущие измерения (мгновенные данные измерений и связанные метаданные) и применяемую конфигурацию измерений, связанную с этим радиоканалом, например, измерения помех и нагрузки. Каждый экземпляр класса Channel может иметь только один экземпляр класса ChannelMeasurements в качестве члена.

- Antenna

Этот класс содержит информацию о выборе антенны. Каждый экземпляр класса Link должен иметь по крайней мере один экземпляр класса Antenna в качестве члена (1..*).

- AntennaProfile

Этот класс содержит общие сведения об определенной антенне, такие как порт антенны, применимый диапазон частот и коэффициент усиления антенны. Каждый экземпляр класса Antenna может иметь только один экземпляр класса AntennaProfile в качестве члена.

- RCConfiguration

Этот класс содержит сведения о текущей конфигурации радиокомпьютера. Каждый экземпляр класса RadioComputer может иметь только один экземпляр класса RConfiguraton в качестве члена.

- Link

Этот класс содержит информацию об одном активном URA и соответствующем соединении между реконфигурируемым MD (Mobile Device) и сетью радиодоступа (RAN). Каждый экземпляр класса RConfiguration имеет ноль, один или несколько экземпляров класса Link в качестве членов (0..*). Каждый экземпляр класса Link связан с одним экземпляром класса Channel.

- LinkProfile

Этот класс содержит общие сведения об этом активном соединении, например, идентификатор соединения (ID), идентификатор обслуживающей ячейки, используемый канал и т. д. Каждый экземпляр класса Link может иметь только один экземпляр класса LinkProfile в качестве члена.

- LinkMeasurements

Этот класс содержит текущие измерения (мгновенные данные измерений и связанные метаданные), связанные с этим активным соединением, такие как частота блоковых ошибок (BLER), мощность и измерение отношения сигнал/помехи плюс шум (SINR). Каждый экземпляр класса Link может иметь только один экземпляр класса LinkMeasurements в качестве члена.

- RFConfiguration

Этот класс содержит информацию о конфигурации радиочастотного приемопередатчика. Каждый экземпляр класса Link может иметь только один экземпляр класса RFConfiguration в качестве члена.

- TxPath

Этот класс содержит сведения об одном канале передачи. Каждый экземпляр класса RFConfiguration имеет ноль или один экземпляр класса TxPath в качестве члена (0,1).

- RxPath

Этот класс содержит сведения об одном канале приема. Каждый экземпляр класса RFConfiguration может иметь только один экземпляр класса RxPath в качестве члена.

- RCMeasurements

Этот класс содержит текущие измерения (мгновенные данные измерений и связанные метаданные), связанные с реконфигурируемым мобильным устройством, такие как емкость аккумулятора, мобильность пользователя, определение местоположения устройства и сведения об истории соединений. Каждый экземпляр класса RadioComputer может иметь только один экземпляр класса RCMeasurements в качестве члена.

- RCProfile

Этот класс содержит общую информацию о радио компьютере, например, идентификаторе радиоустройства. Каждый экземпляр класса RadioComputer может иметь только один экземпляр класса RCProfile в качестве члена.

Основываясь на описании классов радиопередающего компьютера можно выделить несколько задач, которые должны решать функции интерфейса RRFI:

1. Предоставление однозначного доступа для каждого из классов радиопередающего компьютера.
2. Предоставление понятных выходных данных каждому классу
3. Прием всех данных, которые могут содержаться в классах радиопередающего компьютера.

3.3 Функции интерфейса RRFI

Функции интерфейса RRFI подразделяются на 4 службы: SpectrumControlServices, PowerControlServices, AntennaManagementServices, TxRxChainControlServices. На рисунке 8 представлена UML диаграмма интерфейса RRFI.

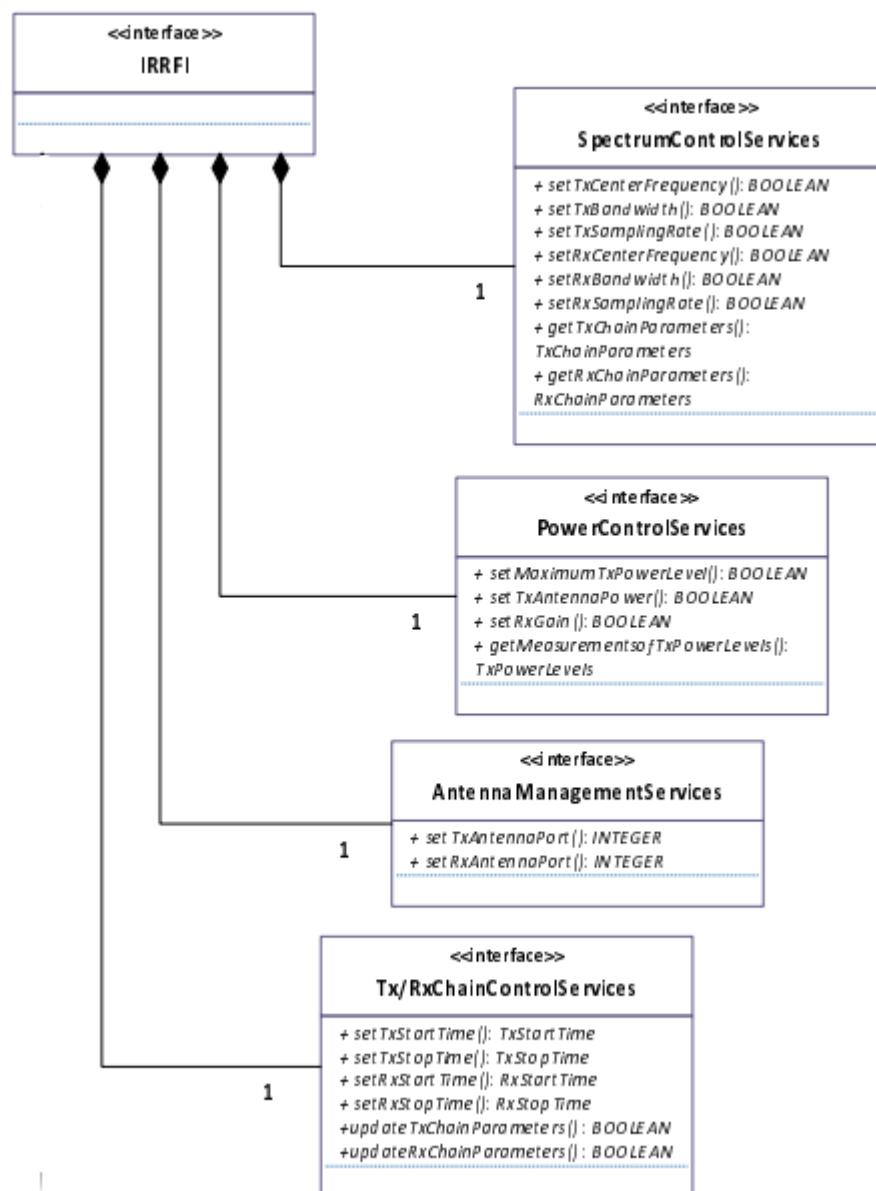


Рисунок 8 – UML диаграмма интерфейса IRRFI

Каждая служба содержит функции определенного типа, например, PowerControlServices конфигурирует параметры мощности и усиления антенны. Рассмотрим функции каждой службы по отдельности.

3.3.1 SpectrumControlServices

- setTxCenterFrequency/setRxCenterFrequency

Это функция настройки несущей частоты. Поскольку диапазон частот, используемый каким-либо определенным протоколом связи, может быть

изменен в зависимости от окружения пользователя, эта функция обеспечивает настройку несущей частоты в диапазоне частот протокола связи.

- SetTxBandwidth/SetRxBandwidth

Это функция настройки полосы пропускания. Поскольку диапазон частот, используемый каким-либо определенным протоколом связи, может быть изменен в зависимости от среды пользователя, эта функция обеспечивает настройку полосы пропускания канала протокола связи.

- SetTxSamplingRate/SetRxSamplingRate

В зависимости от используемого протокола связи эта функция обеспечивает настройку частоты дискретизации.

- GetTxChainParameters/GetRxChainParameters

Информацию о параметрах канала связи, таких как несущую частоту, полосу пропускания, частоту дискретизации для используемого протокола связи могут запрашивать не только URA, но и другие компоненты радиокомпьютера. Эта функция службы управления спектром предоставляет данную информацию.

На рисунке 9 представлены сценарии использования этих функций службы контроля спектра между URA и радиотрансивером, а также сообщения, которыми обмениваются эти два компонента.

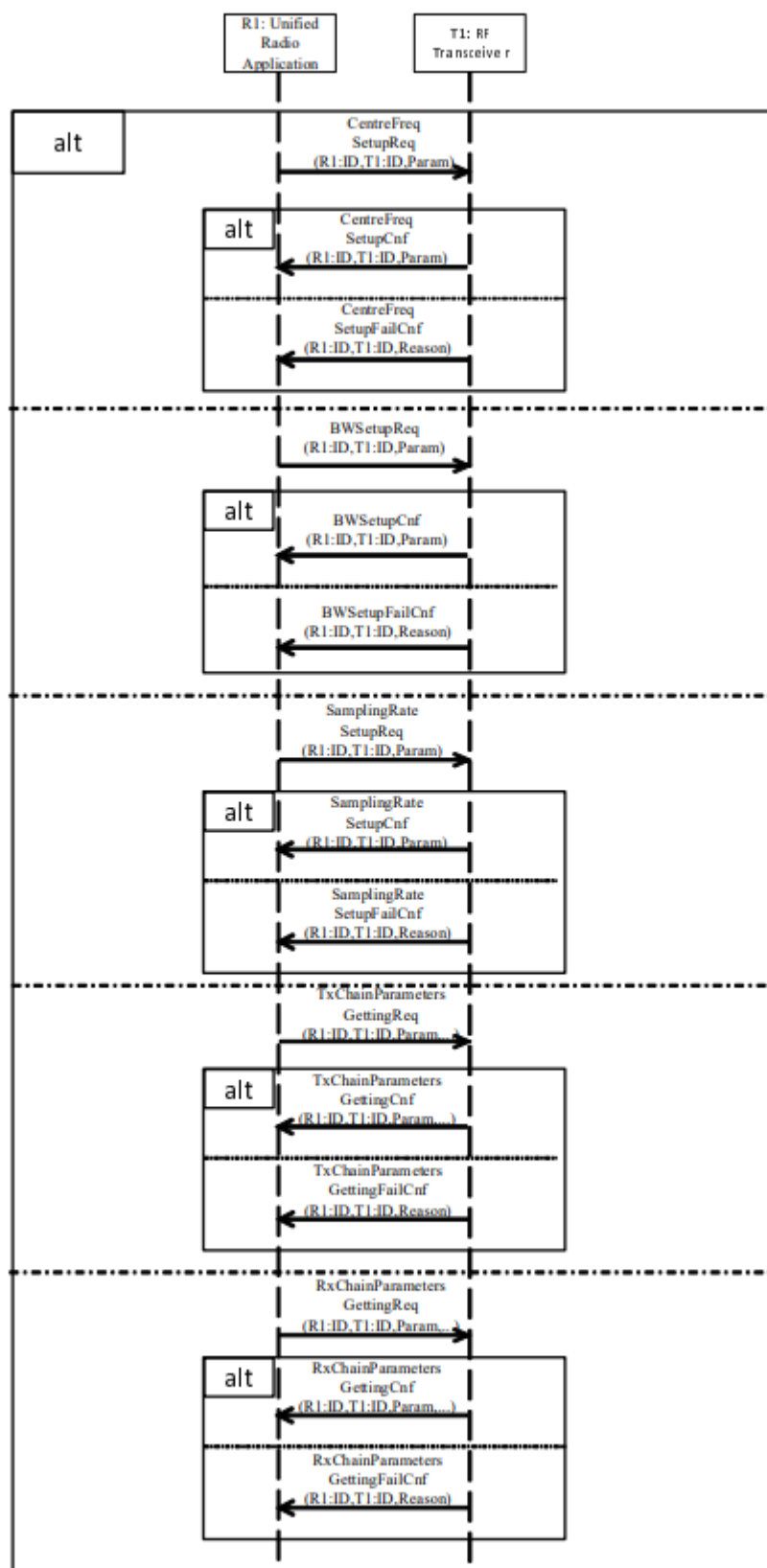


Рисунок 9 – Сценарии взаимодействия службы контроля спектра

Сообщения, которыми обмениваются два компонента радиокomпьютера включают в себя запросы от URA к радиоустройству на установку либо

получение различных параметров, например, запрос на настройку несущей частоты, а также включают в себя ответы от радиоустройства к URA оповещающие об успешном или неуспешном выполнении операции.

3.3.2 PowerControlServices

- SetMaximumTxPower

Поскольку каждый протокол связи может иметь свой определенный максимальный уровень мощности, эта функция обеспечивает настройку максимальной мощности передачи для каждого URA. Максимальный уровень мощности передачи может динамически меняться в зависимости от пользовательского окружения

- SetTxAntennaPower

Нескольким технологиям для антенн, таким как множественный вход, множественный выход (MIMO), различным диаграммам направленности и т.д. может потребоваться различная мощность передачи на каждой антенне. Эта функция обеспечивает настройку мощности передачи для определенной антенны. Алгоритм распределения мощности антенн может быть определен в зависимости от исходного кода радиоприложения, которое предоставляется третьей стороной.

- SetRxGain

Поскольку требуемый уровень мощности сигнала Rx может отличаться для каждого протокола связи, эта функция обеспечивает настройку коэффициента усиления приема, чтобы иметь возможность работать с слишком низким или слишком высоким уровнем мощности сигнала приема

- GetMeasurementsOfTxPowerLevels

Эта функция предоставляет информацию об уровне мощности сигнала передачи, и максимальном уровне мощности передачи.

На рисунке 10 представлены сценарии использования этих функций службы управления мощностью между URA и радиотрансивером, а также сообщения, которыми обмениваются эти два компонента.

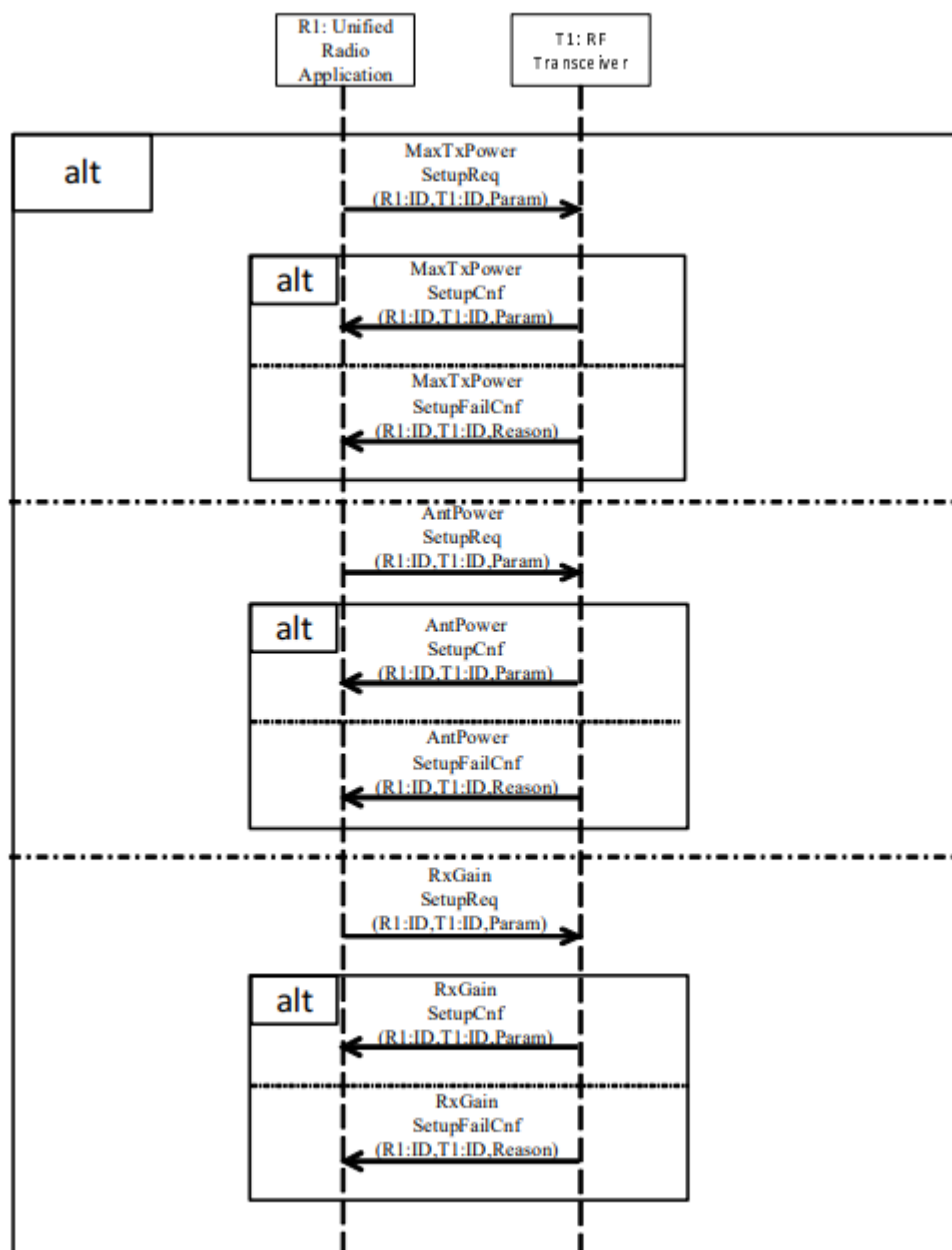


Рисунок 10 – Сценарии взаимодействия со службой управления мощностью

Сообщения, которыми обмениваются два компонента радиокomпьютера включают в себя запросы от URA к радиоустройству на установку параметров мощности или усиления сигнала, и ответы от радиоустройства к URA оповещающие об успешном или неуспешном выполнении операции.

3.3.3 Antenna Management Services

- SetTxAntennaPort/SetRxAntennaPort

В случае, когда на устройстве установлено нескольких антенн, каждая из которых может быть связана с различными характеристиками приема или передачи, должно быть доступно любое подмножество нескольких антенн. Эта функция обеспечивает выбор портов антенны для каждого URA. В зависимости от окружения пользователя один и тот же URA может использовать разные подмножества антенн.

