

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

МОиПЭВМ

Направленность	Программная инженерия
----------------	-----------------------

Программный имитатор для отладки взаимодействия
устройства доверенного управления
аппаратурой передачи данных с другими разрабатываемыми
устройствами

(фамилия, инициалы)

(фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
Пензенский государственный университет
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«20» декабря 2024 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Калугин Артём Андреевич гр. 21ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программный имитатор для отладки взаимодействия устройства
доверенного управления аппаратурой передачи данных с другими разрабатываемыми
устройствами

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

2. Руководитель работы к.т.н., доцент кафедры "МО и ПЭВМ" Гурьянов Л. В.

3. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение: программное средство предназначено для отладки программного
обеспечения другого разрабатываемого устройства

Основные функции:

- 1) Выбрать Ethernet-адаптер, COM-порт, LocalServer
- 2) Принять/отправить/обработать команду
- 3) Отобразить события имитатора
- 4) Записать всех событий имитатора в лог-файл

Технологии разработки: ООП, QT, рсар

Язык программирования: C++

Инструментальные средства разработки: Microsoft Visual Studio 2022 Community Edition,
QT Creator

Среда исполнения: Windows 10

4. Перечень подлежащих разработке вопросов

1. Анализ предметной области и требований к системе. Постановка задачи
2. Проектирование программных средств
3. Реализация программных средств
4. Тестирование программных средств
5. Оформление пояснительной записки

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Планирование разработки и оценка бюджета	5%	23.05.2025 – 25.05.2025	
6	Оформление пояснительной записки	15%	26.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 62 листов, 22 рисунка, 14 таблиц, 20 использованных источников, 2 приложений.

ПРОГРАММИРОВАНИЕ, ИМИТАТОР, ПЕРЕДАЧА ДАННЫХ, ОТЛАДКА ВЗАИМОДЕЙСТВИЯ, ИНТЕРФЕЙС, ТЕСТИРОВАНИЕ, QT, C++

Предмет работы: разработка программного имитатора для отладки взаимодействия устройства доверенного управления аппаратурой передачи данных с другими разрабатываемыми устройствами.

Объект работы: программный имитатор для отладки взаимодействия устройства доверенного управления аппаратурой передачи данных с другими разрабатываемыми устройствами.

Цель работы: разработать программный имитатор для отладки взаимодействия устройства доверенного управления аппаратурой передачи данных с другими разрабатываемыми устройствами.

Область применения: приложение может использоваться для отладки взаимодействия устройства доверенного управления аппаратурой передачи данных с другими разрабатываемыми устройствами.

Технологии разработки: ООП, QT, рсар

Язык программирования: C++

Инструментальные средства разработки: Microsoft Visual Studio 2022 Community Edition, QT Creator

Среда исполнения: Windows 10

					ПГУ 09.03.04 - 08BP211.12 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Программный имитатор для отладки взаимодействия устройства доверенного управления аппаратурой передачи данных с другими разрабатываемыми устройствами. Пояснительная записка	Лит.	Лист	Листов
Разраб.		Калугин А.А.					4	62
Провер.		Гурьянов Л. В.				гр. 21ВП1		
Реценз.								
Н. Контр.		Такташкин Д.В.						
Утв.		Козлов А.Ю.						

Содержание

Введение	6
1 Постановка задачи разработки приложения.....	8
1.1 Анализ системы разрабатываемых устройств и постановка задачи.....	8
1.2 Анализ функциональных требований.....	9
2 Проектирование приложения.....	15
2.1 Выбор технологии разработки.....	15
2.2 Проектирование структуры классов приложения	15
2.3 Проектирование графического интерфейса	18
3 Реализация приложения.....	27
4 Тестирование программного имитатора	32
4.1 Тестирование функционала программного имитатора	32
4.1.1 Классы эквивалентности.....	32
4.1.2 Функциональная диаграмма и таблица решений	34
4.2 Анализ сложности кода открытия СОМ-порта.....	38
4.3 Вычисление и определение метрики Холстеда	40
4.4 Автоматизированное тестирование	43
5 Планирование разработки и оценка бюджета.....	47
Заключение	50
Список использованных источников.....	52
Приложение А (обязательное) Исходный код приложения	54
Приложение Б (рекомендуемое) Акт внедрения	62

Введение

В современном мире разработка проектов это сложных и комплексный процесс, сопряженных с различными сложностями.

На некоторых стадиях разработки различных проектов может возникать необходимость промежуточной отладки устройств. Для одиночного разрабатываемого устройства такое тестирование не вызывает трудностей, но для нескольких устройств, тесно взаимодействующих друг с другом, такое тестирование может вызвать трудности. Часто возникает необходимость проверки корректного взаимодействия устройств между собой, но некоторые устройства могут быть ещё не разработаны или разрабатываться другой компанией в другом городе, что делает дорогостоящим доставку устройства для отладки.

Для решения данной задачи часто используется имитатор недоступного устройства, который берет на себя задачу имитировать поведение устройства и заменять его при отладке разрабатываемого устройства.

Данный имитатор позволяет сэкономить время и деньги, а так же произвести более глубокую отладку разрабатываемого устройства путем регулирования поведения имитатора, в соответствии с ситуацией, которую необходимо отладить.

Так же имитатор может отображать все происходящие с ним события, что позволяет составить более полную картину происходящих событий и быстрее обнаружить ошибку в разрабатываемом устройстве, а при изменении концепции корректирование поведения имитатора можно произвести на месте, не ожидая другую команду разработчиков.

Для создания имитатора требуется решить следующие задачи:

- выполнить анализ системы устройств;
- определить требования к разрабатываемой программе;
- разработать диаграмму вариантов использования;
- выбрать технологии для разработки;
- спроектировать архитектуру приложения;

- разработать диаграммы классов;
- спроектировать графический интерфейс приложения;
- выполнить оценку бюджета разработки;
- разработать диаграмму Ганта;
- реализовать спроектированное приложение;
- протестировать разработанное приложение.

В результате выполнения вышеперечисленных задач будет спроектирован, разработан и протестирован программный имитатор на языке программирования C++ с использованием фреймворка QT, а также произведен анализ задач и оценка бюджета.

1 Постановка задачи разработки приложения

Для успешного создания программного имитатора, нужно провести анализ целей его разработки, а так же системы разрабатываемых устройств и то, какие функции выполняет устройство, имитатор которого необходимо разработать. Помимо этого, нужно сформулировать задачу на проектирование приложения и выполнить анализ функциональных требований.

1.1 Анализ системы разрабатываемых устройств и постановка задачи

Для определения требуемых функций имитатора необходима проанализировать систему разрабатываемых устройств, а именно, устройство, заменяемое имитатором, и устройства взаимодействующие с ним. Устройство заменяемое имитатором – Устройство управления(УУ). На рисунке 1 видно, что УУ взаимодействует с Автоматизированным рабочим местом управления с помощью COM-порта RS-422, обработчиком запросов с помощью LocalServer и Аппаратурой передачи данных с помощью Ethernet. Основная функция УУ отправка команда аппаратуре передачи данных и обработка команд Автоматизированного рабочего места управления.

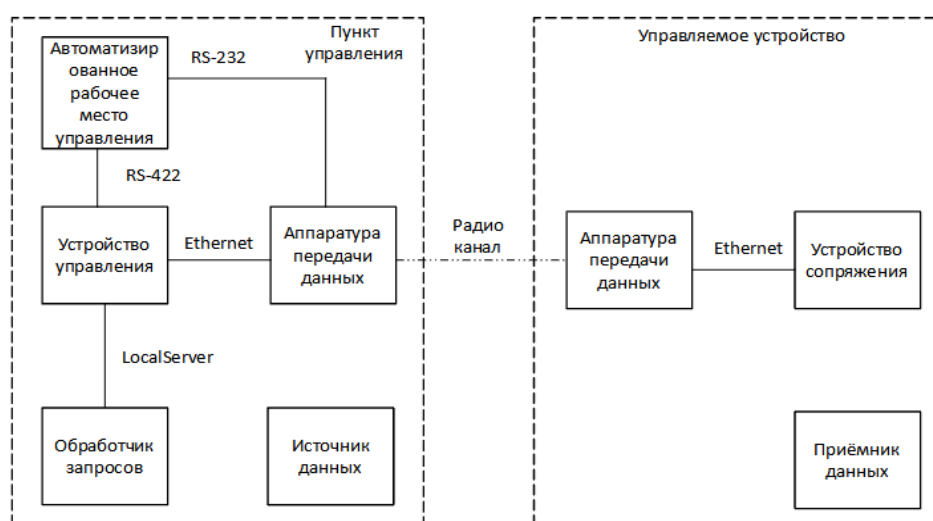


Рисунок 1 – Система разрабатываемых устройств

Так как имитатор должен уметь взаимодействовать с данными устройствами необходимо, чтобы имитатор поддерживал работу с Ethernet-адаптером, LocalServer и COM-портом RS-422. Кроме того имитатор должен уметь отправлять команды и корректно обрабатывать команды от Автоматизированного рабочего места управления. Так как основная функция имитатора – это отладка, следует добавить функцию логирования и ввода сообщения из файлов.

1.2 Анализ функциональных требований

В приложении должна быть возможность:

- выбрать для захвата Ethernet-адаптер;
- выбрать для захвата COM-порт;
- выбрать имя LocalServer и подключиться к нему;
- принять команду;
- отправить команду;
- загрузить команду из файла;
- загрузить команду из файла в формате Нех;
- увидеть все происходящие с имитатором события;
- записать в все происходящие с имитатором события в лог-файл.

Диаграмма вариантов использования (англ. use-case diagram) – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей [1]. Эта диаграмма используется для визуализации программы, ее функций и объектов работы. Диаграмма вариантов использования наглядно показывает функции, которые необходимо реализовать, взаимосвязи этих функций друг с другом и актеров, которые пользуются этими функциями.

На основе сформулированных требований к имитатору составлена диаграмма вариантов использования (рисунок 2).



Рисунок 2 – Диаграмма вариантов использования

Далее описаны основные варианты использования для актёра «Пользователь».

В таблице 1 представлено описание варианта использования «Выбрать имя LocalServer и подключиться к нему».

Таблица 1 – Вариант использования «Выбрать имя LocalServer и подключиться к нему»

Вариант использования	Выбрать имя LocalServer и подключиться к нему
ID	1
Краткое описание	Пользователь выбирает имя LocalServer и подключается к нему

Продолжение таблицы 1

Главные актёры	Пользователь
Второстепенные актёры	Нет
Предусловия	Нет
Основной поток	<ol style="list-style-type: none"> 1. Пользователь переходит на модальное окно выбора имени LocalServer и вводит его 2. Система считывает введенное имя 3. Система подключается к LocalServer
Постусловия	Было выполнено подключение к LocalServer

В таблице 2 представлено описание варианта использования «Выбрать для захвата COM-порт».

Таблица 2 – Вариант использования «Выбрать для захвата COM-порт»

Вариант использования	Выбрать для захвата COM-порт
ID	2
Краткое описание	Пользователь выбирает COM-порт из предоставленного списка и захватывает его
Главные актёры	Пользователь
Второстепенные актёры	Нет
Предусловия	Было выполнено подключение к LocalServer

Продолжение таблицы 2

Основной поток	1. Пользователь переходит на модальное окно выбора COM-порта и выбирает его 2. Система считывает введенный COM-порт Система захватывает и конфигурирует COM-порт
Постусловия	Был выполнен захват COM-порта

В таблице 3 представлено описание варианта использования «Выбрать для захвата Ethernet-адаптер».

Таблица 3 – Вариант использования «Выбрать для захвата Ethernet-адаптер»

Вариант использования	Выбрать для захвата Ethernet-адаптер
ID	3
Краткое описание	Пользователь выбирает Ethernet-адаптер из предоставленного списка и захватывает его
Главные актёры	Пользователь
Второстепенные актёры	Нет
Предусловия	Был выполнен захват COM-порта
Основной поток	1. Пользователь переходит на модальное окно выбора Ethernet-адаптера 2. Система считывает введенный Ethernet-адаптер Система захватывает Ethernet-адаптер

В таблице 4 представлено описание варианта использования «выбрать для захвата COM-порт».

Таблица 4 – Вариант использования «Отправить команду»

Вариант использования	Отправить команду
ID	4
Краткое описание	Пользователь отправляет введенную команду другому устройству
Главные актёры	Пользователь
Второстепенные актёры	Нет
Предусловия	Был выполнен захват Ethernet-адаптера
Основной поток	<ol style="list-style-type: none"> 1. Пользователь выбирает режим работы отправки 2. Пользователь вводит команду и нажимает отправить 3. Система отправляет введенную команду через COM-порт.
Постусловия	Была отправлена команда

В таблице 5 представлено описание варианта использования «Записать все происходящие с имитатором события в лог-файл».

Таблица 5 – Вариант использования «Записать все происходящие с имитатором события в лог-файл»

Вариант использования	Записать все происходящие с имитатором события в лог-файл
ID	5
Краткое описание	Пользователь записывает все происходящие с имитатором события в лог-файл

Продолжение таблицы 5

Главные актёры	Пользователь
Второстепенные актёры	Нет
Предусловия	Был выполнен захват Ethernet-адаптера
Основной поток	<ol style="list-style-type: none"> 1. Пользователь выбирает любой режим работы 2. Пользователь выбивает в меню имя файла лога <p>Система сохраняет информацию о всех произошедших событиях в файл</p>
Постусловия	Были записаны в лог-файл все происходящие с имитатором события

Вышеперечисленные варианты использования необходимы для проектирования приложения.

2 Проектирование приложения

При проектировании приложения необходимо выбрать технологии разработки наиболее подходящие по совокупности таких факторов, как конечное быстродействие приложения, возможности выполнения всех поставленных требований и удобства разработки. Так же необходимо продумать структуру классов приложения.

2.1 Выбор технологии разработки

Для разработки графического интерфейса приложения был выбран фреймворк Qt, так как он предоставляет удобный визуальный редактор приложения и позволяет контролировать многие события, такие как нажатие кнопки, изменение текста в текстовом окне и выбор пункта меню [2, 3]. Также одной из возможностей Qt является поддержка взаимодействия с LocalServer. Немаловажным фактором является то, что Qt является фреймворком C++, языка, сильными сторонами которого являются высокое быстродействие и контроль памяти, а также широкие возможности [4 – 6].

