

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

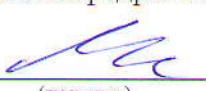
**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ
РАБОТА БАКАЛАВРА**

на тему

**Система передачи посредством RS-232 с использованием
алгоритмов восстановления данных**

Студент  03.06.2020 Иванов Никита Алексеевич
(подпись, дата)
Руководитель  03.06.2020 Гурин Е.И.
(подпись, дата)
Нормоконтролёр  Кучин А.В.
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 03.06.2020 г. № 12)

Заведующий кафедрой  М.А. Митрохин
(подпись)

Работа защищена с отметкой Отлично (протокол заседания ГЭК от
17.06.2020 № 10)

Секретарь ГЭК  Е.И. Гурин
(подпись)

Пенза 2020

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 Митрохин М.А.
«30» апреля 2020 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

1. Студент Иванов Никита Алексеевич гр. 16BB1 факультета вычислительной техники.
2. Тема работы: «Система передачи посредством RS-232 с использованием алгоритмов восстановления данных».

Тема утверждена приказом ПГУ №0335а/о от "30" апреля 2020 г.

3. Руководитель бакалаврской работы д.т.н., профессор каф. ВТ ПГУ Гурин Е.И.

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

Назначение разработки – программное обеспечение, позволяющее обмениваться информацией с использованием последовательного порта RS-232. Необходимо предусмотреть возможность восстановления данных, искаженных во время передачи. Приложение должно иметь удобный пользовательский интерфейс, должно задавать параметры для работы с СОМ портом (номер порта, скорость передачи, число информационных бит, количество стоповых бит, контроль четности). Должна быть предусмотрена возможность выбора способа кодирования передаваемой информации.

5. Перечень подлежащих разработке вопросов

Произвести анализ предметной области, определить язык программирования, разработать программное обеспечение для передачи данных с использованием последовательного порта, выбрать алгоритмы кодирования, декодирования и восстановления данных, произвести отладку и тестирование приложения, оформить выпускную работу.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10	30.04.2020 – 02.05.2020	
2.	Изучение методов кодирования, декодирования информации и алгоритмов восстановления данных	10	03.05.2020 – 05.05.20	
3.	Разработка программного обеспечения для передачи данных с использованием последовательного порта	20	06.05.2020 – 16.05.2020	
4.	Внедрение кода Боуза – Чоудхури – Хоквингема	20	17.05.2020 – 23.05.2020	
5.	Внедрение кода Рида – Соломона	10	24.05.2020 – 27.05.2020	
6.	Отладка и тестирование	10	28.05.2020 – 30.05.2020	
7.	Оформление выпускной работы	20	31.05.2020 – 03.06.2020	

Дата выдачи задания "30" апреля 2020 г.

Руководитель бакалаврской работы  30.04.2020 Гурин Е.И.
(подпись, дата)

Задание к исполнению принял студент  30.04.2020 Иванов Н.А.
(подпись, дата)

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

РЕФЕРАТ

Бакалаврская работа содержит 88 листов, 17 рисунков, 3 таблицы, 7 формул, 12 источников, 2 приложения.

RS-232, QT, ПРИЁМНИК, ПЕРЕДАТЧИК, КОДИРОВАНИЕ, ДЕКОДИРОВАНИЕ, БЧХ КОД, КОД РИДА – СОЛОМОНА.

Цель работы – разработать приложение, осуществляющее обмен сообщениями посредством последовательного порта RS-232.

Объектом разработки является приложение с удобным для пользователя графическим интерфейсом, позволяющее осуществлять прием и отправку сообщений с использованием последовательного порта стандарта RS-232, а также выполнять обработку сообщений.

В результате проведенной работы реализовано приложение, удовлетворяющее поставленной задаче, а именно, разработано приложение, осуществляющее обмен кодированными сообщениями по СОМ порту.

Данное приложение рекомендуется использовать для обмена закодированными данными между отправителем и приемником. Если передаваемое сообщение было подвержено воздействиям внешней среды, вследствие чего было искажено, то приемник имеет возможность восстановить часть утерянной информации. Это возможно благодаря использованию циклических кодов Боуза – Чоудхури – Хоквингема и Риды – Соломона.

					ПГУ 2.090301.16.001 ПЗ					
Изм.	Лист	№ докум.	Подпись	Дата	Система передачи посредством RS-232 с использованием алгоритмов восстановления данных Пояснительная записка			Лит.	Лист	Листов
Разраб.	Иванов Н.А.									
Провер.	Гурин Е.И.								2	88
Н. Контр.	Кучин А.В.							ФВТ, каф. ВТ гр. 16ВВ1		
Утверд.	Митрохин М.А.									

Перечень принятых сокращений

БЧХ код – код Боуза – Чоудхури – Хоквингема;

UART (Universal Asynchronous Receiver-Transmitter) – универсальный асинхронный приёмопередатчик;

ООП – объектно-ориентированное программирование;

DTE (Data Terminal Equipment) – оконечное оборудование данных;

DCE (Data Communication Equipment) – оборудование передачи данных;

ГОСТ – региональный стандарт, принятый Межгосударственным советом по стандартизации, метрологии и сертификации Содружества Независимых Государств;

GUI (Graphical User Interface) – графический пользовательский интерфейс;

RS-232 (Recommended Standard 232, EIA232) - стандарт физического уровня для асинхронного интерфейса (UART);

DSR (Data Set Ready) – сигнал состояния готовности источника данных;

DTR (Data Terminal Ready) – сигнал готовности приемника данных;

ПО – программное обеспечение;

COM порт (Communication port) – стандартный двунаправленный последовательный порт компьютера, используемый для передачи данных согласно протоколу RS-232.

					ПГУ 2.090301.16.001 ПЗ	Лист
						3
Изм.	Лист	№ докум.	Подпись	Дата		

Содержание

РЕФЕРАТ	2
Перечень принятых сокращений	3
Содержание	4
Введение	6
1. Анализ предметной области	8
1.1. Последовательный порт RS-232.....	8
1.1.1. Принцип работы RS-232.....	8
1.1.2. Преимущества и недостатки использования последовательного интерфейса RS-232.....	10
1.2. Циклические коды	11
1.3. Поле Галуа	12
1.4. Кодирование и декодирование данных	15
1.5. Выбор языка программирования и платформы для разработки.....	16
2. Реализация	18
2.1. Описание работы приложения	18
2.2. Описание функций	24
2.2.1. Класс MainWindow.....	25
2.2.2. Класс Setting.....	27
2.2.3. Класс ComPort.....	28
2.2.4. Классы BCH и RS	29
2.2.5. Меню программы	30
2.2.6. Функции обмена данными.....	31
2.3. Отладка и тестирование	32
3. Инструкция пользователя	35
3.1. Главное окно программы.....	35
3.2. Окно настроек последовательного порта.....	37
3.3. Меню приложения.....	38
3.4. Обработка исключительных ситуаций	39

4. Безопасность жизнедеятельности	41
Заключение	42
Список используемых источников	43
Приложение А. Листинги программы.....	44
Файл «mainwindow.h»	44
Файл «mainwindow.cpp»	45
Файл «comport.h»	56
Файл «comport.cpp».....	57
Файл «setting.h»	61
Файл «setting.cpp»	61
Файл «bch.h»	63
Файл «bch.cpp»	64
Файл «rscode.h»	69
Файл «rscode.cpp»	69
Приложение В. Презентация.....	81

Введение

Обмен данными является неотъемлемой частью современной жизни. Развитие сети интернет позволило передавать данные на большие расстояния с высокой скоростью. Однако часто бывает необходимо связываться напрямую, не разворачивая компьютерные сети.

На сегодняшний день в компьютерных системах используются различные интерфейсы, способные обеспечивать обмен информацией между устройствами. Одним из таких является интерфейс стандарта RS-232. Данный интерфейс используется в компьютерах достаточно давно, он зарекомендовал себя как простой и надежный интерфейс [1].

Несмотря на развитие технологий, нельзя обеспечить стопроцентную достоверность передаваемой информации. При передаче данные могут быть подвержены изменениям. К ним можно отнести затухание, глушение сигнала или искажение полезного сигнала шумом из внешней среды. При использовании проводных средств причиной возникновения данных проблем может служить расстояние, на которое передается сообщение, то есть длина кабеля, а также его качество.

Решением данных проблем является использование алгоритмов кодирования и декодирования информации, а также восстановления данных. К ним относятся код Боуза – Чоудхури – Хоквингема (БЧХ) и его частный случай код Рида – Соломона.

Использование кодов БЧХ дополняет передаваемые данные проверочными битами во время кодирования информации. Во время декодирования избыточная информация позволяет определить, было искажено сообщение или нет. В случае, если данные были подвержены воздействию внешней среды, есть возможность восстановить часть утерянной информации. БЧХ код позволяет работать с двоичными данными, а его частный случай, код Рида – Соломона, с

					ПГУ 2.090301.16.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

шестнадцатеричными. Это позволяет работать с большим количеством информации.

Целью данной работы является разработка системы передачи данных посредством последовательного порта с использованием кодов БЧХ и Рида – Соломона. Приложение позволяет передавать сообщение как введенное с клавиатуры, так и данные из текстового файла. Передаваемая информация может быть отправлена тремя способами:

1. В исходном виде.
2. Закодированная с использованием кода БЧХ.
3. Закодированная с использованием кода Рида – Соломона.

Полученные приемником данные декодируются и выводятся пользователю на экран.

					ПГУ 2.090301.16.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

1. Анализ предметной области

Данный раздел содержит обзор интерфейса стандарта RS-232, его принцип работы, достоинства и недостатки. В этом разделе рассматриваются методы циклического кодирования (систематическое, несистематическое), декодирования, а также коды, применяемые для защиты информации от ошибок (БЧХ код, код Рида – Соломона).

1.1. Последовательный порт RS-232

RS – 232 – стандарт физического уровня для асинхронного интерфейса UART. Широко известен как последовательный порт персональных компьютеров.

1.1.1. Принцип работы RS-232

При передаче данных каждый символ обрамляется с двух сторон сигналами, ему должен предшествовать стартовый сигнал, а заканчиваться передача должна стоповым сигналом. Стартовый сигнал предназначен для того, чтобы сообщить принимающему устройству о том, что следующие за ним биты представляют собой данные. После символа передается стоп бит, сигнализирующий об окончании передачи. Пример передаваемого символа показан на рисунке 1 [7].

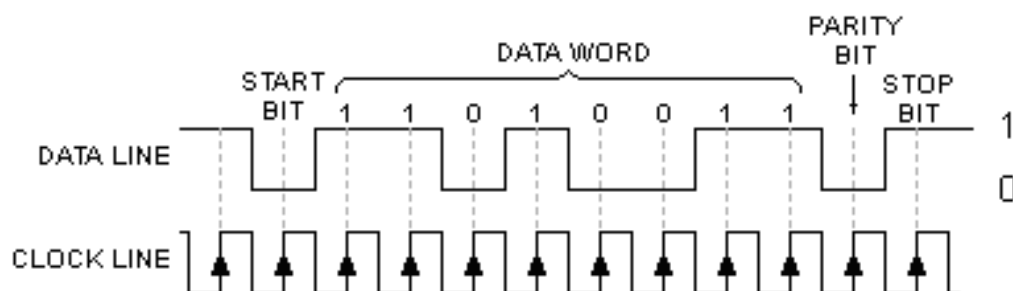


Рисунок 1 - Пример передаваемой информации.

В принимающем устройстве символы распознаются по стартовому и стоповому сигналам. Таким образом, в передаваемой информации около 20%

не содержит в себе каких-либо данных, а используется для идентификации начала и окончания символа.

Последовательным интерфейс называется потому, что связь осуществляется по одиночному проводу, и биты передаются последовательно, один за другим. Информация передается по проводам двоичным сигналом с двумя уровнями напряжения. Логическому нулю соответствует положительное напряжение (от +5 до +15 В), а логической единице - отрицательное (от -5 до -15 В). Эта зависимость для передатчика, для приемника она выглядит следующим образом: логический ноль - от +3 В и выше, а единица - от -3 В и ниже. Данная зависимость продемонстрирована на рисунке 2 [7].

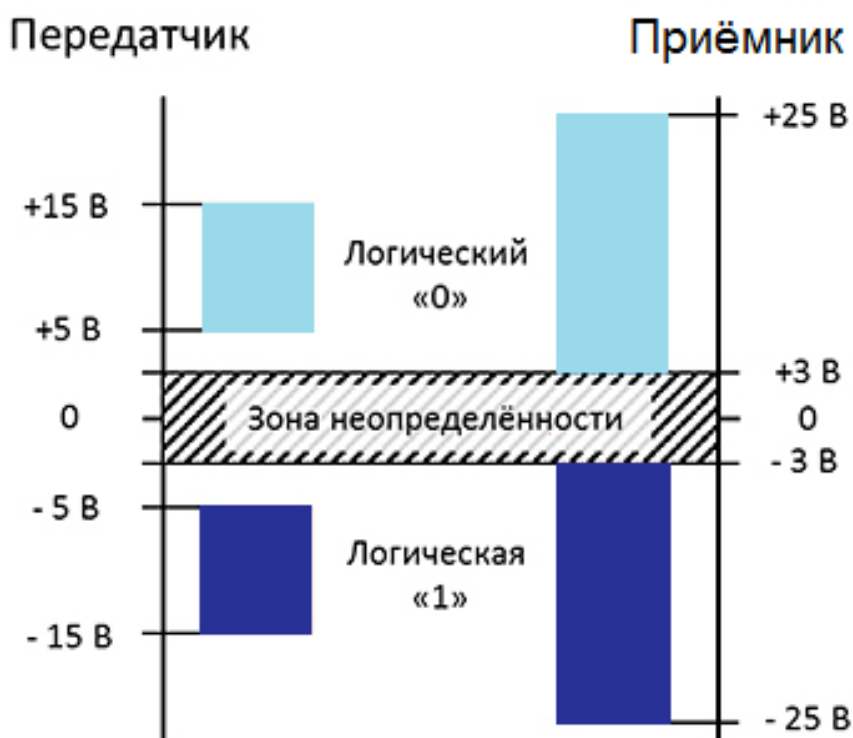


Рисунок 2 – Значения логического «0» и «1» для приемника и передатчика.

RS-232 обеспечивает соединение двух устройств:

- DTE (Data Terminal Equipment) – Оконечное Оборудование Данных
- DCE (Data Communication Equipment) – Оборудование Передачи

Данных

Для успешного обмена данными следующие параметры протокола должны быть заданы одинаково на стороне приемника и передатчика:

- Скорость обмена данными (от 1200 до 115200 бит/с)
- Количество бит данных (от 5 до 8)
- Контроль четности
- Длина стоп бита (1 / 1,5 / 2)

1.1.2. Преимущества и недостатки использования последовательного интерфейса RS-232

В настоящее время RS-232 используется для подключения к компьютерам широкого спектра оборудования, нетребовательного к скорости обмена. Протокол интерфейса предполагает два режима передачи данных: синхронный и асинхронный, а также два метода управления обменом данными: аппаратный и программный. В протоколе также предполагается вариант управления передачей данных по специальным сигналам (DSR – сигнал состояния готовности, DTR– сигнал готовности передачи данных).

Достоинством RS-232 является его простота и надежность, он часто используется в промышленности. Обычно промышленные компьютеры имеют один или несколько портов RS-232. Интерфейс RS-232 является хорошо изученным и достаточно подробно описан в литературе, к его достоинствам можно отнести то, что он позволяет управлять процессом передачи на аппаратном уровне.

К недостаткам RS-232 можно отнести то, что он реализует связь типа «точка-точка» с низкой на сегодняшний день скоростью (до 115200 бит/с), и работает на небольшие расстояния (до 15 метров). Расстояние увеличивается пропорционально снижению скорости (Таблица 1) [7]. К недостаткам RS-232 можно также отнести то, что для коммутации требуется выключение питания компьютера.

Таблица 1 – Соответствие скорости и длины кабеля.

Скорость (бит/с)	Длина (м)
19200	15
9600	150
4800	300
2400	900

1.2. Циклические коды

Циклические коды являются эффективными кодами, способные обнаружить как одиночные, так и пачки ошибок. Данные коды обладают свойством цикличности, то есть при каждой циклической перестановке кодового слова результат будет так же являться кодовым словом.

Выделяют два способа кодирования:

- Несистематическое кодирование;
- Систематическое кодирование.