На рисунке 11 представлен сценарий использования этой функции службы управления антеннами между URA и радиотрансивером, а также сообщения, которыми обмениваются эти два компонента.

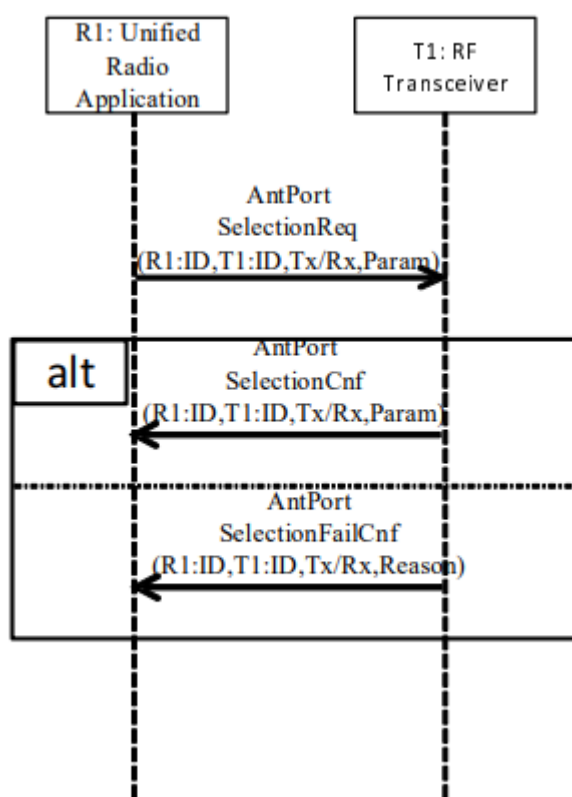


Рисунок 11 – Сценарий взаимодействия со службой управления антеннами

Сообщения, которыми обмениваются два компонента радиокomпьютера включают в себя запрос от URA к радиоустройству на выбор порта антенны и ответы от радиоустройства к URA оповещающие об успешном или неуспешном выполнении операции.

3.3.4 Tx/Rx Chain Control Services

- TxStartTime/RxStartTime

Поскольку поток передачи или приема данных может быть активен в течение заданного времени, эта функция обеспечивает настройку времени запуска соответствующего потока приема или передачи.

- TxStopTime/RxStopTime

Поскольку поток передачи или приема данных может быть активен в течение заданного времени, эта функция обеспечивает настройку времени остановки соответствующего потока приема или передачи.

- UpdateTxChainParameters/UpdateRxChainParameters

Учитывая тот факт, что конкретный набор параметров на запущенном в настоящее время URA может быть изменен, эта функция предоставляет запрос на обновление параметров потока приема или передачи, таких как центральная частота, полоса пропускания и т.д. в реальном времени выполнения URA, которое при возможности их изменит.

На рисунке 12 представлены сценарии использования этих функций службы управления потоками приема/передачи между URA и радиотрансивером, а также сообщения, которыми обмениваются эти два компонента.

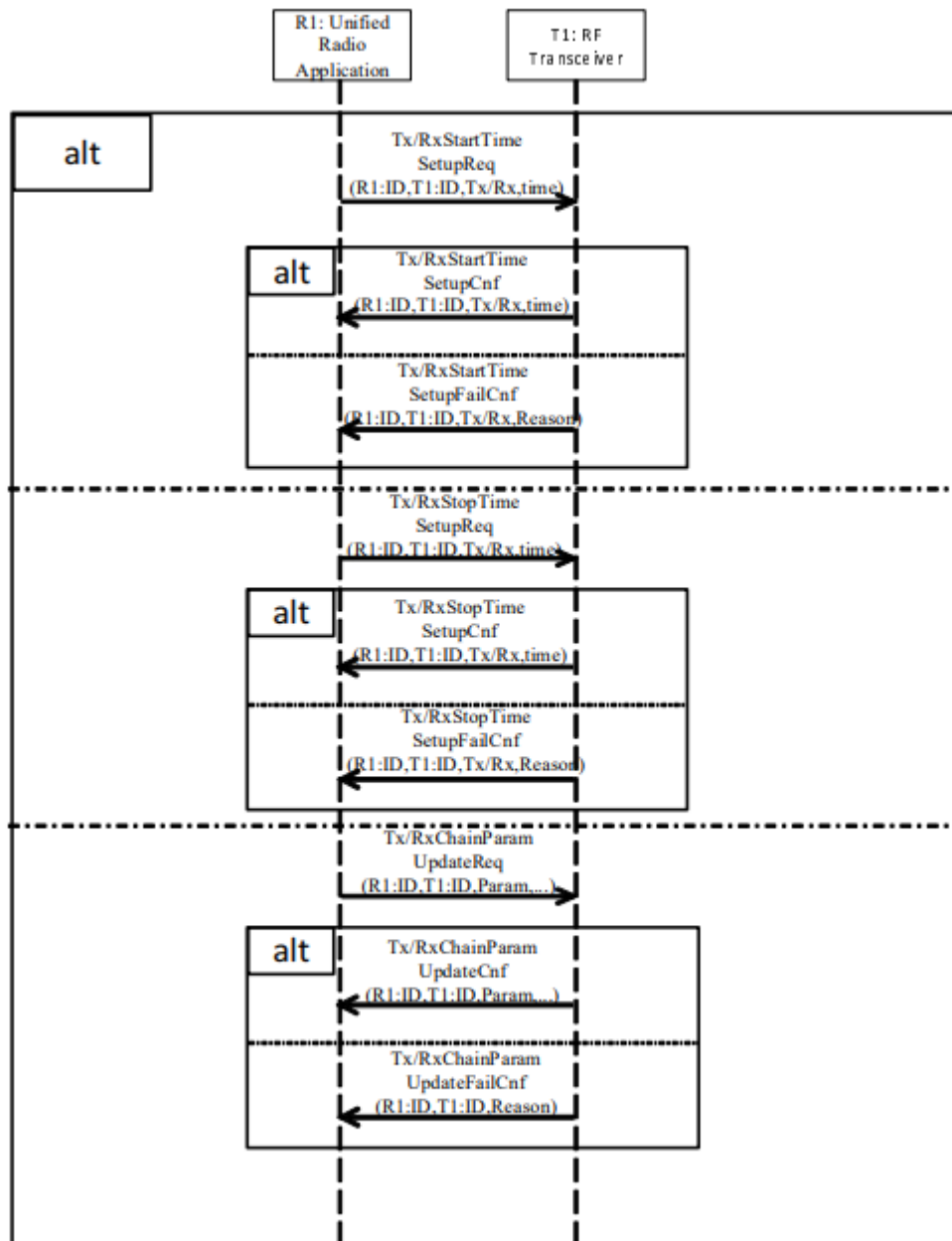


Рисунок 12 – Сценарии взаимодействия службы управления потоками приема/передачи

Сообщения, которыми обмениваются два компонента радиокomпьютера включают в себя запросы от URA к радиоустройству на установку старта или остановки времени приема/передачи, запрос на обновление параметров потока приема/передачи, а также ответы от радиоустройства к URA оповещающие об успешном или неуспешном выполнении операции.

3.4. Используемые инструменты разработки

3.4.1 CLion IDE (Integrated development environment)

Для реализации программ с большой информационной моделью, множеством количеством классов и компонент появляется необходимость использовать среду разработки, которая упростит задачу написания кода, компиляции и связи всех компонент проекта.

CLion — умная интегрированная среда разработки, созданная и поддерживаемая компанией JetBrains и предназначенная для разработки на языках C и C++ на платформах Linux, Windows и macOS. Включает в себя поддержку системы сборки кроссплатформенных проектов CMake, Gradle C++ и compilation database, возможности анализа кода на лету, разнообразные рефакторинги и унифицированный интерфейс для работы с самыми популярными системами контроля версий. На рисунке 13 представлен интерфейс этой программы. [11]

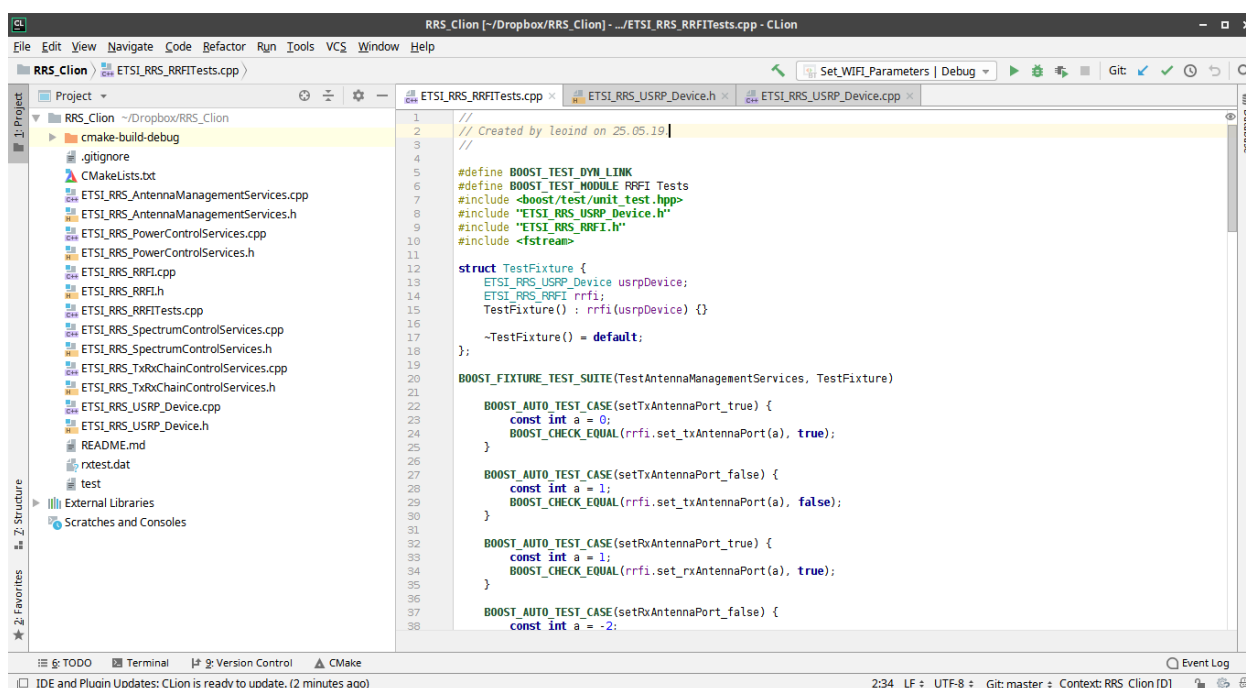


Рисунок 13 – Интерфейс программы CLion

Программная модель интерфейса RRFI реализована на языке C++11, утвержденным в 2011 году международной организацией по стандартизации

(ISO). Это компилируемый, статически типизированный язык программирования общего назначения и на данный момент является одним из самых популярных языков.

3.4.2 Библиотека BOOST

BOOST — собрание библиотек классов, использующих функциональность языка C++ и предоставляющих удобный кроссплатформенный высокоуровневый интерфейс для лаконичного кодирования различных повседневных подзадач программирования. Проект был создан после принятия стандарта C++, когда многие были недовольны отсутствием некоторых библиотек в STL (Standard Template Library). Проект является своего рода «испытательным полигоном» для различных расширений языка и части библиотек, которые являются кандидатами на включение в следующий стандарт C++. [12]

Из полезных функций этой библиотеки можно отметить поддержку линейной алгебры, удобную генерацию псевдослучайных чисел, работа с текстом (регулярные выражение, формат текста), алгоритмы для графов, многопоточность.

3.4.3 Оборудование USRP B210

Плата USRP B210 представляет собой полностью интегрированное универсальное периферийное устройство программного радио (USRP – Universal Software Radio Peripheral), которое покрывает частоты радиоспектра от 70 МГц до 6 ГГц, имеет программируемую FPGA SPARTAN6, и работает по протоколу USB 3.0. Эта платформа позволяет проводить эксперименты с широким спектром сигналов, включая радио и телевизионное вещание, сотовую связь, Wi-Fi и многими другими. Платы USRP B210 имеет 2 канала на прием и 2 каналы на передачу, включает интерфейс ввода/вывода общего назначения, и поддерживает электропитание от внешнего источника. Она может работать с полосой пропускания до 56 МГц в режиме использования

одного канала на прием и одного на передачу и до 30,72 МГц в режиме использования двух каналов на прием и двух каналов на передачу. [13] На рисунке 14 представлен внешний вид этой платы

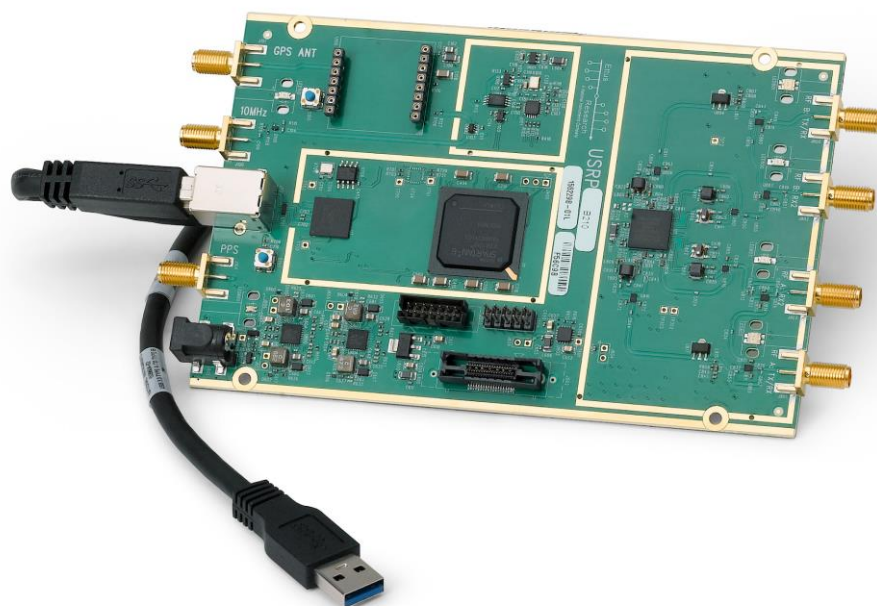


Рисунок 14 – USRP B210

3.4.4 Драйвер UHD (USRP Hardware Driver)

Для подключения платы к компьютеру и возможности с ней взаимодействовать необходима программная составляющая обеспечивающая эти функции. UHD — это кроссплатформенный драйвер с открытым исходным кодом, который может работать в Windows, Linux и MacOS. Он предоставляет общий API, который используется различными программными фреймворками, например, GNU Radio. [14]

Благодаря открытому исходному коду и поддержке API пользователи могут сотрудничать с активным сообществом энтузиастов, студентов и

профессионалов, которые для разработки и экспериментов используют продукты USRP. Являясь членами этого сообщества, пользователи могут находить помощь в разработке приложений, обмениваться знаниями для дальнейшего развития технологии SDR или вносить свои собственные инновации.

3.5 Программная реализация

Информационная модель интерфейса RRFI построена на основе нескольких классов: оборудования USRP, служб RRFI и самого интерфейса. Проект был создан в IDE CLion и компилируется с помощью системы сборки CMake, написан на языке C++ с использованием библиотеки BOOST и драйвера UHD. Также проект имеет файл, который является исполняемым в проекте, для тестирования всех функций и классов интерфейса.

Ниже будут представлены некоторые функции служб интерфейса RRFI, а также результаты работы тестов для этих функций.

3.5.1 Функции класса SpectrumControlServices

Этот класс должен содержать функции, описанные в разделе 3.3.1, которые подразумевают настройку параметров радиоспектра, таких как несущая частота, полоса пропускания и частота дискретизации. Класс содержит заголовочный файл и файл с исходным кодом функций. Все функции можно подразделить на 2 части: установки параметров (setters) и их получения (getters).

Функции установки параметров (set):

- set_rxCenterFrequency
- set_txCenterFrequency
- set_rxBandwidth
- set_txBandwidth
- set_rxSamplingRate
- set_txSamplingRate

Каждая функция возвращает значение типа `bool` и принимает в себя актуальное значение параметра типа `double` и номер канала типа `int` для которого настраивается параметр. На рисунке 15 показан пример исходного кода одной из функций этого класса.

```
bool ETSI_RRS_SpectrumControlServices::set_rxBandwidth(double actualrxBandwidth, int channel) {
    if (this->usrpDevice->min_rx_bandwidth > actualrxBandwidth) {
        this->usrpDevice->usrp->set_rx_bandwidth(actualrxBandwidth, size_t(channel));
        cout << "rx_bandwidth is less than min value, rx_bandwidth is min value" << endl;
        return false;
    } else {
        if (this->usrpDevice->max_rx_bandwidth < actualrxBandwidth) {
            this->usrpDevice->usrp->set_rx_bandwidth(actualrxBandwidth, size_t(channel));
            cout << "rx_bandwidth is larger than max value, rx_bandwidth is max value" << endl;
            return false;
        } else {
            this->usrpDevice->usrp->set_rx_bandwidth(actualrxBandwidth, size_t(channel));
            cout << "rx_bandwidth changed successfully" << endl;
            return true;
        }
    }
}
```

Рисунок 15 – Исходный код функции `set_rxBandwidth`

Каждая функция из категории установки параметров построена одинаково. Внутри тела функции находится ветвление, которое проверяет вхождение устанавливаемого значения в рамки минимального и максимального значения для каждого параметра. В случае если устанавливаемый параметр входит в эти рамки, то функция возвращает значение `true`, в ином случае устанавливается минимальное или максимальное число для параметра и функция возвращает значение `false`, а также уведомляет об этом пользователя. Устанавливаемый параметр заносится предопределенной функцией из UHD API в объект устройства USRP и хранится в нём же.

Функции получения параметров (`get`):

- `get_rxCenterFrequency`
- `get_rxCenterFrequency`

- `get_rxBandwidth`
- `get_txBandwidth`
- `get_rxSamplingRate`
- `get_txSaplingRate`

Каждая функция возвращает значение параметра типа `double`, которое хранится в объекте устройства USRP и принимает значение номера канала типа `int`, для которого запрашивается параметр. На рисунке 16 показан пример исходного кода одной из функций.

```
double ETSI_RRS_SpectrumControlServices::get_rxBandwidth(int channel) {
    return this->usrpDevice->usrp->get_rx_bandwidth((size_t)channel);
}
```

Рисунок 16 – Исходный код функции `get_rxBandwidth`

Каждая функция из категории получения параметров построена одинаково и содержит в себе операцию получения значения для определенного канала из памяти объекта устройства.