Для взаимодействия с COM-портом был выбран Windows API – набор базовых функций операционной системы Windows, который предоставляет интерфейс взаимодействия с последовательными портами [7].

Для взаимодействия с Ethernet была выбрана библиотека pcap за то, что позволяет с легкостью получить список доступный Ethernet-адаптеров, захватить нужный, отправлять и получать Ethernet пакеты [8].

2.2 Проектирование структуры классов приложения

При проектировании классов было принято решение о разбиении программы на четыре логически отдельных класса. По одному классу для взаимодействия с другими устройствами по LocalServer, Ethernet и Com-порту. Кроме этого еще один класс отвечает за графический интерфейс и контроль остальных классов.

Структура классов показана на диаграмме классов (рисунок 3) [9].

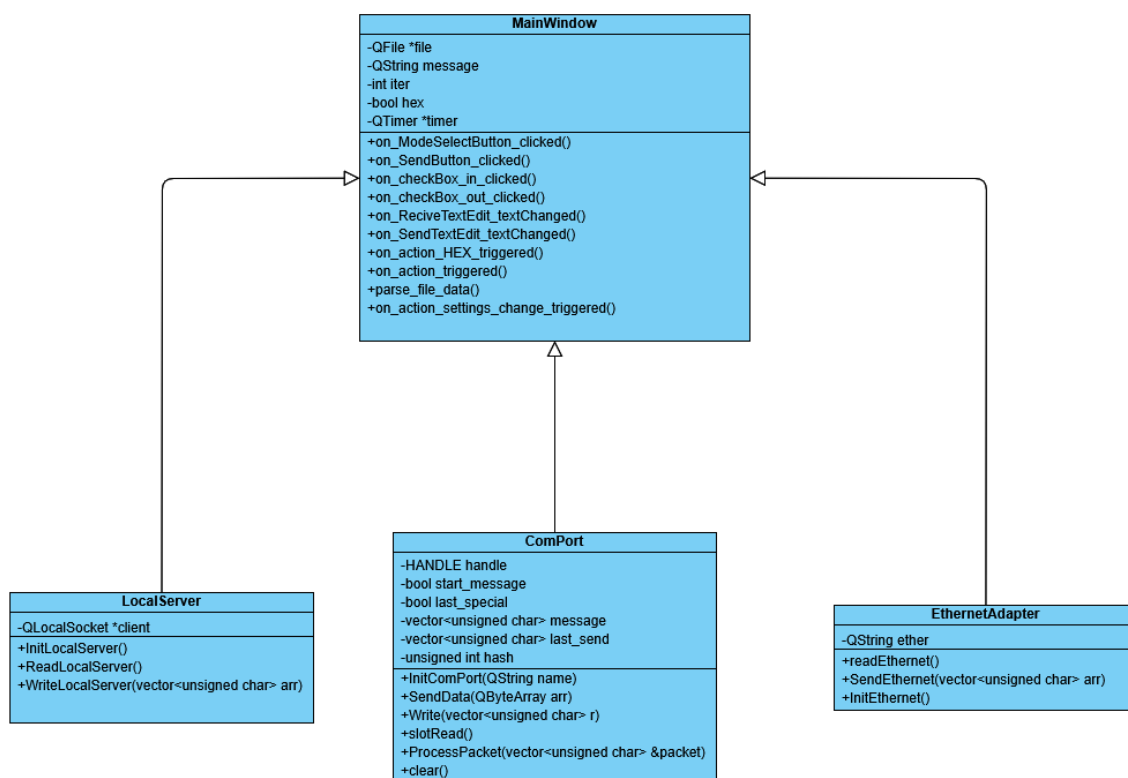


Рисунок 3 – Диаграмма классов

Название класса: MainWindow.

Назначение класса: Управляет классами, отвечающие за взаимодействие с другими устройствами (ComPort, EthernetAdapter, LocalServer), отвечает за графический интерфейс.

Методы класса:

- on_ModeSelectButton_clicked(): метод выбора режима работы имитатора(отправка или приём), вызывается при нажатии на кнопку;
- on_SendButton_clicked(): отправляет сообщение другому устройству, вызывается при нажатии на кнопку;
- on_checkBox_in_clicked(): выбор режима работы «Приём»;
- on_checkBox_out_clicked(): выбор режима работы «Отправка» ;
- on_ReciveTextEdit_textChanged(): вызывается при изменении текста в текстовом окне приёма, очищает текст, если в окне более 100000 строк;
- on_SendTextEdit_textChanged(): вызывается при изменении текста в текстовом окне отправки, очищает текст, если в окне более 100000 строк;

- on_action_HEX_triggered(): изменяет режим отображения текста на HEX;
- on_action_triggered(): изменяет режим отображения текста на ASCII;
- parse_file_data(): метод разбиения файла на отдельные сообщения;
- on_action_settings_change_triggered(): метод изменения настроек программы.

Название класса: LocalServer.

Назначение класса: отвечает за управление LocalServer и связь с другим устройством с его помощью.

Методы класса:

- InitLocalServer(): отвечает за начальную настройку, подключение и конфигурацию LocalServer;
- ReadLocalServer(): чтение сообщений из LocalServer;
- WriteLocalServer(vector<unsigned char> arr): отправляет сообщение другому устройству через LocalServer.

Название класса: ComPort.

Назначение класса: отвечает за управление ComPort и связь с другим устройством с его помощью.

Методы класса:

- InitComPort(QString name): отвечает за начальную настройку, захват и конфигурацию ComPort;
- SendData(QByteArray arr): отправка сообщения через ComPort;
- Write(vector<unsigned char> r): изменение сообщения по протоколу и его отправка через ComPort;
- slotRead(): чтение сообщения через ComPort;
- ProcessPacket(vector<unsigned char> &packet): обработка сообщения;
- clear(): очистка буферов класса.

Название класса: EthernetAdapter.

Назначение класса: отвечает за управление EthernetAdapter и связь с другим устройством с его помощью.

Методы класса:

- InitEthernet(): отвечает за начальную настройку, захват и конфигурацию EthernetAdapter;
- readEthernet(): чтение сообщений из EthernetAdapter;
- SendEthernet(vector<unsigned char> arr): отправляет сообщение другому устройству через EthernetAdapter.

2.3 Проектирование графического интерфейса

Графический интерфейс приложения должен обеспечивать понятное представление информации и позволять совершать действия, необходимые для выполнения поставленных перед приложением задач [10, 11].

При запуске приложения появляется окно выбора COM-порта (рисунок 4, 5).

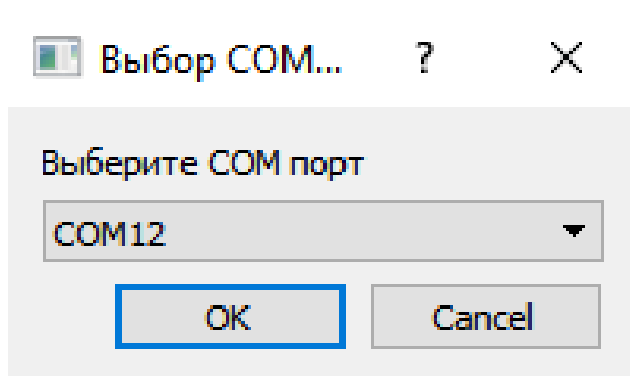


Рисунок 4 – Окно выбора COM-порта

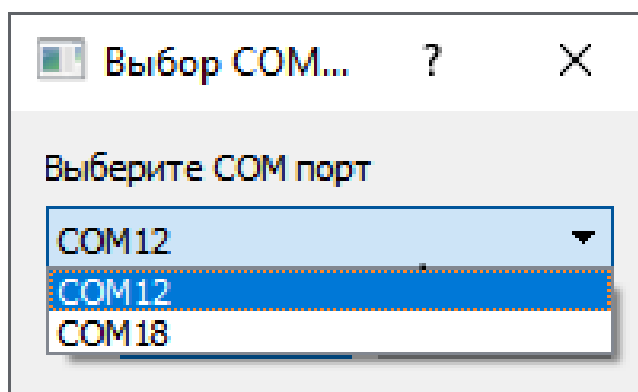


Рисунок 5 – Варианты выбора COM-порта

После выбора COM-порта появляется окно выбора Ethernet-адаптера (рисунок 6, 7).

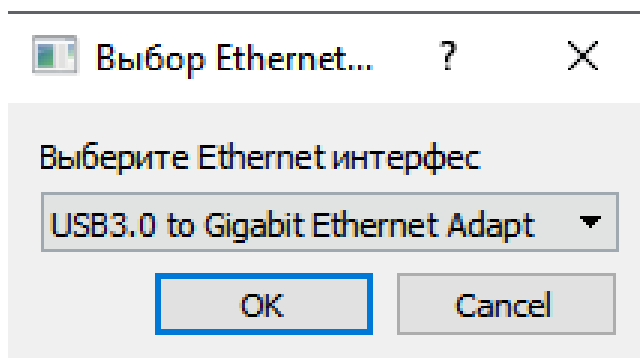


Рисунок 6 – Окно выбора Ethernet-адаптера

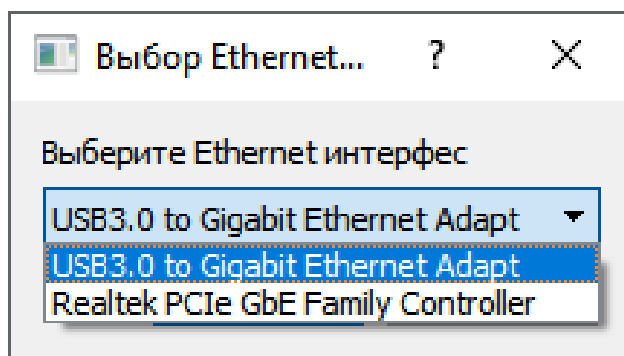


Рисунок 7 – Варианты выбора Ethernet-адаптера

Последним необходимо выбрать имя LocalServer и подключиться к нему (рисунок 8).

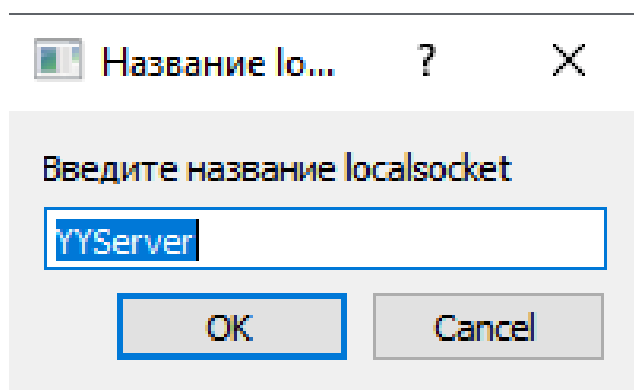


Рисунок 8 – Окно выбора имени Ethernet-адаптера

После ввода всех необходимых данных должно появиться основное окно программы с выбором режима работы (рисунок 9).

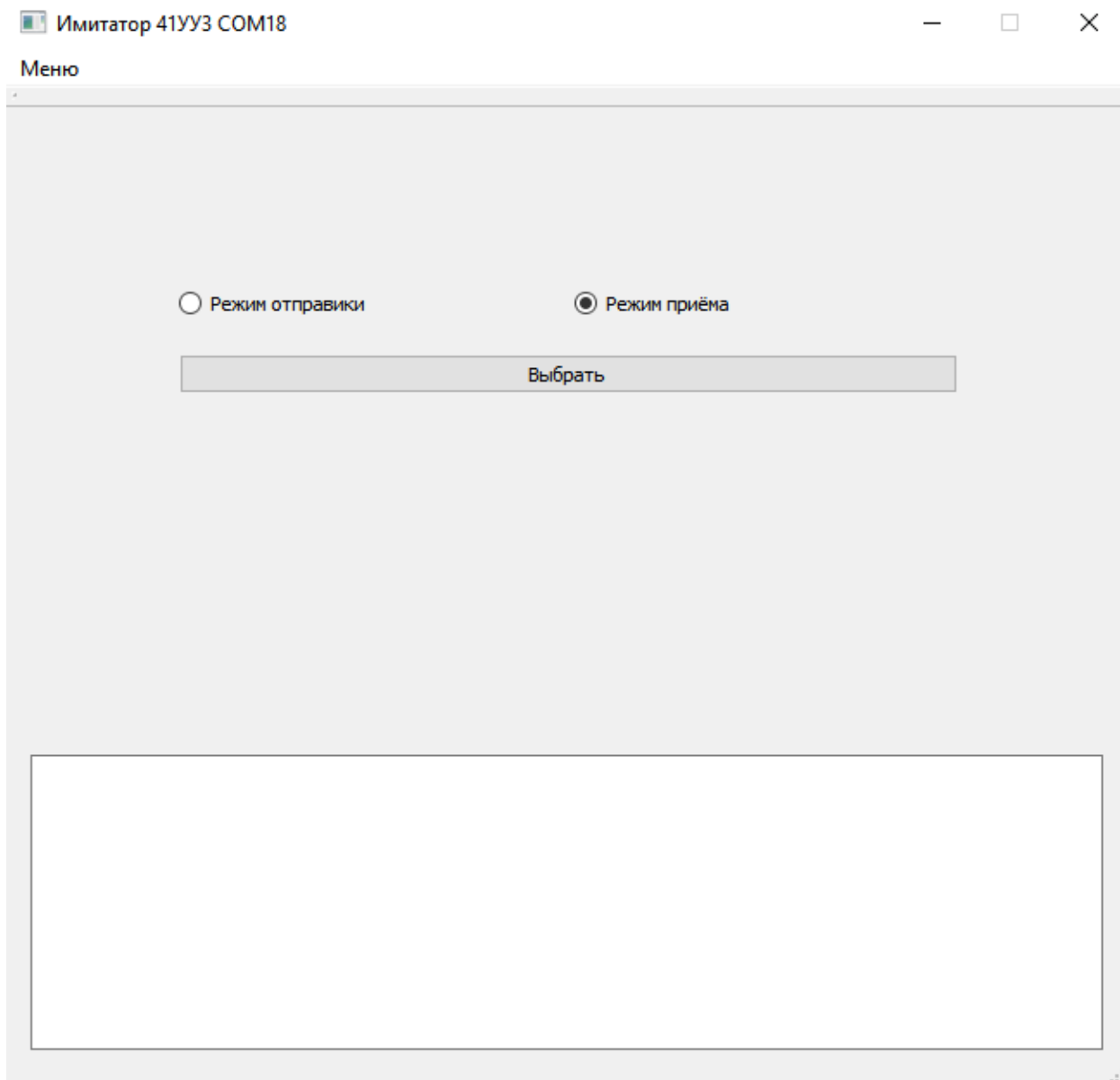


Рисунок 9 – Окно выбора имени Ethernet-адаптера

После выбора режима работы, в зависимости от выбора, приложение отображается по-разному. При выборе режима отправки (рисунок 10) в приложении будут отображаться три текстовых поля и кнопку отправки команды. В поле, содержащие текст на отправку можно вводить команды и нажатием на кнопку отправки отсылать их. Поле принятых данных будет обновляться в процессе работы приложения, отображая принятые данные. Под кнопкой отправить располагается текстовое поле лога, в которое будет записываться все происходящие с имитатором события, включая прием и отправку команд.