При несистематическом кодировании закодированное слово можно получить путем умножения информационного полинома на порождающий [8].

$$c(x) = m(x)g(x) \quad (1)$$

Где $m(x)$ – информационный полином, $g(x)$ – порождающий.

При систематическом кодировании кодовое слово состоит из информационного и проверочного блоков. Информационный блок занимает старшие степени кодового слова. Для того чтобы получить блок проверочной информации $s(x)$ необходимо:

1. Сдвинуть информационный блок на r разрядов, где r – число проверочных битов.
2. Получить остаток от деления сдвинутого полинома на порождающий.

$$s(x) = x^r m(x) \bmod g(x) \quad (2)$$

Для получения кодового слова необходимо к сдвинутому информационному полиному прибавить блок проверочных данных.

$$c(x) = x^r m(x) + s(x) \quad (3)$$

Где $m(x)$ – информационный блок, $s(x)$ – блок проверочных данных.

Двоичный код Боуза – Чоудхури – Хоквингема и его частный случай код Рида – Соломона являются циклическими кодами, применяемые для защиты данных от ошибок. Для построения кода задается общее количество символов N и полезное количество символов K , разница $N - K$ представляет собой избыточный код, предназначенный для восстановления ошибок. Одной из характеристик данных кодов является расстояние Хэмминга. Это расстояние характеризует минимальное число различий между двумя различными кодовыми словами. Расстояние Хэмминга определяет другую, не менее важную характеристику – корректирующую способность, способную определить количество исправляемых ошибок.

$$t = \frac{d-1}{2}, \quad (4)$$

где t – корректирующая способность, d – расстояние Хэмминга. Для построения данных кодов применяются поля Галуа.

1.3. Поле Галуа

Для помехоустойчивого кодирования широко используются конечные поля, называемые полями Галуа.

Поле Галуа, обозначаемое $GF(q)$, представляет собой конечное множество, состоящее из q элементов, обладающих свойствами поля. Если характеристика q является степенью простого числа p , то есть $q = p^m$, где m — целое, то элементами поля $GF(p^m)$ будут многочлены степени $(m-1)$ вида

$$a_0 + a_1 x + a_2 x^2 + \dots + a_{m-1} x^{m-1} \quad (5)$$

где все коэффициенты a_i принадлежат простому полю $GF(p)$ [2]. В этом случае p называется основанием поля, а m — степенью поля.

При работе с цифровыми данными используются двоичные поля Галуа, то есть поля Галуа по основанию 2 $GF(2^m)$.

					ПГУ 2.090301.16.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		12

Для построения поля Галуа $GF(2^m)$ используется порождающий многочлен $p(x)$, который является неприводимой примитивной функцией степени m . Так как в дальнейшем будут строиться коды БЧХ и Рида – Соломона длиной 15, то необходимо построить поле Галуа $GF(16)$. Для этого степень m выбирается равной 4, в качестве образующего полинома четвертой степени выбирается полином

$$p(x) = x^4 + x + 1 \quad (6)$$

Исходя из этого элемент α конечного поля $GF(16)$ можно представить как

$$\alpha^n = \alpha^{n-3} + \alpha^{n-4} \quad (7)$$

Где n является целым числом в диапазоне от 0 до 15. Результат построения поля Галуа $GF(2^4)$ представлен в таблице 2.

Таблица 2 – Поле Галуа $GF(2^4)$. $p(x) = x^4 + x + 1$.

Элемент поля	Полином	Двоичный вид	Десятичный вид
α^0	1	0001	1
α^1	x	0010	2
α^2	x^2	0100	4
α^3	x^3	1000	8
α^4	$x+1$	0011	3
α^5	x^2+x	0110	6
α^6	x^3+x^2	1100	12
α^7	x^3+x+1	1011	11
α^8	x^2+1	0101	5
α^9	x^3+x	1010	10
α^{10}	x^2+x+1	0111	7
α^{11}	x^3+x^2+x	1110	14
α^{12}	x^3+x^2+x+1	1111	15
α^{13}	x^3+x^2+1	1101	13
α^{14}	x^3+1	1001	9
α^{15}	1	0001	1

При кодировании и декодировании информации с использованием кодов БЧХ и Рида – Соломона применяются следующие операции над элементами поля Галуа:

- сложение элементов;
- перемножение элементов;
- возведение элемента в степень.

Самой простой является операция сложения, которая является побитовым сложением по модулю 2, то есть операцией исключающего «или». Схема сложения представлена на рисунке 3 [2].

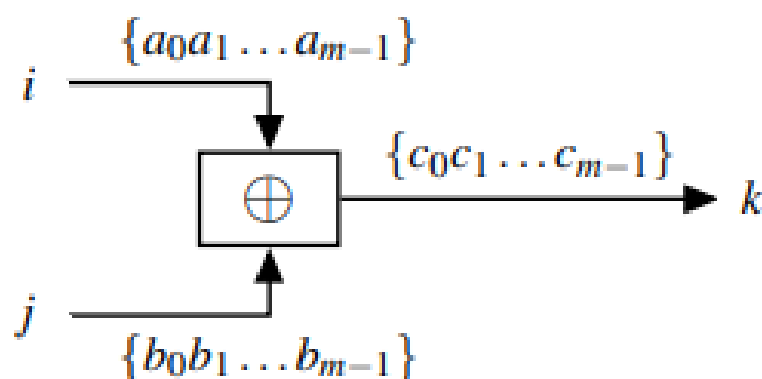


Рисунок 3 – Схема сложения элементов.

Умножение производится путем сложения степеней элементов, при этом сумма степеней приводится по модулю $2^m - 1$ (Рисунок 4).

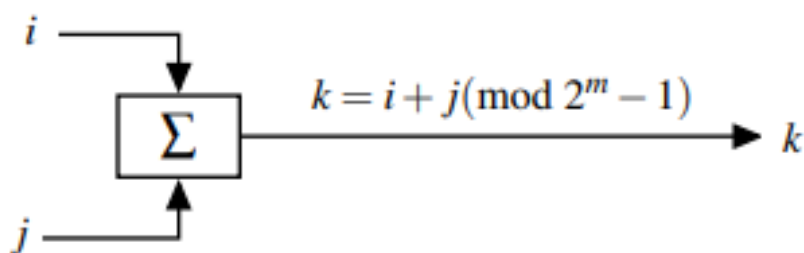


Рисунок 4 – Схема умножения элементов.

Возведение элемента в степень осуществляется путем перемножения степеней, а произведение степеней приводится по модулю $2^m - 1$.

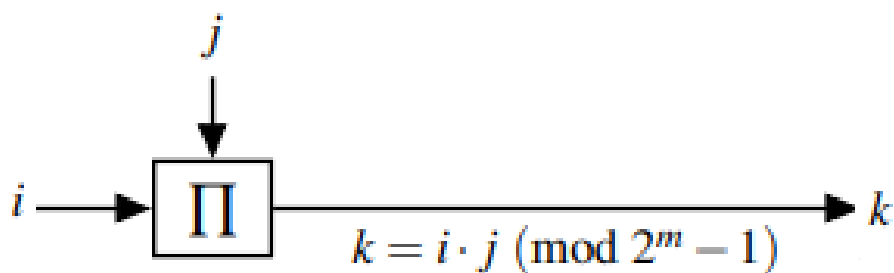


Рисунок 5 – Схема возведения элемента в степень.

1.4. Кодирование и декодирование данных

При кодировании данные дополняются проверочной информацией, которая необходима для обнаружения и исправления ошибки. При декодировании в случае если данные не были искажены, то информация извлекается путем отсечения избыточных данных, если же ошибка была, то используются алгоритмы по обнаружению и исправлению ошибок. Общая схема обмена кодированной информацией представлена на рисунке 6.

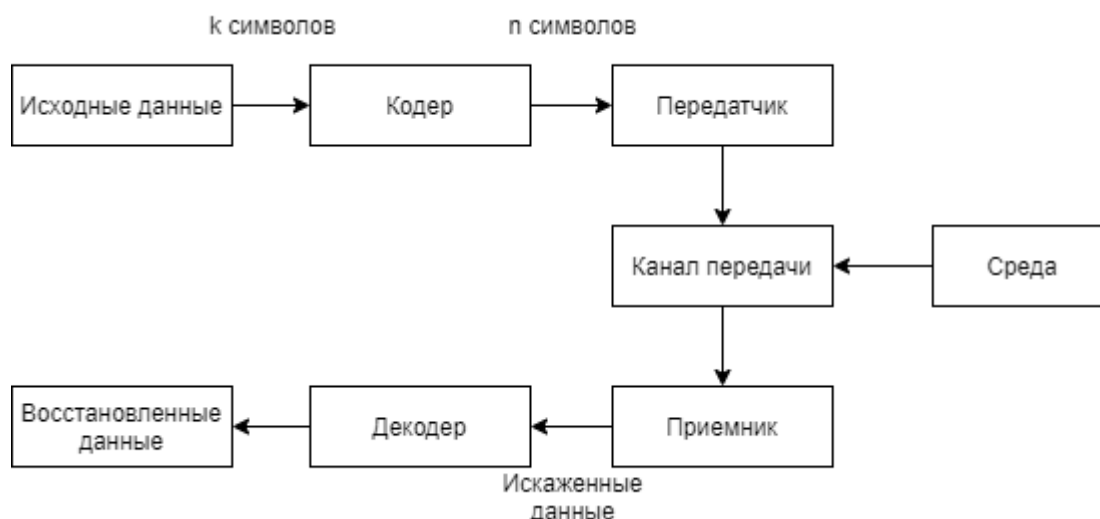


Рисунок 6 – Схема обмена кодированной информацией.

В силу того, что коды БЧХ и Рида – Соломона являются циклическими кодами, то кодирование схоже с кодированием циклических кодов. Кодирование производится систематическим методом, результатом которого является кодовое слово, состоящее из блока проверочных данных и блока информационных данных.

Декодирование информации является более сложным действием, чем кодирование. Для декодирования информации необходимо последовательно выполнить ряд действий.

1. Вычислить синдром ошибки.
2. Определить коэффициенты полинома локаторов ошибок $\sigma(x)$, то есть решить ключевое уравнение.
3. Найти позиции ошибок.
4. Исправить ошибки.

В результате будет получена декодированная и исправленная от ошибок информация.

1.5. Выбор языка программирования и платформы для разработки

Учитывая требования, необходимые для реализации данного приложения был выбран кроссплатформенный фреймворк Qt предназначенный для разработки программного обеспечения на языке C++.

Qt - это бесплатный набор инструментов и виджетов с открытым исходным кодом для создания графических пользовательских интерфейсов (GUI), а также кроссплатформенных приложений, которые работают на различных программных и аппаратных платформах [5].

Qt позволяет запускать написанное с его помощью программное обеспечение в большинстве современных операционных систем путём простой компиляции программы для каждой системы без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Является полностью объектно-ориентированным и расширяемым.

Язык программирования C++ широко используется в различных сферах программирования, от написания драйверов до создания игр. Он позволяет

использовать различные технологии и стили программирования, такие как ООП, обобщенное программирование и процедурное программирование. Использование данного языка совместно с Qt предоставляет широкий спектр возможностей для разработки ПО [4].

					ПГУ 2.090301.16.001 ПЗ	Лист
						17
Изм.	Лист	№ докум.	Подпись	Дата		

2. Реализация

В данной главе описана реализация разрабатываемого приложения. Особое внимание уделяется выделению и описанию различных сущностей и их взаимодействию между собой. При этом освещены основные особенности реализации, связанные с программированием на языке C++. В данном разделе объясняется принцип работы и внутреннее устройство системы в целом.

2.1. Описание работы приложения

При старте приложения в первую очередь необходимо задать параметры для работы с портом. Для этого используется функция *ComOpenPort()*, отображающая на экране окно для ввода параметров. Для корректной работы с последовательным портом необходимо задать номер порта, его скорость, количество бит данных, длина стоп бита и контроль четности. После установки параметров функция генерирует сигнал *ComConnection()*, который связан со слотом *ComConnectPort()*, осуществляющим открытие порта с заданными параметрами.

Для отправки введенного в поле ввода сообщения используется функция *on_ComSendDataButton_clicked()*. Прежде чем сообщение будет передаваться оно кодируется в зависимости от нажатой радиокнопки. Приложение позволяет передать данные тремя способами.

1. Без кодирования.
2. С кодированием при помощи БЧХ кода.
3. С кодированием при помощи кода Рида – Соломона.

Для того, чтобы приемник определил способ кодирования, предварительно отправляется сообщение, позволяющее определить метод кодирования данных.

При получении этого сообщения приложение определяет способ декодирования полученного сообщения. Если во время передачи данные были подвержены внешним воздействиям и исказились, то при декодировании с

помощью циклических кодов, данные по возможности будут восстановлены. После чего сообщение отобразится в поле полученных данных.

Кроме того, приложение позволяет работать с содержимым текстовых файлов, для этого используется функция *on_ComSendFileButton_clicked()*. В результате выполнения данной функции необходимо выбрать файл, содержимое которого, будет отображено в поле для ввода сообщения, закодировано в соответствии с выбранным способ кодирования и отправлено. Принятую информацию также можно записать в файл, для этого предназначена функция *on_ComSaveDataButton_clicked()*.

Язык программирования C++, выбранный для реализации, является объектно-ориентированным языком программирования, что позволяет разбить программу на классы. Это обеспечивает многомодульность программы, что в свою очередь повышает наглядность и легкость восприятия кода программы. Рассмотрим классы, используемые в разрабатываемом приложении.

MainWindow – это основной класс приложения. Данный класс отвечает за внешний вид главного экрана приложения. На нем расположены основные элементы взаимодействия пользователя с приложением.

ComPort – класс для работы с последовательным портом. Данный класс отвечает за открытие и закрытие порта, а также запись и чтение данных.

Settings – класс для задания параметров порта. Отвечает за внешний вид окна с настройками.

BSH – это класс, осуществляющий кодирование и декодирование сообщения с помощью кода БЧХ.

RS – это класс, осуществляющий кодирование и декодирование сообщения с помощью кода Рида – Соломона.

UML-диаграмма классов реализованной программы представлена на рисунке 7.

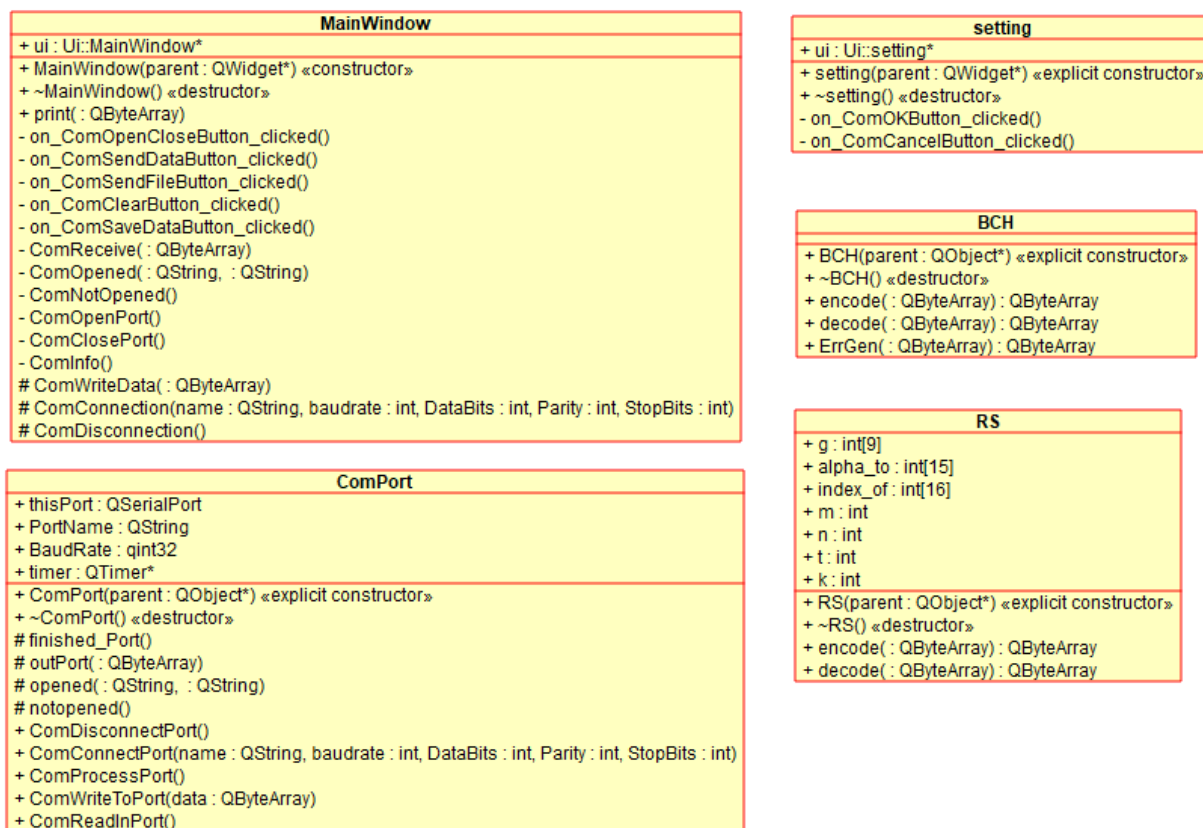


Рисунок 7 – UML-диаграмма классов.