Полный исходный код класса `SpectrumControlServices` представлен в приложении А.

3.5.2 Функции класса `PowerControlServices`

Этот класс должен содержать функции, описанные в разделе 3.3.2, которые подразумевает настройку параметров мощности излучения антенны, таких как максимальная мощность излучения, мощность передающей антенны и усиление принимающей антенны. Класс содержит заголовочный файл и файл с исходным кодом функций. Все функции также, как и в прошлом классе можно подразделить на 2 части: установки параметров (setters) и их получения (getters).

Функции установки параметров:

- `set_maxTxPowerLevel`

- set_txPowerlevel
- set_rxGain

Каждая функция возвращает значение типа bool и принимает в себя актуальное значение параметра типа double, а также номер канала типа int для функций установки актуальной мощности излучения и усиления сигнала. На рисунке 17 показан пример исходного кода функции установки мощности излучения передающей антенны этого класса.

```
bool ETSI_RRS_PowerControlServices::set_txPowerLevel(double actualTxGain, int channel) {
    if (this->usrpDevice->min_tx_gain > actualTxGain) {
        usrpDevice->usrp->set_tx_gain(actualTxGain, size_t(channel));
        cout << "tx_powerlevel is less than min value, tx_powerlevel is min value" << endl;
        return false;
    } else {
        if (this->usrpDevice->max_tx_gain < actualTxGain) {
            usrpDevice->usrp->set_tx_gain(actualTxGain, size_t(channel));
            cout << "tx_powerlevel is larger than max value, tx_powerlevel is max value" << endl;
            return false;
        } else {
            usrpDevice->usrp->set_tx_gain(actualTxGain, size_t(channel));
            cout << "tx_powerlevel changed successful" << endl;
            return true;
        }
    }
}
```

Рисунок 17 – Исходный код функции set_txPowerLevel

Так как оборудование USRP не поддерживает возможность установки значения мощности эта функция работает с усилением передающей антенны. В ней содержится ветвление, которое проверяет вхождение в рамки минимального и максимального значения для параметра. В случае если устанавливаемый параметр входит в эти рамки, то функция возвращает значение true, в ином случае устанавливается минимальное или максимальное число для параметра и функция возвращает значение false, а также уведомляет об этом пользователя. Устанавливаемый параметр заносится предопределенной функцией из UHD API в объект устройства USRP и хранится в нём же. Функция установки значения усиления сигнала для принимающей антенны построена похожим образом.

Функция установки максимального значения мощности передающей антенны также работает с установкой максимального уровня усиления передающего сигнала и заносится в объект класса для работы с устройством USRP.

Функции получения параметров (get):

- `get_maxTxPowerLevel`
- `get_txPowerlevel`
- `get_rxGain`

Каждая функция возвращает значение параметра типа `double`, которое хранится в объекте устройства USRP либо в объекте класса для работы с устройством, также, функции получения актуальных параметров принимают значение номера канала типа `int`, для которого запрашивается параметр. Построение функций идентично функциям из класса `SpectrumControlServices` описанных в разделе 3.5.1.

Полный исходный код класса `PowerControlServices` представлен в приложении Б.

3.5.3 Функции класса `AntennaManagementServices`

Этот класс должен содержать функции, описанные в разделе 3.3.3, которые подразумевают выбор конкретной антенны для приема или передачи. Класс содержит заголовочный файл и файл с исходным кодом функций. Класс содержит заголовочный файл и файл с исходным кодом функций. Функции также как и в прошлом классе можно подразделить на 2 части: установки параметров (setters) и их получения (getters). Также из-за того, что оборудование USRP поддерживает указание антенны только в текстовом виде, в классе представлены вспомогательные функции, выполняющие декодирование числового значения в текстовую строку, которую можно передать на устройство USRP.

Функции декодирования числового значения:

- `decode_tx_port_num`

- decode_rx_port_num

Эти функции принимают числовое значение типа `int` и возвращают текстовое значение типа `string` определенного вида. Внутри функции имеется проверка числового значения и если оно не сопоставимо ни с одной текстовой строкой, то возвращается пустая строка, которая впоследствии не воспримется функцией UHD API.

Функции установки значений (set):

- set_txAntennaPort
- set_rxAntennaPort

Каждая функция возвращает значение типа `bool` и принимает значение типа `int`, а также номер канала типа `int`. На рисунке 18 показан пример исходного кода функции выбора передающей антенны.

```
bool ETSI_RRS_AntennaManagementServices::set_txAntennaPort(int actualTxAntennaPort, int channel) {

    this->txAntennaPort = decode_tx_port_num(actualTxAntennaPort);
    if (this->txAntennaPort != "") {
        cout << format("Setting TX Antenna: %s") % this->txAntennaPort << endl;
        usrpDevice->tx_ant = this->txAntennaPort;
        usrpDevice->usrp->set_tx_antenna(this->txAntennaPort, size_t(channel));
        cout << format("Actual TX Antenna: %s, on channel %s") % usrpDevice->usrp->get_tx_antenna(size_t(channel)) % channel << endl;
        return true;
    } else {
        cout << "error" << endl;
        return false;
    }
}
```

Рисунок 18 – Исходный код функции set_txAntennaPort

Вначале эта функция декодирует полученное числовое значение для выбора порта антенны в строку, которая потом используется для передачи в предопределенную функцию UHD API для установки нужной антенны. В случае если было получено значение пустой строки функция возвращает значение `false` и уведомляет пользователя об ошибке, а если операция прошла успешно, то функция возвращает значение `true`. Функция выбора принимающей антенны построена аналогичным образом.

Функции получения значения (get):

- get_txAntennaPort
- get_rxAntennaPort

Эти функции возвращают значение типа string, которое содержит в себе текстовое значение установленного значения антенны, а также принимает значение канала типа int, для которого поступает запрос. На рисунке 19 представлен исходный код функции получения значения передающей антенны.

```
string ETSI_RRS_AntennaManagementServices::get_txAntennaPort(int channel) {  
  
    if (this->txAntennaPort == usrpDevice->usrp->get_tx_antenna(size_t(channel))) {  
        return txAntennaPort;  
    } else {  
        cout << "error" << endl;  
    }  
  
}
```

Рисунок 19 – Исходный код функции get_txAntennaPort

В этой функции проверяется значение, установленное в объекте класса этой службы и значение которое хранится в объекте самого устройства. Если они совпадают, то функция возвращает значение антенны из класса службы, в ином случае функция уведомляет пользователя об ошибке. Функция получения значения принимающей антенны построена аналогичным образом.

Полный исходный код класса AntennaManagementServices представлен в приложении В.

3.5.4 Функции класса TxRxChainControlServices

Этот класс должен содержать функции начала и конца времени передачи и приема данных, а также функции на запрос изменения параметров. UHD API не имеет возможность устанавливать непосредственно время передачи и приема данных, в том числе также имеется необходимость для использования пользователем самих функций приема и передачи. Поэтому было принято решение реализации функций приема и передачи из файла или буфера данных и в файл или буфер данных соответственно. Эти функции подразумевают, что

в случае их вызова будет начата процедура передачи или приема, а затем будет остановлена по истечению переданных или принятых кадров. Класс содержит заголовочный файл и файл с исходным кодом функций.

Функции приема и передачи в буфер и из него:

- tx_from_buff
- rx_to_buff

Функция передачи данных из буфера возвращает значение типа bool, которое извещает об успешности выполнения функции передачи, и принимает массив данных типа short вместе с количеством кадров типа size_t, которые необходимо передать.

Функция приема данных в буфер возвращает принятый массив данных типа short и принимает значения размера буфера типа size_t, времени установки соединения типа float и количество запрашиваемых кадров типа int.

На рисунке 20 представлен исходный код функции передачи данных из буфера.

```
bool ETSI_RRS_TxRxChainControlServices::tx_from_buff(std::vector<short> buff, size_t samps_per_buff) {
    usrpDevice->tx_samps_per_buff = samps_per_buff;
    cout << "Sending bytes..." << endl;
    if (usrpDevice->tx_stream->send(&buff.front(), samps_per_buff, usrpDevice->tx_md) != samps_per_buff) {
        return false;
    }
    else {
        cout << "\e[1m" << "Bytes was sent successfully" << "\e[0m" << endl;
        return true;
    }
}
```

Рисунок 20 – Исходный код функции tx_from_buff

Функции передачи и приема подразумевают работу с объектами потоков передачи или приема, которые находятся в объекте класса работы с устройством USRP. В этой функции применяется сравнение переданных кадров определенной функцией UHD API и значения количества кадров, которые передавались в функцию до этого. Если они оказываются не равны,

то функция возвращает значение `false`, в ином случае функция возвращает значение `true` и уведомляет пользователя об успешном выполнении операции.

Функции приема и передачи в файл и из него:

- `tx_from_file`
- `rx_to_file`

Функция передачи данных из файла возвращает значение типа `bool`, которое извещает об успешности выполнения операции и принимает значение типа `string` с путем до файла, а также количество кадров для буфера через который будут отправляться данные. Эта функция работает подобным образом, как функция передачи из буфера, за исключением того, что она работает с файлом и данные из файла сначала передаются в буфер, который впоследствии передается в предопределенную функцию передачи UHD API.

Функция приема данных в файл возвращает значение типа `void` и принимает те же значения, что функция приема данных в буфер, а также значение типа `string` с путем до файла назначения. Эта функция также работает подобным образом, как и функция приема данных в буфер, за исключением того, что данные которые принимаются в буфер сразу же записываются в файл до тех пор, пока не будет достигнуто значение количества запрашиваемых кадров.

Все функции приема и передачи также поддерживают возможность преждевременного окончания операции приема или передачи аппаратным сигналом о прерывании процесса.

Функции изменения параметров потоков приема и передачи:

- `change_tx_stream_args`
- `change_rx_stream_args`

Эти функции подразумевают возможность изменения некоторых параметров потока приема и передачи, таких как тип данных, которые передаются или принимаются, а также значение квадратурной модуляции. Они изменяют параметры аргументов для потоков, которые хранятся в объекте класса работы с устройством USRP.

Полный исходный код класса TxRxChainControlServices представлен в приложении Г.

3.5.5 Функции класса RRFI

Этот класс агрегирует в себе все функции классов, описанных ранее в разделах 3.5.1-3.5.4. Это сделано для удобства взаимодействия программиста с интерфейсом, потому что не требуется обращаться к определенному классу для вызова какой-либо из функций. Класс содержит заголовочный файл и файл с исходным кодом функций.

Названия функций этого класса идентичны названиям из описанных выше классов. Каждая из них вызывает функцию определенного класса и принимает необходимые значения. Также некоторые из них содержат значения по умолчанию для упрощения задания некоторых параметров.

Конструктор этого класса создает объекты каждого из классов служб и передает в них объект класса для работы с устройством USRP, который будет описан в разделе 3.5.6.

Полный исходный код класса RRFI представлен в приложении Д.

3.5.6 Класс USRP_Device

Этот класс не описан в информационной модели интерфейса RRFI, но он необходим для установления соединения, начальной настройки и работы с устройством USRP. Он содержит все используемые в работе других классов значения, такие как максимальные и значения параметров, названия антенн и т.д., а также объекты, например, потоки передачи и приема данных, их параметры и т.д.

Интерес в этом классе представляет конструктор, который устанавливает соединение с устройством USRP, создает его объект, затем проводит установку параметров по умолчанию и получает границы максимальных и минимальных значений радиопараметров. Параллельно вся информация о выполненных операциях и полученных значениях выводится для пользователя в консоль.

Работа с интерфейсом RRFI представляет собой установку параметров на определенном канале с значением по умолчанию 0, но класс USRP_Device предоставляет возможность изменить это значение для работы с другим каналом устройства. Функция `changeChannel` возвращает значение `void` и выполняет практически те же операции, которые выполняются в конструкторе за исключением того, что все они направлены на канал, значение типа `size_t` которого принимается функцией.

Полный исходный код класса USRP_Device представлен в приложении Е.

3.5.7 Файл RRFI_Tests

Программа не содержит исполняемой функции `main`, поскольку подразумевается, что функции интерфейса будут вызываться из программ к которым он будет подключен. Поэтому входная точка программы находится в этом файле, который является исполняемым и содержит тесты, которые покрывают все функции интерфейса RRFI и проверяют возможность установки радиопараметров определенного протокола связи, например, Bluetooth или WiFi. В нём используется фреймворк для тестирования из библиотеки BOOST, который позволяет проводить тесты путём сравнения заданных значений и тех, которые возвращают тестируемые функции.

Исходный код подразделен на наборы (*suites*) и отдельные случаи (*cases*) различных тестов. Наборы тестируют функции какого-либо определенного класса, а случаи — определенные ситуации, например, установку радиопараметров протокола связи.

Для работы с тестами необходимо создание объектов класса работы с устройством USRP_Device и класса интерфейса RRFI для работы с функциями. Библиотека BOOST не поддерживает создание объектов в самих тестах, но имеет возможность создание структуры, в которой хранятся эти объекты, называемой *fixture*. Эта структура, которая представлена на рисунке

21, описывается в начале файла и содержит конструктор создающий необходимые объекты.

```
struct TestFixture {  
    ETSI_RRS_USRP_Device usrpDevice;  
    ETSI_RRS_RRFI rrfi;  
    TestFixture() : rrfi(usrpDevice) {}  
  
    ~TestFixture() = default;  
};
```

Рисунок 21 – Структура TestFixture

Затем в файле представлены тесты, которые обрабатывают все функции интерфейса с различными значениями, вызывающими разное поведение функций. Каждый случай и набор тестов содержит название и структуру fixture, которая была упомянута выше. Каждый набор тестов начинается с макроса BOOST_FIXTURE_TEST_CASE, который указывает на то, что начинается описание набора тестов с использованием fixture, затем в нём объявляются макросы BOOST_AUTO_TEST_CASE, содержащими название тестового случая и указывающими на то, что рассматривается отдельный тестовый случай. В нем устанавливаются необходимые проверяемые значения и используется макрос BOOST_CHECK_EQUAL, который проводит непосредственно проверку. Этот макрос не подразумевает аварийного завершения автоматизируемого тестирования в случае, если значения не сходятся, но выводит предупреждение и продолжает дальнейшую работы. В фреймворке тестирования библиотеки BOOST в том числе имеются макросы, которые поддерживают эту возможность.

На рисунке 22 представлена часть исходного кода набора тестов класса SpectrumControlServices в котором проверяются 4 значения установки параметра несущей частоты сигнала.

```

BOOST_FIXTURE_TEST_SUITE(TestSpectrumControlServices, TestFixture)

BOOST_AUTO_TEST_CASE(setRxCenterFrequency) {
    double a = usrpDevice.min_rx_frequency;
    BOOST_CHECK_EQUAL(rrfi.set_rxCenterFrequency(a), true);
    a = a - 1;
    BOOST_CHECK_EQUAL(rrfi.set_rxCenterFrequency(a), false);
    a = usrpDevice.max_rx_frequency;
    BOOST_CHECK_EQUAL(rrfi.set_rxCenterFrequency(a), true);
    a = a + 1;
    BOOST_CHECK_EQUAL(rrfi.set_rxCenterFrequency(a), false);
}

```

Рисунок 22 – Исходный код тестового случая setRxCenterFrequency

В каждой проверке этого случая тестирования подставляются значения для определения поведения функции. Например, сперва проверяется граничное минимальное значение несущей частоты при котором функция должна возвращать значение true, затем это значение уменьшается на единицу, которое в итоге становится меньше значения границы, следовательно, функция должна возвращать значение false. Подобным образом сделана проверка максимального значения параметра несущей частоты. Наборы тестов для остальных классов реализованы идентично, за исключением тестирования класса TxRxChainControlServices. В нём проверяется только успешность выполнения функций передачи, так как для функций приема невозможно предопределить данные, которые будут получены из радиоспектра.