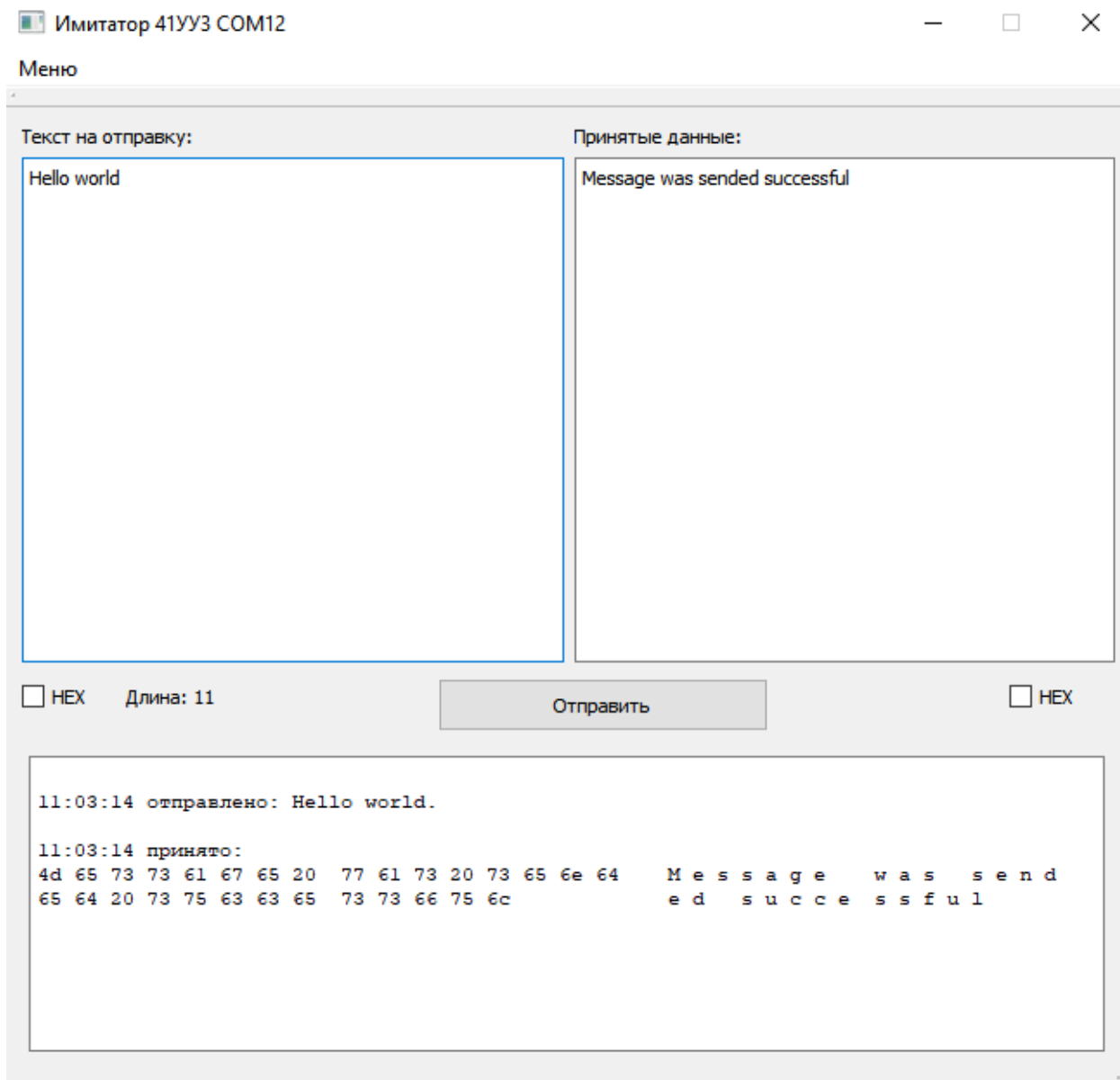


Рисунок 10 – Окно режима отправки

При выборе режима отправки приложение не сможет отправлять команды, в связи с чем пропадет кнопка отправки команд, а поле для текста на отправку будет недоступно для изменения (рисунок 11).

Поле принятых данных работает одинаково в обоих режимах, как и поле лога. Также в режиме приема пропадет кнопка перевода окна текста на отправку в байтовый вид, а также не будет отображаться его длина.

Текстовые окна также имеют ограничение на количество хранимой в них информации. После превышения количества отображенных строк окно автоматически очищается во избежание зависаний программы.

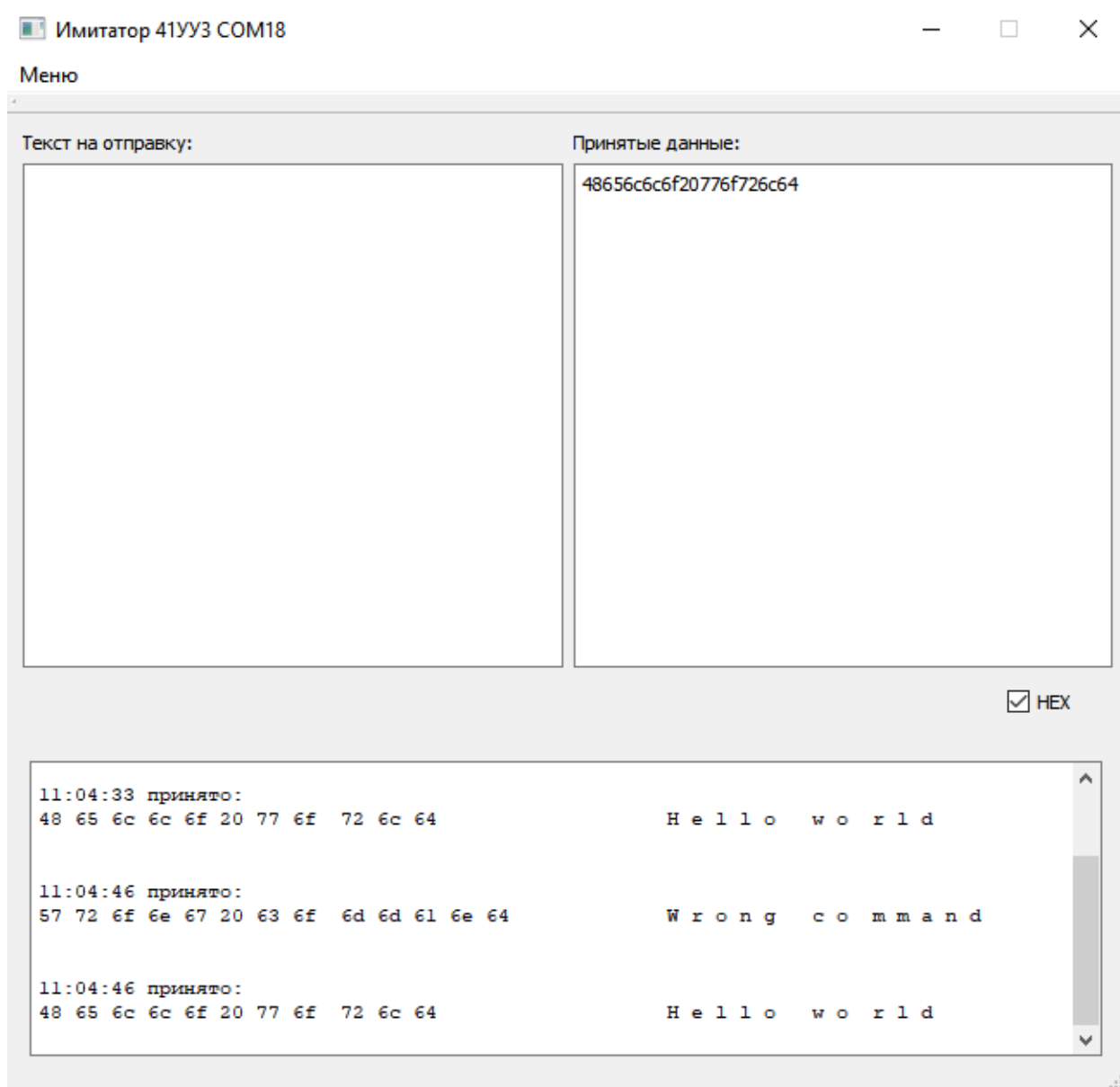


Рисунок 11 – Окно режима приема

Под окнами с отображением текста есть кнопка с названием «HEX». При нажатии этой кнопки текст в окне будет перекодирован в вид hex (рисунок 12). Эта функция необходима для того, чтобы корректно отображать команды, содержащие байты, которые некорректно отображаются в виде ASCII. Например некоторые команды могут содержать нулевые байты или байты, содержащие длину всей команды. Для корректного отображения команды в таких случаях следует отображать ее в байтовом виде, в данном случае в виде hex.

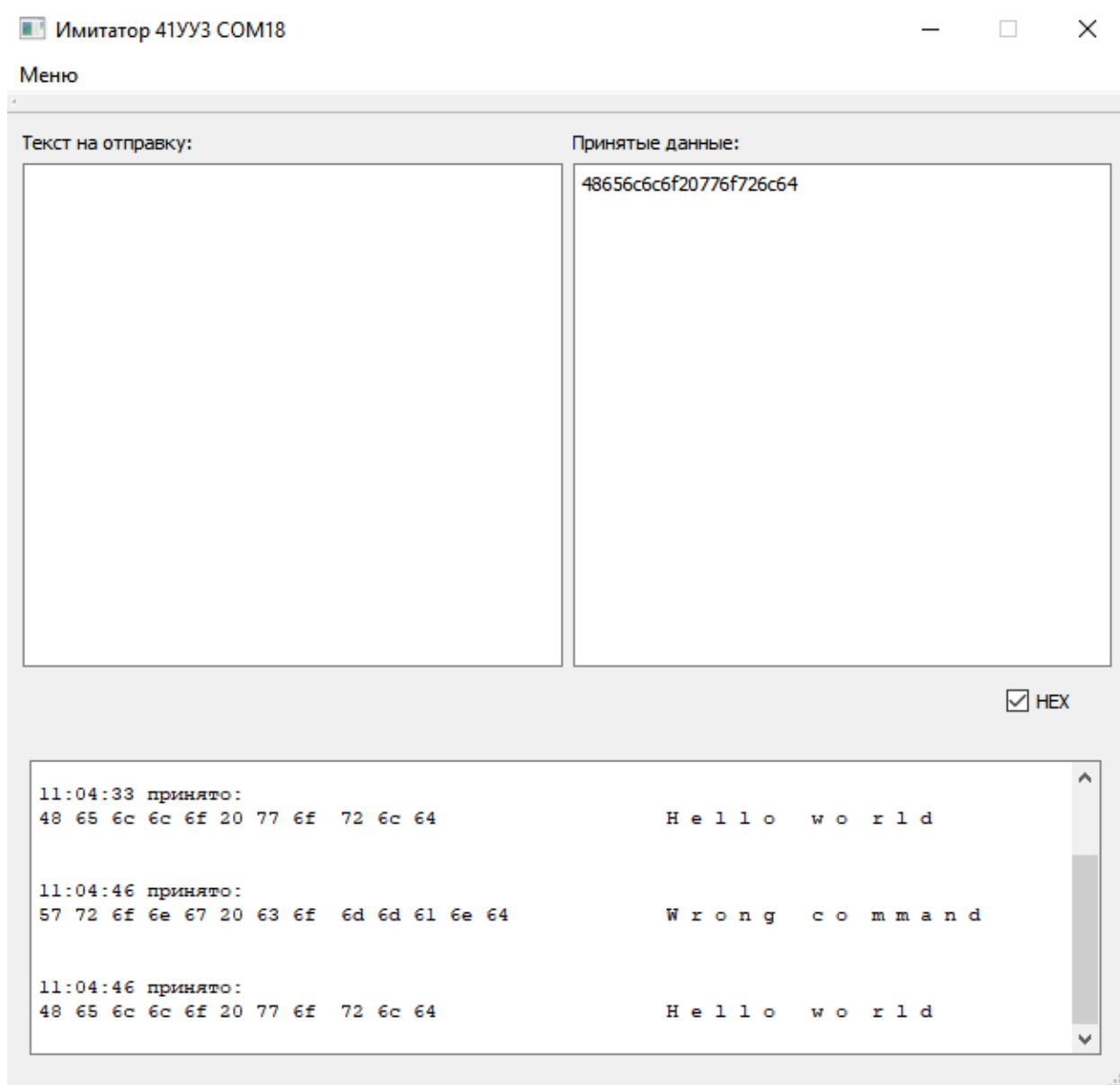


Рисунок 12 – Окно принятых данных в виде hex

После просмотра команды в байтовом виде можно повторно нажать кнопку «HEX» (рисунок 13). Это действие сменит вид отображения обратно на ASCII, что позволит причитать текстовую часть команды, если таковая имеется. У текстового поля лога, находящегося внизу приложения, отсутствует кнопка смена вида отображения. Вместо этого любая информация записываемая в него сразу отображается в обоих видах. В левой половине поле данные записаны в ASCII виде, справа находятся те же данные, но в виде hex.