При обмене информацией между отправителем и получателем выполняется ряд действий, который показан на рисунке 8. На этом рисунке показан процесс передачи сообщения от отправителя к получателю. Перед отправкой отправитель выбирает метод кодирования, после чего вводит сообщение и отправляет его. Приложение, в зависимости от выбранного метода, выполняет кодирование информации по методу БЧХ или Рида – Соломона. Если кодирование не задано, то сообщение передается без изменений. Перед отправкой сообщения получателю сообщается метод кодирования. На стороне получателя приложение, получив сообщение о методе кодирования, выбирает метод декодирования, а затем при получении сообщения ПО декодирует его в соответствии с выбранным методом и выводит пользователю на экран. На основе данного описания была разработана UML-диаграмма последовательности, которая показана на рисунке 8.

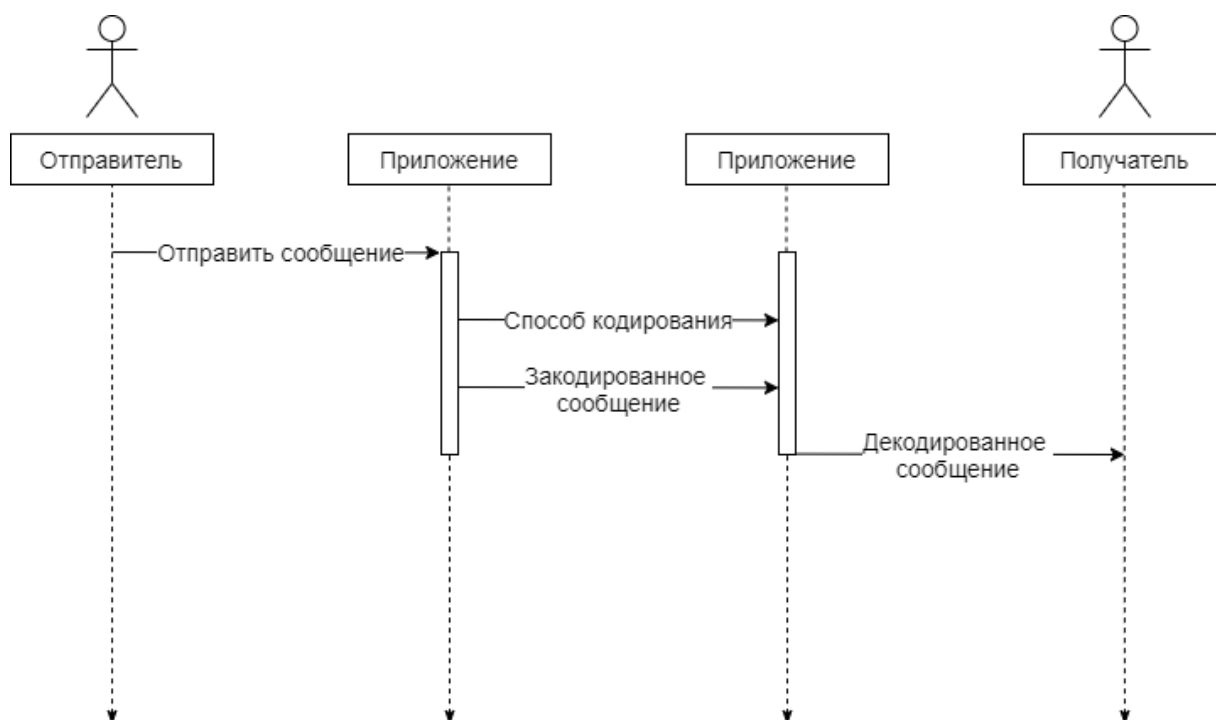


Рисунок 8 – UML-диаграмма последовательности.

На рисунке 9 в виде UML-диаграммы показаны возможности, предоставляемые пользователю по обмену информацией с использованием последовательного порта.

Приложение позволяет пользователю выполнить следующие действия:

- Открыть порт – данное действие открывает окно с настройками порта, после чего открывает порт с заданными параметрами, такими как номер порта, скорость передачи, количество информационных бит, количество стоп бит и контроль четности.
- Закреть порт – осуществляет закрытие порта. Эта процедура выполняется при временной приостановке обмена информацией или при окончании работы.
- Выбрать метод кодирования – позволяет выбрать метод кодирования данных.
- Отправить сообщение – осуществляет кодирование сообщения, отправку метода кодирования и сообщения.

- Отправить файл – позволяет выбрать файл, содержимое которого будет выведено в окно ввода, закодировано и отправлено получателю.
- Сохранить в файл – позволяет сохранить полученные данные из окна вывода в выбранный файл.
- Очистить поля – осуществляет очистку полей ввода и вывода информации.



Рисунок 9 – UML-диаграмма вариантов использования.

Жизненный цикл программы отображает диаграмма деятельности, показанная на рисунке 10.

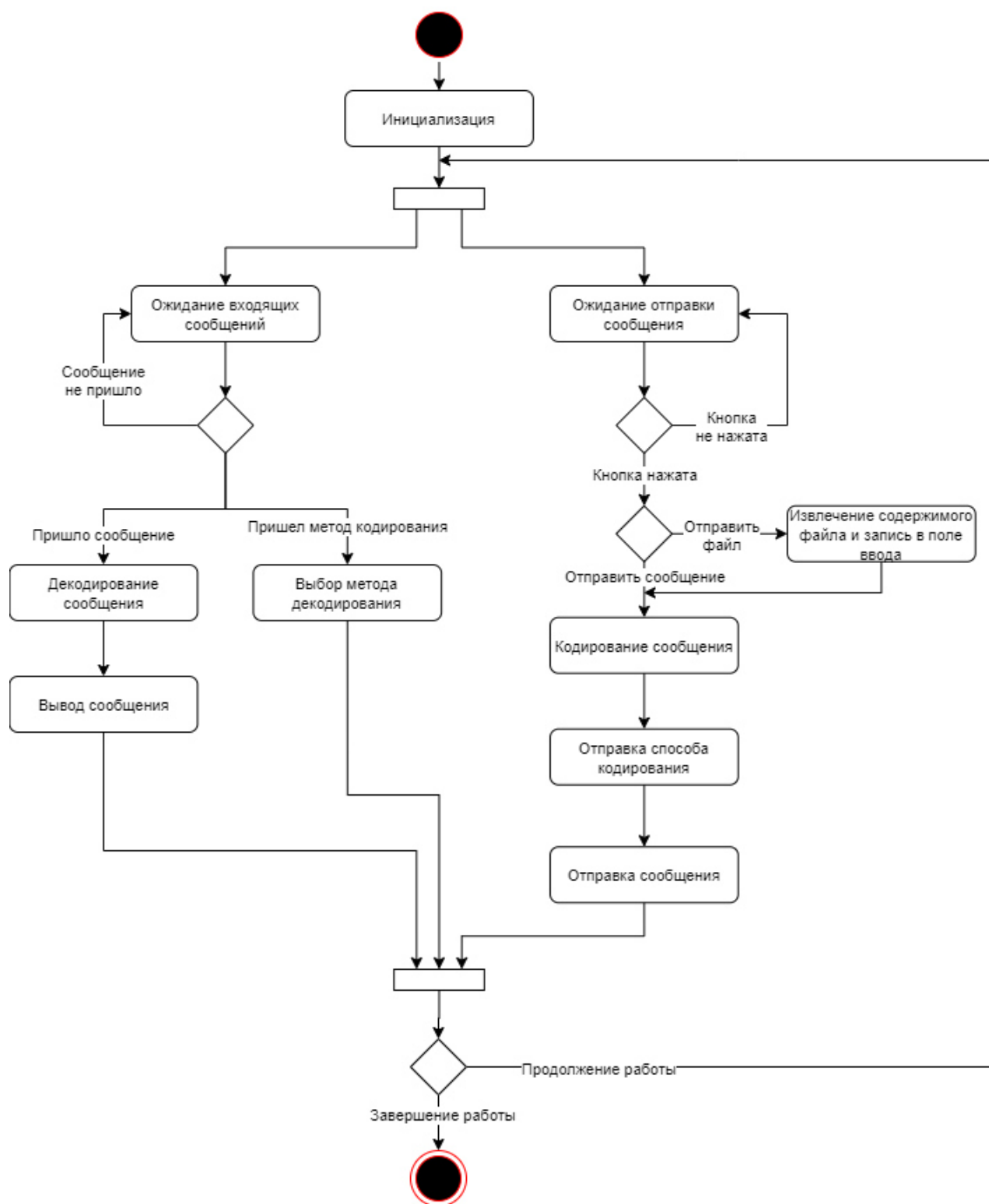


Рисунок 10 – UML-диаграмма деятельности.

Из диаграммы деятельности и описания работы можно выделить следующие состояния приложения:

1. Состояние ожидания отправки сообщения. В этом состоянии приложение ожидает, пока пользователь нажмет кнопку «Отправить сообщение» или «Отправить файл...».

2. Состояние извлечения данных из файла. В этом состоянии приложение считывает содержимое выбранного файла и выводит его на экран в окно ввода информации.

3. Состояние кодирования сообщения. В этом состоянии приложение осуществляет кодирование сообщения при помощи выбранного пользователем метода.

4. Состояние отправки способа кодирования. В этом состоянии приложение отправляет сообщение о методе кодирования, выбранным пользователем.

5. Состояние отправки сообщения. В этом состоянии приложение выполняет отправку закодированного сообщения.

6. Состояние ожидания входящих сообщений. В этом состоянии приложение ожидает получения сообщений от отправителя.

7. Состояние выбора метода декодирования. В этом состоянии приложение, основываясь на полученном сообщении о методе кодирования, выбирает способ декодирования сообщений.

8. Состояние декодирования сообщения. В этом состоянии приложение выполняет декодирование принятого сообщения.

9. Состояние вывода сообщения. В этом состоянии приложение осуществляет вывод декодированного сообщения на экран.

2.2. Описание функций

Рассмотрим ряд функций, реализованных для решения поставленной задачи.

В программировании графического интерфейса при изменении одного виджета необходимо, чтобы остальные виджеты получили об этом уведомление. Для выполнения данного действия в Qt используются сигналы и

слоты. Сигнал вырабатывается при определенном событии, а слот – это функция, вызываемая в ответ на сигнал. Виджеты Qt имеют множество предопределенных сигналов и слотов, но всегда можно сделать их самостоятельно.

2.2.1. Класс MainWindow

Класс `MainWindow` это основной класс приложения. Данный класс отвечает за внешний вид главного экрана приложения и взаимодействие пользователя с элементами приложения. В его состав входят функции: `on_ComOpenCloseButton_clicked()`, `on_ComSendDataButton_clicked()`, `on_ComSendFileButton_clicked()`, `on_ComClearButton_clicked()`, `on_ComSaveDataButton_clicked()`, `ComReceive(QByteArray)`, `ComOpened(QString,QString)`, `ComNotOpened()`, `ComOpenPort()`, `ComClosePort()`, `ComInfo()`, а также сигналы: `ComWriteData(QByteArray)`, `ComConnection(QString, int, int, int, int)`, `ComDisconnection()`.

Рассмотрим слоты, используемые в классе `MainWindow`.

- `ComReceive(QByteArray)` – используется для вывода полученного сообщения на экран. В качестве параметров принимает массив `QByteArray`, содержащий полученное сообщение. Полный код данной функции приведен в приложении А.
- `ComOpened(QString,QString)` – выполняется в том случае, если удалость открыть последовательный порт. В качестве параметров принимает две строки `QString`, содержащие номер порта и его скорость, которые выводятся в строку состояния. Также делает активными кнопки для работы с данными («Отправить сообщение», «Отправить файл...», «Сохранить в файл») и изменяет название кнопки «Открыть порт...» на «Закрыть порт». Подпункт меню «Открыть...» становится неактивным, а подпункты «Закрыть», «Отправить данные», «Отправить файл...», «Сохранить в файл» активными. Полный код данной функции приведен в приложении А.

- *ComNotOpened()* – выполняется в том случае, если последовательный порт открыть не удалось. Пользователю выводится сообщение об ошибке. Полный код данной функции приведен в приложении А.

- *ComOpenPort()* – открывает окно с настройками параметров порта, если параметры были заданы, подает сигнал для открытия порта *ComConnection()*, использующий в качестве параметров настройки СОМ порта, а именно номер порта, скорость передачи, количество информационных бит, количество стоп бит, контроль четности. Полный код данной функции приведен в приложении А.

- *ComClosePort()* – подает сигнал на закрытие порта *ComDisconnection()*. После чего очищает строку состояния и делает кнопки для работы с данными неактивными. Изменяет название кнопки «Закрыть порт» на «Открыть порт...». Подпункт меню «Открыть...» становится активным, а подпункты «Закрыть», «Отправить данные», «Отправить файл...», «Сохранить в файл» неактивными. Полный код данной функции приведен в приложении А.

- *ComInfo()* – используется для отображения окна с информацией о программе. Полный код данной функции приведен в приложении А.

Остальные слоты используются для обработки нажатия клавиш.

- *on_ComOpenCloseButton_clicked()* – обработка нажатия кнопки «Открыть порт.../Закрыть порт». Если кнопка имеет название «Открыть порт...», то вызывает функцию *ComOpenPort()*, если «Закрыть порт», то *ComClosePort()*. Полный код данной функции приведен в приложении А.

- *on_ComSendDataButton_clicked()* – обработка нажатия кнопки «Отправить данные». Отправляет сообщение с методом кодирования, выбранным пользователем, после чего производится кодирование сообщения с помощью функции *encode()* класса BCH или RS. Далее генерируется сигнал *ComWriteData(QByteArray)*, где в качестве параметра передается закодированное сообщение. Данный сигнал приводит к срабатываю слота

ComWriteToPort(QByteArray), осуществляющего запись сообщения в порт. Полный код данной функции приведен в приложении А.

- *on_ComSendFileButton_clicked()* – обработка нажатия кнопки «Отправить файл...». Предоставляет пользователю возможность выбрать файл, содержимое которого выводится в окно ввода, далее вызывается функция *on_ComSendDataButton_clicked()*. Полный код данной функции приведен в приложении А.

- *on_ComClearButton_clicked()* – обработка нажатия кнопки «Очистить поля». Функция выполняет очистку полей ввода и вывода информации. Полный код данной функции приведен в приложении А.

- *on_ComSaveDataButton_clicked()* – обработка нажатия кнопки «Сохранить в файл...». Предоставляет пользователю возможность выбрать файл, в который будет записана принятая информация. Полный код данной функции приведен в приложении А.

Помимо слотов в данном классе также используются сигналы.

- *ComWriteData(QByteArray)* – используется для вызова слота, отвечающего за запись данных в порт. В качестве параметра передается отправляемое сообщение.

- *ComConnection(QString, int, int, int, int)* – используется для вызова слота, отвечающего за открытие порта. В качестве параметров передаются настройки открываемого порта (номер, скорость, количество информационных бит, количество стоп бит, контроль четности).

- *ComDisconnection()* – используется для вызова слота, отвечающего за закрытие порта.

2.2.2. Класс Setting

Класс Setting используется для отображения графического интерфейса с настройками COM порта. Класс содержит пять *comboBox*, позволяющих выбрать параметры открываемого порта, а также две кнопки для

подтверждения выбранных настроек или отмены. В данном классе используется два слота:

- *on_ComOKButton_clicked()* – используется для обработки нажатия на кнопку подтверждения «Ок»;
- *on_ComCancelButton_clicked()* - используется для обработки нажатия на кнопку «Отмена»;

2.2.3. Класс ComPort

ComPort – класс для работы с последовательным портом. Данный класс отвечает за открытие и закрытие порта, а также запись и чтение данных. В его состав входят функции *ComDisconnectPort()*, *ComConnectPort()*, *ComWriteToPort()*, *ComReadInPort()*, а также сигналы *finished_Port()*, *outPort()*, *opened()*, *notopened()*.