На рисунке 23 представлен исходный код тестового случая установки радиопараметров протокола связи WiFi.


```
BOOST_FIXTURE_TEST_CASE(Set_WIFI_Parameters, TestFixture) {
    const double freq = 2400000000;
    const double bw = 20000000;
    const double sr = 2000000;
    const double gain = 60;

    BOOST_CHECK_EQUAL(rrfi.set_txAntennaPort(0), true);
    BOOST_CHECK_EQUAL(rrfi.set_txCenterFrequency(freq), true);
    BOOST_CHECK_EQUAL(rrfi.get_txCenterFrequency(), freq);
    BOOST_CHECK_EQUAL(rrfi.set_txBandwidth(bw), true);
    BOOST_CHECK_EQUAL(rrfi.get_txBandwidth(), bw);
    BOOST_CHECK_EQUAL(rrfi.set_txSamplingRate(sr), true);
    BOOST_CHECK_EQUAL(rrfi.get_txSamplingRate(), sr);
    BOOST_CHECK_EQUAL(rrfi.set_txPowerLevel(gain), true);
    BOOST_CHECK_EQUAL(rrfi.get_txPowerLevel(), gain);

    BOOST_CHECK_EQUAL(rrfi.set_rxAntennaPort(1), true);
    BOOST_CHECK_EQUAL(rrfi.set_rxCenterFrequency(freq), true);
    BOOST_CHECK_EQUAL(rrfi.get_rxCenterFrequency(), freq);
    BOOST_CHECK_EQUAL(rrfi.set_rxBandwidth(bw), true);
    BOOST_CHECK_EQUAL(rrfi.get_rxBandwidth(), bw);
    BOOST_CHECK_EQUAL(rrfi.set_rxSamplingRate(sr), true);
    BOOST_CHECK_EQUAL(rrfi.get_rxSamplingRate(), sr);
    BOOST_CHECK_EQUAL(rrfi.set_rxGain(gain), true);
    BOOST_CHECK_EQUAL(rrfi.get_rxGain(), gain);

}
```

Рисунок 23 – Исходный код тестового случая Set_WIFI_Parameters

В этом случае проверяется возможность установки радиопараметров для приема и передачи:

- Несущая частота — 2,4 ГГц
- Полоса пропускания — 20 МГц
- Частота дискретизации — 2 МГц
- Усиление сигнала — 60 дБ

Все функции должны возвращать значение true, так как работа с реконфигурируемым мобильным устройством должна подразумевать

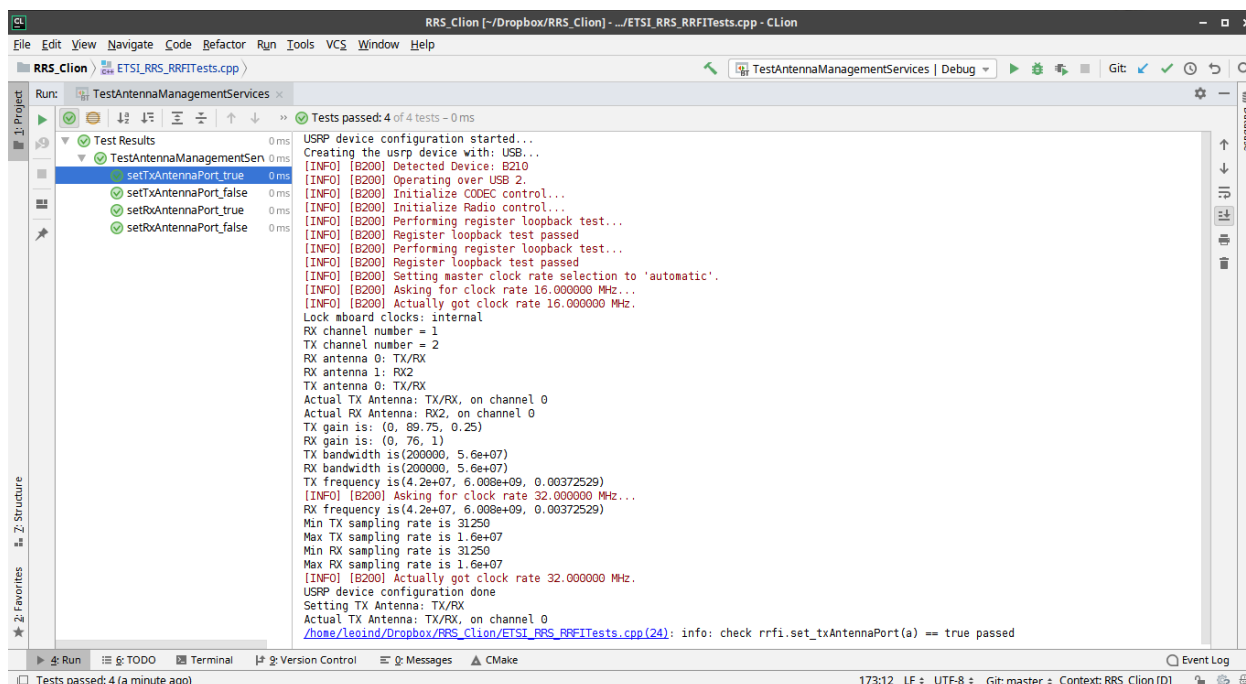
возможность использования параметров современных протоколов связи. Также каждое установленное значение проверяется на соответствие с заданным, чтобы удостовериться, что все значения установлены корректно.

Полный исходный код файла `RRFI_Tests`, текстовый вывод работы этого файла представлен в приложении Ж.

3.6 Результаты работы программы

Работоспособность программы проверяется запуском наборов тестов из файла `RRFI_Tests`, который был описан в разделе 3.5.7. Ниже будут представлены результаты работы тестов для некоторых тестовых случаев.

На рисунке 24 представлен результат работы набора тестов класса `AntennaManagementServices` и подробный результат работы положительного теста выбора передающей антенны.



```
RRS_Client [~/Dropbox/RRS_Client] - .../ETSI_RRS_RRFITests.cpp - CClient
File Edit View Navigate Code Refactor Run Tools VCS Window Help
RRS_Client > ETSI_RRS_RRFITests.cpp
Run: TestAntennaManagementServices
Test Results
TestAntennaManagementServices
  setTxAntennaPort_true 0 ms
  setTxAntennaPort_false 0 ms
  setRxAntennaPort_true 0 ms
  setRxAntennaPort_false 0 ms
Tests passed: 4 of 4 tests - 0 ms
USRP device configuration started...
Creating the usrp device with: USB...
[INFO] [B200] Detected Device: B210
[INFO] [B200] Operating over USB 2.
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Setting master clock rate selection to 'automatic'.
[INFO] [B200] Asking for clock rate 16.000000 MHz...
[INFO] [B200] Actually got clock rate 16.000000 MHz.
Lock mboard clocks: internal
RX channel number = 1
TX channel number = 2
RX antenna 0: TX/RX
RX antenna 1: RX2
TX antenna 0: TX/RX
Actual TX Antenna: TX/RX, on channel 0
Actual RX Antenna: RX2, on channel 0
TX gain is: (0, 89.75, 0.25)
RX gain is: (0, 76, 1)
TX bandwidth is(200000, 5.6e+07)
RX bandwidth is(200000, 5.6e+07)
TX frequency is(4.2e+07, 6.008e+09, 0.00372529)
[INFO] [B200] Asking for clock rate 32.000000 MHz...
RX frequency is(4.2e+07, 6.008e+09, 0.00372529)
Min TX sampling rate is 31250
Max TX sampling rate is 1.6e+07
Min RX sampling rate is 31250
Max RX sampling rate is 1.6e+07
[INFO] [B200] Actually got clock rate 32.000000 MHz.
USRP device configuration done
Setting TX Antenna: TX/RX
Actual TX Antenna: TX/RX, on channel 0
/home/leopard/Dropbox/RRS_Client/ETSI_RRS_RRFITests.cpp(24): info: check rrfi.set_txAntennaPort(a) == true passed
Tests passed: 4 (a minute ago) 173:12 LF : UTF-8 : Git: master : Context: RRS_Client [D]
```

Рисунок 24 – Результат работы теста `setTxAntennaPort_true`

В левой части рисунка расположены все тестовые случаи набора, которые помечаются зеленым цветом, если тесты пройдены положительно и красным, если тесты оказались не пройденными. В центре рисунка представлен консольный вывод, который содержит всю информацию, выводимую программой, большую часть которой составляет информация

установки соединения и первоначальной настройки устройства USRP. Самой последней строчкой вывода является информация о том, что тест успешно пройден.

На рисунке 25 представлен результат работы набора тестов класса `SpectrumControlServices` и подробный результат работы тестового случая установки несущей частоты.

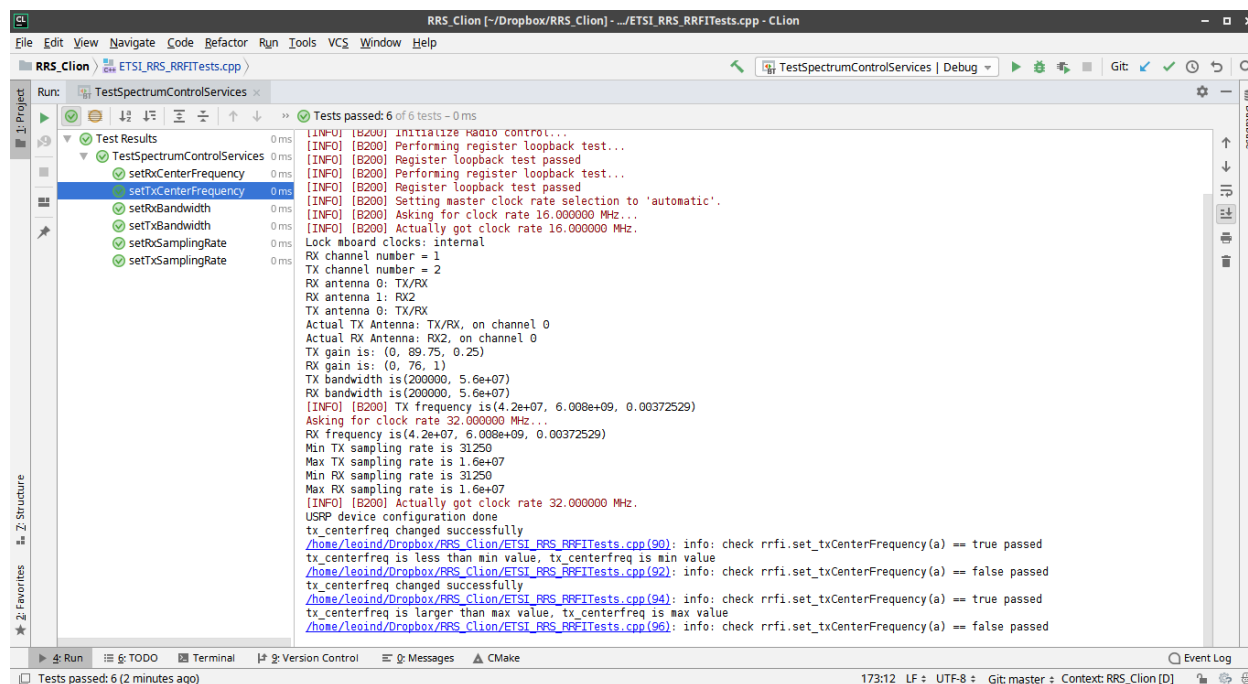


Рисунок 25 – Результат работы теста `setTxCenterFrequency`

В левой части рисунка также, как и в прошлом результате расположена информация обо всех случаях набора тестов. В центре рисунка расположен консольный вывод, содержащий информацию о первоначальной настройке устройства USRP и результаты каждой проверки, где указано, что все они пройдены успешно.

На рисунке 26 представлен результат работы тестового случая установки параметров протокола связи WiFi.

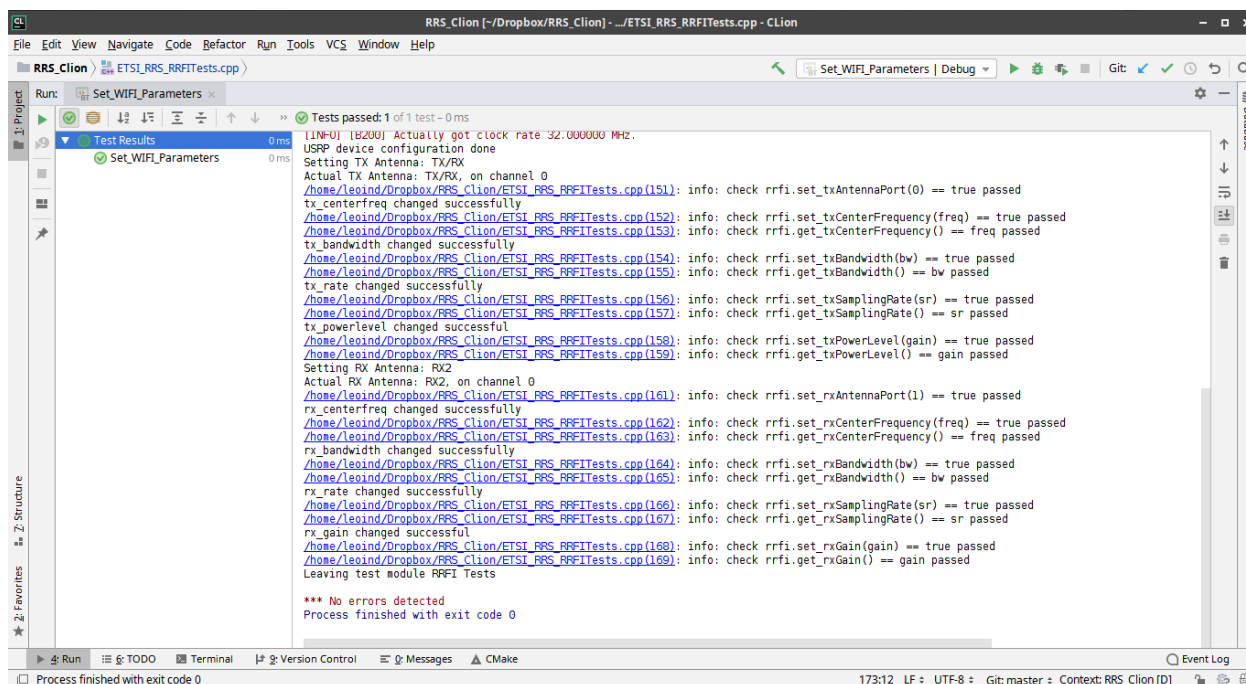


Рисунок 26 – Результат работы теста Set_WIFI_Parameters

В центральной части рисунка расположен консольный вывод работы теста содержащий информацию о результате всех проверок значений параметров, описанных в разделе 3.5.7. На этой основе можно сделать вывод о том, что все радиопараметры протокола WiFi установлены верно, следовательно, реконфигурируемое устройство может работать с этим протоколом.

В конце консольного вывода работы каждого из тестов расположена строка, оповещающая пользователя о количестве ошибок, возникших при тестировании. В случае если их не произошло выводится сообщение “No errors detected”, в иной ситуации указан тест, в котором произошла ошибка.

Также реализован аварийный выход из программы в случае, который показан на рисунке 27, когда устройство USRP не подключено или имеется проблема в установлении соединения с ним. В этой ситуации выводится сообщение о необходимости корректно подсоединить устройство.

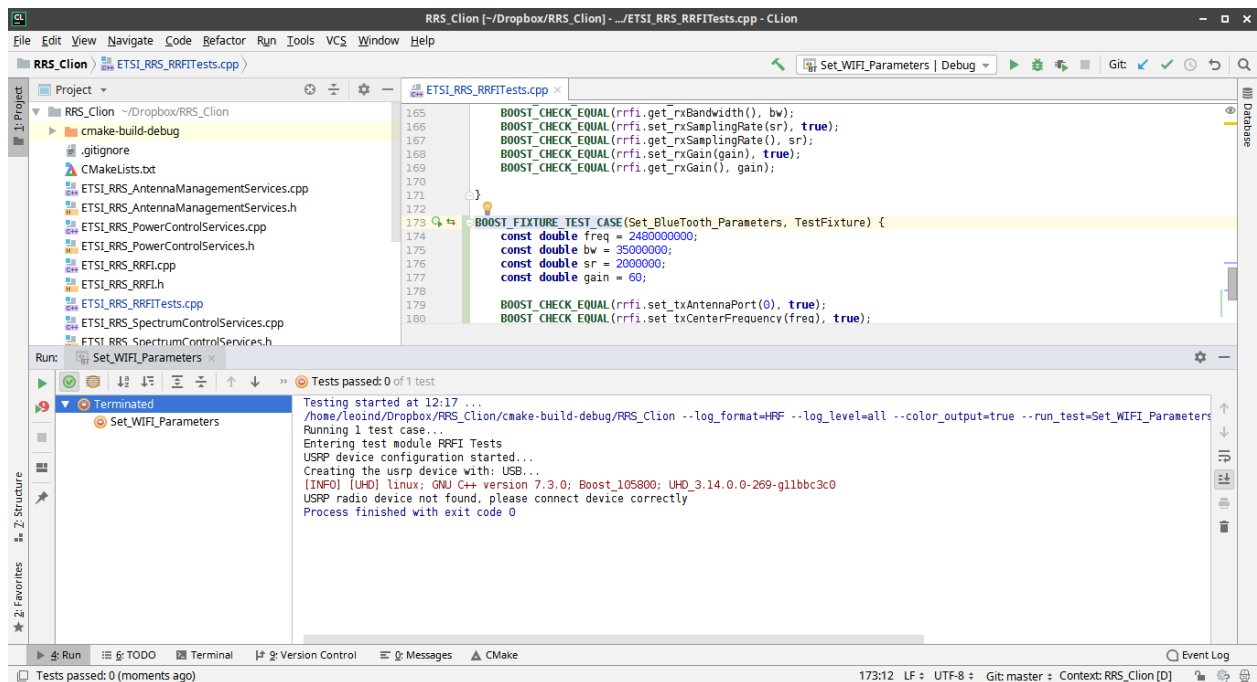


Рисунок 27 – Результат работы при неподключенном устройстве USRP

3.7 Руководство программиста-разработчика

Для последующей разработки программной модели остальных компонент радиоперсонального компьютера с использованием интерфейса RRFI необходимо встроить его в разрабатываемое приложение. Ниже будут указаны действия, которое необходимо выполнить программисту, для успешного подключения интерфейса, а также информация о сборке программы и возможности ее дополнения.

Для работы интерфейса необходимо подключить к проекту два заголовочных файла: ETSI_RRS_USRP_Device.h и ETSI_RRS_RRFI.h. При создании объекта класса ETSI_RRS_USRP_Device обеспечится подключение к устройству USRP и будет выполнена его первоначальная настройка. Объект самого устройства находится в этом классе и имеет название usrpDevice. Через него можно обращаться непосредственно к необходимым функциям UHD API. При создании объекта класса ETSI_RRS_RRFI в его конструктор необходимо передать указатель на объект класса работы с устройством USRP, который был создан ранее. После этих действий можно использовать все функции

интерфейса RRFI, обращаясь к созданному объекту и через символ точки к нужному функционалу.

Программная модель интерфейса имплементирована в среде разработки CLion, которая использует систему сборки проектов CMake, позволяющую автоматически скомпилировать сразу все файлы и подключить необходимые библиотеки. CMake для конфигурации использует файл CMakeLists.txt, в котором содержатся данные о файлах для сборки, библиотеках, наличии тестов, использовании версии языка программирования и о других параметрах. Этот файл содержит четкую структуру построения, при нарушении которой компиляция будет происходить с ошибкой.

В проекте интерфейса RRFI функционал системы CMake сначала проверяет наличие необходимых библиотек BOOST и UHD API, выводит информацию о них, затем применяет необходимые флаги для компиляции, добавляет компилируемые файлы и по итогу, если библиотеки были найдены, привязывает их к проекту. Минимальная требуемая версия системы сборки CMake — 3.5, версия библиотеки BOOST — 1.58.0, требуемые компоненты библиотеки — system, filesystem, unit_test_framework. Также в файле присутствует директива enable_testing(), указывающая на то, что в проекте присутствуют элементы тестирования.

Исходный код данного проекта является открытым, поэтому впоследствии может дополняться сторонними разработчиками. К данному интерфейсу может прибавляться новый функционал, не рассмотренный в информационной модели интерфейса. К нему обязательно должны быть разработаны и применены тесты, которые будут покрывать весь новый исходный код. Также в проекте может модифицироваться уже присутствующий функционал, но тесты должны продолжать проходить успешно. В том числе приветствуется добавление новых тестов и проверочных данных для будущего развития интерфейса. Все данные о классах, представленных в интерфейсе описаны в разделе 3.5.