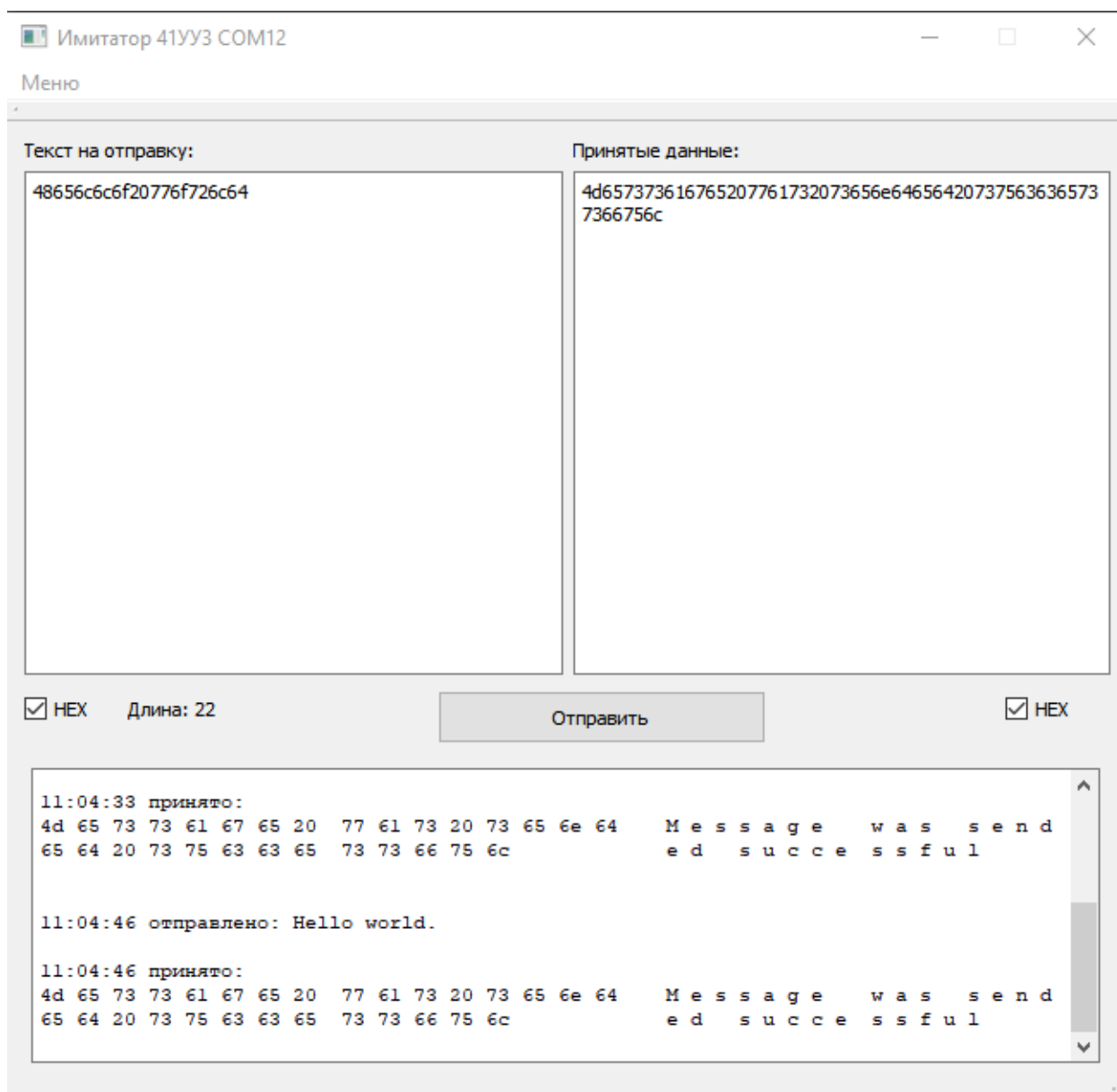


Рисунок 13 – Окно отправляемого текста в виде hex

В верхнем левом углу приложения есть меню, при наведении на которое будут доступны кнопки изменения настроек в обоих режимах и отправки их файла в режиме отправки (рисунки 14 – 16). В режиме отправки в меню будет доступна отправка команд из файлов и изменение настроек, а в режиме приема доступна только функция изменения настроек.

При выборе изменения настроек программа перезапустится и будут заново вызваны окна выбора COM-порта, Ethernet-адаптера и LocalServer (смотри рисунки 4 – 8).

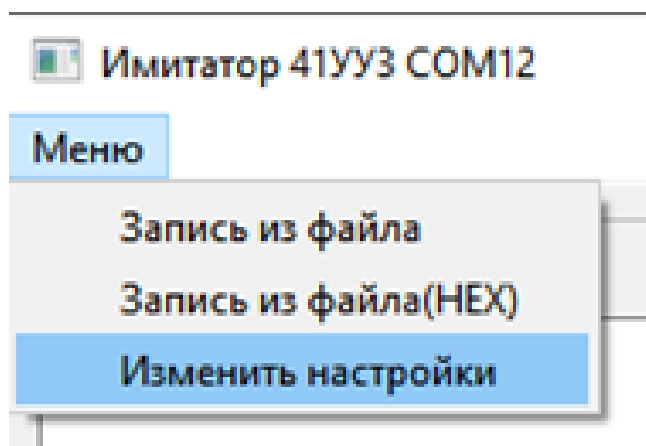


Рисунок 14 – Меню режима отправки

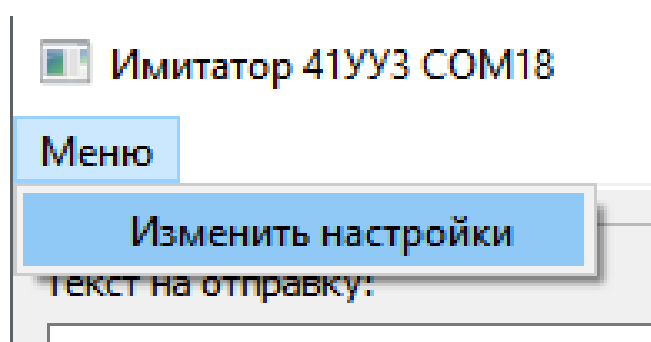


Рисунок 15 – Меню режима приема

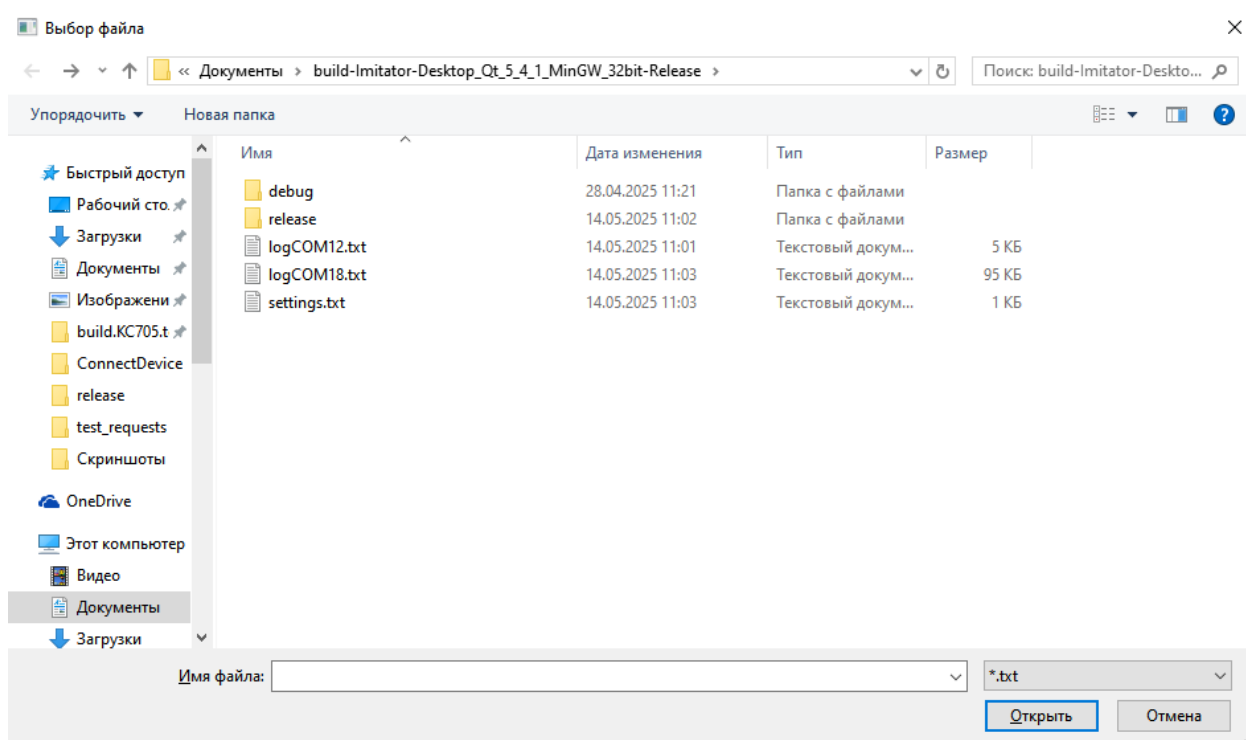


Рисунок 16 – Выбор файла при выборе опции отправки из файла

Таким образом был разработан графический интерфейс приложения, обладающий необходимым для корректной и удобной работы функционалом.

3 Реализация приложения

При реализации имитатора была составлена диаграмма компонентов (рисунок 17) [12].

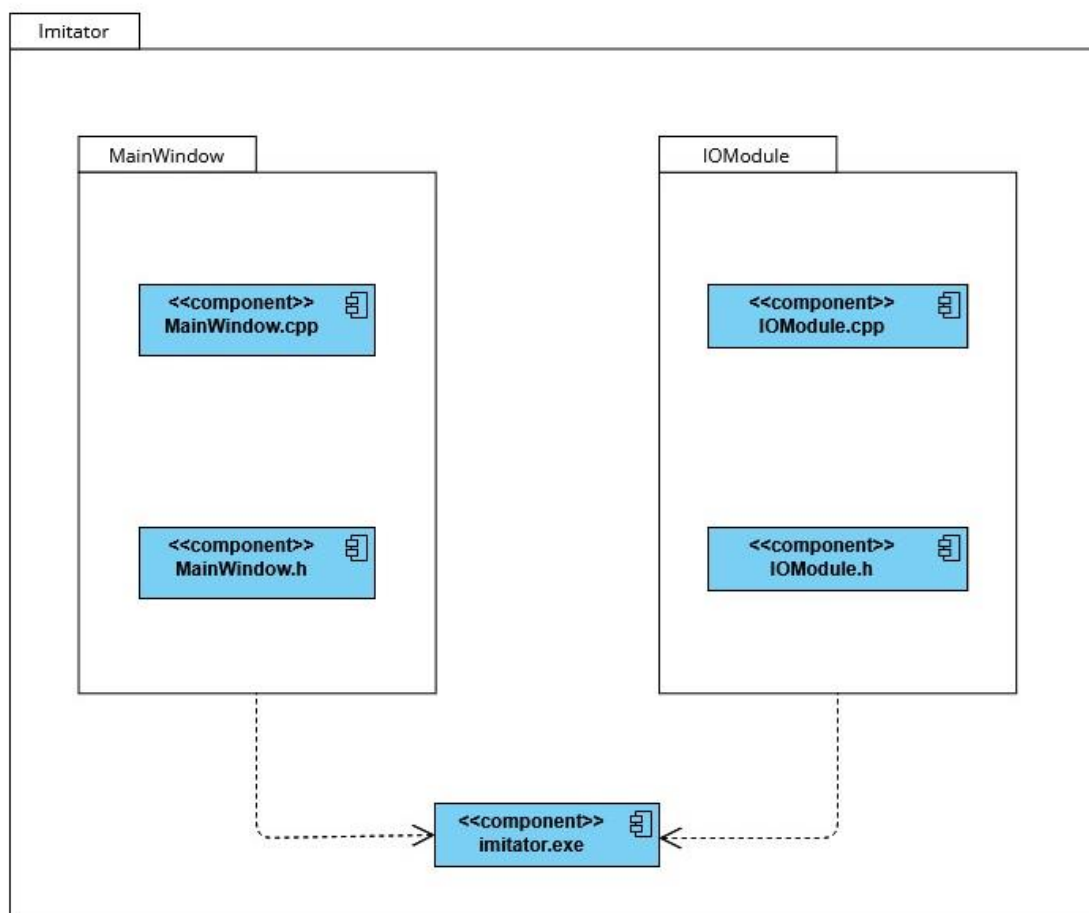


Рисунок 17 – Диаграмма компонентов

В таблице 6 приведено краткое описание для компонентов системы, описывающее каждый компонент.

Таблица 6 – Описание компонентов системы

Название компонента	Описание компонента
MainWindow.h	Заголовочный файл класса MainWindow, который отвечает за контроль IOModule и графический интерфейс

Продолжение таблицы 6

Название компонента	Описание компонента
MainWindow.cpp	Реализационный файл класса MainWindow, который отвечает за контроль IOModule и графический интерфейс
IOModule.h	Заголовочный файл, который содержит классы ComPort, EthernteAdapter, LocalServer
IOModule.h	Реализационный файл, который содержит реализации классов ComPort, EthernteAdapter, LocalServer
imitator.exe	Исполняемый файл скомпилированной и готовой к использованию программы.

При запуске программы необходимо выполнить захват Ethernet-адаптера (листинг 1) и COM-порта (листинг 2), а также считать имя LocalServer и подключиться к нему (листинг 3).