Рассмотрим используемые слоты.

- *ComDisconnectPort()* – осуществляет закрытие порта и останавливает таймер, по которому производится чтение из порта. Полный код данной функции приведен в приложении А.
- *ComConnectPort(QString, int, int, int, int)* – производит открытие порта с настройками, переданными в качестве параметров данному слоту (номер порта, скорость передачи, количество информационных бит, количество стоп бит, контроль четности). Если порт открылся, то генерируется сигнал *opened()* и запускается таймер, по которому происходит считывание данных. Если порт открыть не удалось, то генерируется сигнал *notopened()*. Полный код данной функции приведен в приложении А.
- *ComWriteToPort(QByteArray)* – осуществляет запись данных, полученных в качестве параметра, в порт. Данный слот срабатывает при генерации сигнала *ComWriteData()*. Полный код данной функции приведен в приложении А.
- *ComReadInPort()* – считывает данные из порта. Если прочитанные данные содержат метод кодирования сообщения, то выбирается метод

декодирования, иначе производится декодирование полученного сообщения с использованием функции *decode()* класса BCH или RS. После декодирования генерируется сигнал *outPort()*, в качестве параметра которого передается декодированное сообщение. Функция *ComReadInPort()* срабатывает по истечении времени таймера по сигналу *timeout()*. Полный код данной функции приведен в приложении А.

Также используются сигналы:

- *outPort(QByteArray)* – используется для вызова функции вывода на экран принятых и декодированных сообщений;
- *opened(QString,QString)* – сигнализирует об успешном открытии порта, в качестве параметров передает номер порта и его скорость;
- *notopened()* – сигнализирует о неудачной попытке открыть порта.

2.2.4. Классы BCH и RS

Рассмотренные ранее классы используются исключительно для работы с последовательным портом. Для работы с кодированием и декодированием используются классы BCH и RS имеющие идентичный набор функций.

- *encode(QByteArray)* – функция кодирования данных, в качестве параметра принимает массив данных для кодирования.
- *decode(QByteArray)* – функция декодирования данных, в качестве параметра принимает массив данных, которые необходимо декодировать.

Для кодирования данных в приложении используются код БЧХ (15,7,5) и код Рида – Соломона (15,11,5).

Кодирование осуществляется систематическим методом, то есть кодовое слово состоит из блока проверочных данных и блока информации. Для кода БЧХ блок информации состоит из 7 символов, а для Рида – Соломона из 11. Функция *encode()* сначала вычисляет блок проверочной информации, а затем добавляет его к входной информации, в результате чего на выходе получается кодовое слово длиной 15.

При декодировании функция *decode()* сначала вычисляет синдром, если ошибок нет, то информация получается путем удаления блока проверочных данных. Если ошибки все же были обнаружены, то определяются коэффициенты полинома локаторов ошибок с помощью алгоритма Берлекэмпа – Мэсси. Далее находятся позиции ошибок при помощи алгоритма Ченя и исправляются ошибки. Для кода Рида – Соломона присутствует дополнительный шаг, после вычисления позиций ошибок вычисляются значения данных ошибок при помощи алгоритма Форни. Необходимо учитывать, что в процессе декодирования возможно исправить до 2 ошибок, так как расстояние Хэмминга в нашем случае равно 5. Если ошибок окажется больше, то будет исправлено максимально допустимое количество ошибок, а остальные останутся неисправленными.

2.2.5. Меню программы

Для создания строки меню используется функция *createMenuBar()*. В ней создаются объекты класса *QAction* и к ним устанавливаются сигналы *&QAction::triggered* и соответствующие функции, которые необходимо вызвать в том или ином событии. К событиям относятся:

- *gOpenPort* – «Открыть порт...», вызывает функцию открытия последовательного порта *ComOpenPort()*;
- *gClosePort* – «Закрыть порт», вызывает функцию закрытия порта *ComClosePort()*;
- *sendData* – «Отправить данные», осуществляет кодирование и отправку введенного сообщения с помощью функции *on_ComSendDataButton_clicked()*;
- *sendFile* – «Отправить файл...», осуществляет вывод содержимого выбранного файла в поле ввода, кодирование и запись в порт с помощью функции *on_ComSendFileButton_clicked()*;

- *saveData* – «Сохранить в файл...», осуществляет сохранение полученного сообщения в файл, выбранный пользователем, для данного действия используется функция *on_ComSaveDataButton_clicked()*;
- *clearWindow* – «Очистить поля», осуществляет очищение полей ввода и вывода данных с использованием функции *on_ComClearButton_clicked()*;
- *info* – «?», осуществляет вывод на экран сообщения с информацией о приложении.

Функции, вызываемые при нажатии на пункты меню, аналогичны тем, что выполняются при нажатии кнопок с тем же названием. Исключением является пункт меню «?», который вызывает функцию *info()*, содержащую информацию о программе.

2.2.6. Функции обмена данными

Необходимо подробнее рассмотреть каким образом происходит прием и передача сообщений. Получение сообщения происходит в отдельном потоке *QThread*, создаваемом при запуске приложения. Считывание данных происходит по таймеру, когда время истекает генерируется сигнал *timeout()* и вызывается слот *ComReadInPort()*. В данной функции происходит проверка полученных данных, если это сигнал, то выбирается метод декодирования данных, если данные, то происходит декодирование и генерация сигнала *outPort()*. Сигнал связан со слотом *ComReceive()* класса *MainWindow*, который осуществляет вывод сообщения на экран.

При отправке сообщение кодируется с помощью функции *encode()* класса *BSH* или *RS*, после отправляется сигнал о методе кодирования передаваемого сообщения. Сигнал представляет собой шестнадцатеричный ASCII код символов, не имеющих печатного кода. Соответствие сигнала и метода кодирования представлено в таблице 3. Для отправки сообщения генерируется сигнал *ComWriteData()*, который связан со слотом *ComWriteToPort()* класса *ComPort*, осуществляющий запись сообщения в порт.

Таблица 3 – Соответствие сигнала и метода кодирования.

Сигнал	Метод кодирования
0x11	Без кодирования
0x12	БЧХ код
0x13	Код Рида – Соломона

2.3. Отладка и тестирование

Отладка разработанного приложения выполнялась с использованием последовательного порта RS-232 двумя способами:

- с использованием кабеля стандарта RS-232;
- при помощи создания виртуального COM порта.

При разработке использовался только один компьютер, поэтому при отладке с использованием кабеля последовательный порт замыкался сам на себя. Для того чтобы обеспечить замыкание порта, одевалась заглушка, соединяющая контакты TxD (Transmit Data) и RxD (Receive Data), которые обеспечивают отправку и прием данных. Таким образом, переданное сообщение возвращалось обратно отправителю.

Кроме того, использовалась отладка с применением виртуальных портов. Для данной цели использовалась программа «Virtual Serial Port Driver» от компании Eltima Software. Данное ПО позволяет создать пару виртуальных последовательных портов и отслеживать количество передаваемых байтов (Рисунок 11).

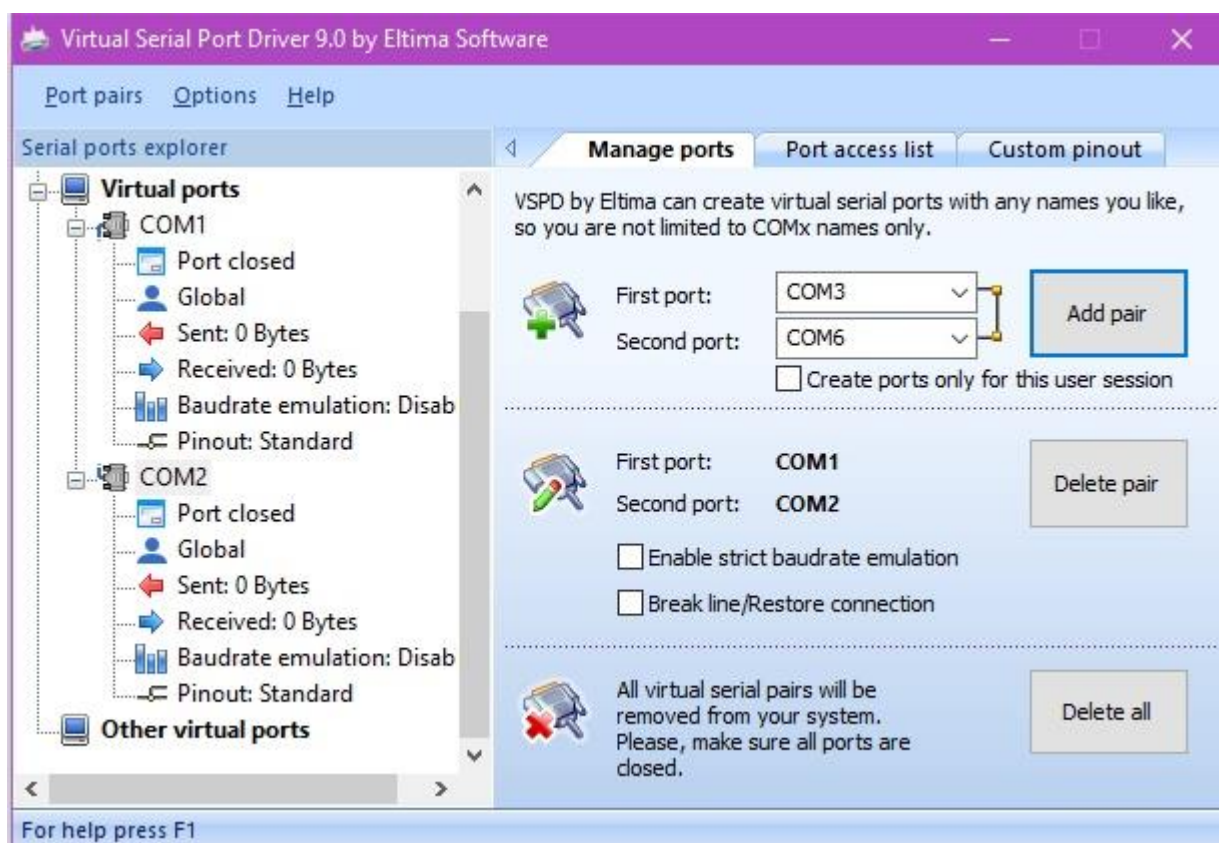


Рисунок 11 – Virtual Serial Port Driver.

Для проверки правильности работы по обмену информацией приложение запускалось в двух экземплярах, в которых открывалась пара виртуальных портов, созданная программой Virtual Serial Port Driver.

В ходе тестирования приложения были выявлены проблемы, которые необходимо было решить. Одной из проблем было то, что прием данных осуществлялся при наличии в порте данных по сигналу *ReadyRead()*, который вызывал функцию отображения сообщения на экран, но данный сигнал никогда не срабатывал. Были проверены некоторые предположения об источнике ошибки. Изменялась длина кабеля и порт для подсоединения, но это не дало результата. В дальнейшем было сделано предположение, что данная ошибка была вызвана неправильной работой установленной версии Qt. Была установлена другая версии Qt, что решило проблему и сигнал *ReadyRead()* начал работать корректно.

Другой проблемой в результате тестирования стала работа с рассмотренным ранее сигналом *ReadyRead()*. Срабатывание данного сигнала происходило слишком часто, что приводило к большому количеству вызова функции вывода сообщения на экран и сильно нагружало процессор. Было принято решение производить чтение из последовательного порта не по сигналу *ReadyRead()*, а по таймеру. В результате нагрузка на процессор снизилась, а эффективность работы приложения не уменьшилась.

Еще одной проблемой стал обмен информацией между окнами приложения. Из окна настроек необходимо передавать параметры порта, которые задал пользователь, в основное окно, где часть информации отображается на экране, а часть используется в качестве параметров функции открытия порта. На помощь пришли сигналы и слоты, используемые в Qt.

После предпринятых мер разработанное приложение работало надежно и передавало сообщение без ошибок.

3. Инструкция пользователя

3.1. Главное окно программы

После запуска приложения перед пользователем появляется главное окно (Рисунок 12).

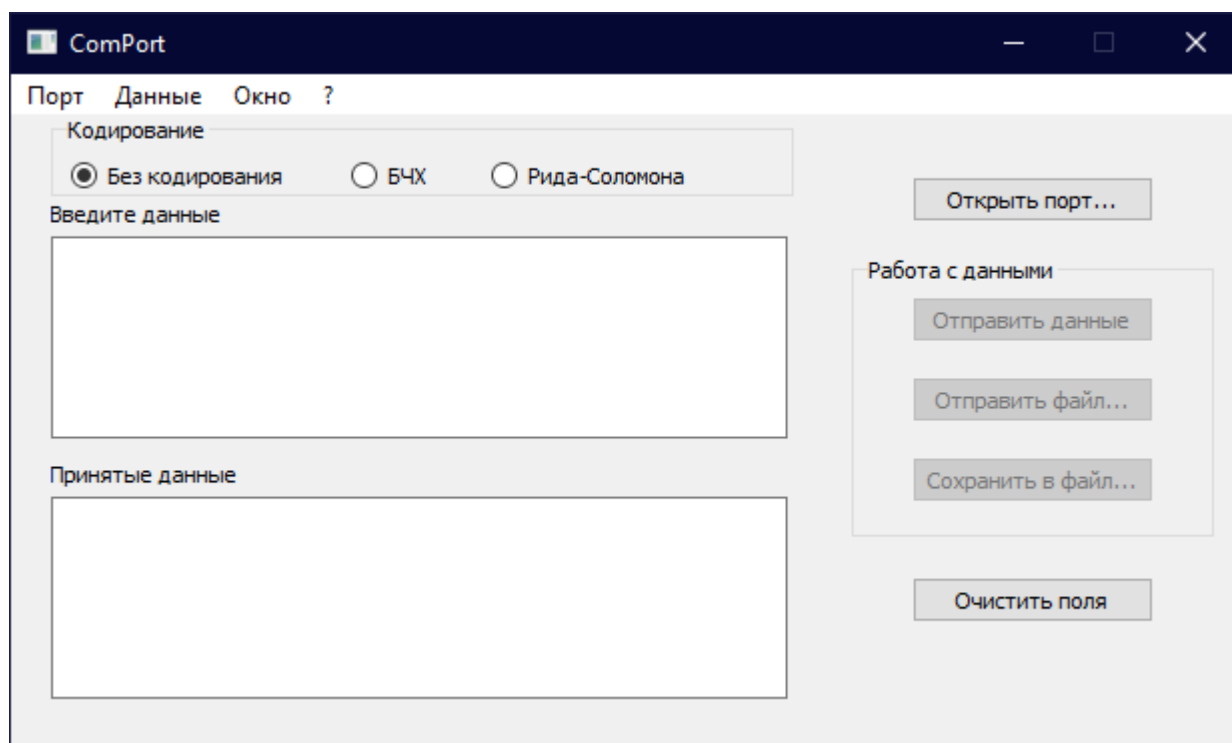


Рисунок 12 – Главное окно программы.

Окно программы состоит из следующих компонентов:

- Строка меню.
- Группа «Кодирование».
- Группа «Работа с данными».
- Поля для ввода и вывода данных.
- Кнопка «Открыть порт...» / «Заккрыть порт».
- Кнопка очищения полей данных.
- Строка состояния.

Группа «Кодирование» включает в себя 3 радиокнопки, которые служат для выбора метода кодирования отправляемого сообщения.

- Без кодирования – данные передаются в исходном виде без кодирования.

- БЧХ – кодирование с помощью кода БЧХ.

- Рида – Соломона – кодирование с помощью кода Рида – Соломона.

Кнопка «Открыть порт...» / «Заккрыть порт» служит для выполнения действий, соответствующих названию кнопки.

Группа «Работа с данными» предоставляет возможность работать с передаваемыми и принимаемыми данными. Данная группа является неактивной до тех пор, пока не будет открыт порт. Для отправки сообщения, введенного в поле ввода, необходимо нажать кнопку «Отправить данные». Сообщение будет закодировано в соответствии с выбранным методом и отправлено получателю. Кнопка «Отправить файл...» позволяет выбрать текстовый файл, содержимое которого будет сперва выведено в поле ввода, затем закодировано и отправлено. Нажатие кнопки «Сохранить в файл...» дает возможность сохранить принятое сообщение из поля вывода в файл, выбранный пользователем.