3.8 Руководство пользователя

Разработанный интерфейс не является самостоятельным автономным приложением и подразумевает его использование в разработке компонент реконфигурируемых систем и других приложений. Но для пользователя может представлять интерес проверка работоспособности интерфейса, т.е. запуска необходимых тестов. Ожидается, что он имеет базовые навыки программирования и работы с интегрированными системами разработки. Ниже будут описаны действия, которые нужно выполнить пользователю для работы с тестами.

Загрузив исходный код пользователю необходимо импортировать проект в IDE CLion кнопкой Import Project. Затем установив курсор в теле нужного набора тестов необходимо нажать комбинацию клавиш Ctrl+Shift+F10 для автоматического создания конфигурации запуска. Откроется окно настройки конфигурации в котором можно ее название и параметры запуска. Если используется библиотека BOOST версии ниже 1.62.0, то для корректной работы тестов необходимо добавить параметры `--log_format=HRF --log_level=all`. После этого нажать кнопку Apply, а затем вызвать конфигурацию через меню Run.

3.9 Дальнейшая разработка и применение интерфейса

Разработанный программный модуль имеет перспективу дальнейшего развития. С дополнением спецификации интерфейса могут добавляться новые функции, открывающие другие возможности работы с реконфигурируемыми устройствами. Также этот интерфейс планируется применять при дальнейшей разработке общей реконфигурируемой радиосистемы для использования в комплексе с другими компонентами, в частности для построения модели полезной нагрузки для коммуникационных спутников.

ЗАКЛЮЧЕНИЕ

В данной работе было проведено исследование программного интерфейса, работающего с радиоспектром в реконфигурируемых мобильных устройствах, а также возможность его имплементации для будущего встраивания в радиосистемы. В том числе были рассмотрены уже существующие технологии программного радио.

На основе информационной модели радиокomпьютера была построена модель интерфейса, которая включила в себя необходимые функции для настройки радиотрансивера.

При имплементации интерфейса были использованы возможности современных языков программирования, библиотек и оборудования. В том числе были разработаны тесты, проверяющие возможность использования тех или иных протоколов связи, что привело к результатам, которые дают понять, что интерфейс с оборудованием с которым он работает, можно успешно применять для использования и дальнейшей разработки реконфигурируемых систем.

Интерфейс позволяет использовать реконфигурируемое оборудование USRP для его настройки и дальнейшей работы с ним, упрощает взаимодействие с оборудованием и количество строк кода, которое необходимо написать программисту для этой цели. Интерфейс имеет возможность расширения и встраивания в любые приложения.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Силин Алексей. «Технология Software Defined Radio. Теория, принципы и примеры аппаратных платформ» – Беспроводные Технологии, № 7, 2007, 22-27 с.;
2. What is Software Defined Radio
URL: <https://www.wirelessinnovation.org/assets/documents/SoftwareDefinedRadio.pdf>;
3. SDRF Cognitive Radio Definitions / SDRF-06-R-0011-V1.0.0, 2007;
4. Andrew Feickert. «The Joint Tactical Radio System (JTRS) and the Army's Future Combat System (FCS): Issues for Congress», Library of Congress, Congressional Research Service, 2005;
5. SCA Platform Abstraction
URL: <https://nordiasoft.com/knowledge-center/sca/platform-abstraction>;
6. Software Communication Architecture Specification. User's Guide/ Joint Tactical Networking Center, 2015. 9-11 с.;
7. By Richard C. Reinhart, Sandra K. Johnson, Thomas J. Kacpura, Charles S. Hall, Carl R. Smith, John Liebetreu. «Open Architecture Standard for NASA's Software-Defined Space Telecommunications Radio Systems», Proceedings of the IEEE, Vol. 95 Issue 10 Oct. 2007, 1986-1993 с.;
8. DigRF Dual-Mode 2.5G / 3G Baseband / RF IC Interface standard // DigRF Working Group. V.3.09. 2006. 17-19 с.;
9. Reconfigurable Radio Systems (RRS). Radio Reconfiguration related Architecture for Mobile Devices / ETSI EN 303 095 V1.3.1., 2017. 11-13 с.;
10. Reconfigurable Radio Systems (RRS). Mobile Device Information Models and Protocols. Part 2: Reconfigurable Radio Frequency Interface (RRFI) / ETSI EN 303 146-2 V1.1.5, 2016. 10-28 с.;

11. CLion IDE
URL: <https://www.jetbrains.com/clion/>
12. BOOST Libraries
URL: <https://www.boost.org/>
13. USRP B210
URL: http://files.ettus.com/manual/page_usrp_b200.html
14. UHD API
URL: http://files.ettus.com/manual/page_uhd.html

ПРИЛОЖЕНИЕ А

Файл .h

```
//
// Spectrum Control Services header
//

#ifndef RRS_CLION_SPECTRUMCONTROLSERVICES_H
#define RRS_CLION_SPECTRUMCONTROLSERVICES_H

#include <iostream>
#include <string>
#include "ETSI_RRS_USRP_Device.h"

using namespace std;

class ETSI_RRS_SpectrumControlServices {

private:
    ETSI_RRS_USRP_Device* usrpDevice;
    string chainParameters;

public:
    ETSI_RRS_SpectrumControlServices(ETSI_RRS_USRP_Device&);
    bool set_rxCenterFrequency(double, int);
    bool set_txCenterFrequency(double, int);
    double get_rxCenterFrequency(int);
    double get_txCenterFrequency(int);
    bool set_rxBandwidth(double, int);
    bool set_txBandwidth(double, int);
    double get_rxBandwidth(int);
    double get_txBandwidth(int);
    bool set_rxSamplingRate(double, int);
    bool set_txSamplingRate(double, int);
    double get_rxSamplingRate(int);
    double get_txSamplingRate(int);
    ~ETSI_RRS_SpectrumControlServices();

};

#endif //RRS_CLION_SPECTRUMCONTROLSERVICES_H
```

Файл .cpp

```
//
// Spectrum Control Services implementation
//

#include "ETSI_RRS_SpectrumControlServices.h"

using namespace std;

ETSI_RRS_SpectrumControlServices::ETSI_RRS_SpectrumControlServices(ETSI_RRS_USRP_Device &actualusrpDevice) {
    usrpDevice = &actualusrpDevice;
}
```

```

bool ETSI_RRS_SpectrumControlServices::set_rxCenterFrequency(double
actualrxCenterFrequency, int channel) {

    if (this->usrpDevice->min_rx_frequency > actualrxCenterFrequency) {
        usrpDevice->usrp-
>set_rx_freq(uhd::tune_request_t(actualrxCenterFrequency), size_t(channel));
        cout << "rx_centerfreq is less than min value, rx_centerfreq is min
value" << endl;
        return false;
    } else {
        if (this->usrpDevice->max_rx_frequency < actualrxCenterFrequency) {
            usrpDevice->usrp-
>set_rx_freq(uhd::tune_request_t(actualrxCenterFrequency), size_t(channel));
            cout << "rx_centerfreq is larger than max value, rx_centerfreq is
max value" << endl;
            return false;
        } else {
            usrpDevice->usrp-
>set_rx_freq(uhd::tune_request_t(actualrxCenterFrequency), size_t(channel));
            cout << "rx_centerfreq changed successfully" << endl;
            return true;
        }
    }
}

bool ETSI_RRS_SpectrumControlServices::set_txCenterFrequency(double
actualtxCenterFrequency, int channel) {
    if (this->usrpDevice->min_tx_frequency > actualtxCenterFrequency) {
        usrpDevice->usrp-
>set_tx_freq(uhd::tune_request_t(actualtxCenterFrequency), size_t(channel));
        cout << "tx_centerfreq is less than min value, tx_centerfreq is min
value" << endl;
        return false;
    } else {
        if (this->usrpDevice->max_tx_frequency < actualtxCenterFrequency) {
            usrpDevice->usrp-
>set_tx_freq(uhd::tune_request_t(actualtxCenterFrequency), size_t(channel));
            cout << "tx_centerfreq is larger than max value, tx_centerfreq is
max value" << endl;
            return false;
        } else {
            usrpDevice->usrp-
>set_tx_freq(uhd::tune_request_t(actualtxCenterFrequency), size_t(channel));
            cout << "tx_centerfreq changed successfully" << endl;
            return true;
        }
    }
}

bool ETSI_RRS_SpectrumControlServices::set_rxBandwidth(double
actualrxBandwidth, int channel) {
    if (this->usrpDevice->min_rx_bandwidth > actualrxBandwidth) {
        this->usrpDevice->usrp->set_rx_bandwidth(actualrxBandwidth,
size_t(channel));
        cout << "rx_bandwidth is less than min value, rx_bandwidth is min
value" << endl;
        return false;
    } else {
        if (this->usrpDevice->max_rx_bandwidth < actualrxBandwidth) {
            this->usrpDevice->usrp->set_rx_bandwidth(actualrxBandwidth,
size_t(channel));
            cout << "rx_bandwidth is larger than max value, rx_bandwidth is
max value" << endl;

```

```

        return false;
    } else {
        this->usrpDevice->usrp->set_rx_bandwidth(actualrxBandwidth,
size_t(channel));
        cout << "rx_bandwidth changed successfully" << endl;
        return true;
    }
}

}

bool ETSI_RRS_SpectrumControlServices::set_txBandwidth(double
actualtxBandwidth, int channel) {
    if (this->usrpDevice->min_tx_bandwidth > actualtxBandwidth) {
        this->usrpDevice->usrp->set_tx_bandwidth(actualtxBandwidth,
size_t(channel));
        cout << "tx_bandwidth is less than min value, tx_bandwidth is min
value" << endl;
        return false;
    } else {
        if (this->usrpDevice->max_tx_bandwidth < actualtxBandwidth) {
            this->usrpDevice->usrp->set_tx_bandwidth(actualtxBandwidth,
size_t(channel));
            cout << "tx_bandwidth is larger than max value, tx_bandwidth is
max value" << endl;
            return false;
        } else {
            this->usrpDevice->usrp->set_tx_bandwidth(actualtxBandwidth,
size_t(channel));
            cout << "tx_bandwidth changed successfully" << endl;
            return true;
        }
    }
}

bool ETSI_RRS_SpectrumControlServices::set_rxSamplingRate(double
actualrxRate, int channel) {
    if (this->usrpDevice->min_rx_rate > actualrxRate) {
        this->usrpDevice->usrp->set_rx_rate(actualrxRate, size_t(channel));
        cout << "rx_rate is less than min value, rx_rate is min value" <<
endl;
        return false;
    } else {
        if (this->usrpDevice->max_rx_rate < actualrxRate) {
            this->usrpDevice->usrp->set_rx_rate(actualrxRate,
size_t(channel));
            cout << "rx_rate is larger than max value, rx_rate is max value"
<< endl;
            return false;
        } else {
            this->usrpDevice->usrp->set_rx_rate(actualrxRate,
size_t(channel));
            cout << "rx_rate changed successfully" << endl;
            return true;
        }
    }
}

bool ETSI_RRS_SpectrumControlServices::set_txSamplingRate(double
actualtxRate, int channel) {
    if (this->usrpDevice->min_tx_rate > actualtxRate) {
        this->usrpDevice->usrp->set_tx_rate(actualtxRate, size_t(channel));
        cout << "tx_rate is less than min value, tx_rate is min value" <<
endl;

```

```

        return false;
    } else {
        if (this->usrpDevice->max_tx_rate < actualtxRate) {
            this->usrpDevice->usrp->set_tx_rate(actualtxRate,
size_t(channel));
            cout << "tx_rate is larger than max value, tx_rate is max value"
<< endl;
            return false;
        } else {
            this->usrpDevice->usrp->set_tx_rate(actualtxRate,
size_t(channel));
            cout << "tx_rate changed successfully" << endl;
            return true;
        }
    }
}

double ETSI_RRS_SpectrumControlServices::get_rxCenterFrequency(int channel) {
    return this->usrpDevice->usrp->get_rx_freq((size_t)channel);
}

double ETSI_RRS_SpectrumControlServices::get_txCenterFrequency(int channel) {
    return this->usrpDevice->usrp->get_tx_freq((size_t)channel);
}

double ETSI_RRS_SpectrumControlServices::get_rxBandwidth(int channel) {
    return this->usrpDevice->usrp->get_rx_bandwidth((size_t)channel);
}

double ETSI_RRS_SpectrumControlServices::get_txBandwidth(int channel) {
    return this->usrpDevice->usrp->get_tx_bandwidth((size_t)channel);
}

double ETSI_RRS_SpectrumControlServices::get_rxSamplingRate(int channel) {
    return this->usrpDevice->usrp->get_rx_rate((size_t)channel);
}

double ETSI_RRS_SpectrumControlServices::get_txSamplingRate(int channel) {
    return this->usrpDevice->usrp->get_tx_rate((size_t)channel);
}

ETSI_RRS_SpectrumControlServices::~ETSI_RRS_SpectrumControlServices() {
}

```

ПРИЛОЖЕНИЕ Б

Файл .h

```
//
// Power Control Services header
//

#ifndef RRS_CLION_POWERCONTROLSERVICES_H
#define RRS_CLION_POWERCONTROLSERVICES_H

#include "ETSI_RRS_USRP_Device.h"

using namespace std;

class ETSI_RRS_PowerControlServices {

private:
    ETSI_RRS_USRP_Device *usrpDevice;

public:
    ETSI_RRS_PowerControlServices(ETSI_RRS_USRP_Device&);
    bool set_maxTxPowerLevel(double);
    bool set_txPowerLevel(double, int);
    bool set_rxGain(double, int);
    double get_maxTxPowerLevel();
    double get_txPowerLevel(int);
    double get_rxGain(int);
    ~ETSI_RRS_PowerControlServices();

};

#endif //RRS_CLION_POWERCONTROLSERVICES_H
```

Файл .cpp

```
//
// Power Control Services implementation
//

#include <iostream>
#include "ETSI_RRS_PowerControlServices.h"

using namespace std;

ETSI_RRS_PowerControlServices::ETSI_RRS_PowerControlServices(ETSI_RRS_USRP_De
vice &actualusrpDevice) {
    usrpDevice = &actualusrpDevice;
}

bool ETSI_RRS_PowerControlServices::set_maxTxPowerLevel(double
actualMaxTxPowerLevel) {
    usrpDevice->max_tx_gain = actualMaxTxPowerLevel;
    cout << "Max gain: " << usrpDevice->max_tx_gain << endl;
    return true;
}

bool ETSI_RRS_PowerControlServices::set_txPowerLevel(double actualTxGain, int
channel) {
    if (this->usrpDevice->min_tx_gain > actualTxGain) {
```

```

        usrpDevice->usrp->set_tx_gain(actualTxGain, size_t(channel));
        cout << "tx_powerlevel is less than min value, tx_powerlevel is min
value" << endl;
        return false;
    } else {
        if (this->usrpDevice->max_tx_gain < actualTxGain) {
            usrpDevice->usrp->set_tx_gain(actualTxGain, size_t(channel));
            cout << "tx_powerlevel is larger than max value, tx_powerlevel is
max value" << endl;
            return false;
        } else {
            usrpDevice->usrp->set_tx_gain(actualTxGain, size_t(channel));
            cout << "tx_powerlevel changed successful" << endl;
            return true;
        }
    }
}

bool ETSI_RRS_PowerControlServices::set_rxGain(double actualRxGain, int
channel) {
    if (this->usrpDevice->min_rx_gain > actualRxGain) {
        usrpDevice->usrp->set_rx_gain(actualRxGain, size_t(channel));
        cout << "rx_gain is less than min value, rx_gain is min value" <<
endl;
        return false;
    } else {
        if (this->usrpDevice->max_rx_gain < actualRxGain) {
            usrpDevice->usrp->set_rx_gain(actualRxGain, size_t(channel));
            cout << "rx_gain is larger than max value, rx_gain is max value"
<< endl;
            return false;
        } else {
            usrpDevice->usrp->set_rx_gain(actualRxGain, size_t(channel));
            cout << "rx_gain changed successful" << endl;
            return true;
        }
    }
}

double ETSI_RRS_PowerControlServices::get_maxTxPowerLevel() {
    return usrpDevice->max_tx_gain;
}

double ETSI_RRS_PowerControlServices::get_txPowerLevel(int channel) {
    return usrpDevice->usrp->get_tx_gain(size_t(channel));
}

double ETSI_RRS_PowerControlServices::get_rxGain(int channel) {
    return usrpDevice->usrp->get_rx_gain(size_t(channel));
}

ETSI_RRS_PowerControlServices::~ETSI_RRS_PowerControlServices() {
}

```


ПРИЛОЖЕНИЕ В

Файл .h

```
//
// Antenna Management Services header
//

#ifndef RRS_CLION_ANTENNAMANAGEMENTSERVICES_H
#define RRS_CLION_ANTENNAMANAGEMENTSERVICES_H

#include "ETSI_RRS_USRP_Device.h"

using namespace std;

class ETSI_RRS_AntennaManagementServices {

private:
    string txAntennaPort;
    string rxAntennaPort;
    ETSI_RRS_USRP_Device *usrpDevice;

public:
    ETSI_RRS_AntennaManagementServices(ETSI_RRS_USRP_Device&);
    bool set_txAntennaPort(int, int);
    string get_txAntennaPort(int);
    bool set_rxAntennaPort(int, int);
    string get_rxAntennaPort(int);
    string decode_tx_port_num(int); //decode to string name of antenna
    string decode_rx_port_num(int); //decode to string name of antenna
    ~ETSI_RRS_AntennaManagementServices();

};

#endif //RRS_CLION_ANTENNAMANAGEMENTSERVICES_H
```