Листинг 1 – Захват Ethernet-адаптера

```
pcap_if_t *alldevs;
char errbuf[PCAP_ERRBUF_SIZE];
memset(errbuf, 'N', PCAP_ERRBUF_SIZE);

if (pcap_findalldevs(&alldevs, errbuf) != -1)
{
    for (pcap_if_t *d = alldevs; d != NULL; d = d->next)
        items << d->description;
    pcap_freealldevs(alldevs);
}

item = QInputDialog::getItem(Q_NULLPTR, "Выбор Ethernet
интерфеса", "Выберите Ethernet интерфейс", items, 0, false, &ok);

if (!ok || item.isEmpty()) exit(0);

if (pcap_findalldevs(&alldevs, errbuf) != -1)
```

```

{
    for (pcap_if_t *d = alldevs; d != NULL; d = d->next)
        if (item == d->description) {
            item = d->name;
            break;
        }
    pcap_freealldevs(alldevs);
    ether = item;
}

```

Листинг 2 – Захват COM-порта

```

for (QSerialPortInfo info: QSerialPortInfo().availablePorts())
    items << info.portName();

QString item = QInputDialog::getItem(Q_NULLPTR, "Выбор COM
порта", "Выберите COM порт", items, 0, false, &ok);

items.clear();
if (!ok || item.isEmpty()) exit(0);

auto ret = item;

```

Листинг 3 – Считывание имя LocalServer и подключение к нему

```

QString server_name = QInputDialog::getText(Q_NULLPTR,
"Название localsocket", "Введите название localsocket",
QLineEdit::Normal, tr("ServerWidget"));
client = new QLocalSocket(this);
client->connectToServer(server_name);

```

Кроме подключение COM-порт нуждается в настройке, прежде он будет готов к работе (листинг 4).

Листинг 4 – Настройка COM-порта

```

bool ComPort::InitComPort(QString name){
    QString deviceString = name.startsWith("COM") ? "\\.\\" +
name : name;
    handle = CreateFileW(reinterpret_cast<const
wchar_t*>(deviceString.utf16()),
                                FILE_SHARE_READ |
FILE_SHARE_WRITE, 0, Q_NULLPTR, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, Q_NULLPTR);

    qDebug() << GetLastError();
    if (handle == INVALID_HANDLE_VALUE) return false;

```

```

qDebug() << deviceString;

DCB dcb;
memset(&dcb, 0, sizeof(DCB));
dcb.DCBlength = sizeof(DCB);

if (GetCommState(handle, &dcb)){
    if (GetLastError()) return false;
    dcb.BaudRate = 115200;
    dcb.StopBits = TWOSTOPBITS;
    dcb.Parity = PARITY_NONE;
    dcb.ByteSize = DATABITS_8;
    SetCommState(handle, &dcb);
    if (GetLastError()) return false;
    qDebug() << GetLastError();
    memset(&dcb, 0, sizeof(DCB));
    GetCommState(handle, &dcb);
    if (GetLastError()) return false;
}
else return false;

COMMTIMEOUTS cto;
cto.ReadIntervalTimeout = 1;
cto.ReadTotalTimeoutConstant = 1;
cto.ReadTotalTimeoutMultiplier = 1;
cto.WriteTotalTimeoutConstant = 1;
cto.WriteTotalTimeoutMultiplier = 1;
SetCommTimeouts(handle, &cto);
if (GetLastError()) return false;
}

```

Также для работоспособности имитатора были реализованы протоколы взаимодействия с другими устройствами с использованием вышеописанных устройств.

Кроме этого был реализовано управление классами ComPort, EthernetAdapter, LocalServer классом MainWindow с помощью функционала слотов фреймворка Qt. Фрагмент связывания класса ComPort с MainWindow приведен в листинге 5.

Листинг 5 – Связывание класса ComPort с MainWindow

```

connect(this, &this->send_message, com, &ComPort::SendData);
connect(com, &ComPort::was_read_data, this, &this->settext);
connect(com, &ComPort::was_write_data, this, &this->
was_write);

```

Таким образом при реализации программы неоднократно были использованы возможности фреймворка Qt и библиотеки rcar, подтверждая необходимость использование их, при создании имитатора. Кроме того были реализованы интерфейсы взаимодействия с другими разрабатываемыми устройствами, что является одной из основных задач имитатора.

4 Тестирование программного имитатора

4.1 Тестирование функционала программного имитатора

Тестирование методом чёрного ящика определяется как методика тестирования, при которой функциональность тестируемого приложения (AUT) тестируется без учета внутренней структуры кода, деталей реализации и знания внутренних путей программного обеспечения. Этот тип тестирования полностью основан на требованиях и спецификациях программного обеспечения. В тестирование методом чёрного ящика фокусируются на входах и выходах программной системы, не заботясь о внутренних знаниях программ. Вышеупомянутый чёрного ящик может быть любой программной системой, которую нужно протестировать. Например, операционная система, такая как Linux, вебсайт, такой как Google, база данных, такая как Oracle, или даже просто разработанное приложение. В тестировании методом чёрного ящика можно протестировать эти приложения, просто сосредоточившись на входах и выходах, не зная их внутренней реализации кода [13].

4.1.1 Классы эквивалентности

При передачи байтовой команды, требуется ввести его в поле ввода команды. Поле «Команды» должно содержать 2 – 10000 символов: цифры, латинские буквы, кириллицу, специальные символы. Должно начинаться с буквы или цифры. Для дальнейшего составление тест-кейсов необходимо составить классы эквивалентности. Класс эквивалентности (Equivalence Class) – это группа входных или выходных данных программы, которые обрабатываются одинаковым образом и приводят к схожему поведению системы.

Другими словами, если программа должна вести себя одинаково для определённого диапазона значений, то все эти значения относятся к одному классу эквивалентности.

Составление классов эквивалентности необходимо для определение количества необходимых тест-кейсов, которые будут перекрывать все

возможные варианты обработки программой входных данных и предоставления выходных.

Классы эквивалентности представлены в таблице 7.

Таблица 7 – Классы эквивалентности

	Правильные КЭ	Неправильные КЭ
Длина передаваемого сообщения	2 – 10000 (1)	0 – 1(2) 10001 – ∞ (3)
Структура	Начинается с буквы (4) Начинается с цифры(5)	Начинается со специального символа (6)
Содержание	Буквы Латинские (7) Кириллицу (8) Цифры (9) Специальные символы (10)	Остальное (11)

После выявления классов эквивалентности и граничных значений необходимо составить тест-кейсы перекрывающие различный комбинации классов эквивалентности для последующего использования этих тест-кейсов в целях тестирования и выявления некорректной работы программы (таблица 8). Каждый тест кейс должен представлять различные классы эквивалентности или их комбинации.

При помощи формирования тест-кейсов, удовлетворяющих этому условию, возможно спроектировать минимальное количество тест-кейсов, которые полностью протестируют всю программу, перекрыв все возможные варианты её поведения. Уменьшение количество тест-кейсов повышает скорость и удобство тестирования программы, а также позволяет точнее выявить ее ошибки.

Таблица 8 – Тест – кейсы

Номер теста	Классы Эквивалентности	Входные данные	Выходные данные
1	1,4,8,10	Привет!)	Успешно
2	2,4,7	Ы	Ошибка: Меньше 2х символов
3	3,5,9	<20000>	Ошибка: Больше 10000х символов
4	1,6,9,10	№1	Ошибка: Начинается со специального символа
5	1,11	文7676	Ошибка: Содержит неподдерживаемый символ

Таким образом были спроектированы тест-кейсы.

4.1.2 Функциональная диаграмма и таблица решений

Чтобы отправить команду, пользователь должен провести первоначальную настройку имитатора. Для этого он должен выбрать COM-порт, Ethernet-адаптер и LocalServer для получения и передачи информации. COM-порт, Ethernet-адаптер и LocalServer должны быть корректны и доступны для взаимодействия. Далее пользователь должен ввести команду длиной от 2 до 10000 символов. Команда должна состоять из букв, цифр, специальных символов.

Для составления таблицы решений необходимо описать все возможные причины и следствия поведения программной логики. После этого

необходимо связать причины и следствия связями. Набор причин и следствий приведен в таблице 9.

Таблица 9 – Набор причин и следствий

П1	Выбран корректный COM- порт
П2	Выбран корректный Ethernet-адаптер
П3	Выбрано корректное имя LocalServer
П4	Программа смогла открыть COM-порт
П5	Программа смогла захватить Ethernet-адаптер
П6	Программа смогла подключиться к LocalServer
П7	Корректная длина введенной команды
П8	Введенная команда из корректных символов
С1	Сообщение: «Программа не смогла открыть COM-порт»
С2	Сообщение: «Программа не смогла захватить Ethernet-адаптер»
С3	Сообщение: «Программа не смогла подключиться к LocalServer»
С4	Сообщение: «Некорректная длина введенной команды»
С5	Сообщение: «Некорректные символы в введенной команде»
С6	Сообщение: «Команда успешно отправлена»
С1	Сообщение: «Программа не смогла открыть COM-порт»

В качестве графического представление причин и следствий приведена функциональная диаграмма [14] на рисунке 18. Эта диаграмма отображает

связи между причинами и следствиями, что помогает определить поведение программы в конкретных случаях и позволяет составить тест-кейсы.

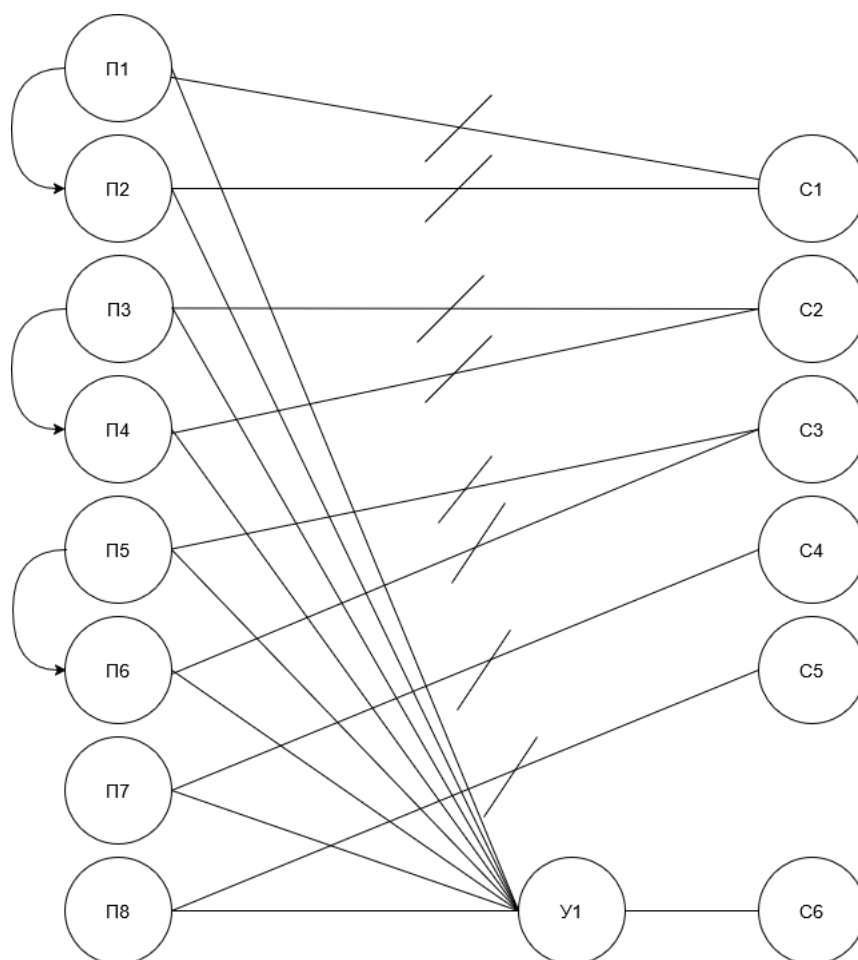


Рисунок 18 – Функциональная диаграмма

Таблицы с представлением допустимых ограничений в случае полного перебора всех комбинаций определяют 256 тестовых сценария. С целью сокращения количества вхождений во множество допустимых тестов используется комбинаторный метод тестирования программного обеспечения, называемый попарным тестированием, который для каждой пары входных параметров системы проверяет все возможные дискретные комбинации этих параметров. Данная техника позволяет уменьшить количество тестовых сценариев с минимальным вмешательством в качество проводимого тестирования.

Результат применения техники попарного тестирования оставил из 256 тестовых сценариев лишь 21. Тестовые сценарии после применения техники попарного тестирования представлены на таблице 10.

Таблица 10 – Тестовые сценарии после применения техники попарного тестирования

	П1	П2	П3	П4	П5	П6	П7	П8
1	Да	Да	Да	Нет	Нет	Нет	0 – 1	A..Z
2	Да	Да	Нет	Нет	Нет	Да	10001 – ∞	0..9
3	Да	Нет	Нет	Нет	Да	Да	0..10000	@, №, !, ()
4	Нет	Нет	Да	Да	Нет	Нет	0..10000	@, №, !, ()
5	Нет	Да	Да	Нет	Да	Да	10001 – ∞	0..9
6	Нет	Нет	Нет	Да	Да	Нет	0 – 1	A..Z
7	Нет	Да	Нет	Да	Да	Нет	0..10000	0..9
8	Нет	Нет	Да	Да	Нет	Да	10001 – ∞	A..Z
9	Да	Да	Нет	Нет	Да	Да	0 – 1	@, №, !, ()
10	Да	Нет	Да	Да	Нет	Да	0..10000	0..9
11	Да	Да	Нет	Да	Да	Нет	10001 – ∞	A..Z
12	Да	Да	Да	Нет	Нет	Да	0 – 1	@, №, !, ()
13	Да	Нет	Нет	Да	Да	Нет	0 – 1	0..9
14	Да	Да	Да	Нет	Да	Да	0..10000	A..Z
15	Да	Нет	Да	Да	Нет	Нет	10001 – ∞	@, №, !, ()
16	Нет	Нет	Нет	Нет	Да	Да	10001 – ∞	@, №, !, ()

Продолжение таблицы 10

	П1	П2	П3	П4	П5	П6	П7	П8
17	Нет	Да	Да	Да	Да	Нет	0 – 1	0..9
18	Нет	Да	Нет	Нет	Нет	Да	0..10000	A..Z
19	Нет	Нет	Да	Да	Да	Да	10001 – ∞	@, №, !, ()
20	Нет	Да	Да	Нет	Нет	Нет	0 – 1	0..9
21	Нет	Нет	Нет	Да	Да	Да	0 – 1	A..Z

Результат выполнения каждого тестового сценария, определенного на таблице 5, привел к успешному выполнению тестирования. Таким образом, можно сделать вывод о том, что результат выполнения технического задания произведен успешно и разработанное программное обеспечение работает корректно.