Поля ввода и вывода информации служат для визуализации отправляемого и получаемого сообщения. Для того чтобы пользователь не мог вносить изменения в принятые данные, поле вывода недоступно для редактирования. Кнопка «Очисти поля» позволяет очистить данные поля.

Строка состояния располагается в нижней части приложения и служит для отображения номера и скорости открытого порта.

Пример обмена сообщениями продемонстрирован далее (Рисунок 13).

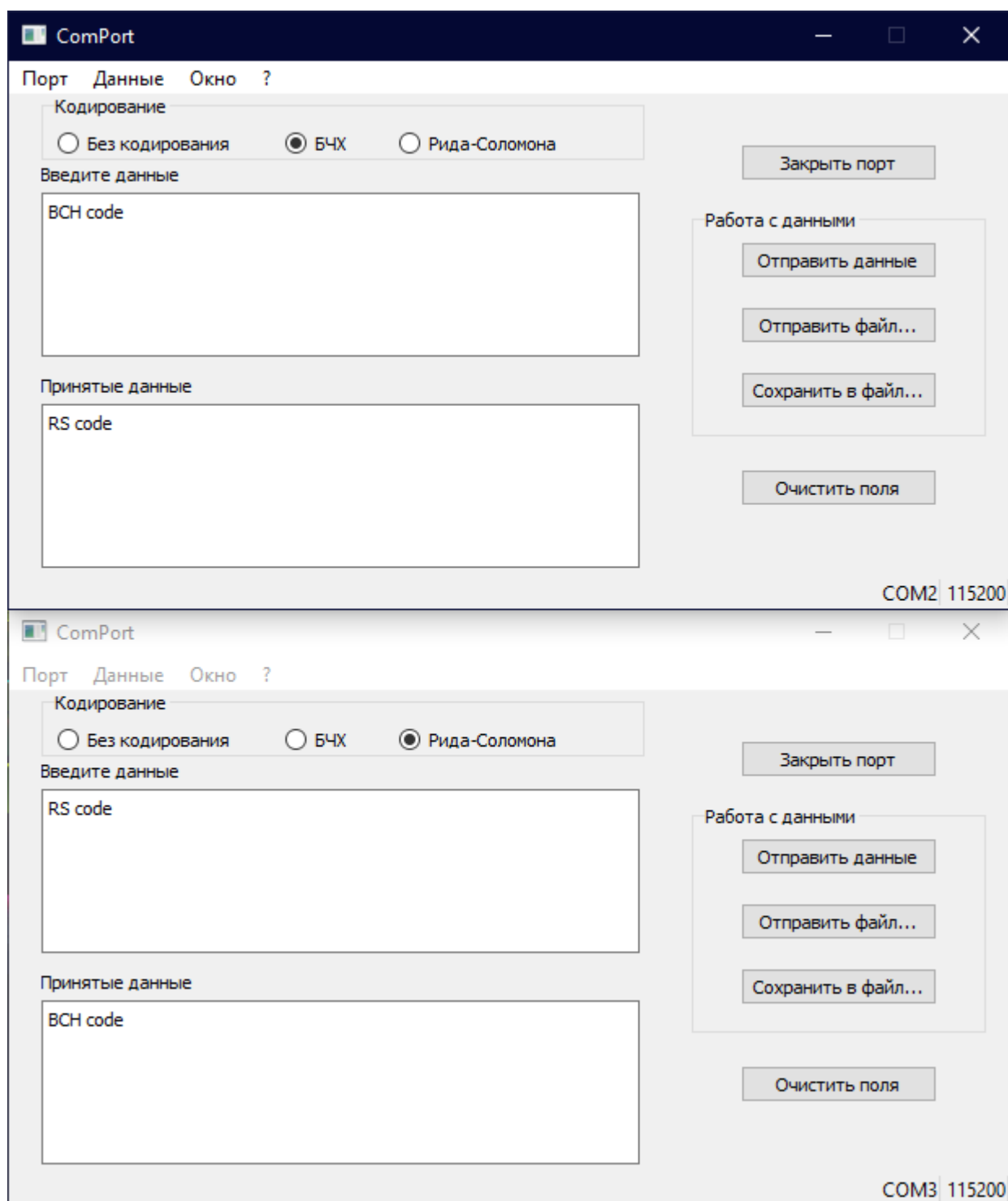


Рисунок 13 – Обмен сообщениями посредством RS-232.

3.2. Окно настроек последовательного порта

При нажатии кнопки «Открыть порт...» перед пользователем появляется окно настроек параметров последовательного порта (Рисунок 14).

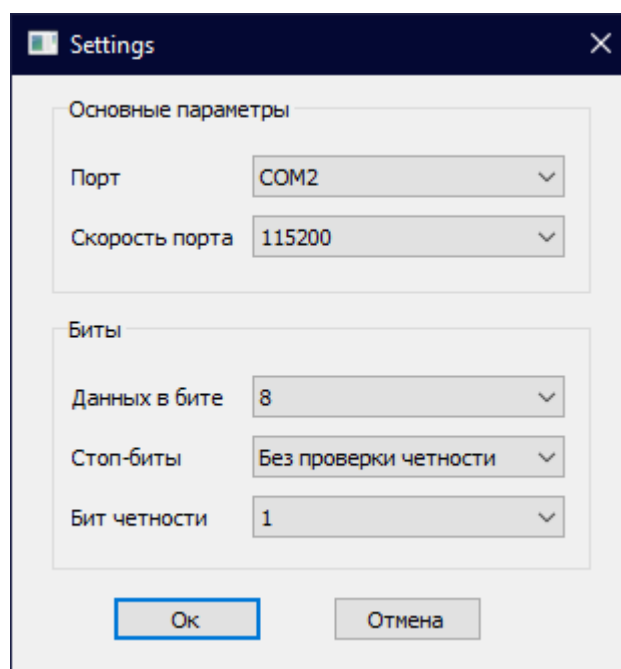


Рисунок 14 – Окно настроек.

Данное окно состоит из двух групп: группы основных параметров и группы настроек бит. К основным параметрам относятся номер порта и его скорость. Группа «Биты» позволяет выбрать количество информационных бит, количество стоп бит и контроль четности.

После выбора параметров необходимо подтвердить свой выбор нажатием кнопки «Ок». Если пользователь передумал открывать последовательный порт, то он может воспользоваться кнопкой «Отмена», что приведет к закрытию данного окна.

3.3. Меню приложения

Меню приложения содержит 4 пункта (Рисунок 12):

- Порт;
- Данные;
- Окно;
- ?.

Для каждого пункта меню существует горячая клавиша, что ускоряет и упрощает работу с приложением.

Пункт меню «Порт» предоставляет возможность открыть порт и закрыть его. В каждый момент времени доступен только 1 из подпунктов (Рисунок 15). Подпункт «Открыть порт...» открывает окно с настройками порта аналогично кнопке «Открыть порт...».

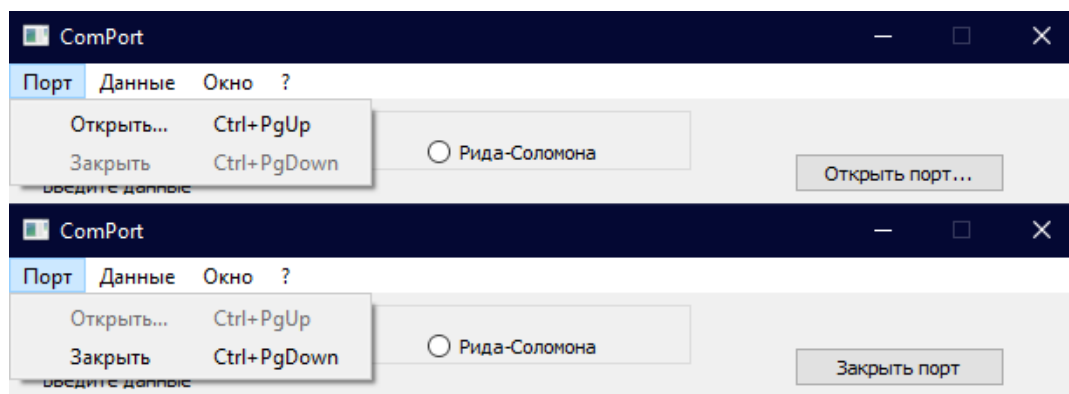


Рисунок 15 – Пункт меню «Порт».

Пункт «Данные» дублирует действия группы «Работа с данными» – «Отправить сообщение», «Отправить файл...», «Сохранить в файл...».

Пункт меню «Окно» содержит единственный подпункт «Очистить поля», осуществляющий удаление данных из полей ввода и вывода информации.

При нажатии на пункт меню «?» пользователю открывается окно с информацией о программе и разработчике (Рисунок 16).

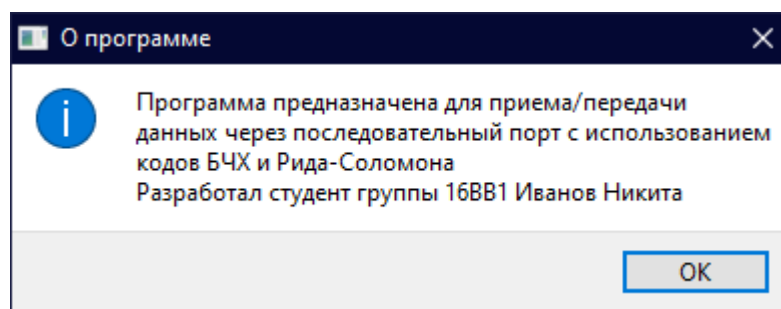


Рисунок 16 – Сообщение «О программе».

3.4. Обработка исключительных ситуаций

В случае, если при открытии порта возникает ошибка, то пользователю выводится сообщение об ошибке (Рисунок 17).

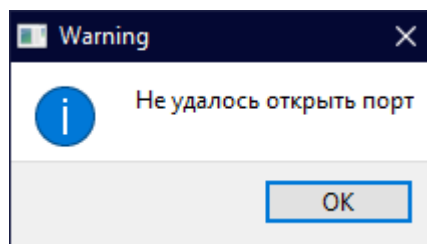


Рисунок 17 – Сообщение об ошибке при открытии порта.

Помимо этого, пользователь не может воспользоваться функционалом группы «Работы с данными» до тех пор, пока не будет открыт порт (Рисунок 12).

4. Безопасность жизнедеятельности

В современном мире стало нормой то, что каждый из нас большинство своего времени находится за компьютером. Для правильного взаимодействия пользователя и программного обеспечения нужно соблюдать ряд рекомендаций по обеспечению общих санитарно-технических норм к помещению и рабочим местам.

Размеры помещения, в котором планируется производиться работа с разработанным приложением, должны соответствовать количеству полностью оснащенных рабочих мест. Нельзя забывать и про соответствующие нормы температуры, освещения, чистоты воздуха, уровня шума и вибрации. На одного работающего сотрудника, основываясь на санитарные нормы труда СН 245-71, должно быть выделено помещение объемом не менее 15 м³, площадью не менее 4,5 м².

Основные требования к организации рабочего места и его оборудования прописаны в ГОСТ 12.2.032 78. Высота рабочей поверхности стола должна регулироваться в пределах 680-800 мм, если такой возможности нет, то высота рабочей поверхности должна составлять 725 мм. Ширина стола соответствует 800, 1000, 1200 или 1400 мм, а глубина – 800 или 1000 мм. Кроме того, пространство для ног высотой в рабочей области должно быть не менее 600 мм, а шириной – не менее 500 мм, глубиной на уровне колен – не менее 450 мм, и на уровне вытянутых ног – не менее 650 мм.

Рабочий стул (кресло) должен быть подъёмно-поворотным и регулируемым по высоте и углам наклона сиденья и спинки, а также по расстоянию спинки до переднего края сиденья. Основные требования к размерам и конструкции рабочего стула в зависимости от вида выполняемых работ приведены в ГОСТ 12.2.032 78 и ГОСТ 21889 76.

					ПГУ 2.090301.16.001 ПЗ	Лист
						41
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В ходе выполнения выпускной квалификационной работы были изучены методы кодирования и декодирования данных, возможность передачи данных посредством последовательного порта стандарта RS-232 и восстановление данных в случае их искажения при передаче. Также были получены навыки объектно-ориентированного программирования на языке C++ в среде разработки Qt.

Результатом выполненной работы является приложение, позволяющее обмениваться кодированной информацией с помощью последовательного порта. Разработанное программное обеспечение позволяет восстанавливать искаженные данные, полученные под действием внешней среды, благодаря использованию кодов БЧХ и Рида – Соломона.

Была выполнена экспериментальная проверка работоспособности. Для отладки программы использовался кабель стандарта RS-232 замкнутый сам на себя, а также пара виртуальных СОМ портов. В процессе разработки приложения были выявлены различные проблемы. При попытке чтения данных с порта данная операция не выполнялась. Данная проблема была связана с устаревшей версией Qt, решением стала установка более новой версии. Помимо этого, при чтении данных по сигналу *ReadyRead()* данная операция сильно нагружала процессор. Было принято решение осуществлять чтение данных по таймеру.

В дальнейшем приложение может быть усовершенствовано путем добавления более тонкой настройки параметров последовательного порта. Также можно расширить функционал работы с файлами, что добавит дополнительные поддерживаемые форматы.

Список используемых источников

1. Кузьминов А.Ю. Интерфейс RS232: Связь между компьютером и микроконтроллером: От DOS к WINDOWS98/XP. – М.: Издательский дом «ДМК-пресс» - 320 с.
2. Владимиров С. Математические основы теории помехоустойчивого кодирования: Курс лекций – СПб ГУТ, 2014.
3. Охорзин В., Кукунин Д., Новодворский М. Построение каскадных кодов на основе кодов Боуза – Чоудхури – Хоквингема и Рида – Соломона: Курс лекций – СПб ГУТ, 2013.
4. Бланшет Ж., Саммерфилд М. Qt 4: программирование GUI на C++. Пер. с англ. 2-е изд., доп. — М.: КУДИЦ-ПРЕСС, 2008. — 736 с.
5. Боровский А.Н. Qt4.7+. Практическое программирование на C++. – СПб.: БХВ-Петербург, 2012. — 496 с.: ил. — (Профессиональное программирование).
6. Лафоре Р. Объектно-ориентированное программирование в C++. 4-е изд. – СПб.: Питер, 2016. – 928 с.: ил. – (Серия «Классика computer science»).
7. Справочник по электронным компонентам – URL: <http://www.gaw.ru>.
8. Сидельников В. Теория кодирования: Справочник по принципам и методам кодирования – МГУ. – 2006 г., 289 с.
9. Qt, справочная документация – URL: <https://doc.qt.io/>.
10. Форум программистов и сисадминов Киберфорум – URL: <https://www.cyberforum.ru/>.
11. Система вопросов и ответов о программировании Stack Overflow – URL: <https://stackoverflow.com/>.
12. Научная библиотека избранных естественно-научных изданий – URL: <https://scask.ru/>.