Файл .cpp

```
//
// Antenna Management Services implementation
//

#include <iostream>
#include "ETSI_RRS_AntennaManagementServices.h"
#include "ETSI_RRS_USRP_Device.h"

using namespace std;
using namespace boost;

ETSI_RRS_AntennaManagementServices::ETSI_RRS_AntennaManagementServices(ETSI_RRS_USRP_Device &actualusrpDevice) {
    usrpDevice = &actualusrpDevice;
}

bool ETSI_RRS_AntennaManagementServices::set_txAntennaPort(int actualTxAntennaPort, int channel) {

    this->txAntennaPort = decode_tx_port_num(actualTxAntennaPort);
    if (this->txAntennaPort != "") {
        cout << format("Setting TX Antenna: %s") % this->txAntennaPort << endl;
    }
}
```

```

        usrpDevice->tx_ant = this->txAntennaPort;
        usrpDevice->usrp->set_tx_antenna(this->txAntennaPort,
size_t(channel));
        cout << format("Actual TX Antenna: %s, on channel %s") % usrpDevice-
>usrp->get_tx_antenna(size_t(channel)) % channel << endl;
        return true;
    } else {
        cout << "error" << endl;
        return false;
    }
}

string ETSI_RRS_AntennaManagementServices::get_txAntennaPort(int channel) {

    if (this->txAntennaPort == usrpDevice->usrp-
>get_tx_antenna(size_t(channel))) {
        return txAntennaPort;
    } else {
        cout << "error" << endl;
    }

}

bool ETSI_RRS_AntennaManagementServices::set_rxAntennaPort(int
actualRxAntennaPort, int channel) {

    this->rxAntennaPort = decode_rx_port_num(actualRxAntennaPort);
    if (this->rxAntennaPort != "") {
        cout << format("Setting RX Antenna: %s") % this->rxAntennaPort <<
endl;
        usrpDevice->rx_ant = this->rxAntennaPort;
        usrpDevice->usrp->set_rx_antenna(this->rxAntennaPort,
size_t(channel));
        cout << format("Actual RX Antenna: %s, on channel %s") % usrpDevice-
>usrp->get_rx_antenna(size_t(channel)) % channel << endl;
        return true;
    } else {
        cout << "error" << endl;
        return false;
    }

}

string ETSI_RRS_AntennaManagementServices::get_rxAntennaPort(int channel) {

    if (this->rxAntennaPort == usrpDevice->usrp-
>get_rx_antenna(size_t(channel))) {
        return rxAntennaPort;
    } else {
        cout << "Error" << endl;
    }

}

string ETSI_RRS_AntennaManagementServices::decode_tx_port_num(int num) {

    if (num == 0) {
        return "TX/RX";
    } else {
        return "";
    }

}

```

```

string ETSI_RRS_AntennaManagementServices::decode_rx_port_num(int num) {
    if (num == 0) {
        return "TX/RX";
    }
    else if (num == 1) {
        return "RX2";
    } else {
        return "";
    }
}

ETSI_RRS_AntennaManagementServices::~ETSI_RRS_AntennaManagementServices() {
}

```

ПРИЛОЖЕНИЕ Г

Файл .h

```
//
// Tx/Rx Chain Control Services header
//

#ifndef RRS_CLION_TXRXCHAINCONTROLSERVICES_H
#define RRS_CLION_TXRXCHAINCONTROLSERVICES_H

#include "ETSI_RRS_USRP_Device.h"

using namespace std;

static bool stop_signal_called = false;

class ETSI_RRS_TxRxChainControlServices {
private:
    ETSI_RRS_USRP_Device *usrpDevice;

    bool txStartTime;
    bool txStopTime;
    bool rxStartTime;
    bool rxStopTime;

public:
    ETSI_RRS_TxRxChainControlServices(ETSI_RRS_USRP_Device&);
    //TX and RX from file functions
    bool tx_from_buff(std::vector<short>, size_t);
    bool tx_from_file(string, size_t);
    void change_tx_stream_args(string, string); //fc64 - complex<double> fc32
- complex<float>
    void change_rx_stream_args(string, string); //sc16 - complex<int16_t> sc8
- complex<int8_t>
    std::vector<short> rx_to_buff(size_t, float, int);
    void rx_to_file(string, size_t, float, int);
    ~ETSI_RRS_TxRxChainControlServices();

};

#endif //RRS_CLION_TXRXCHAINCONTROLSERVICES_H
```

Файл .cpp

```
//
// Tx/Rx Chain Control Services implementation
//

#include <iostream>
#include <fstream>
#include <uhd/usrp/multi_usrp.hpp>
#include "ETSI_RRS_TxRxChainControlServices.h"

using namespace std;

void sig_int_handler(int){stop_signal_called = true;}

ETSI_RRS_TxRxChainControlServices::ETSI_RRS_TxRxChainControlServices(ETSI_RRS_USRP_Device &actualusrpdevice) {
```

```

        usrpDevice = &actualusrpdevice;
    }

bool ETSI_RRS_TxRxChainControlServices::tx_from_buff(std::vector<short> buff,
size_t samps_per_buff) {
    usrpDevice->tx_samps_per_buff = samps_per_buff;
    cout << "Sending bytes..." << endl;
    if (usrpDevice->tx_stream->send(&buff.front(), samps_per_buff,
usrpDevice->tx_md) != samps_per_buff) {
        return false;
    }
    else {
        cout << "\e[1m" << "Bytes was sent successfully" << "\e[0m" << endl;
        return true;
    }
}

bool ETSI_RRS_TxRxChainControlServices::tx_from_file(string actualfilename,
size_t samps_per_buff) {
    cout << "Sending file" << endl;
    usrpDevice->tx_samps_per_buff = samps_per_buff;
    usrpDevice->tx_filename = actualfilename;
    ifstream infile(usrpDevice->tx_filename.c_str(), ifstream::binary);
    if (!infile) {
        cout << "Cannot open file" << endl;
        return false;
    }
    std::vector<short> buff(usrpDevice->tx_samps_per_buff);
    usrpDevice->tx_md.end_of_burst = infile.eof();

    while(not usrpDevice->tx_md.end_of_burst and not stop_signal_called){

        infile.read((char*)&buff.front(), buff.size()*sizeof(short));
        size_t num_tx_samps = infile.gcount()/sizeof(short);

        usrpDevice->tx_md.end_of_burst = infile.eof();

        usrpDevice->tx_stream->send(&buff.front(), num_tx_samps, usrpDevice-
>tx_md);
        cout << ".";
    }
    infile.close();
    cout << "\e[1m" << "\nFile was sent successfully" << "\e[0m" << endl;
    return true;
}

void ETSI_RRS_TxRxChainControlServices::change_tx_stream_args(string
cpu_format, string otw_format) {
    usrpDevice->tx_stream_args.cpu_format = cpu_format;
    usrpDevice->tx_stream_args.otw_format = otw_format;
}

void ETSI_RRS_TxRxChainControlServices::change_rx_stream_args(string
cpu_format, string otw_format) {
    usrpDevice->rx_stream_args.cpu_format = cpu_format;
    usrpDevice->rx_stream_args.otw_format = otw_format;
}

std::vector<short> ETSI_RRS_TxRxChainControlServices::rx_to_buff(size_t
samps_per_buff, float settling_time, int num_requested_samps) {
    usrpDevice->rx_samps_per_buff = samps_per_buff;
    float timeout = settling_time + 0.1;
    int num_total_samps = 0;

```

```

        bool overflow_message = true;
        uhd::stream_cmd_t stream_cmd((num_requested_samps == 0)?

uhd::stream_cmd_t::STREAM_MODE_START_CONTINUOUS:

uhd::stream_cmd_t::STREAM_MODE_NUM_SAMPS_AND_DONE
    );
    stream_cmd.num_samps = num_requested_samps;
    stream_cmd.stream_now = false;
    stream_cmd.time_spec = uhd::time_spec_t(settling_time);
    usrpDevice->rx_stream->issue_stream_cmd(stream_cmd);

    std::vector<short> buff(usrpDevice->rx_samps_per_buff);

    size_t num_rx_samps = usrpDevice->rx_stream->recv(&buff.front(),
buff.size(), usrpDevice->rx_md, timeout);
    timeout = 0.1; //small timeout for subsequent recv
    cout << "Received samps = " << num_rx_samps << endl;

    if (usrpDevice->rx_md.error_code ==
uhd::rx_metadata_t::ERROR_CODE_TIMEOUT) {
        std::cout << boost::format("Timeout while streaming") << std::endl;
        return buff;
    }
    if (usrpDevice->rx_md.error_code ==
uhd::rx_metadata_t::ERROR_CODE_OVERFLOW) {
        if (overflow_message) {
            overflow_message = false;
            std::cerr << boost::format(
                "Got an overflow indication. Please consider the following:\n"
                "  Your write medium must sustain a rate of %fMB/s.\n"
                "  Dropped samples will not be written to the file.\n"
                "  Please modify this example for your purposes.\n"
                "  This message will not appear again.\n"
            ); //% (usrpDevice->usrp->get_rx_rate()*sizeof((short)/1e6);
        }
    }
    if (usrpDevice->rx_md.error_code != uhd::rx_metadata_t::ERROR_CODE_NONE) {
        throw std::runtime_error(str(boost::format(
            "Receiver error %s"
        ) % usrpDevice->rx_md.strerror()));
    }

    return buff;
}

void ETSI_RRS_TxRxChainControlServices::rx_to_file(string actualfilename,
size_t samps_per_buff, float settling_time, int num_requested_samps) {
    usrpDevice->rx_samps_per_buff = samps_per_buff;
    usrpDevice->rx_filename = actualfilename;
    float timeout = settling_time + 0.1;
    int num_total_samps = 0;
    std::ofstream outfile(usrpDevice->rx_filename.c_str(),
std::ofstream::binary);
    bool overflow_message = true;
    uhd::stream_cmd_t stream_cmd((num_requested_samps == 0)?

uhd::stream_cmd_t::STREAM_MODE_START_CONTINUOUS:

uhd::stream_cmd_t::STREAM_MODE_NUM_SAMPS_AND_DONE
    );
    stream_cmd.num_samps = num_requested_samps;
    stream_cmd.stream_now = false;

```

```

        stream_cmd.time_spec = uhd::time_spec_t(settling_time);
        usrpDevice->rx_stream->issue_stream_cmd(stream_cmd);

        std::vector<short> buff(usrpDevice->rx_samps_per_buff);

        while(not stop_signal_called and (num_requested_samps != num_total_samps
or num_requested_samps == 0)){
            size_t num_rx_samps = usrpDevice->rx_stream->recv(&buff.front(),
buff.size(), usrpDevice->rx_md, timeout);
            timeout = 0.1; //small timeout for subsequent recv

            if (usrpDevice->rx_md.error_code ==
uhd::rx_metadata_t::ERROR_CODE_TIMEOUT) {
                std::cout << boost::format("Timeout while streaming") <<
std::endl;
                return;
            }
            if (usrpDevice->rx_md.error_code ==
uhd::rx_metadata_t::ERROR_CODE_OVERFLOW) {
                if (overflow_message) {
                    overflow_message = false;
                    std::cerr << boost::format(
                        "Got an overflow indication. Please consider the
following:\n"
                        "  Your write medium must sustain a rate of
%fMB/s.\n"
                        "  Dropped samples will not be written to the
file.\n"
                        "  Please modify this example for your purposes.\n"
                        "  This message will not appear again.\n"
                    ); // % (usrpDevice->usrp->get_rx_rate()*sizeof((short)/1e6);
                }
                continue;
            }
            if (usrpDevice->rx_md.error_code !=
uhd::rx_metadata_t::ERROR_CODE_NONE) {
                throw std::runtime_error(str(boost::format(
                    "Receiver error %s"
                ) % usrpDevice->rx_md.strerror()));
            }

            num_total_samps += num_rx_samps;

            outfile.write((const char*)&buff.front(),
num_rx_samps*sizeof(short));
        }

        outfile.close();
        return;
    }

    ETSI_RRS_TxRxChainControlServices::~ETSI_RRS_TxRxChainControlServices() {
}

```

ПРИЛОЖЕНИЕ Д

Файл .h

```
//
// General ETSI_RRS_RRFI header
//

#ifndef RRS_CLION_RRFI_H
#define RRS_CLION_RRFI_H

#include "ETSI_RRS_SpectrumControlServices.h"
#include "ETSI_RRS_AntennaManagementServices.h"
#include "ETSI_RRS_PowerControlServices.h"
#include "ETSI_RRS_TxRxChainControlServices.h"
#include "ETSI_RRS_USRP_Device.h"

using namespace std;

class ETSI_RRS_RRFI {

private:
    ETSI_RRS_SpectrumControlServices *spectrumControlServices;
    ETSI_RRS_AntennaManagementServices *antennaManagementServices;
    ETSI_RRS_PowerControlServices *powerControlServices;
    ETSI_RRS_TxRxChainControlServices *txRxChainControlServices;
    ETSI_RRS_USRP_Device* usrpDevice;
    int channel = 0;
    //ETSI_RRS_USRP_Device usrpDevice;

public:
    ETSI_RRS_RRFI(ETSI_RRS_USRP_Device&);

    void changeChannel(int);

    //Set parameters functions
    //Antenna management Services
    bool set_txAntennaPort(int);
    bool set_rxAntennaPort(int);
    //Power Control Services
    bool set_maxTxPowerLevel(double);
    bool set_txPowerLevel(double);
    bool set_rxGain(double);
    //Spectrum Control Services
    bool set_rxCenterFrequency(double);
    bool set_txCenterFrequency(double);
    bool set_rxBandwidth(double);
    bool set_txBandwidth(double);
    bool set_rxSamplingRate(double);
    bool set_txSamplingRate(double);

    //Get parameters functions
    // Antenna management Services
    string get_txAntennaPort();
    string get_rxAntennaPort();
    //Power Control Services
    double get_maxTxPowerLevel();
    double get_txPowerLevel();
    double get_rxGain();
    //Spectrum Control Services
    double get_rxCenterFrequency();
    double get_txCenterFrequency();
    double get_rxBandwidth();
```



```

        double get_txBandwidth();
        double get_rxSamplingRate();
        double get_txSamplingRate();
        //TX and RX from file functions
        bool tx_from_file(string, size_t);
        bool tx_from_buff(std::vector<short>, size_t);
        void change_tx_stream_args(string, string); //fc64 - complex<double> fc32
- complex<float>
        void change_rx_stream_args(string, string); //sc16 - complex<int16_t> sc8
- complex<int8_t>
        std::vector<short> rx_to_buff(size_t, float, int);
        void rx_to_file(string, size_t, float, int);

        ~ETSI_RRS_RRFI();

};
#endif //RRS_CLION_RRFI_H

```

Файл .cpp

```

//
// General ETSI_RRS_RRFI Implementation
//
#include <uhd/usrp/multi_usrp.hpp>
#include <fstream>
#include "ETSI_RRS_RRFI.h"
#include "ETSI_RRS_SpectrumControlServices.h"
#include "ETSI_RRS_AntennaManagementServices.h"
#include "ETSI_RRS_PowerControlServices.h"
#include "ETSI_RRS_TxRxChainControlServices.h"

using namespace std;
using namespace boost;

ETSI_RRS_RRFI::ETSI_RRS_RRFI(ETSI_RRS_USRP_Device &actualusrpdevice) {
    spectrumControlServices = new
    ETSI_RRS_SpectrumControlServices(actualusrpdevice);
    antennaManagementServices = new
    ETSI_RRS_AntennaManagementServices(actualusrpdevice);
    powerControlServices = new
    ETSI_RRS_PowerControlServices(actualusrpdevice);
    txRxChainControlServices = new
    ETSI_RRS_TxRxChainControlServices(actualusrpdevice);
    usrpDevice = &actualusrpdevice;
}

void ETSI_RRS_RRFI::changeChannel(int actualChannel) {
    this->channel = actualChannel;
    usrpDevice->changeChannel(size_t(actualChannel));
}

//Set parameters functions implementation

bool ETSI_RRS_RRFI::set_txAntennaPort(int actualTxAntennaPort) {
    return antennaManagementServices->set_txAntennaPort(actualTxAntennaPort,
    this->channel);
}

bool ETSI_RRS_RRFI::set_rxAntennaPort(int actualRxAntennaPort) {
    return antennaManagementServices->set_rxAntennaPort(actualRxAntennaPort,
    this->channel);
}

```

```

bool ETSI_RRS_RRFI::set_maxTxPowerLevel(double actualMaxTxPowerLevel) {
    return powerControlServices->set_maxTxPowerLevel(actualMaxTxPowerLevel);
}

bool ETSI_RRS_RRFI::set_txPowerLevel(double actualTxPowerLevel) {
    return powerControlServices->set_txPowerLevel(actualTxPowerLevel, this->channel);
}

bool ETSI_RRS_RRFI::set_rxGain(double actualRxGain) {
    return powerControlServices->set_rxGain(actualRxGain, this->channel);
}

bool ETSI_RRS_RRFI::set_rxCenterFrequency(double actualRxCenterFrequency) {
    return spectrumControlServices->set_rxCenterFrequency(actualRxCenterFrequency, this->channel);
}

bool ETSI_RRS_RRFI::set_txCenterFrequency(double actualTxCenterFrequency) {
    return spectrumControlServices->set_txCenterFrequency(actualTxCenterFrequency, this->channel);
}

bool ETSI_RRS_RRFI::set_rxBandwidth(double actualRxBandwidth) {
    return spectrumControlServices->set_rxBandwidth(actualRxBandwidth, this->channel);
}

bool ETSI_RRS_RRFI::set_txBandwidth(double actualTxBandwidth) {
    return spectrumControlServices->set_txBandwidth(actualTxBandwidth, this->channel);
}

bool ETSI_RRS_RRFI::set_rxSamplingRate(double actualRxRate) {
    return spectrumControlServices->set_rxSamplingRate(actualRxRate, this->channel);
}

bool ETSI_RRS_RRFI::set_txSamplingRate(double actualTxRate) {
    return spectrumControlServices->set_txSamplingRate(actualTxRate, this->channel);
}

//Get parameters functions implementation

string ETSI_RRS_RRFI::get_txAntennaPort() {
    return antennaManagementServices->get_txAntennaPort(this->channel);
}

string ETSI_RRS_RRFI::get_rxAntennaPort() {
    return antennaManagementServices->get_rxAntennaPort(this->channel);
}

double ETSI_RRS_RRFI::get_maxTxPowerLevel() {
    return powerControlServices->get_maxTxPowerLevel();
}

double ETSI_RRS_RRFI::get_txPowerLevel() {
    return powerControlServices->get_txPowerLevel(this->channel);
}

double ETSI_RRS_RRFI::get_rxGain() {