4.2 Анализ сложности кода открытия СОМ-порта

Для измерения характеристик и критериев качества используют метрики. Метрика качества программ – это система измерений качества программ. Эти измерения могут проводиться на уровне критериев качества программ или на уровне отдельных характеристик качества. В первом случае система измерений позволяет непосредственно сравнивать программы по качеству, при этом сами измерения не могут быть проведены без субъективных оценок свойств программ. Во втором случае измерения характеристик можно выполнить объективно и достоверно, но оценка качества ПО в целом будет связана с субъективной интерпретацией получаемых оценок. В листинге 6 приведён код, проверяющий корректность введённого СОМ-порта, на основании которого будет производится расчёт метрик [15].

Листинг 6 – Класс и перечисление, проверяющие корректность введённого

```
bool ComPort::InitComPort(QString name){
    QString deviceString = name.startsWith("COM") ? "\\.\\" +
name : name;
```

```

    handle = CreateFileW(reinterpret_cast<const
wchar_t*>(deviceString.utf16()),
                                FILE_SHARE_READ |
FILE_SHARE_WRITE, 0, Q_NULLPTR, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, Q_NULLPTR);

    qDebug() << GetLastError();
    if (handle == INVALID_HANDLE_VALUE) return false;
    qDebug() << deviceString;

    DCB dcb;
    memset(&dcb, 0, sizeof(DCB));
    dcb.DCBlength = sizeof(DCB);

    if (GetCommState(handle, &dcb)){
        if (GetLastError()) return false;
        dcb.BaudRate = 115200;
        dcb.StopBits = TWOSTOPBITS;
        dcb.Parity = PARITY_NONE;
        dcb.ByteSize = DATABITS_8;
        SetCommState(handle, &dcb);
        if (GetLastError()) return false;
        qDebug() << GetLastError();
        memset(&dcb, 0, sizeof(DCB));
        GetCommState(handle, &dcb);
        if (GetLastError()) return false;
    }
    else return false;

    COMMTIMEOUTS cto;
    cto.ReadIntervalTimeout = 1;
    cto.ReadTotalTimeoutConstant = 1;
    cto.ReadTotalTimeoutMultiplier = 1;
    cto.WriteTotalTimeoutConstant = 1;
    cto.WriteTotalTimeoutMultiplier = 1;
    SetCommTimeouts(handle, &cto);
    if (GetLastError()) return false;
}

```

Расчет метрики SLOC на примере кода функции из листинга 1 представлен на таблице 11 [16].

Таблица 11 – Расчет метрики SLOC для кода из листинга 1

Метрика	Результат
Физические строки кода	40
Исходные строки кода	37

Продолжение таблица 11

Метрика	Результат
Строки комментариев	0
Пустые строки	4

Результат расчета метрики показывает, что код функции, представленный на листинге 1, не имеет достаточной сложности, его просто анализировать и использовать.

4.3 Вычисление и определение метрики Холстеда

Приведённый фрагмент кода состоит из попытки открытия СОМ-порта по имени и проверки отсутствия ошибок. Затем СОМ-порт конфигурируется путем задания стандартных настроек и повторной проверки отсутствия ошибок. После этого происходит настройка задержек для записи и чтения СОМ-порта и, в случае отсутствия ошибок, происходит возврат информации о успешном открытии СОМ-порта.

Традиционной характеристикой размера программ является количество строк исходного текста. Под строкой понимается любой оператор программы, поскольку именно оператор, а не отдельно взятая строка является тем интеллектуальным «квантом» программы, опираясь на который можно строить метрики сложности ее создания.

Оценка размера программы недостаточна для принятия решения о ее сложности, но вполне применима для классификации программ, существенно различающихся объемами. При уменьшении различий в объеме программ на первый план выдвигаются оценки других факторов, оказывающих влияние на сложность. Таким образом, оценка размера программы есть оценка по номинальной шкале, на основе которой определяются только категории

программ без уточнения оценки для каждой категории. К группе оценок размера программ можно отнести метрику Холстеда [17].

Основу метрики Холстеда составляют четыре измеряемых характеристики программы:

n_1 – число уникальных операторов программы, включая символы;

разделители, имена процедур и знаки операций (словарь операторов), n_2

– число уникальных операндов программы (словарь операндов)

N_1 – общее число операторов в программе, N_2 – общее число операндов в программе.

В таблице 12 приведены результаты расчёта количества операторов и операндов для тестируемого блока кода.

Таблица 12 – Операторы и операнды

№ п/п	Операторы	Число операторов	Операнды	Число операндов
1	startsWith	1	deviceString	3
2	CreateFileW	1	name	4
3	GetLastError	5	dcb	11
4	if	5	handle	6
5	<<	3	cto	7
6	(	26	QString	2
7	)	26	DCB	4
8	{	2	0	3
9	}	2	"\\\\.\\"	1
10	qDebug	3	"COM"	1
11	=	11	FILE_SHARE_READ	1
12	==	1	FILE_SHARE_WRITE	1
13	?:	1	Q_NULLPTR	2
14	reinterpret_cast	1	OPEN_EXISTING	1
15	+	1	FILE_ATTRIBUTE_NORMAL	1
16	===	2	BaudRate	1
17	&	5	115200	1
18		1	StopBits	1
19	return	6	Parity	1
20	false	6	TWOSTOPBITS	1
21	else	1	PARITY_NONE	1
22	sizeof	3	ByteSize	1
23	GetCommState	2	DATABITS_8	1
24	const	1	COMMTIMEOUTS	1
25	wchar_t	1	1	5
26	*	1	ReadIntervalTimeout	1
27	.	12	ReadTotalTimeoutConstant	1
28	;	28	ReadTotalTimeoutMultiplier	1
29	SetCommState	1	WriteTotalTimeoutConstant	1
30	memset	2	WriteTotalTimeoutMultiplier	1
31			115200	1

Произведем расчет метрики Холстеда на примере кода из листинга 6.

Длина кода программы рассчитывается по формуле:

$$N=N_1+N_2=155+64=219,$$

где N_1 – общее число операторов в программе, а N_2 – общее число операндов в программе.

Словарь программы рассчитывается по формуле:

$$\eta=\eta_1+\eta_2=31+30=61,$$

где η_1 – число уникальных операторов программы, а η_2 – число уникальных операндов программы.

Потенциальный объем программы:

$$\hat{N}=\eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 = 300.9327.$$

Объем кода программы рассчитывается по формуле:

$$V=N \log_2 \eta = 1298.8317,$$

где N – длина программы;

η_1 – словарь программы.

Сложность кода программы рассчитывается по формуле:

$$D=\frac{\eta_1}{2} * \frac{N_2}{\eta_2} = 33.07.$$

Усилие для написания и понимания кода рассчитывается по формуле:

$$E=D*V=42952.364319.$$

Время написания кода рассчитывается по формуле:

$$T=\frac{E}{18}=2386.2424.$$

Количество доставленных кодом ошибок рассчитывается по формуле:

$$B=\frac{V}{3000}=0.432439.$$

С помощью объема Холстеда (HV), цикломатической сложности (CC) и количества строк куда можно рассчитать одну из самых популярных метрик для оценки кода – комплексный показатель качества кода (Maintainability Index) [18]. Это комплексный показатель качества кода может принимать значения от 0 до 100 и показывает относительную сложность поддержки кода.

Чем больше значение этой метрики, тем легче поддерживать код. Эта метрика рассчитывается метрика по следующей формуле:

$$MI = \text{MAX}(0, (171 - 5.2 * \ln(HV) - 0.23 * CC - 16.2 * \ln(LoC)) * 100 / 171).$$

Объем Холстеда, рассчитанный выше, равняется 1298.8317. Цикломатическая сложность рассчитывается как количество операторов if плюс количество операторов цикла плюс 1 и равняется 4. Количество строк кода равняется 41. Таким образом комплексный показатель качества кода равняется 42.47964104.

В результате расчёта сложности поддержки кода по метрике комплексного показателя качества кода вышло, что программный код рассматриваемой функции оценивается как среднетяжелый. Удовлетворение требуемым ограничениям определяет код рассматриваемой программы как успешный вариант реализации.

4.4 Автоматизированное тестирование

Для автоматизированного тестирования отправки сообщения используется отправка сообщений из файла с заданной задержкой. Это позволяет автоматически произвести тестирование на большом количестве сообщений, а вывод информации о сообщении, который происходит в любом режиме работы программы, позволяет увидеть последние сообщения, время их отправки и описание ошибок. Функция `MainWindow::on_action_triggered` используется для выбора файла с тестовыми сообщениями. Функция `MainWindow::parse_file_data` читает файл и отправляет заданные в нём сообщения с заданной в файле задержкой.

В листинге 7 приведён код основных тестовых функций.

Листинг 7 – Функции тестирования корректности отправки сообщений

```
void MainWindow::on_action_triggered()
{
    iter = 0;
    file_data.clear();
    hex = false;
    timer->stop();
}
```

```

    QString filename = QFileDialog::getOpenFileName(this, "Выбор
файла", "./", "*.txt");
    QFile *f = new QFile(filename);
    f->open(QFile::ReadOnly);
    file_data = f->readAll();
    f->close();
    parse_file_data();
}

void MainWindow::parse_file_data(){
    qDebug() << "parse_file_data";
    qDebug() << QTime::currentTime().toString();
    timer->stop();
    QByteArray arr;
    bool was = false;
    int time = 0;
    for (; iter < file_data.size(); ++iter){
        if (was){
            if ('0' <= file_data[iter] && file_data[iter] <=
'9') time = time * 10 + (file_data[iter] - '0');
            else break;
            continue;
        }

        if (file_data[iter] == '\\n') break;
        if (file_data[iter] == '#'){
            was = true;
            continue;
        }

        arr.append(file_data[iter]);
    }
    qDebug() << "arr" << arr;
    qDebug() << "time" << time;

    if (arr.size() == 0) return;

    if (hex) arr = fromHex(arr);

    ui->pushButton_2->setEnabled(false);
    this->send_message(arr);
    QString t = "\\n" + QTime::currentTime().toString() + "
отправлено: " + arr + ".";
    file->write(t.toUtf8());
    file->flush();
    ui->textEdit_3->append(t);

    if (time != 0){
        qDebug() << "setInterval";
        timer->setInterval(time);
        timer->start();
    }

    if (iter == file_data.size()){

```

```

qDebug() << "iter == file_data.size()";
iter = 0;
file_data.clear();
hex = false;
timer->stop();
}
}

```

После выбора файла произойдет отправка команд, которые он содержит. После отправки команд, приложение выведет лог на экран, в поле принятых данных и нижнее текстовое поле, содержащее более подробную информацию. Нижнее поле отображает всю принятую и отправленную информацию в виде читаемого текста (ASCII) и байтовом виде (HEX). Результаты автоматизированного тестирования представлены на рисунках 19.

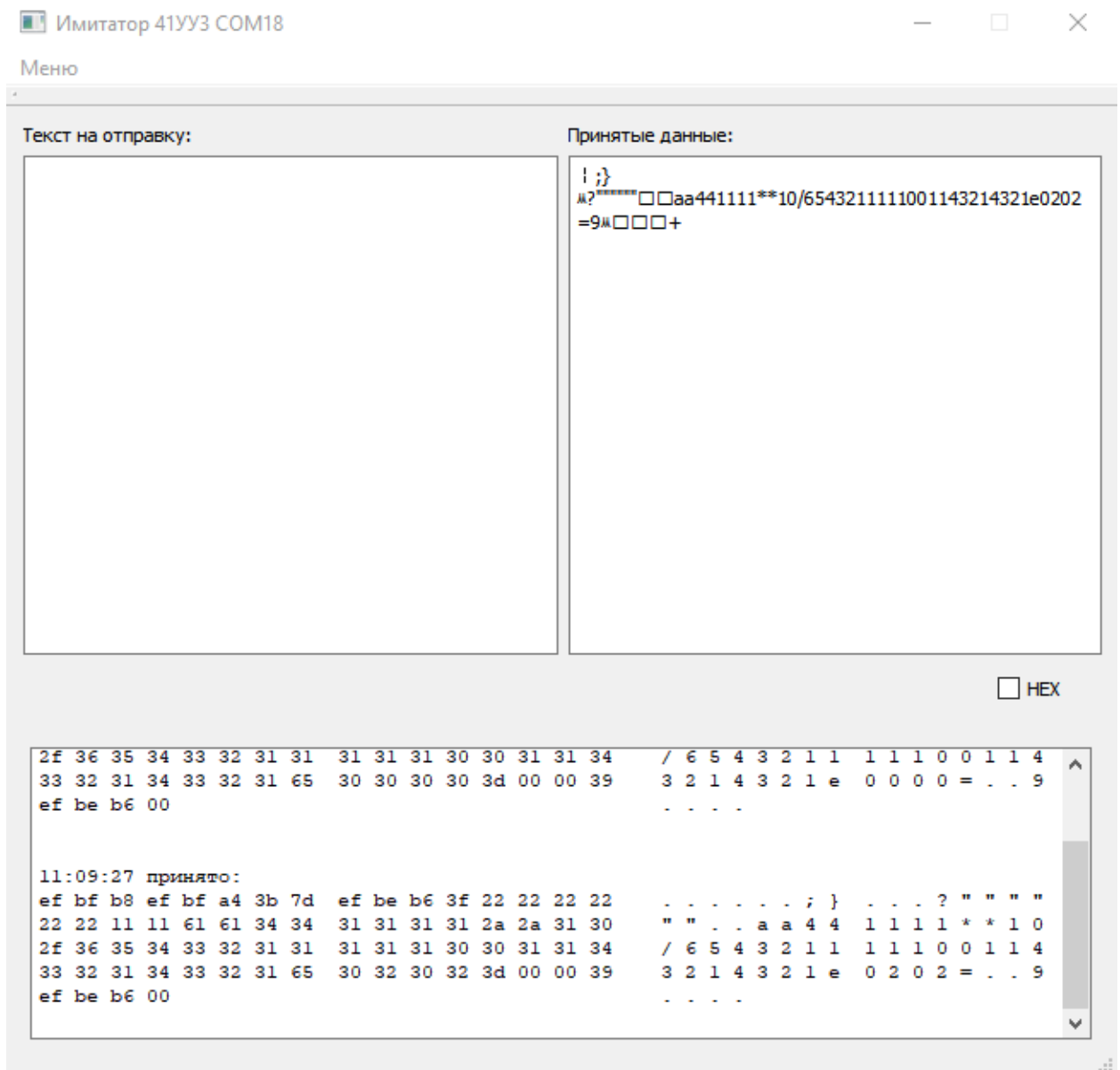


Рисунок 19 – Лог тестирования

Также результаты автоматизированного тестирования с помощью другого файла представлены на рисунке 20.