Приложение А. Листинги программы

Файл «mainwindow.h»

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSerialPort>
#include <QDebug>
#include <QFileDialog>
#include <QFile>
#include <QMenu>
#include <QMenuBar>
#include <QAction>
#include <comport.h>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    void print(QByteArray);
    Ui::MainWindow *ui;

private slots:
    void on_ComOpenCloseButton_clicked();
    void on_ComSendDataButton_clicked();
    void on_ComSendFileButton_clicked();
    void on_ComClearButton_clicked();
    void on_ComSaveDataButton_clicked();
    void ComReceive(QByteArray);
    void ComOpened(QString, QString);
    void ComNotOpened();
    void ComOpenPort();
    void ComClosePort();
    void ComInfo();

signals:
    void ComWriteData(QByteArray);
    void ComConnection(QString name, int baudrate, int DataBits, int
Parity, int StopBits);
    void ComDisconnection();
```

					ПГУ 2.090301.16.001 ПЗ	Лист
						44
Изм.	Лист	№ докум.	Подпись	Дата		

```

};

#endif // MAINWINDOW_H
Файл «mainwindow.cpp»

/*
*/

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "setting.h"
#include "ui_setting.h"
#include "bch.h"
#include "rs.h"

/*
*/

QLabel *gName; ///< Хранит имя порта и используется в статусбаре
QLabel *gSpeed; ///< Хранит скорость порта и используется в статусбаре
QThread *gThread = new QThread; ///< Поток для работы с портом
QAction *gOpenPort; ///< Подпункт меню "Открыть..."
QAction *gClosePort; ///< Подпункт меню "Закрыть"
QAction *gSendData; ///< Подпункт меню "Отправить данные"
QAction *gSendFile; ///< Подпункт меню "Отправить файл"
QAction *gSaveData; ///< Подпункт меню "Сохранить в файл"

MainWindow::MainWindow( QWidget *parent )
    : QMainWindow( parent )
    , ui(new Ui::MainWindow)
{
    /// Задаем фиксированный размер окна
    QWidget::setFixedSize( 615 , 337 );

    ui->setupUi( this );
    ui->radioButton->setChecked(true);
    /// Пункт меню "Порт"
    QMenu *port = menuBar()->addMenu("Порт");
    /// Подпункт меню "Открыть..."
    gOpenPort = new QAction(tr("Открыть..."));
    /// Назначения горячих клавиш
    gOpenPort->setShortcut(tr("Ctrl+PgUp"));
    /// Подпункт меню "Закрыть"
    gClosePort = new QAction(tr("Закрыть"));
    /// Назначения горячих клавиш
    gClosePort->setShortcut(tr("Ctrl+PgDown"));
    /// Недоступна для нажатия
    gClosePort->setEnabled(false);

    /// Пункт меню "Данные"

```

```

QMenu *file = menuBar()->addMenu("Данные");
/// Подпункт меню "Отправить данные"
sendData = new QAction(tr("Отправить данные"));
/// Назначения горячих клавиш
sendData->setShortcut(tr("Ctrl+Alt+D"));
/// Кнопка недоступна
sendData->setEnabled(false);
/// Подпункт меню "Отправить файл..."
sendFile = new QAction(tr("Отправить файл..."));
/// Назначения горячих клавиш
sendFile->setShortcut(tr("Ctrl+Alt+F"));
/// Кнопка недоступна
sendFile->setEnabled(false);
/// Подпункт меню "Сохранить данные..."
saveData = new QAction(tr("Сохранить данные..."));
/// Назначения горячих клавиш
saveData->setShortcut(tr("Ctrl+S"));
/// Кнопка недоступна
saveData->setEnabled(false);

/// Пункт меню "Окно"
QMenu *window = menuBar()->addMenu("Окно");
/// Подпункт меню "Очистить поля"
QAction *clearWindow = new QAction(tr("Очистить поля"));
/// Назначения горячих клавиш
clearWindow->setShortcut(tr("Ctrl+Del"));

/// Пункт меню "?"
QMenu *inf = menuBar()->addMenu("?");
/// Подпункт меню "О программе..."
QAction *info = new QAction(tr("О программе..."));

/// Связь подпунктов меню с выполняемыми функциями

/// Нажатие подпункта "Открыть..." вызывает функцию ComOpenPort
connect(gOpenPort, SIGNAL(triggered()), this, SLOT(ComOpenPort()));
/// Нажатие подпункта "Закрыть" вызывает функцию ComClosePort
connect(gClosePort, SIGNAL(triggered()), this, SLOT(ComClosePort()));

/// Нажатие подпункта "Отправить данные" вызывает функцию
/// on_ComSendDataButton_clicked
connect(sendData, SIGNAL(triggered()),
        this, SLOT(on_ComSendDataButton_clicked()));
/// Нажатие подпункта "Отправить файл..." вызывает функцию
/// on_ComSendFileButton_clicked
connect(sendFile, SIGNAL(triggered()),
        this, SLOT(on_ComSendFileButton_clicked()));
/// Нажатие подпункта "Сохранить данные..." вызывает функцию

```

```

/// on_ComSaveDataButton_clicked
connect(saveData,SIGNAL(triggered()),
        this,SLOT(on_ComSaveDataButton_clicked()));

/// Нажатие подпункта "Очистить поля" вызывает функцию
/// on_ComClearButton_clicked
connect(clearWindow,SIGNAL(triggered()),
        this,SLOT(on_ComClearButton_clicked()));

/// Нажатие подпункта "О программе..." вызывает функцию ComInfo
connect(info,SIGNAL(triggered()),this,SLOT(ComInfo()));

/// Связь пунктов меню подпунктами

port->addAction(gOpenPort);
port->addAction(gClosePort);

file->addAction(sendData);
file->addAction(sendFile);
file->addAction(saveData);

window->addAction(clearWindow);

inf->addAction(info);

/// Экземпляр класса comport
ComPort *PortNew = new ComPort();
/// Перенос comport в отдельный поток
PortNew->moveToThread( gThread );
/// Перенос qserialport в отдельный поток
PortNew->thisPort.moveToThread( gThread );

/// Сигналы для управления работой программы

/// Запуск потока приводит к выполнению функции process_Port()
connect(gThread,SIGNAL(started()),PortNew,SLOT(ComProcessPort()));
/// Появления сигнала finished_Port приводит к завершению
потока
connect(PortNew,SIGNAL(finished_Port()),gThread,SLOT(quit()));
/// Если поток завершен, происходит удаление PortNew

connect(gThread,SIGNAL(finished()),PortNew,SLOT(deleteLater()));
/// Появления сигнала finished_Port приводит к удалению потока

connect(PortNew,SIGNAL(finished_Port()),gThread,SLOT(deleteLater()
));

```

```

    /// Появления сигнала ComWriteData приводит к записи данных в
порт
connect(this, SIGNAL(ComWriteData(QByteArray)),
        PortNew, SLOT(ComWriteToPort(QByteArray)));
    /// Появления сигнала outPort приводит к чтению из порта
connect(PortNew, SIGNAL(outPort(QByteArray)),
        this, SLOT(ComReceive(QByteArray)));
    /// Появления сигнала ComConnection приводит к открытию порта
connect(this, SIGNAL(ComConnection(QString,int,int,int,int)),
        PortNew, SLOT(ComConnectPort(QString,int,int,int,int)));
    /// Появления сигнала opened говорит о успешном открытии порта
connect(PortNew, SIGNAL(opened(QString,QString)),
        this, SLOT(ComOpened(QString,QString)));
    /// Появления сигнала notopened говорит, что порт не открылся

connect(PortNew, SIGNAL(notopened()), this, SLOT(ComNotOpened()));
    /// Появления сигнала ComDisconnection приводит к закрытию
порта

connect(this, SIGNAL(ComDisconnection()), PortNew, SLOT(ComDisconnect
Port()));
}

MainWindow::~MainWindow()
{
    delete ui;
}

/*
*/
/**
    \fn          void
MainWindow::on_ComOpenCloseButton_clicked()
    \brief       Нажатие кнопки открытия/закрытия порта.
    \note       Кнопка имеет 2 состояния: "Открыть порт..." и
"Закрыть порт",
                отвечающие за соответствующее действие с портом.
    \remarks    Если порт не удалось открыть выводиться сообщения об
ошибке.
*/
void MainWindow::on_ComOpenCloseButton_clicked()
{
    // Проверка состояния кнопки Открытия/Закрытия порта
    if ( ui->ComOpenCloseButton->text() == "Открыть порт..." )
    {
        /// Вызов функции открытия порта
        MainWindow::ComOpenPort();
    }
}

```

```

else if (ui->ComOpenCloseButton->text() == "Заккрыть порт")
{
    /// Вызов функции закрытия порта
    MainWindow::ComClosePort();
}
} // on_ComOpenCloseButton_clicked
/*
*/

/*
*/
/**
    \fn          void          MainWindow::ComOpened(QString
PortName,QString BaudRate)
    \brief       Обновляет статус бар.
    \param       PortName - имя открываемого порта
                BaudRate - скорость порта
    \note        Выполняется в случае открытия порта.
*/
void MainWindow::ComOpened(QString PortName,QString BaudRate)
{
    /// Изменение текста кнопки
    ui->ComOpenCloseButton->setText("Заккрыть порт");
    gName=new QLabel(this);
    gName->setText(PortName);
    gSpeed=new QLabel(this);
    gSpeed->setText(BaudRate);
    /// Добавление данные в строку состояния
    ui->statusBar->addWidget(gSpeed);
    ui->statusBar->addWidget(gName);
    /// Изменение возможности использования кнопок меню
    gOpenPort->setEnabled(false);
    gClosePort->setEnabled(true);
    ui->ComSendDataButton->setEnabled(true);
    ui->ComSendFileButton->setEnabled(true);
    ui->ComSaveDataButton->setEnabled(true);
    sendData->setEnabled(true);
    sendFile->setEnabled(true);
    saveData->setEnabled(true);
} // ComOpened
/*
*/

/*
*/
/**
    \fn          void MainWindow::ComNotOpened()
    \brief       Сообщение об ошибке.
    \note        Выполняется в случае не открытия порта.
*/

```

```

void MainWindow::ComNotOpened()
{
    /// Сообщение об ошибке
    QMessageBox::information(nullptr, "Warning", "Не удалось
открыть порт");
} // ComNotOpened
/*
*/

/*
*/
/**
\fn          void MainWindow::on_ComSendDataButton_clicked()
\brief       Нажатие кнопки отправки данных.
\note       Функция выполняет запись данных, введенных в окно
данных,
                в порт.
*/
void MainWindow::on_ComSendDataButton_clicked()
{
    /// Записать данных из окна ввода в строку
    QString message = ui->textEdit->toPlainText();
    /// Преобразование строки в QByteArray
    QByteArray mes;

    if (ui->radioButton->isChecked())
    {
        QByteArray changestate;

        changestate.append("11");

        emit ComWriteData(changestate);
        QThread::msleep(550);

        mes = message.toLocal8Bit();
        mes = mes.toHex();

        emit ComWriteData(mes);
    }

    else if (ui->radioButton_2->isChecked())
    {
        QByteArray changestate;
        changestate.append("12");

        emit ComWriteData(changestate);
        QThread::msleep(550);
    }
}

```

```

QByteArray encoding;
QByteArray message = message.toLocal8Bit();;
BCH *a;
int mas[8];

// Пока не обрабатывается каждый символ
for (int i = 0; i < message.length(); i++)
{
    QByteArray result;
    // Получение кода символа
    int d= static_cast<unsigned char>(message[i]);
    // Получение двоичного представления
    for (int i = 0; i < 8; i++)
        mas[7 - i] = (d >> i) & 0x01;
    // Удаление символа
    message.remove(0, 0);

    for (int i = 0; i < 8; i++)
        mes.push_back(mas[i] + '0');

    while(mes.length()>0)
    {
        if(mes.length()<7)
        {
            int need = 7 - mes.length();
            for(int i=need; i>0;i--)
            {
                mes.push_back('0');
            }
        }
        for(int i = 0;i<7;i++)
        {
            encoding[i]=0;
        }
        for(int i = 0;i<7;i++)
        {
            encoding[i]=mes[i];
        }
        QByteArray mesa = a->encode(encoding);
        mes.remove(0,7);
        mesa += encoding;
        result += a->ErrGen(mesa);
        //result += mesa;
    }

    emit ComWriteData(result);
}

else if (ui->radioButton_3->isChecked())
{

```

```

    QByteArray changestate;
    changestate.append("13");

    emit ComWriteData(changestate);
    QThread::msleep(550);

    QByteArray encoding;
    QByteArray message = message.toLocal8Bit();
    RS* a;

    while (message.length() > 0)
    {
        QByteArray toencode;
        if (message.length() < 5)
        {
            for (int i=0; i<message.length(); i++)
            {
                toencode[i] = message[i];
            }
            encoding = a->encode(toencode);
            message.remove(0, 5);
            emit ComWriteData(encoding);
        }
        else {
            for (int i=0; i<5; i++)
            {
                toencode[i] = message[i];
            }
            encoding = a->encode(toencode);
            message.remove(0, 5);
            emit ComWriteData(encoding);
        }
    }
}

} // on_ComSendDataButton_clicked
/*
*/

/*
*/
/**
    \fn          void MainWindow::on_ComSendFileButton_clicked()
    \brief       Нажатие кнопки отправки файла.
    \note       Функция выполняет запись выбранного в диалоговом
окне
                файла в порт.
    \remarks    Данные записывают в textbox, а затем в порт.
*/

```

					ПГУ 2.090301.16.001 ПЗ	Лист
						52
Изм.	Лист	№ докум.	Подпись	Дата		

```

void MainWindow::on_ComSendFileButton_clicked()
{
    /// Получение пути к файлу
    QString path= QFileDialog::getOpenFileName();
    QFile file(path);
    /// Существует ли файлы и открыт ли он для чтения
    if ( (file.exists()) && (file.open(QIODevice::ReadOnly)) )
    {
        QByteArray mes = file.readAll();
        qDebug() << mes;
        ui->textEdit->setText(QString::fromLocal8Bit(mes));

        file.close();
        MainWindow::on_ComSendDataButton_clicked();
    }

} // on_ComSendFileButton_clicked
/*
*/

/*
*/
/**
 \fn          void MainWindow::on_ComClearButton_clicked()
 \brief       Нажатие кнопки очистки полей.
 \note       Функция очищает textbox ввода и вывода информации.
*/
void MainWindow::on_ComClearButton_clicked()
{
    ///Очищение окон ввода и вывода информации
    ui->textEdit->setText("");
    ui->textEdit_2->setText("");
} // on_ComClearButton_clicked
/*
*/

/*
*/
/**
 \fn          void MainWindow::on_ComSaveDataButton_clicked()
 \brief       Нажатие кнопки сохранения в файл.
 \note       Функция выполняет сохранение данных из окна вывода
              данных в выбранный файл.
*/
void MainWindow::on_ComSaveDataButton_clicked()
{
    /// Получение пути к файлу
    QString path= QFileDialog::getOpenFileName();
    QFile file(path);

```

```

// Открыт ли файл для записи
if ( file.open(QIODevice::WriteOnly) )
{
    QTextStream out(&file);
    out<<ui->textEdit_2->toPlainText();

    /// Заккрытие файла
    file.close();
}
} // on_ComSaveDataButton_clicked
/*
*/

/*
*/
/**
 \fn          void MainWindow::receive(QString str)
 \brief       Вывод считанной информации из порта в окно
вывода
                данных.
 \param       str - строка принятых данных.
 \note        Функция выполняется по сигналу outPort(QString).
*/
void MainWindow::ComReceive(QByteArray str)
{
    /// Вывод данных в окно вывода
    ui->textEdit_2->setText(ui->textEdit_2->toPlainText()          +
QString::fromLocal8Bit(str));
} // ComReceive
/*
*/

/*
*/
/**
 \fn          void MainWindow::ComOpenPort()
 \brief       Функция открытия порта.
 \note        Выполняет открытие порта.
 \remarks     Если порт не удалось открыть выводиться сообщения об
ошибке.
*/
void MainWindow::ComOpenPort()
{
    // Проверка состояния кнопки Открытия/Закрытия порта
    if ( ui->ComOpenCloseButton->text() == "Открыть порт..." )
    {
        setting *set=new setting();
        if (set->exec() == QDialog::Accepted)

```

```

    {
        /// Запуск потока
        gThread->start();
        /// Генерация сигнала ComConnection
        emit ComConnection(set->ui->comboBox->currentText(),
                           set->ui->comboBox_2->
>currentText().toInt(),
                           set->ui->comboBox_3->
>currentText().toInt(),
                           set->ui->comboBox_4->
>currentText().toInt(),
                           set->ui->comboBox_5->
>currentText().toInt());
    }
}
} // ComOpenPort
/*
*/

/*
*/

/*
*/
void MainWindow::ComClosePort()
{
    // Проверка состояния кнопки Открытия/Закрытия порта
    if ( ui->ComOpenCloseButton->text() == "Закрыть порт" )
    {
        /// Генерация сигнала ComDisconnection
        emit ComDisconnection();
        /// /// Изменение текста кнопки
        ui->ComOpenCloseButton->setText("Открыть порт...");
        /// Удаление данных из строки состояния
        ui->statusBar->removeWidget(gSpeed);
        ui->statusBar->removeWidget(gName);
        /// Изменение возможности использования кнопок меню
        gOpenPort->setEnabled(true);
        gClosePort->setEnabled(false);
        ui->ComSendDataButton->setEnabled(false);
        ui->ComSendFileButton->setEnabled(false);
        ui->ComSaveDataButton->setEnabled(false);
        sendData->setEnabled(false);
        sendFile->setEnabled(false);
        saveData->setEnabled(false);
    }
} // ComClosePort
/*
*/

```