```

```

        return powerControlServices->get_rxGain(this->channel);
    }

    double ETSI_RRS_RRFI::get_rxCenterFrequency() {
        return spectrumControlServices->get_rxCenterFrequency(this->channel);
    }

    double ETSI_RRS_RRFI::get_txCenterFrequency() {
        return spectrumControlServices->get_txCenterFrequency(this->channel);
    }

    double ETSI_RRS_RRFI::get_rxBandwidth() {
        return spectrumControlServices->get_rxBandwidth(this->channel);
    }

    double ETSI_RRS_RRFI::get_txBandwidth() {
        return spectrumControlServices->get_txBandwidth(this->channel);
    }

    double ETSI_RRS_RRFI::get_rxSamplingRate() {
        return spectrumControlServices->get_rxSamplingRate(this->channel);
    }

    double ETSI_RRS_RRFI::get_txSamplingRate() {
        return spectrumControlServices->get_txSamplingRate(this->channel);
    }

    bool ETSI_RRS_RRFI::tx_from_file(string actualfilename, size_t samps_per_buff
= 100) {
        return txRxChainControlServices->tx_from_file(actualfilename,
samps_per_buff);
    }

    bool ETSI_RRS_RRFI::tx_from_buff(std::vector<short> buff, size_t
samps_per_buff = 100) {
        return txRxChainControlServices->tx_from_buff(buff, samps_per_buff);
    }

    void ETSI_RRS_RRFI::change_tx_stream_args(string cpu_format, string
otw_format) {
        txRxChainControlServices->change_tx_stream_args(cpu_format, otw_format);
    }

    void ETSI_RRS_RRFI::change_rx_stream_args(string cpu_format, string
otw_format) {
        txRxChainControlServices->change_rx_stream_args(cpu_format, otw_format);
    }

    void ETSI_RRS_RRFI::rx_to_file(string actualfilename, size_t samps_per_buff,
float settling_time, int num_requested_samps = 0) {
        txRxChainControlServices->rx_to_file(actualfilename, samps_per_buff,
settling_time, num_requested_samps);
    }

    std::vector<short> ETSI_RRS_RRFI::rx_to_buff(size_t samps_per_buff, float
settling_time, int num_requested_samps = 0) {
        return txRxChainControlServices->rx_to_buff(samps_per_buff,
settling_time, num_requested_samps);
    }

    ETSI_RRS_RRFI::~ETSI_RRS_RRFI() {
        delete spectrumControlServices;
        delete antennaManagementServices;
    }

```

```

delete powerControlServices;
delete txRxChainControlServices;

}

```

ПРИЛОЖЕНИЕ Е

Файл .h

```

//
// Class for global USRP device parameters
//

#ifndef RRS_CLION_ETSI_RRS_USRPDEVICE_H
#define RRS_CLION_ETSI_RRS_USRPDEVICE_H
#include <uhd/usrp/multi_usrp.hpp>

using namespace std;

class ETSI_RRS_USRP_Device {
public:
    ETSI_RRS_USRP_Device(); //first config of usrp device
    void changeChannel(size_t);

    string deviceArgs = "";
    string ref = "internal";
    string subdev = "A:B";
    string rx_ant = "TX:RX";
    string tx_ant = "TX:RX";
    double rx_gain = 0;
    double tx_gain = 0;
    double max_rx_gain, min_rx_gain, max_tx_gain, min_tx_gain;
    double max_rx_bandwidth, min_rx_bandwidth, max_tx_bandwidth,
min_tx_bandwidth;
    double max_rx_frequency, min_rx_frequency, max_tx_frequency,
min_tx_frequency;
    double min_tx_rate, max_tx_rate, min_rx_rate, max_rx_rate;
    uhd::usrp::multi_usrp::sptr usrp;

    uhd::tx_streamer::sptr tx_stream;
    uhd::rx_streamer::sptr rx_stream;
    uhd::tx_metadata_t tx_md;
    uhd::rx_metadata_t rx_md;
    uhd::stream_args_t tx_stream_args;
    uhd::stream_args_t rx_stream_args;
    size_t tx_samps_per_buff;
    size_t rx_samps_per_buff;
    string tx_filename;
    string rx_filename;

};

#endif //RRS_CLION_ETSI_RRS_USRPDEVICE_H

```

Файл .cpp

```

//
// Global USRP device parameters implementation
//

#include "ETSI_RRS_USRP_Device.h"

```

```

using namespace boost;
using namespace std;

ETSI_RRS_USRP_Device::ETSI_RRS_USRP_Device() {

    cout << "\e[1m" << "USRP device configuration started..." << "\e[0m" <<
endl;

    //Creating connection with USRP device
    try {
        cout << format("Creating the usrp device with: USB...") << endl;
        usrp = uhd::usrp::multi_usrp::make(deviceArgs);
    } catch (uhd::lookup_error) {
        cout << "\e[31m" << "USRP radio device not found, please connect
device correctly" << "\e[0m" << endl;
        exit(0);
    }

    // Lock mboard clocks
    try {
        cout << format("Lock mboard clocks: %f") % ref << endl;
        usrp->set_clock_source(ref);
    } catch (uhd::value_error) {
        cout << "\e[31m" << "Source for clock is invalid, please check
clock_source option" << "\e[0m" << endl;
        exit(0);
    }

    //always select the subdevice first, the channel mapping affects the
other settings
    //cout << format("Subdevice set to: %f") % subdev << endl;
    usrp->set_rx_subdev_spec(subdev);
    //cout << format("Using Subdevice: %s") % usrp->get_pp_string() << endl;

    cout << "RX channel number = " << usrp->get_rx_num_channels() << endl;
    cout << "TX channel number = " << usrp->get_tx_num_channels() << endl;

    for(int i=0; i<usrp->get_rx_antennas().size(); ++i)
        cout << "RX antenna "<< i << ": " << usrp->get_rx_antennas()[i] <<
endl;

    for(int i=0; i<usrp->get_tx_antennas().size(); ++i)
        cout << "TX antenna "<< i << ": " << usrp->get_tx_antennas()[i] <<
endl;

    usrp->set_tx_antenna("TX/RX", size_t(0));
    cout << format("Actual TX Antenna: %s") % usrp->get_tx_antenna(size_t(0))
<< ", on channel 0" << endl;

    usrp->set_rx_antenna("RX2", size_t(0));
    cout << format("Actual RX Antenna: %s") % usrp->get_rx_antenna(size_t(0))
<< ", on channel 0" << endl;

    cout << "TX gain is: " << usrp->get_tx_gain_range().to_pp_string();
    this->min_tx_gain = usrp->get_tx_gain_range().start();
    this->max_tx_gain = usrp->get_tx_gain_range().stop();
    cout << "RX gain is: " << usrp->get_rx_gain_range().to_pp_string();
    this->min_rx_gain = usrp->get_rx_gain_range().start();
    this->max_rx_gain = usrp->get_rx_gain_range().stop();

    cout << "TX bandwidth is" << usrp-
>get_tx_bandwidth_range().to_pp_string();

```

```

    this->min_tx_bandwidth = usrp->get_tx_bandwidth_range().start();
    this->max_tx_bandwidth = usrp->get_tx_bandwidth_range().stop();
    cout << "RX bandwidth is" << usrp-
>get_rx_bandwidth_range().to_pp_string();
    this->max_rx_bandwidth = usrp->get_rx_bandwidth_range().start();
    this->max_rx_bandwidth = usrp->get_rx_bandwidth_range().stop();

    cout << "TX frequency is" << usrp->get_tx_freq_range().to_pp_string();
    this->min_tx_frequency = usrp->get_tx_freq_range().start();
    this->max_tx_frequency = usrp->get_tx_freq_range().stop();
    cout << "RX frequency is" << usrp->get_rx_freq_range().to_pp_string();
    this->min_rx_frequency = usrp->get_rx_freq_range().start();
    this->max_rx_frequency = usrp->get_rx_freq_range().stop();

    //cout << "TX Sapling rate is" << usrp->get_tx_rates().to_pp_string() <<
endl;
    this->min_tx_rate = usrp->get_tx_rates().start();
    cout << "Min TX sampling rate is " << this->min_tx_rate << endl;
    this->max_tx_rate = usrp->get_tx_rates().stop();
    cout << "Max TX sampling rate is " << this->max_tx_rate << endl;
    //cout << "RX Sapling rate is" << usrp->get_rx_rates().to_pp_string() <<
endl;
    this->min_rx_rate = usrp->get_rx_rates().start();
    cout << "Min RX sampling rate is " << this->min_rx_rate << endl;
    this->max_rx_rate = usrp->get_rx_rates().stop();
    cout << "Max RX sampling rate is " << this->max_rx_rate << endl;

    tx_stream_args.cpu_format = "sc16";
    tx_stream_args.otw_format = "sc16";
    rx_stream_args.cpu_format = "sc16";
    rx_stream_args.otw_format = "sc16";
    tx_stream = usrp->get_tx_stream(tx_stream_args);
    rx_stream = usrp->get_rx_stream(rx_stream_args);
    tx_md.start_of_burst = false;
    tx_md.end_of_burst = false;

    cout << "\e[1m" << "USRP device configuration done" << "\e[0m" << endl;
}

void ETSI_RRS_USRP_Device::changeChannel(size_t actualChannel) {
    cout << "\e[1m" << "Channel changing for " << actualChannel << " channel"
<< "\e[0m" << endl;

    usrp->set_tx_antenna("TX/RX", size_t(actualChannel));
    cout << format("Actual TX Antenna: %s") % usrp-
>get_tx_antenna(actualChannel) << endl;

    usrp->set_rx_antenna("RX2", actualChannel);
    cout << format("Actual RX Antenna: %s") % usrp-
>get_rx_antenna(actualChannel) << endl;

    cout << "TX gain is: " << usrp-
>get_tx_gain_range(actualChannel).to_pp_string();
    this->min_tx_gain = usrp->get_tx_gain_range(actualChannel).start();
    this->max_tx_gain = usrp->get_tx_gain_range(actualChannel).stop();
    cout << "RX gain is: " << usrp-
>get_rx_gain_range(actualChannel).to_pp_string();
    this->min_rx_gain = usrp->get_rx_gain_range(actualChannel).start();
    this->max_rx_gain = usrp->get_rx_gain_range(actualChannel).stop();

```

```

        cout << "TX bandwidth is" << usrp-
>get_tx_bandwidth_range(actualChannel).to_pp_string();
        this->min_tx_bandwidth = usrp-
>get_tx_bandwidth_range(actualChannel).start();
        this->max_tx_bandwidth = usrp-
>get_tx_bandwidth_range(actualChannel).stop();
        cout << "RX bandwidth is" << usrp-
>get_rx_bandwidth_range(actualChannel).to_pp_string();
        this->max_rx_bandwidth = usrp-
>get_rx_bandwidth_range(actualChannel).start();
        this->max_rx_bandwidth = usrp-
>get_rx_bandwidth_range(actualChannel).stop();

        cout << "TX frequency is" << usrp-
>get_tx_freq_range(actualChannel).to_pp_string();
        this->min_tx_frequency = usrp->get_tx_freq_range(actualChannel).start();
        this->max_tx_frequency = usrp->get_tx_freq_range(actualChannel).stop();
        cout << "RX frequency is" << usrp-
>get_rx_freq_range(actualChannel).to_pp_string();
        this->min_rx_frequency = usrp->get_rx_freq_range(actualChannel).start();
        this->max_rx_frequency = usrp->get_rx_freq_range(actualChannel).stop();
        this->min_tx_rate = usrp->get_tx_rates().start();
        cout << "Min TX sampling rate is " << this->min_tx_rate << endl;
        this->max_tx_rate = usrp->get_tx_rates().stop();
        cout << "Max TX sampling rate is " << this->max_tx_rate << endl;
        //cout << "RX Sapling rate is" << usrp->get_rx_rates().to_pp_string() <<
endl;
        this->min_rx_rate = usrp->get_rx_rates().start();
        cout << "Min RX sampling rate is " << this->min_rx_rate << endl;
        this->max_rx_rate = usrp->get_rx_rates().stop();
        cout << "Max RX sampling rate is " << this->max_rx_rate << endl;

        cout << "\e[1m" << "Channel changing done" << "\e[0m" << endl;

    }

```

ПРИЛОЖЕНИЕ Ж

```
//
// Created by leoind on 25.05.19.
//

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE RRFI Tests
#include <boost/test/unit_test.hpp>
#include "ETSI_RRS_USRP_Device.h"
#include "ETSI_RRS_RRFI.h"
#include <fstream>

struct TestFixture {
    ETSI_RRS_USRP_Device usrpDevice;
    ETSI_RRS_RRFI rrfi;
    TestFixture() : rrfi(usrpDevice) {}

    ~TestFixture() = default;
};

BOOST_FIXTURE_TEST_SUITE(TestAntennaManagementServices, TestFixture)

    BOOST_AUTO_TEST_CASE(setTxAntennaPort_true) {
        const int a = 0;
        BOOST_CHECK_EQUAL(rrfi.set_txAntennaPort(a), true);
    }

    BOOST_AUTO_TEST_CASE(setTxAntennaPort_false) {
        const int a = 1;
        BOOST_CHECK_EQUAL(rrfi.set_txAntennaPort(a), false);
    }

    BOOST_AUTO_TEST_CASE(setRxAntennaPort_true) {
        const int a = 1;
        BOOST_CHECK_EQUAL(rrfi.set_rxAntennaPort(a), true);
    }

    BOOST_AUTO_TEST_CASE(setRxAntennaPort_false) {
        const int a = -2;
        BOOST_CHECK_EQUAL(rrfi.set_rxAntennaPort(a), false);
    }

BOOST_AUTO_TEST_SUITE_END()

BOOST_FIXTURE_TEST_SUITE(TestPowerControlServices, TestFixture)

    BOOST_AUTO_TEST_CASE(setTxMaxPowerLevel_true) {
        const int a = 90;
        BOOST_CHECK_EQUAL(rrfi.set_maxTxPowerLevel(a), true);
    }

    BOOST_AUTO_TEST_CASE(setTxPowerLevel) {
        double a = usrpDevice.min_tx_gain;
        BOOST_CHECK_EQUAL(rrfi.set_txPowerLevel(a), true);
        a = a - 1;
        BOOST_CHECK_EQUAL(rrfi.set_txPowerLevel(a), false);
        a = usrpDevice.max_tx_gain;
        BOOST_CHECK_EQUAL(rrfi.set_txPowerLevel(a), true);
        a = a + 1;
        BOOST_CHECK_EQUAL(rrfi.set_txPowerLevel(a), false);
    }

}
```



```

BOOST_AUTO_TEST_CASE(setRxGain) {
    double a = usrpDevice.min_rx_gain;
    BOOST_CHECK_EQUAL(rrfi.set_rxGain(a), true);
    a = a - 1;
    BOOST_CHECK_EQUAL(rrfi.set_rxGain(a), false);
    a = usrpDevice.max_rx_gain;
    BOOST_CHECK_EQUAL(rrfi.set_rxGain(a), true);
    a = a + 1;
    BOOST_CHECK_EQUAL(rrfi.set_rxGain(a), false);
}

BOOST_AUTO_TEST_SUITE_END()

BOOST_FIXTURE_TEST_SUITE(TestSpectrumControlServices, TestFixture)

BOOST_AUTO_TEST_CASE(setRxCenterFrequency) {
    double a = usrpDevice.min_rx_frequency;
    BOOST_CHECK_EQUAL(rrfi.set_rxCenterFrequency(a), true);
    a = a - 1;
    BOOST_CHECK_EQUAL(rrfi.set_rxCenterFrequency(a), false);
    a = usrpDevice.max_rx_frequency;
    BOOST_CHECK_EQUAL(rrfi.set_rxCenterFrequency(a), true);
    a = a + 1;
    BOOST_CHECK_EQUAL(rrfi.set_rxCenterFrequency(a), false);
}

BOOST_AUTO_TEST_CASE(setTxCenterFrequency) {
    double a = usrpDevice.min_tx_frequency;
    BOOST_CHECK_EQUAL(rrfi.set_txCenterFrequency(a), true);
    a = a - 1;
    BOOST_CHECK_EQUAL(rrfi.set_txCenterFrequency(a), false);
    a = usrpDevice.max_tx_frequency;
    BOOST_CHECK_EQUAL(rrfi.set_txCenterFrequency(a), true);
    a = a + 1;
    BOOST_CHECK_EQUAL(rrfi.set_txCenterFrequency(a), false);
}

BOOST_AUTO_TEST_CASE(setRxBandwidth) {
    double a = usrpDevice.min_rx_bandwidth;
    BOOST_CHECK_EQUAL(rrfi.set_rxBandwidth(a), true);
    a = a - 1;
    BOOST_CHECK_EQUAL(rrfi.set_rxBandwidth(a), false);
    a = usrpDevice.max_rx_bandwidth;
    BOOST_CHECK_EQUAL(rrfi.set_rxBandwidth(a), true);
    a = a + 1;
    BOOST_CHECK_EQUAL(rrfi.set_rxBandwidth(a), false);
}

BOOST_AUTO_TEST_CASE(setTxBandwidth) {
    double a = usrpDevice.min_tx_bandwidth;
    BOOST_CHECK_EQUAL(rrfi.set_txBandwidth(a), true);
    a = a - 1;
    BOOST_CHECK_EQUAL(rrfi.set_txBandwidth(a), false);
    a = usrpDevice.max_tx_bandwidth;
    BOOST_CHECK_EQUAL(rrfi.set_txBandwidth(a), true);
    a = a + 1;
    BOOST_CHECK_EQUAL(rrfi.set_txBandwidth(a), false);
}

BOOST_AUTO_TEST_CASE(setRxSamplingRate) {
    double a = usrpDevice.min_rx_rate;
    BOOST_CHECK_EQUAL(rrfi.set_rxSamplingRate(a), true);
    a = a - 1;