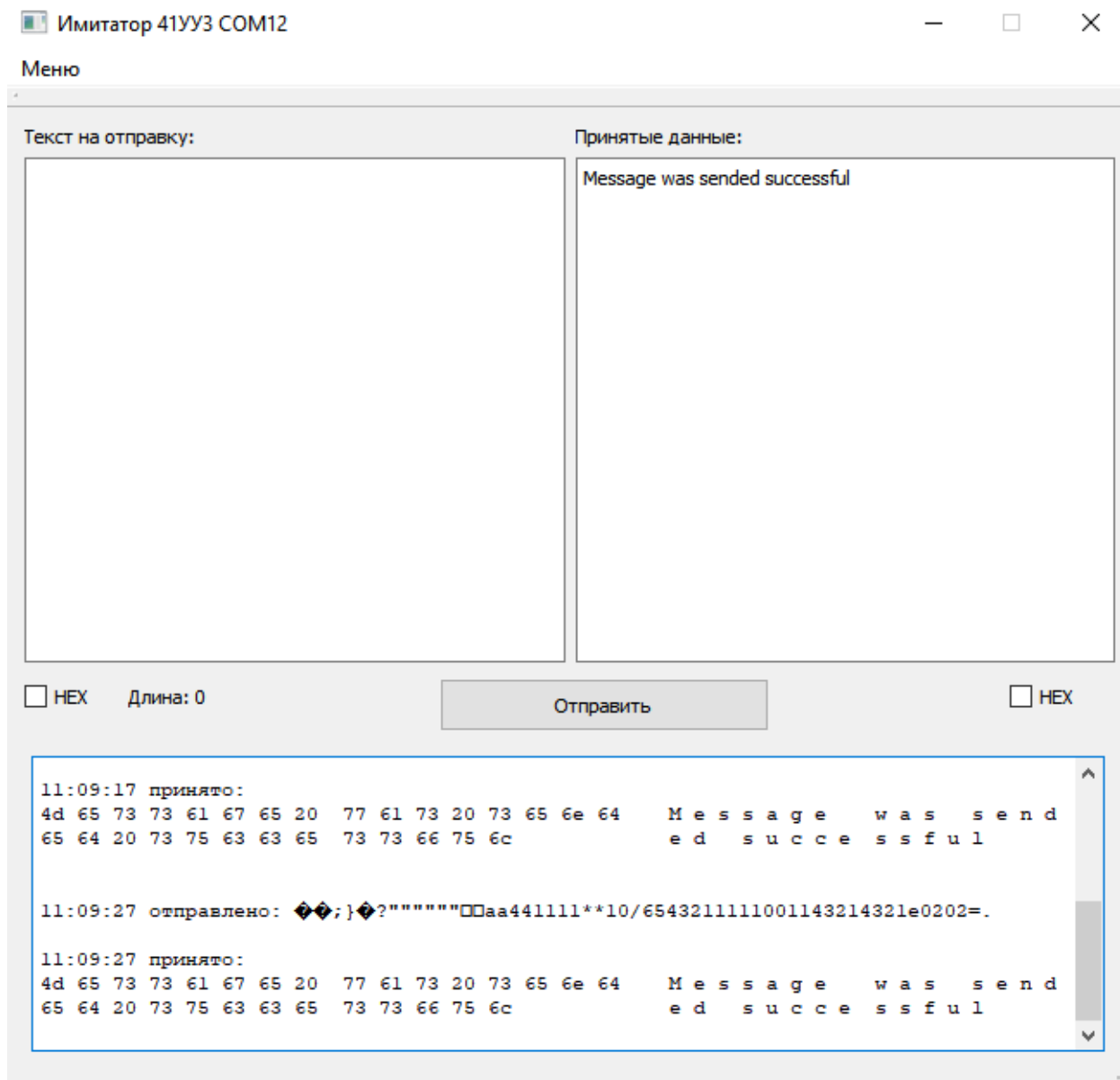


Рисунок 20 – Лог тестирования

В результате проведенного тестирования были выявлены ошибки с одновременным приемом сервером сообщений с различных клиентов. Тестирование отправки сообщений с различными интервалами прошло успешно, так же, как и отправки единичного сообщения. После исправления найденных ошибок рассматриваемая программа оценивается как корректно работающий.

5 Планирование разработки и оценка бюджета

Для расчета бюджета необходимо составить список ресурсов, необходимых для реализации проекта (материальных и трудовых), а также составить список задач проекта.

По данным портала russia.gorodrabot.ru [19] на 2025 год средняя зарплата для вышеперечисленных работников в России приведена в таблице 13.

Таблица 13 – Таблица доходов работников

Профессия	Средняя зарплата за месяц	Средняя зарплата за час (округлённо до 50 ₽/ч)
Аналитик	100 000 ₽	600 ₽/ч
Программист	150 000 ₽	900 ₽/ч
Проектировщик	97 000 ₽	600 ₽/ч
Дизайнер	75 000 ₽	550 ₽/ч
Тестировщик	80 000 ₽	550 ₽/ч

На основании стоимости необходимого оборудования, материалов и доходов работников была составлена таблица 14.

Таблица 14 – Таблица ресурсов

Название ресурса	Тип	Стандартная ставка	Начисление
Аналитик	Трудовой	600,00 ₽/ч	Пропорциональное
Программист	Трудовой	900,00 ₽/ч	Пропорциональное
Тестировщик	Трудовой	550,00 ₽/ч	Пропорциональное
Дизайнер	Трудовой	550,00 ₽/ч	Пропорциональное
Проектировщик	Трудовой	600,00 ₽/ч	Пропорциональное
Компьютер	Материальный	60 000,00 ₽	В начале
Бумага	Материальный	500,00 ₽	В начале

Продолжение таблицы 14

Название ресурса	Тип	Стандартная ставка	Начисление
Принтер	Материальный	5 000,00 Р	В начале
COM-порт	Материальный	1 500,00 Р	В начале
Ethernet-адаптер	Материальный	2 000,00 Р	В начале
Кабель	Материальный	200,00 Р	В начале

На основании ресурсов и оценки сложности планируемых при проектировании работ были выделены задачи и время на их выполнение (рисунок 21).

	Название задачи	Длительность	Начало	Окончание	Названия ресурсов	Предшественники
1	Начало проекта	42 дней	Пн 07.04.25	Вт 03.06.25		
2	Закупка оборудования	1 день	Пн 07.04.25	Пн 07.04.25	COM-порт[1]; Ethernet-адаптер[1]; Бумага[1]; Кабель[1]; Компьютер[1]; Принтер[1]	
3	Постановка задачи разработки приложения	5 дней	Вт 08.04.25	Пн 14.04.25		
4	Анализ системы разрабатываемых устройств и постановка задачи	3 дней	Вт 08.04.25	Чт 10.04.25	Аналитик; Компьютер[1]	2
5	Анализ функциональных требований	2 дней	Пт 11.04.25	Пн 14.04.25	Аналитик; Проектировщик; Компьютер[1]	4
6	Проектирование приложения	9 дней	Вт 15.04.25	Пт 25.04.25		
7	Выбор технологии разработки	3 дней	Вт 15.04.25	Чт 17.04.25	Проектировщик; Аналитик; Программист; Компьютер[1]	5
8	Проектирование архитектуры классов приложения	5 дней	Пт 18.04.25	Чт 24.04.25	Программист; Проектировщик; Бумага[1]; Компьютер[1]; Принтер[1]	7
9	Оценка бюджета разработки приложения	1 день	Пт 25.04.25	Пт 25.04.25	Аналитик; Проектировщик; Компьютер[1]	8
10	Реализация программных средств	11 дней	Пн 28.04.25	Пн 12.05.25		
11	Разработка графического интерфейса	5 дней	Пн 28.04.25	Пт 02.05.25	Дизайнер; Программист; Компьютер[1]; Бумага[1]; Принтер[1]	9
12	Разработка взаимодействия	6 дней	Пн 05.05.25	Пн 12.05.25	Аналитик; Программист; Проектировщик; COM-порт	11
13	Тестирование	8 дней	Вт 13.05.25	Чт 22.05.25		
14	Тестирование функционала	3 дней	Вт 13.05.25	Чт 15.05.25	COM-порт[1]; Ethernet-адаптер[1]; Кабель[1]; Компьютер[1]; Принтер[1]	12
15	Анализ сложности кода	3 дней	Пт 16.05.25	Вт 20.05.25	Бумага[1]; Компьютер[1]; Принтер[1]; Тестировщик	14
16	Автоматизированное тестирование	2 дней	Ср 21.05.25	Чт 22.05.25	Программист; Тестировщик; Компьютер[1]	15
17	Оформление пояснительной записки	8 дней	Пт 23.05.25	Вт 03.06.25	Бумага[1]; Принтер[1]; Компьютер[1]	16
18	Завершение проекта	0 дней	Ср 04.06.25	Ср 04.06.25		17

Рисунок 21 – Список задач проекта

Для визуализации описанного плана разработки была составлена диаграмма Ганта [20], приведенная на рисунке 22.

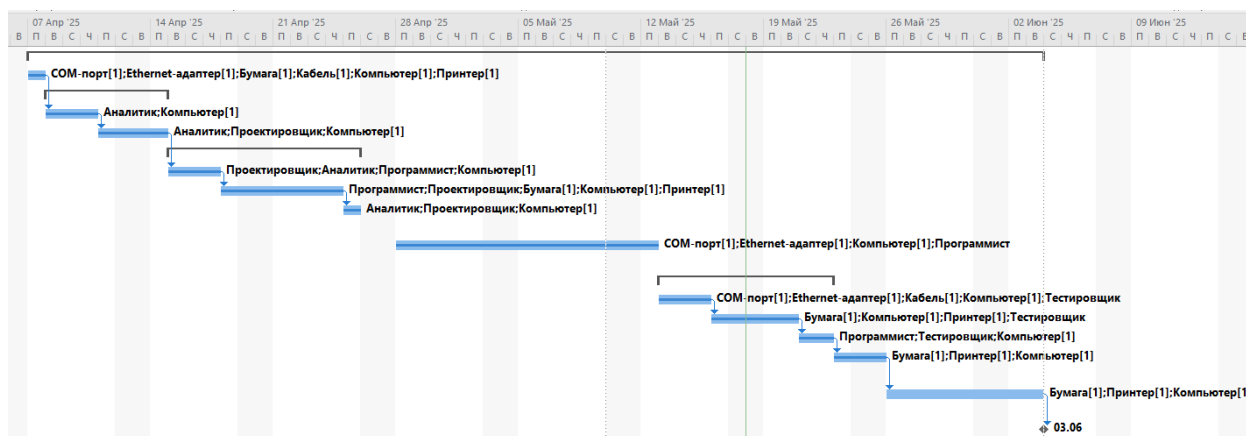


Рисунок 22 – Диаграмма Ганта

Из составленного плана проекта видно, что для его реализации потребуется 42 рабочих дня. Реализация проекта запланирована на период с 7 апреля 2025 года по 3 июня 2025 года и составляет 58 календарных дней с учетом выходных. На основании этих данных был рассчитан бюджет проекта, который составляет 507 300 рублей.

Заключение

В рамках выпускной квалификационной работы был спроектирован и разработан программный имитатор для отладки взаимодействия устройства доверенного управления аппаратурой передачи данных с другими разрабатываемыми устройствами.

Был проведен анализ системы разрабатываемых устройств и выработаны функциональных требований, необходимые для проектирования имитатора. Кроме этого, была разработана диаграмма вариантов использования с основными возможными действиями, основные из которых подробно описанными в таблицах.

На стадии проектирования приложения были выбран язык программирования C++ с фреймворком Qt. Так же было принято решение использовать библиотеку pcar для взаимодействия с Ethernet. Было произведено проектирование структуры классов, создана диаграмма классов, которые были подробно описаны. Также был спроектирован графический интерфейс.

Спроектированный графический интерфейс имеет все, заявленные на этапе функционального проектирования, функции. Также спроектированный интерфейс удобен для использования и предоставляет достаточное для эффективной работы количество информации.

На этапе реализации был написан код для программы. При создании приложения была использована среда разработки QtCreator в операционной системе Windows 10.

Была создана диаграмма компонентов и были реализованы интерфейсы взаимодействия с другими разрабатываемыми устройствами.

Тестирование программного средства проводилось в QtCreator в операционной системе Windows 10. Благодаря некоторым тестам были выявлены ошибки, которые были исправлены. Кроме этого была посчитана метрика кода и создано автоматизированное тестирование.

В ходе работы была произведено планирование разработки и оценка бюджета с Microsoft Project. Длительность проекта составила 58 календарных дней с учетом выходных, а суммарные затраты равны 507 300 рублей.