```

/*
*/
/**
    \fn          void MainWindow::ComInfo()
    \brief       Функция информирования о программе.
    \note       Выводит сообщение с информацией о программе.
*/
void MainWindow::ComInfo()
{
    QMessageBox::information(nullptr, "О программе",
                                "Программа предназначена для
приема/передачи\r\n"
                                "данных через последовательный порт с
использованием\r\n"
                                "кодов БЧХ и Рида-Соломона\r\n"
                                "Разработал студент группы 16ВВ1
Иванов Никита");
} // ComInfo
/*
*/

```

Файл «comport.h»

```

#ifndef COMPORT_H
#define COMPORT_H
#include <QObject>
#include <QtSerialPort/QtSerialPort>
#include <QtSerialPort/QtSerialPortInfo>
#include <QDebug>
#include <QThread>
#include <QTextCodec>
#include <QTimer>

class ComPort : public QObject
{
    Q_OBJECT
public:
    explicit ComPort(QObject *parent = nullptr);
    ~ComPort();
    QtSerialPort thisPort;
    QString PortName;
    qint32 BaudRate;
    QTimer *timer;

signals:
    void finished_Port();
    void outPort(QByteArray);
    void opened(QString, QString);
    void notopened();

```

					ПГУ 2.090301.16.001 ПЗ	Лист
						56
Изм.	Лист	№ докум.	Подпись	Дата		

```

public slots:
    void ComDisconnectPort();
    void ComConnectPort(QString name,int baudrate,int DataBits,int
Parity, int StopBits);
    void ComProcessPort();
    void ComWriteToPort(QByteArray data);
    void ComReadInPort();
};

#endif // COMPORT_H
Файл «comport.cpp»

/* _____ */

#include "comport.h"
#include "bch.h"
#include "rs.h"

/* _____ */
int state=0;

ComPort::ComPort(QObject *parent) : QObject(parent)
{
}

ComPort::~ComPort()
{
    emit finished_Port();
}

/* _____ */
/**
    \fn          void ComPort::ComProcessPort()
    \brief       Функция чтения данных по таймеру.
    \note        Производит чтение данных из порта по таймеру.
    \remarks     Запускается при запуске потока.
*/
void ComPort::ComProcessPort()
{
    /// Таймер
    timer = new QTimer();
    /// Связь таймера и функции чтения данных
    connect(timer,SIGNAL(timeout()),this,SLOT(ComReadInPort()));
} // ComProcessPort
/* _____ */

/* _____ */
/**

```

```

        \fn                                void ComPort::ComConnectPort(QString
name,int baudrate,
                                                int DataBits,int Parity,
int StopBits)
        \brief                            Функция открытия порта.
        \param                            name - имя открываемого порта
                                         baudrate - скорость порта
                                         DataBits - данные в бите
                                         Parity - четность
                                         StopBits - стоп биты.
        \note                            Открывает порт с заданными параметрами.
*/
void ComPort::ComConnectPort(QString name,int baudrate,int
DataBits,int Parity, int StopBits)
{
    /// Настройка параметров порта
    thisPort.setPortName(name);

    thisPort.setBaudRate(static_cast<QSerialPort::BaudRate>(baudrate))
;

    thisPort.setDataBits(static_cast<QSerialPort::DataBits>(DataBits))
;
    thisPort.setParity(static_cast<QSerialPort::Parity>(Parity));

    thisPort.setStopBits(static_cast<QSerialPort::StopBits>(StopBits))
;
    thisPort.setFlowControl(QSerialPort::NoFlowControl);

    /// Открытие порта
    thisPort.open(QIODevice::ReadWrite);

    // Открылся ли порт
    if (thisPort.isOpen())
    {
        PortName = name;
        BaudRate = baudrate;
        QString Baudrate = QString::number(BaudRate);
        /// Генерация сигнала opened
        emit opened(PortName,Baudrate);
        /// Запуск таймера
        timer->start(500);
    }

    else
    {
        /// Генерация сигнала notopened
        emit notopened();
    }

} // ComConnectPort

```

```

/* _____ */

/* _____ */
/**
    \fn                void ComPort::ComDisconnectPort()
    \brief            Функция закрытия порта.
    \note            Закрывает порт.
*/
void ComPort::ComDisconnectPort()
{
    // Открыт ли порт
    if (thisPort.isOpen())
    {
        /// Закрытие порта
        thisPort.close();
        timer->stop();
    }
} // ComDisconnectPort
/* _____ */

/* _____ */
/**
    \fn                void ComPort::ComWriteToPort(QByteArray
data)
    \brief            Функция записи данных.
    \param            data - данные для записи.
    \note            Записывает данные в порт.
*/
void ComPort::ComWriteToPort(QByteArray data)
{
    // Открыт ли порт
    if (thisPort.isOpen())
    {
        /// Запись данных в порт
        thisPort.write(data);
        /// Ожидание записи данных в порт
        //thisPort.waitForBytesWritten(100);
        //this->thread()->msleep(50);
    }
} // ComWriteToPort
/* _____ */

/* _____ */
/**
    \fn                void ComPort::ComReadInPort()
    \brief            Функция чтения данных.
    \note            Считывает данные из порта по таймеру.
*/

```

```

void ComPort::ComReadInPort()
{
    // Открыт ли порт
    if (thisPort.isOpen())
    {
        const QByteArray data = thisPort.readAll();

        // Если данные есть
        if (data != "")
        {
            if (data.length() == 2)
            {
                QByteArray siganal = QByteArray::fromHex(data);

                if(siganal == "\\021")
                {
                    state = 1;
                }
                else if(siganal == "\\022")
                {
                    state = 2;
                }
                else if(siganal == "\\023")
                {
                    state = 3;
                }

                else
                {
                    QByteArray rezult = QByteArray::fromHex(data);
                    emit outPort(rezult);
                }
            }

            else if (state == 1)
            {
                QByteArray rezult = QByteArray::fromHex(data);
                emit outPort(rezult);
            }

            else if (state == 2)
            {
                BCH* a;
                QByteArray result;
                result = a->decode(data);
                /// Генерация сигнала outPort
                emit outPort(result);
            }

            else if (state == 3)
            {

```

					ПГУ 2.090301.16.001 ПЗ	Лист
						60
Изм.	Лист	№ докум.	Подпись	Дата		

```

        RS* a;
        QByteArray result;
        result = a->decode(data);
        result = QByteArray::fromHex(result);
        emit outPort(result);
    }

}

}
} // ComReadInPort
/* _____ */

```

Файл «setting.h»

```

#ifndef SETTING_H
#define SETTING_H

#include <QDialog>
#include <QtSerialPort/QSerialPort>
#include <QtSerialPort/QSerialPortInfo>
#include <QString>
#include <QMessageBox>

namespace Ui {
class setting;
}

class setting : public QDialog
{
    Q_OBJECT

public:
    explicit setting(QWidget *parent = nullptr);
    ~setting();
    Ui::setting *ui;

private slots:
    void on_ComOKButton_clicked();
    void on_ComCancelButton_clicked();
};

#endif // SETTING_H

```

Файл «setting.cpp»

```

#include "setting.h"
#include "ui_setting.h"
#include "mainwindow.h"

setting::setting(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::setting)
{

```

```

    /// Задаем фиксированный размер окна
    QWidget::setFixedSize(309,300);

    ui->setupUi(this);

    /// Связь кнопки "OK" со слотом
    connect(ui->ComOKButton,SIGNAL(clicked()),this,SLOT(accept()));

    /// Заполнение comboBox имени порта
    foreach ( const QSerialPortInfo &info ,
    QSerialPortInfo::availablePorts() )
    {
        ui->comboBox->addItem(info.portName());
    }

    /// Заполнение comboBox скорости порта
    ui->comboBox_2->addItem(QString("1200"),QSerialPort::Baud1200);
    ui->comboBox_2->addItem(QString("2400"),QSerialPort::Baud2400);
    ui->comboBox_2->addItem(QString("4800"),QSerialPort::Baud4800);
    ui->comboBox_2->addItem(QString("9600"),QSerialPort::Baud9600);
    ui->comboBox_2->addItem(QString("19200"),QSerialPort::Baud19200);
    ui->comboBox_2->addItem(QString("38400"),QSerialPort::Baud38400);
    ui->comboBox_2->addItem(QString("57600"),QSerialPort::Baud57600);
    ui->comboBox_2->addItem(QString("115200"),QSerialPort::Baud115200);
    /// Задается текущее значение comboBox
    ui->comboBox_2->setCurrentIndex(7);

    /// Заполнение comboBox данных в бите
    ui->comboBox_3->addItem(QString("5"),QSerialPort::Data5);
    ui->comboBox_3->addItem(QString("6"),QSerialPort::Data6);
    ui->comboBox_3->addItem(QString("7"),QSerialPort::Data7);
    ui->comboBox_3->addItem(QString("8"),QSerialPort::Data8);
    /// Задается текущее значение comboBox
    ui->comboBox_3->setCurrentIndex(3);

    /// Заполнение comboBox проверки четности
    ui->comboBox_4->addItem(QString("Без проверки
четности"),QSerialPort::NoParity);
    ui->comboBox_4->addItem(QString("Проверка по
четности"),QSerialPort::EvenParity);
    ui->comboBox_4->addItem(QString("Проверка по
нечетности"),QSerialPort::OddParity);

```

```

        ui->comboBox_4->addItem(QString("Проверка четности по
метке"),QSerialPort::MarkParity);
        ui->comboBox_4->addItem(QString("Проверка четности по
паузе"),QSerialPort::SpaceParity);

        /// Заполнение comboBox стоп битов
        ui->comboBox_5->addItem(QString("1"),QSerialPort::OneStop);
        ui->comboBox_5-
>addItem(QString("1.5"),QSerialPort::OneAndHalfStop);
        ui->comboBox_5->addItem(QString("2"),QSerialPort::TwoStop);
    }

setting::~setting()
{
    delete ui;
}

/* _____ */
/**
    \fn                void setting::on_ComOKButton_clicked()
    \brief             Нажатие кнопки "ОК"
*/
void setting::on_ComOKButton_clicked()
{} // on_ComOKButton_clicked
/* _____ */

/* _____ */
/**
    \fn                void
setting::on_ComCancelButton_clicked()
    \brief             Нажатие кнопки "Отмена"
    \note              Закрывает окно настроек
*/
void setting::on_ComCancelButton_clicked()
{
    /// Закрывает окно
    this->close();
} // on_ComCancelButton_clicked
/* _____ */

```

Файл «bch.h»

```

#ifndef BCH_H
#define BCH_H
#include <QObject>

class BCH : public QObject
{
    Q_OBJECT
public:
    explicit BCH(QObject *parent = nullptr);

```

```

~BCH();
QByteArray encode(QByteArray);
QByteArray decode(QByteArray);
QByteArray ErrGen(QByteArray);

};
#endif // BCH_H

    Файл «bch.cpp»
#include "bch.h"
#include <QDebug>

BCH::BCH(QObject *parent) : QObject(parent)
{
}

BCH::~BCH()
{
}

/* _____ */
/**
    \fn                QByteArray BCH::encode(QByteArray)
    \brief             Функция кодирования БЧХ кода.
    \note              Производит кодирование входного сообщения длиной
7 и возвращает
                        сообщение длиной 15.
*/
QByteArray BCH::encode(QByteArray message)
{
    int    i, j;
    int    feedback;
    int bb[9];
    int g[9] = {1,0,0,0,1,0,1,1,1};
    for (i = 0; i < 8; i++)
        bb[i] = 0;
    for (i = 6; i >= 0; i--) {
        feedback = (message[i] - 48) ^ bb[7];
        if (feedback != 0) {
            for (j = 7; j > 0; j--)
                if (g[j] != 0)
                    bb[j] = bb[j - 1] ^ feedback;
            else
                bb[j] = bb[j - 1];
            bb[0] = g[0] && feedback;
        }
        else {
            for (j = 7; j > 0; j--)
                bb[j] = bb[j - 1];
            bb[0] = 0;
        }
    }
}

```

```

    }
    QByteArray s;
    for(int i=0;i<9;i++)
    {
        if (bb[i] == 0)
        {
            s.push_back('0');
        }
        else if (bb[i] == 1)
        {
            s.push_back('1');
        }
    }
    return s;
} // encode
/* _____ */

/* _____ */
/**
    \fn          QByteArray BCH::decode(QByteArray)
    \brief       Функция декодирования БЧХ кода.
    \note        Производит декодирование входного сообщения
    длиной 15 и возвращает
                    сообщение длиной 7 с исправлением до 2 ошибок.
*/
QByteArray BCH::decode(QByteArray message)
{
    QByteArray output;
    int i, j, u, q, t2, count = 0, syn_error = 0, t=2, n=15;
    int elp[500][500], d[1026], l[1026], u_lu[1026], s[6];
    int root[200], loc[200], reg[201];

    int alpha_to[15]={1,2,4,8,3,6,12,11,5,10,7,14,15,13,9};
    int index_of[16]={-1,0,1,4,2,8,5,10,3,14,9,7,6,13,11,12};
    int simbol[8];

    t2 = 2 * t;

    while (message.length()>0)
    {
        for(int x=0;x<2;x++)
        {
            int mes[15];

            for (int i=0; i<15; i++)
            {
                mes[i]=0;
            }

            for (int i=0; i<15; i++)
            {

```

```

        if (message[i]=='1')
            mes[i]=1;
        else if (message[i]=='1')
        {
            mes[i]=0;
        }
    }
    message.remove(0,15);

    /// Получение синдрома
    for (i = 1; i <= t2; i++) {
        s[i] = 0;
        for (j = 0; j < 15; j++)
            if (mes[j] != 0)
                s[i] ^= alpha_to[(i * j) % n];
        if (s[i] != 0)
            syn_error = 1; ///< устанавливается если
синдром не 0
        s[i] = index_of[s[i]];
    }
    /// если ошибка есть
    /// выполняется алгоритм Берлекэмпа - Мэсси
    if (syn_error) {
        d[0] = 0;
        d[1] = s[1];
        elp[0][0] = 0;
        elp[1][0] = 1;
        for (i = 1; i < t2; i++) {
            elp[0][i] = -1;
            elp[1][i] = 0;
        }
        l[0] = 0;
        l[1] = 0;
        u_lu[0] = -1;
        u_lu[1] = 0;
        u = 0;

        do {
            u++;
            if (d[u] == -1) {
                l[u + 1] = l[u];
                for (i = 0; i <= l[u]; i++) {
                    elp[u + 1][i] = elp[u][i];
                    elp[u][i] = index_of[elp[u][i]];
                }
            }
            else
            {
                q = u - 1;
                while ((d[q] == -1) && (q > 0))

```

```

        q--;
    if (q > 0) {
        j = q;
        do {
            j--;
            if ((d[j] != -1) && (u_lu[q] <
u_lu[j]))

                q = j;
        } while (j > 0);
    }

    if (l[u] > l[q] + u - q)
        l[u + 1] = l[u];
    else
        l[u + 1] = l[q] + u - q;

    for (i = 0; i < t2; i++)
        elp[u + 1][i] = 0;
    for (i = 0; i <= l[q]; i++)
        if (elp[q][i] != -1)
            elp[u + 1][i + u - q] =
alpha_to[(d[u] + n - d[q] +
elp[q][i]) % n];

        for (i = 0; i <= l[u]; i++) {
            elp[u + 1][i] ^= elp[u][i];
            elp[u][i] = index_of[elp[u][i]];
        }
    u_lu[u + 1] = u - l[u + 1];

    if (u < t2) {
        if (s[u + 1] != -1)
            d[u + 1] = alpha_to[s[u + 1]];
        else
            d[u + 1] = 0;
        for (i = 1; i <= l[u + 1]; i++)
            if ((s[u + 1 - i] != -1) && (elp[u +
1][i] != 0))

                d[u + 1] ^= alpha_to[(s[u + 1 - i]
+ index_of[elp[u + 1][i]]) %
n];

        d[u + 1] = index_of[d[u + 1]];
    }
} while ((u < t2) && (l[u + 1] <= t));

u++;
/// если число науденных ошибок не превышает
максимальное число ошибок
    if (l[u] <= t) {
        for (i = 0; i <= l[u]; i++)
            elp[u][i] = index_of[elp[u][i]];

```