```

```

        BOOST_CHECK_EQUAL(rrfi.set_rxSamplingRate(a), false);
        a = usrpDevice.max_rx_rate;
        BOOST_CHECK_EQUAL(rrfi.set_rxSamplingRate(a), true);
        a = a + 1;
        BOOST_CHECK_EQUAL(rrfi.set_rxSamplingRate(a), false);
    }

    BOOST_AUTO_TEST_CASE(setTxSamplingRate) {
        double a = usrpDevice.min_tx_rate;
        BOOST_CHECK_EQUAL(rrfi.set_txSamplingRate(a), true);
        a = a - 1;
        BOOST_CHECK_EQUAL(rrfi.set_txSamplingRate(a), false);
        a = usrpDevice.max_tx_rate;
        BOOST_CHECK_EQUAL(rrfi.set_txSamplingRate(a), true);
        a = a + 1;
        BOOST_CHECK_EQUAL(rrfi.set_txSamplingRate(a), false);
    }

BOOST_AUTO_TEST_SUITE_END()

BOOST_FIXTURE_TEST_CASE(Set_WIFI_Parameters, TestFixture) {
    const double freq = 2400000000;
    const double bw = 20000000;
    const double sr = 2000000;
    const double gain = 60;

    BOOST_CHECK_EQUAL(rrfi.set_txAntennaPort(0), true);
    BOOST_CHECK_EQUAL(rrfi.set_txCenterFrequency(freq), true);
    BOOST_CHECK_EQUAL(rrfi.get_txCenterFrequency(), freq);
    BOOST_CHECK_EQUAL(rrfi.set_txBandwidth(bw), true);
    BOOST_CHECK_EQUAL(rrfi.get_txBandwidth(), bw);
    BOOST_CHECK_EQUAL(rrfi.set_txSamplingRate(sr), true);
    BOOST_CHECK_EQUAL(rrfi.get_txSamplingRate(), sr);
    BOOST_CHECK_EQUAL(rrfi.set_txPowerLevel(gain), true);
    BOOST_CHECK_EQUAL(rrfi.get_txPowerLevel(), gain);

    BOOST_CHECK_EQUAL(rrfi.set_rxAntennaPort(1), true);
    BOOST_CHECK_EQUAL(rrfi.set_rxCenterFrequency(freq), true);
    BOOST_CHECK_EQUAL(rrfi.get_rxCenterFrequency(), freq);
    BOOST_CHECK_EQUAL(rrfi.set_rxBandwidth(bw), true);
    BOOST_CHECK_EQUAL(rrfi.get_rxBandwidth(), bw);
    BOOST_CHECK_EQUAL(rrfi.set_rxSamplingRate(sr), true);
    BOOST_CHECK_EQUAL(rrfi.get_rxSamplingRate(), sr);
    BOOST_CHECK_EQUAL(rrfi.set_rxGain(gain), true);
    BOOST_CHECK_EQUAL(rrfi.get_rxGain(), gain);
}

BOOST_FIXTURE_TEST_CASE(Set_BlueTooth_Parameters, TestFixture) {
    const double freq = 2480000000;
    const double bw = 35000000;
    const double sr = 2000000;
    const double gain = 50;

    BOOST_CHECK_EQUAL(rrfi.set_txAntennaPort(0), true);
    BOOST_CHECK_EQUAL(rrfi.set_txCenterFrequency(freq), true);
    BOOST_CHECK_EQUAL(rrfi.get_txCenterFrequency(), freq);
    BOOST_CHECK_EQUAL(rrfi.set_txBandwidth(bw), true);
    BOOST_CHECK_EQUAL(rrfi.get_txBandwidth(), bw);
    BOOST_CHECK_EQUAL(rrfi.set_txSamplingRate(sr), true);
    BOOST_CHECK_EQUAL(rrfi.get_txSamplingRate(), sr);
    BOOST_CHECK_EQUAL(rrfi.set_txPowerLevel(gain), true);
    BOOST_CHECK_EQUAL(rrfi.get_txPowerLevel(), gain);
}

```

```

BOOST_CHECK_EQUAL(rrfi.set_rxAntennaPort(1), true);
BOOST_CHECK_EQUAL(rrfi.set_rxCenterFrequency(freq), true);
BOOST_CHECK_EQUAL(rrfi.get_rxCenterFrequency(), freq);
BOOST_CHECK_EQUAL(rrfi.set_rxBandwidth(bw), true);
BOOST_CHECK_EQUAL(rrfi.get_rxBandwidth(), bw);
BOOST_CHECK_EQUAL(rrfi.set_rxSamplingRate(sr), true);
BOOST_CHECK_EQUAL(rrfi.get_rxSamplingRate(), sr);
BOOST_CHECK_EQUAL(rrfi.set_rxGain(gain), true);
BOOST_CHECK_EQUAL(rrfi.get_rxGain(), gain);

}

BOOST_FIXTURE_TEST_CASE(Tx_Test, TestFixture) {
    const double freq = 2480000000;
    const double bw = 35000000;
    const double sr = 2000000;
    const double gain = 50;

    BOOST_CHECK_EQUAL(rrfi.set_txAntennaPort(0), true);
    BOOST_CHECK_EQUAL(rrfi.set_txCenterFrequency(freq), true);
    BOOST_CHECK_EQUAL(rrfi.get_txCenterFrequency(), freq);
    BOOST_CHECK_EQUAL(rrfi.set_txBandwidth(bw), true);
    BOOST_CHECK_EQUAL(rrfi.get_txBandwidth(), bw);
    BOOST_CHECK_EQUAL(rrfi.set_txSamplingRate(sr), true);
    BOOST_CHECK_EQUAL(rrfi.get_txSamplingRate(), sr);
    BOOST_CHECK_EQUAL(rrfi.set_txPowerLevel(gain), true);
    BOOST_CHECK_EQUAL(rrfi.get_txPowerLevel(), gain);

    const std::vector<short> buff(5000);
    for (int i = 0; i < 100; i++) {
        BOOST_CHECK_EQUAL(rrfi.tx_from_buff(buff, 50), true);
    }

    BOOST_CHECK_EQUAL(rrfi.tx_from_file("/home/leoind/Dropbox/RRS_Clion/pic.png",
    100), true);
}

```

Текстовый вывод:

Проблема подключения:

```

Testing started at 12:17 ...
/home/leoind/Dropbox/RRS_Clion/cmake-build-debug/RRS_Clion --log_format=HRF -
-log_level=all --color_output=true --run_test=Set_WIFI_Parameters --
color_output=false --report_format=HRF --show_progress=no
Running 1 test case...
Entering test module RRFI Tests
USRP device configuration started...
Creating the usrp device with: USB...
[INFO] [UHD] linux; GNU C++ version 7.3.0; Boost_105800; UHD_3.14.0.0-269-
g11bbc3c0
USRP radio device not found, please connect device correctly
Process finished with exit code 0

```

Тест параметров Bluetooth:

```

Testing started at 12:14 ...

```

```

/home/leoind/Dropbox/RRS_Clion/cmake-build-debug/RRS_Clion --log_format=HRF -
-log_level=all --color_output=true --run_test=Set_BlueTooth_Parameters --
color_output=false --report_format=HRF --show_progress=no
Running 1 test case...
Entering test module RRFI Tests
USRP device configuration started...
Creating the usrp device with: USB...
[INFO] [UHD] linux; GNU C++ version 7.3.0; Boost_105800; UHD_3.14.0.0-269-
g11bbc3c0
[INFO] [B200] Detected Device: B210
[INFO] [B200] Operating over USB 2.
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Setting master clock rate selection to 'automatic'.
[INFO] [B200] Asking for clock rate 16.000000 MHz...
[INFO] [B200] Actually got clock rate 16.000000 MHz.
Lock mboard clocks: internal
RX channel number = 1
TX channel number = 2
RX antenna 0: TX/RX
RX antenna 1: RX2
TX antenna 0: TX/RX
Actual TX Antenna: TX/RX, on channel 0
Actual RX Antenna: RX2, on channel 0
TX gain is: (0, 89.75, 0.25)
RX gain is: (0, 76, 1)
TX bandwidth is(200000, 5.6e+07)
RX bandwidth is(200000, 5.6e+07)
TX frequency is(4.2e+07, 6.008e+09, 0.00372529)
[INFO] [B200] RX frequency is(4.2e+07, 6.008e+09, 0.00372529)
Min TX sampling rate is 31250
Max TX sampling rate is 1.6e+07
Asking for clock rate 32.000000 MHz...
Min RX sampling rate is 31250
Max RX sampling rate is 1.6e+07
[INFO] [B200] Actually got clock rate 32.000000 MHz.
USRP device configuration done
Setting TX Antenna: TX/RX
Actual TX Antenna: TX/RX, on channel 0
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(179): info: check
rrfi.set_txAntennaPort(0) == true passed
tx_centerfreq changed successfully
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(180): info: check
rrfi.set_txCenterFrequency(freq) == true passed
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(181): info: check
rrfi.get_txCenterFrequency() == freq passed
tx_bandwidth changed successfully
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(182): info: check
rrfi.set_txBandwidth(bw) == true passed
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(183): info: check
rrfi.get_txBandwidth() == bw passed
tx_rate changed successfully
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(184): info: check
rrfi.set_txSamplingRate(sr) == true passed
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(185): info: check
rrfi.get_txSamplingRate() == sr passed
tx_powerlevel changed successful
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(186): info: check
rrfi.set_txPowerLevel(gain) == true passed

```

```

/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(187): info: check
rrfi.get_txPowerLevel() == gain passed
Setting RX Antenna: RX2
Actual RX Antenna: RX2, on channel 0
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(189): info: check
rrfi.set_rxAntennaPort(1) == true passed
rx_centerfreq changed successfully
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(190): info: check
rrfi.set_rxCenterFrequency(freq) == true passed
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(191): info: check
rrfi.get_rxCenterFrequency() == freq passed
rx_bandwidth changed successfully
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(192): info: check
rrfi.set_rxBandwidth(bw) == true passed
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(193): info: check
rrfi.get_rxBandwidth() == bw passed
rx_rate changed successfully
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(194): info: check
rrfi.set_rxSamplingRate(sr) == true passed
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(195): info: check
rrfi.get_rxSamplingRate() == sr passed
rx_gain changed successful
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(196): info: check
rrfi.set_rxGain(gain) == true passed
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(197): info: check
rrfi.get_rxGain() == gain passed

Leaving test module RRFI Tests
*** No errors detected
Process finished with exit code 0

```

Тест параметров WiFi:

```

Testing started at 12:16 ...
/home/leoind/Dropbox/RRS_Clion/cmake-build-debug/RRS_Clion --log_format=HRF -
-log_level=all --color_output=true --run_test=Set_WIFI_Parameters --
color_output=false --report_format=HRF --show_progress=no
Running 1 test case...
Entering test module RRFI Tests
USRP device configuration started...
Creating the usrp device with: USB...
[INFO] [UHD] linux; GNU C++ version 7.3.0; Boost_105800; UHD_3.14.0.0-269-
g11bbc3c0
[INFO] [B200] Detected Device: B210
[INFO] [B200] Operating over USB 2.
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Setting master clock rate selection to 'automatic'.
[INFO] [B200] Asking for clock rate 16.000000 MHz...
[INFO] [B200] Actually got clock rate 16.000000 MHz.
Lock mboard clocks: internal
RX channel number = 1
TX channel number = 2
RX antenna 0: TX/RX
RX antenna 1: RX2
TX antenna 0: TX/RX
Actual TX Antenna: TX/RX, on channel 0

```

```

Actual RX Antenna: RX2, on channel 0
TX gain is: (0, 89.75, 0.25)
RX gain is: (0, 76, 1)
TX bandwidth is(200000, 5.6e+07)
RX bandwidth is(200000, 5.6e+07)
TX frequency is(4.2e+07, 6.008e+09, 0.00372529)
[INFO] [B200] Asking for clock rate 32.000000 MHz...
RX frequency is(4.2e+07, 6.008e+09, 0.00372529)
Min TX sampling rate is 31250
Max TX sampling rate is 1.6e+07
Min RX sampling rate is 31250
Max RX sampling rate is 1.6e+07
[INFO] [B200] Actually got clock rate 32.000000 MHz.
USRP device configuration done
Setting TX Antenna: TX/RX
Actual TX Antenna: TX/RX, on channel 0
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(151): info: check
rrfi.set_txAntennaPort(0) == true passed
tx_centerfreq changed successfully
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(152): info: check
rrfi.set_txCenterFrequency(freq) == true passed
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(153): info: check
rrfi.get_txCenterFrequency() == freq passed
tx_bandwidth changed successfully
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(154): info: check
rrfi.set_txBandwidth(bw) == true passed
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(155): info: check
rrfi.get_txBandwidth() == bw passed
tx_rate changed successfully
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(156): info: check
rrfi.set_txSamplingRate(sr) == true passed
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(157): info: check
rrfi.get_txSamplingRate() == sr passed
tx_powerlevel changed successful
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(158): info: check
rrfi.set_txPowerLevel(gain) == true passed
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(159): info: check
rrfi.get_txPowerLevel() == gain passed
Setting RX Antenna: RX2
Actual RX Antenna: RX2, on channel 0
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(161): info: check
rrfi.set_rxAntennaPort(1) == true passed
rx_centerfreq changed successfully
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(162): info: check
rrfi.set_rxCenterFrequency(freq) == true passed
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(163): info: check
rrfi.get_rxCenterFrequency() == freq passed
rx_bandwidth changed successfully
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(164): info: check
rrfi.set_rxBandwidth(bw) == true passed
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(165): info: check
rrfi.get_rxBandwidth() == bw passed
rx_rate changed successfully
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(166): info: check
rrfi.set_rxSamplingRate(sr) == true passed
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(167): info: check
rrfi.get_rxSamplingRate() == sr passed
rx_gain changed successful
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(168): info: check
rrfi.set_rxGain(gain) == true passed
/home/leoind/Dropbox/RRS_Clion/ETSI_RRS_RRFITests.cpp(169): info: check
rrfi.get_rxGain() == gain passed

```

```
Leaving test module RRFI Tests  
*** No errors detected  
Process finished with exit code 0
```

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«Санкт-Петербургский государственный университет
аэрокосмического приборостроения»

ОТЗЫВ РУКОВОДИТЕЛЯ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

на тему Исследование программного интерфейса радиочастотной части мобильного
мультирадиотерминала

выполненную студентом группы № 1740М

Шатуновым Леонидом Владимировичем

фамилия, имя, отчество студента

по направлению подготовки /
специальности

09.04.01
(код)

Информатика и вычислительная техника
(наименование направления подготовки/специальности)

(наименование направления подготовки/специальности)

Актуальность темы работы:

Новая директива RED Европейской Комиссии определяют бизнес условия для разработки и применения реконфигурируемых радиоустройств, способных менять свои существенные радио характеристики (модуляцию, кодирование, полосы частот и т.д.). Эта директива определила актуальность работы, которая связана с разработкой и оптимизацией интерфейса для реконфигурирования радиочастотной части устройства.

Цель и задачи работы: Изучение концепции реконфигурируемых радиосистем; исследование, имплементация и оптимизация программного интерфейса радиочастотной части реконфигурируемых мобильных устройств.

Общая оценка выполнения поставленной перед студентом задачи, основные достоинства и недостатки работы:
В процессе работы над диссертацией Леонид изучил европейские стандарты, связанные с архитектурой реконфигурируемых радио устройств. Были изучены различные подходы к реконфигурации на основе концепций JTRS SDR и ETSI RRS. Основной фокус в работе был сделан на исследование и реализации интерфейса к радиочастотной части устройства. Было показано, что существует возможность реконфигурации параметров радио трансивера в процессе работы. Работа имеет большое прикладное значение в частности для разработки реконфигурируемой коммуникационной полезной нагрузки спутников. В процессе работы Леонид выполнил все поставленные перед ним задачи. Основное

достоинство работы в практической реализации интерфейса и получении рабочего прототипа радиоустройства с реконфигурируемой радиочастотной частью в соответствии с европейскими стандартами.

Степень самостоятельности и способности к исследовательской работе студента (умение и навыки поиска, обобщения, анализа материала и формулирования выводов):
В процессе работы над диссертацией Леонид продемонстрировал умение самостоятельно собирать, анализировать и обобщать материалы исследования.

Проверка текста выпускной квалификационной работы с использованием системы guar.antiplagiat.ru, проводившаяся «14» июня 2019 г. показывает оригинальность содержания на уровне 92.51%.

Степень грамотности изложения и оформления материала:

Диссертация изложена грамотно и оформление материала произведено корректно.

Оценка деятельности студента в период подготовки выпускной квалификационной работы (добросовестность, работоспособность, ответственность, аккуратность и т.п.):
Леонид проявил ответственность и добросовестность в процессе работы над диссертацией.

Общий вывод:

Работа выполнена в соответствии с требованиями, предъявляемыми к магистерским диссертационным работам. Общая оценка работы – ОТЛИЧНО.

Руководитель

Доцент, к.т.н.

должность, уч. степень, звание

 17.06.19
подпись, дата

Иванов В.Н.

инициалы, фамилия

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

РЕЦЕНЗИЯ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

на тему Исследование программного интерфейса радиочастотной части мобильного
мультирадиотерминала

выполненную студентом группы № 1740М

Шатуновым Леонидом Владимировичем

фамилия, имя, отчество студента

по направлению подготовки / 09.04.01 Информатика и вычислительная техника
специальности (код) (наименование направления подготовки/специальности)

(наименование направления подготовки/специальности)

Актуальность темы исследования:

Тема исследования актуальна в связи с необходимостью прогрессирования и улучшения
существующих стандартов мобильной связи. Также прослеживается потребность в
модифицировании существенных параметров радиосистем в процессе работы.

Краткая характеристика структуры работы и отдельных ее разделов:

Работа состоит из 2 теоретических разделов и 1 практического, списков сокращений и
используемых источников и 7 приложений, содержащих разработанный исходный код.

Достоинства работы (интересные материалы, положения, выводы, в которых проявились самостоятельность студента, его эрудиция, оригинальность мышления, знание литературы, уровень теоретической подготовки и т.п.):

Теоретические главы детально описаны, опираются на большое количество иностранных
материалов, что говорит о хорошем уровне теоретической подготовки и знания материала.
Было проведено подробное исследование реконфигурируемого мобильного устройства, в
частности интерфейса RRFL. Основным достоинством работы является практическая
реализация этого интерфейса и доказательство возможности эффективного использования
технологий реконфигурируемых радиосистем.

Недостатки работы (по содержанию и по оформлению):

К недостаткам можно отнести не всегда логичное изложение материала, а также оформление некоторых элементов не в соответствии с ГОСТ.

Общий вывод о выпускной квалификационной работе, ее соответствии предъявляемым к данному виду работ требованиям:

ВКР соответствует предъявляемым к ней требованиям, показывает способность студента работать современной иностранной технической литературой и хороший уровень теоретической подготовки.


Мнение рецензента об оценке работы:

Работа мною оценивается на оценку «хорошо», ее автор Шатунов Л.В. заслуживает присвоения квалификации «магистр» по направлению 09.04.01 Информатика и вычислительная техника.

Рецензент

профессор, доцент, д.т.н.

должность, уч. степень, звание



подпись, дата

18.06.19

Балонин Н.А.

инициалы, фамилия