Таким образом в результате был спроектирован и разработан программный имитатор для отладки взаимодействия устройства доверенного управления аппаратурой передачи данных с другими разрабатываемыми устройствами.

Список использованных источников

1. Диаграмма вариантов использования // Flex berry [Электронный ресурс]. URL: https://flexberry.github.io/ru/fd_use-case-diagram.html/ (дата обращения: 02.05.2025).
2. Шлее, М. Qt 5.10. Профессиональное программирование на C++ / Шлее, М. – СПб: БХВ-Петербург, 2018. – 1072 с.
3. Прохоренок, Н. А. Qt 6 Разработка оконных приложений на C++ / М. Шлее. – СПб: БХВ-Петербург, 2022 – 512 с.
4. Рейзлин, В.И. Язык C++ и программирование на нём: учебное пособие / В.И. Рейзлин. – Томск: Изд-во Томского политехнического университета, 2021. – 208 с.
5. Страуструп, Б. Язык программирования C++ / Б. Страуструп. – М.: Бином, 2015. – 1136 с.
6. Шилдт, Г. C++: базовый курс / Г. Шилдт. – М.: Вильямс, 2018. – 624 с.
7. OpenCommPort function (winbase.h) // Microsoft [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-opencommport> (дата обращения: 04.05.2025).
8. tcpdump/libpcap // The Tcpdump Group [Электронный ресурс]. – URL: <https://www.tcpdump.org/> (дата обращения: 12.12.2024).
9. Кендалл, К.Е. Системный анализ и проектирование / К.Е. Кендалл, Дж.Е. Кендалл. – М.: БИНОМ. Лаборатория знаний, 2010. – 743 с.
10. Шнайдерман, Б. Разработка пользовательских интерфейсов / Б. Шнайдерман, К. Плаза. – М.: ДМК Пресс, 2005. – 800 с.
11. Раскин, Дж. Человечный интерфейс: Новые направления в проектировании компьютерных систем / Дж. Раскин. – СПб.: Символ-Плюс, 2000. – 272 с.
12. Фаулер, М. UML. Основы / М. Фаулер. – М.: Символ-Плюс, 2003. – 192 с.

13. Бейзер, Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем / Б. Бейзер. – СПб.: Питер, 2004. – 318 с.
14. Моисеев, А.Н. Основы языка UML: учеб. пособие. // М.И. Литовченко, А.Н. Моисеев. – Томск: Издательство Томского государственного университета, 2023 – 96 с.
15. Гилб, Т. Метрики программного обеспечения / Т. Гилб. – М.: Лори, 1996. – 240 с.
16. Jones, C. Applied Software Measurement: Assuring Productivity and Quality / C. Jones. – New York: McGraw-Hill, 2008. – 640 с.
17. Холстед, М.Х. Начала науки о программах / М.Х. Холстед. – М.: Финансы и статистика, 1981. – 128 с.
18. Coleman, D. Using metrics to evaluate software system maintainability / D. Ash, B. Lowther, P. Oman // Computer. – 1994. – Vol. 27, No. 8. – P. 44–49.
19. Статистика рынка труда в России // ГородРабот.ру [Электронный ресурс]. URL: <https://russia.gorodrabot.ru/salaries/> (дата обращения: 05.05.2025).
20. Гант, Г. Организация производства: Системы учета и контроля / Г. Гант. – М.: Экономика, 1922. – 120 с.

Приложение А
(обязательное)
Исходный код приложения

Листинг A.1 – MainWindow.h

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#define RESTART 100

#include <QMainWindow>
#include <QThread>
#include <QFile>
#include <QTextEdit>
#include <QTime>
#include <QFileDialog>
#include "comport.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_pushButton_clicked();

    void on_pushButton_2_clicked();

    void on_checkBox_in_clicked();

    void on_checkBox_out_clicked();

    void on_textEdit_textChanged();

    void on_textEdit_3_textChanged();

    void on_action_HEX_triggered();

    void on_action_triggered();

    void parse_file_data();

    void on_action_settings_change_triggered();

signals:
    void send_message(QByteArray);

    void changeSettings(bool);

private:
    QFile *file;

```

```

void setttext(QString);
void was_write(QString);
int global_mod = 1;
Ui::MainWindow *ui;
ComPort *com;
QThread *thr;
QString message;
int iter = 0;
QByteArray file_data;
bool hex;
QTimer *timer;
};

```

```
#endif // MAINWINDOW_H
```

Листинг A.2 – MainWindow.cpp

```

#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent):
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    com = new ComPort();
    thr = new QThread();

    //connect(thr, &QThread::started, com, &ComPort::slotInit);
    QString com_name = com->slotInit();

    file = new QFile(QString("./log") + com_name + ".txt");
    file->open(QIODevice::Append);
    QString t = "\n\n" + QTime::currentTime().toString() + "
начата новая сессия.";
    file->write(t.toUtf8());
    file->flush();

    setWindowTitle(com_name);

    timer = new QTimer();
    connect(timer, SIGNAL(timeout()), this,
    SLOT(parse_file_data()));

    com->moveToThread(thr);
    com->timer->moveToThread(thr);

    connect(this, &this->send_message, com, &ComPort::SendData);
    connect(com, &ComPort::was_read_data, this, &this->settext);
    connect(com, &ComPort::was_write_data, this, &this-
>was_write);

```

```

//com->port->moveToThread(thr);
thr->start();

ui->verticalLayoutWidget_2->hide();
ui->checkBox_in->hide();
ui->checkBox_out->hide();
ui->pushButton_2->hide();
ui->label_size->hide();
ui->textEdit->setReadOnly(true);
ui->textEdit_2->setReadOnly(true);
ui->textEdit_3->setReadOnly(true);
}

void MainWindow::on_pushButton_clicked()
{
    if (ui->radioButton->isChecked())
        this->global_mod = 1;
    else if (ui->radioButton_2->isChecked())
        this->global_mod = 2;
    else
        this->global_mod = 3;
    ui->verticalLayoutWidget->hide();

    if (this->global_mod != 3){
        ui->verticalLayoutWidget_2->show();
        ui->checkBox_out->show();
        ui->label->setText("Текст на отправку:");
        ui->label_2->setText("Принятые данные:");
    }
    if (this->global_mod == 1){
        ui->label_size->show();
        ui->textEdit->setReadOnly(false);
        ui->pushButton_2->show();
        ui->checkBox_in->show();
    }
    else{
        ui->action->setVisible(false);
        ui->action_HEX->setVisible(false);
    }
    //ui->textEdit->setText(QString(0x30 + this->global_mod));
}

MainWindow::~MainWindow() {
    thr->quit();
    file->close();
    timer->stop();
    delete timer;
    delete file;
    delete com;
    if (thr->isFinished()){
        delete thr;
        delete ui;
    }
}

```

```

QByteArray toHex(const QByteArray &s){
    QByteArray k;
    for (int i = 0; i < s.size(); ++i){
        unsigned int l = (unsigned char)(s[i]) >> 4;
        if (l <= 9) k.append((char)('0' + l));
        else k.append((char)('a' - 10 + l));
        l = (unsigned char)(s[i]) & 0x0f;
        if (l <= 9) k.append((char)('0' + l));
        else k.append((char)('a' - 10 + l));
    }
    return k;
}

QByteArray fromHex(const QByteArray &s){
    QByteArray k;
    for (int i = 0; i + 1 < s.size(); i += 2){
        char l = s[i], r = s[i + 1];
        if ('0' <= l && l <= '9') l -= '0';
        else if ('a' <= l && l <= 'f') l -= 'a' - 10;
        else l -= 'A' - 10;

        if ('0' <= r && r <= '9') r -= '0';
        else if ('a' <= r && r <= 'f') r -= 'a' - 10;
        else r -= 'A' - 10;

        k.append((l << 4) + r);
    }
    return k;
}

void MainWindow::setText(QString s){
    //message = s;
    qDebug() << s;
    if (s.compare("Message was sended successful") == 0) ui-
    >pushButton_2->setEnabled(true);

    QString t = "\n" + QTime::currentTime().toString() + "
принято: " + s + ".";
    file->write(t.toUtf8());
    ui->textEdit_3->append(t);
    file->flush();

    if (ui->checkBox_out->isChecked())
        ui->textEdit_2->setText(toHex(s.toUtf8()));
    else
        ui->textEdit_2->setText(s);
}

void MainWindow::on_pushButton_2_clicked()
{
    qDebug() << "click";
    qDebug() << ui->textEdit->toPlainText().toUtf8();
    qDebug() << fromHex(ui->textEdit->toPlainText().toUtf8());
}

```

```

    auto k = ui->textEdit->toPlainText().toUtf8();
    if (ui->checkBox_in->isChecked()){
        if (k.size() % 2)
            return;

        for (auto &i: k) if (!('0' <= i && i <= '9' || 'A' <= i
&& i <= 'F' || 'a' <= i && i <= 'f')) return;

        ui->pushButton_2->setEnabled(false);
        QString t = "\n" + QTime::currentTime().toString() + "
отправлено: " + fromHex(k) + ".";
        file->write(t.toUtf8());
        file->flush();
        ui->textEdit_3->append(t);
        this->send_message(fromHex(k));
    }
    else{
        ui->pushButton_2->setEnabled(false);
        this->send_message(k);
        QString t = "\n" + QTime::currentTime().toString() + "
отправлено: " + k + ".";
        file->write(t.toUtf8());
        file->flush();
        ui->textEdit_3->append(t);
    }
}

void MainWindow::on_checkBox_in_clicked()
{
    if (ui->checkBox_in->isChecked())
        ui->textEdit->setText(toHex(ui->textEdit-
>toPlainText().toLatin1()));
    else
        ui->textEdit->setText(fromHex(ui->textEdit-
>toPlainText().toLatin1()));
}

void MainWindow::on_checkBox_out_clicked()
{
    if (ui->checkBox_out->isChecked())
        ui->textEdit_2->setText(toHex(ui->textEdit_2-
>toPlainText().toLatin1()));
    else
        ui->textEdit_2->setText(fromHex(ui->textEdit_2-
>toPlainText().toLatin1()));
}

void MainWindow::on_textEdit_textChanged()
{
    ui->label_size->setText(QString("Длина: ") +
QString::number(ui->textEdit->toPlainText().size()));
}

void MainWindow::on_textEdit_3_textChanged()

```

```

{
    if (ui->textEdit_3->toPlainText().size() >= 10000)
        ui->textEdit_3->setText("");
}

void MainWindow::was_write(QString s){
    QString t = "\n" + QTime::currentTime().toString() + "
отправлено: " + s + ".";
    file->write(t.toUtf8());
    file->flush();
    ui->textEdit_3->append(t);
}

void MainWindow::on_action_HEX_triggered()
{
    iter = 0;
    file_data.clear();
    hex = true;
    timer->stop();
    QString filename = QFileDialog::getOpenFileName(this, "Выбор
файла", "./", "*.txt");
    QFile *f = new QFile(filename);
    f->open(QFile::ReadOnly);
    file_data = f->readAll();
    f->close();
    parse_file_data();
}

void MainWindow::on_action_triggered()
{
    iter = 0;
    file_data.clear();
    hex = false;
    timer->stop();
    QString filename = QFileDialog::getOpenFileName(this, "Выбор
файла", "./", "*.txt");
    QFile *f = new QFile(filename);
    f->open(QFile::ReadOnly);
    file_data = f->readAll();
    f->close();
    parse_file_data();
}

void MainWindow::parse_file_data(){
    qDebug() << "parse_file_data";
    qDebug() << QTime::currentTime().toString();
    timer->stop();
    QByteArray arr;
    bool was = false;
    int time = 0;
    for (; iter < file_data.size(); ++iter){
        if (was){
            if ('0' <= file_data[iter] && file_data[iter] <=
'9') time = time * 10 + (file_data[iter] - '0');

```

```

        else break;
        continue;
    }

    if (file_data[iter] == '\n') break;
    if (file_data[iter] == '#'){
        was = true;
        continue;
    }

    arr.append(file_data[iter]);
}
qDebug() << "arr" << arr;
qDebug() << "time" << time;

if (arr.size() == 0) return;

if (hex) arr = fromHex(arr);

ui->pushButton_2->setEnabled(false);
this->send_message(arr);
QString t = "\n" + QTime::currentTime().toString() + "
отправлено: " + arr + ".";
file->write(t.toUtf8());
file->flush();
ui->textEdit_3->append(t);

if (time != 0){
    qDebug() << "setInterval";
    timer->setInterval(time);
    timer->start();
}

if (iter == file_data.size()){
    qDebug() << "iter == file_data.size()";
    iter = 0;
    file_data.clear();
    hex = false;
    timer->stop();
}
}

void MainWindow::on_action_settings_change_triggered()
{
    QFile::remove(QString("./settings.txt"));
    QApplication::exit(RESTART);
}

```

Приложение Б
(рекомендуемое)
Акт внедрения



АКТ ВНЕДРЕНИЯ

результатов дипломного проекта студента
Пензенского государственного университета
Калугина Артёма Андреевича

«Программный имитатор для отладки взаимодействия устройства доверенного управления аппаратурой передачи данных с другими разрабатываемыми устройствами»

В результате выполнения дипломного проекта был спроектирован и реализован программный имитатор для отладки взаимодействия устройства доверенного управления аппаратурой передачи данных с другими разрабатываемыми устройствами.

Функции разработанного программного средства:

- 1) формирование журнала событий;
- 2) отправка команд в соответствии с разрабатываемым протоколом;
- 3) приём и обработка команд в соответствии с разрабатываемым протоколом;
- 4) обмен сообщениями с аппаратурой передачи данных по интерфейсу Ethernet;
- 5) обмен сообщениями с автоматизированным рабочим местом управления по интерфейсу RS-422.

Использование разработанного программного имитатора в процессе разработки и отладки системы устройств обеспечивает сокращение сроков выполнения работ.

Разработанный программный имитатор представляет интерес для дальнейшего использования в процессе отладки программного обеспечения пульта доверенного управления «ПДУ-01» разрабатываемого в Пензенском филиале АО «НТЦ «Атлас».

Заместитель директора
ПФ АО «НТЦ «Атлас»



Козлов И.А.