```

        /// алгоритм Ченя
        for (i = 1; i <= l[u]; i++)
            reg[i] = elp[u][i];
        count = 0;
        for (i = 1; i <= n; i++) {
            q = 1;
            for (j = 1; j <= l[u]; j++)
                if (reg[j] != -1) {
                    reg[j] = (reg[j] + j) % n;
                    q ^= alpha_to[reg[j]];
                }
            if (!q) {
                root[count] = i;
                loc[count] = n - i;
                count++;
            }
        }
        if (count == l[u])
            for (i = 0; i < l[u]; i++)
                mes[loc[i]] ^= 1;
    }
}
if (x == 0)
{
    for(int i = 0; i < 7; i++)
    {
        simbol[i] = mes[i + 8];
    }
}
else if (x == 1)
{
    simbol[7] = mes[8];
}
}

/// перевод двоичной информации в символ
int rez = 0;
int a = 1;
int k = 8 - 1;
for (int i = 0; i < 9; i++, k--)
    if (simbol[i] == 1)
        rez |= a << k;

char c = rez;
output.push_back(c);

for(int i = 0; i < 8; i++)
{
    simbol[i] = 0;
}

```

```

    }

    }
    return output;
} // decode
/* _____ */

/* _____ */
/**
    \fn          QByteArray BCH::ErrGen(QByteArray)
    \brief       Функция генерации ошибки.
*/
QByteArray BCH::ErrGen(QByteArray message)
{
    int numerr, errpos[2];
    numerr = rand() % 3;
    for (int i = 0; i < numerr; i++)
        errpos[i] = rand() % 15;
    if (numerr)
        for (int i = 0; i < numerr; i++)
        {
            int op = message[errpos[i]];
            op ^= 1;
            message[errpos[i]] = op;
        }
    return message;
} // ErrGen
/* _____ */

```

Файл «rscode.h»

```

#ifndef RSCODE_H
#define RSCODE_H
#include <QObject>

class RS : public QObject
{
    Q_OBJECT
public:
    explicit RS(QObject *parent = nullptr);
    ~RS();
    QByteArray encode(QByteArray);
    QByteArray decode(QByteArray);

    int m = 4;          /* GF(2**4) */
    int n = 15;
    int t = 2;
    int k = 11;
};
#endif // RSCODE_H

```

Файл «rscode.cpp»

```

#include "rs.h"

```

					ПГУ 2.090301.16.001 ПЗ	Лист
						69
Изм.	Лист	№ докум.	Подпись	Дата		

```

RS::RS(QObject *parent) : QObject(parent)
{
}

RS::~~RS()
{
}

/* _____ */
/**
    \fn          QByteArray RS::encode(QByteArray)
    \brief       Функция кодирования РС кода.
    \note       Производит кодирование входного сообщения длиной
11 и возвращает сообщение длиной 15.
*/
QByteArray RS::encode(QByteArray message)
{
    int g[5] = {10,3,6,13,0};
    int alpha_to[16]={1,2,4,8,3,6,12,11,5,10,7,14,15,13,9,0};
    int index_of[16]={-1,0,1,4,2,8,5,10,3,14,9,7,6,13,11,12};

    int b[4];
    int i, j;
    int feedback;
    int data[11];
    QByteArray output;

    for (int i = 0; i < 11; i++)
    {
        data[i]=0;
    }

    for (int i = 0; i < message.length(); i++)
    {
        QByteArray result;

        char asd = message[i];
        result.append(asd);
        result = result.toHex();
        output.append(result);

        for (int k=0;k<2;k++)
        {
            int pos;
            if (k == 0)
                pos=i*2;
            else if (k==1) {
                pos=i*2+1;
            }

```

```

switch (result[k]) {
case '0':
    data[pos] = 0x00;
    break;
case '1':
    data[pos] = 0x01;
    break;
case '2':
    data[pos] = 0x02;
    break;
case '3':
    data[pos] = 0x03;
    break;
case '4':
    data[pos] = 0x04;
    break;
case '5':
    data[pos] = 0x05;
    break;
case '6':
    data[pos] = 0x06;
    break;
case '7':
    data[pos] = 0x07;
    break;
case '8':
    data[pos] = 0x08;
    break;
case '9':
    data[pos] = 0x09;
    break;
case 'a':
    data[pos] = 0x0a;
    break;
case 'b':
    data[pos] = 0x0b;
    break;
case 'c':
    data[pos] = 0x0c;
    break;
case 'd':
    data[pos] = 0x0d;
    break;
case 'e':
    data[pos] = 0x0e;
    break;
case 'f':
    data[pos] = 0x0f;
    break;
default:
    break;

```

```

        }
    }

    if (message.length() < 5)
    {
        int need = 5 - message.length();
        for (i=0; i<need; i++)
        {
            output.append("00");
        }
    }

    for (i = 0; i < 4; i++)
        b[i] = 0;
    for (i = 10; i >= 0; i--)
    {
        int z = data[i];
        int y = b[3];
        int beba = z ^ y;
        feedback = index_of[beba];

        if (feedback != -1)
        {
            for (j = 3; j > 0; j--)
                if (g[j] != -1)
                    b[j] = b[j - 1] ^ alpha_to[(g[j] + feedback) %
15];

            else
                b[j] = b[j - 1];
            b[0] = alpha_to[(g[0] + feedback) % 15];
        }
        else
        {
            for (j = 3; j > 0; j--)
                b[j] = b[j - 1];
            b[0] = 0;
        }
    }

    QByteArray s;
    for(i=4; i>=0; i--)
    {
        switch (b[i]) {
            case 0:
                output.push_front('0');
                break;
            case 1:
                output.push_front('1');
                break;
        }
    }

```

```

        case 2:
            output.push_front('2');
            break;
        case 3:
            output.push_front('3');
            break;
        case 4:
            output.push_front('4');
            break;
        case 5:
            output.push_front('5');
            break;
        case 6:
            output.push_front('6');
            break;
        case 7:
            output.push_front('7');
            break;
        case 8:
            output.push_front('8');
            break;
        case 9:
            output.push_front('9');
            break;
        case 10:
            output.push_front('a');
            break;
        case 11:
            output.push_front('b');
            break;
        case 12:
            output.push_front('c');
            break;
        case 13:
            output.push_front('d');
            break;
        case 14:
            output.push_front('e');
            break;
        case 15:
            output.push_front('f');
            break;
    }
}
output.push_back('0');
output[5]='a';
output[10]='7';

return output;
} // encode
/* _____ */

```

```

/* _____ */
/**
    \fn                QByteArray RS::decode(QByteArray)
    \brief             Функция декодирования РС кода.
    \note              Производит декодирование входного сообщения
диной 15 и возвращет
                        сообщение длиной 11 с исправлением до 2 ошибок.
*/
QByteArray RS::decode(QByteArray message)
{
    int alpha_to[16]={1,2,4,8,3,6,12,11,5,10,7,14,15,13,9,0};
    int index_of[16]={-1,0,1,4,2,8,5,10,3,14,9,7,6,13,11,12};
    int data[15];
    QByteArray result;

    while (message.length()>0)
    {
        for (int i=0; i<15; i++)
        {
            data[i]=0;
        }

        for (int i=0; i<15; i++)
        {
            switch (message[i]) {
            case '0':
                data[i] = 0x00;
                break;
            case '1':
                data[i] = 0x01;
                break;
            case '2':
                data[i] = 0x02;
                break;
            case '3':
                data[i] = 0x03;
                break;
            case '4':
                data[i] = 0x04;
                break;
            case '5':
                data[i] = 0x05;
                break;
            case '6':
                data[i] = 0x06;
                break;
            case '7':
                data[i] = 0x07;
                break;
            case '8':

```

```

        data[i] = 0x08;
        break;
    case '9':
        data[i] = 0x09;
        break;
    case 'a':
        data[i] = 0x0a;
        break;
    case 'b':
        data[i] = 0x0b;
        break;
    case 'c':
        data[i] = 0x0c;
        break;
    case 'd':
        data[i] = 0x0d;
        break;
    case 'e':
        data[i] = 0x0e;
        break;
    case 'f':
        data[i] = 0x0f;
        break;
    default:
        break;
    }
}

int i, j, u, q;
int elp[6][4], d[6], l[6], u_lu[6], s[5];
int count = 0, syn_error = 0, root[2], loc[2], z[3],
err[15], reg[3];

for (i = 0; i < 15; i++)
    data[i] = index_of[data[i]];

/// поиск синдрома
for (i = 1; i <= 4; i++)
{
    s[i] = 0;
    for (j = 0; j < 15; j++)
        if (data[j] != -1)
            s[i] ^= alpha_to[(data[j] + i * j) % 15];
    if (s[i] != 0) syn_error = 1;
    s[i] = index_of[s[i]];
};

if (syn_error)
{
    d[0] = 0;

```

					ПГУ 2.090301.16.001 ПЗ	Лист
						75
Изм.	Лист	№ докум.	Подпись	Дата		

```

d[1] = s[1];
elp[0][0] = 0;
elp[1][0] = 1;
for (i = 1; i < 4; i++)
{
    elp[0][i] = -1;
    elp[1][i] = 0;
}
l[0] = 0;
l[1] = 0;
u_lu[0] = -1;
u_lu[1] = 0;
u = 0;

do
{
    u++;
    if (d[u] == -1)
    {
        l[u + 1] = l[u];
        for (i = 0; i <= l[u]; i++)
        {
            elp[u + 1][i] = elp[u][i];
            elp[u][i] = index_of[elp[u][i]];
        }
    }
    else
    {
        q = u - 1;
        while ((d[q] == -1) && (q > 0)) q--;
        if (q > 0)
        {
            j = q;
            do
            {
                j--;
                if ((d[j] != -1) && (u_lu[q] <
u_lu[j]))
                    q = j;
            } while (j > 0);
        };

        if (l[u] > l[q] + u - q) l[u + 1] = l[u];
        else l[u + 1] = l[q] + u - q;

        for (i = 0; i < 4; i++) elp[u + 1][i] = 0;
        for (i = 0; i <= l[q]; i++)
            if (elp[q][i] != -1)
                elp[u + 1][i + u - q] = alpha_to[(d[u]
+ 15 - d[q] + elp[q][i]) % 15];
        for (i = 0; i <= l[u]; i++)

```

```

        {
            elp[u + 1][i] ^= elp[u][i];
            elp[u][i] = index_of[elp[u][i]];
        }
    }
    u_lu[u + 1] = u - l[u + 1];

    if (u < 4)
    {
        if (s[u + 1] != -1)
            d[u + 1] = alpha_to[s[u + 1]];
        else
            d[u + 1] = 0;
        for (i = 1; i <= l[u + 1]; i++)
            if ((s[u + 1 - i] != -1) && (elp[u + 1][i]
!= 0))
                d[u + 1] ^= alpha_to[(s[u + 1 - i] +
index_of[elp[u + 1][i]]) % 15];
            d[u + 1] = index_of[d[u + 1]];
        }
    } while ((u < 4) && (l[u + 1] <= 2));

    u++;
    if (l[u] <= 2)
    {
        for (i = 0; i <= l[u]; i++)    elp[u][i] =
index_of[elp[u][i]];

        for (i = 1; i <= l[u]; i++)
            reg[i] = elp[u][i];
        count = 0;
        for (i = 1; i <= 15; i++)
        {
            q = 1;
            for (j = 1; j <= l[u]; j++)
                if (reg[j] != -1)
                {
                    reg[j] = (reg[j] + j) % 15;
                    q ^= alpha_to[reg[j]];
                };
            if (!q)
            {
                root[count] = i;
                loc[count] = 15 - i;
                count++;
            };
        };
        if (count == l[u])
        {
            for (i = 1; i <= l[u]; i++)
            {

```

```

        if ((s[i] != -1) && (elp[u][i] != -1))
            z[i] = alpha_to[s[i]] ^
alpha_to[elp[u][i]];
        else if ((s[i] != -1) && (elp[u][i] == -
1))
            z[i] = alpha_to[s[i]];
        else if ((s[i] == -1) && (elp[u][i] != -
1))
            z[i] = alpha_to[elp[u][i]];
        else
            z[i] = 0;
        for (j = 1; j < i; j++)
            if ((s[j] != -1) && (elp[u][i - j] !=
-1))
                z[i] ^= alpha_to[(elp[u][i - j] +
s[j]) % 15];
        z[i] = index_of[z[i]];
    };

    for (i = 0; i < 15; i++)
    {
        err[i] = 0;
        if (data[i] != -1)
            data[i] = alpha_to[data[i]];
        else data[i] = 0;
    }
    for (i = 0; i < l[u]; i++)
    {
        err[loc[i]] = 1;
        for (j = 1; j <= l[u]; j++)
            if (z[j] != -1)
                err[loc[i]] ^= alpha_to[(z[j] + j
* root[i]) % 15];
        if (err[loc[i]] != 0)
        {
            err[loc[i]] = index_of[err[loc[i]]];
            q = 0;
            for (j = 0; j < l[u]; j++)
                if (j != i)
                    q += index_of[l ^
alpha_to[(loc[j] + root[i]) % 15]];
            q = q % 15;
            err[loc[i]] = alpha_to[(err[loc[i]] -
q + 15) % 15];
            data[loc[i]] ^= err[loc[i]];
        }
    }
}
else
    for (i = 0; i < 15; i++)
        if (data[i] != -1)

```

```

        data[i] = alpha_to[data[i]];
        else data[i] = 0;
    }
    else
        for (i = 0; i < 15; i++)
            if (data[i] != -1)
                data[i] = alpha_to[data[i]];
            else data[i] = 0;
    }
    else
    {
        for (i = 0; i < 15; i++)
            if (data[i] != -1)
                data[i] = alpha_to[data[i]];
            else data[i] = 0;
    }

    for(int i=4;i<14;i+=2)
    {
        if ((data[i]==0)&&(data[i+1]==0))
        {}
        else
        {
            int pos;
            for (int j=0;j<2;j++)
            {
                if(j==0)
                    pos = i;
                else if(j==1)
                    pos =i+1;

                switch (data[pos]) {
                case 0:
                    result.push_back('0');
                    break;
                case 1:
                    result.push_back('1');
                    break;
                case 2:
                    result.push_back('2');
                    break;
                case 3:
                    result.push_back('3');
                    break;
                case 4:
                    result.push_back('4');
                    break;
                case 5:
                    result.push_back('5');
                    break;
                case 6:

```

```

        result.push_back('6');
        break;
    case 7:
        result.push_back('7');
        break;
    case 8:
        result.push_back('8');
        break;
    case 9:
        result.push_back('9');
        break;
    case 10:
        result.push_back('a');
        break;
    case 11:
        result.push_back('b');
        break;
    case 12:
        result.push_back('c');
        break;
    case 13:
        result.push_back('d');
        break;
    case 14:
        result.push_back('e');
        break;
    case 15:
        result.push_back('f');
        break;
    }
}

}

message.remove(0,15);
}

return result;

} // decode
/* _____ */

```

					ПГУ 2.090301.16.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В. Презентация

ТЕМА РАБОТЫ: «СИСТЕМА ПЕРЕДАЧИ ПОСРЕДСТВОМ RS-232 С ИСПОЛЬЗОВАНИЕМ АЛГОРИТМОВ ВОССТАНОВЛЕНИЯ ДАННЫХ»

Автор работы: студент группы 16вв1 Иванов Н.А.

Научный руководитель: д.т.н., профессор каф.
ВТ ПГУ Гурин Е.И.

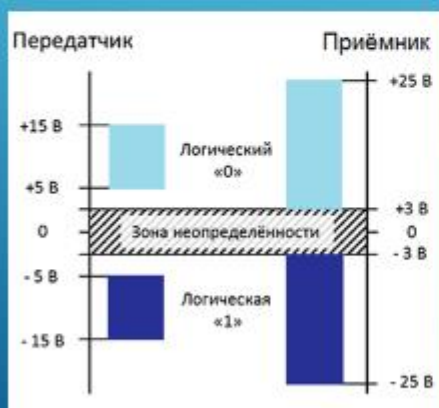
ПОСТАНОВКА ЗАДАЧИ

- ▶ Требуется разработать приложение, осуществляющее обмен кодированными данными посредством последовательного порта.
- ▶ Для передачи информации на большое расстояние необходимо предусмотреть возможность восстановления искаженных данных.
- ▶ Пользователь должен иметь возможность задавать параметры COM порта и выбирать метод кодирования информации.
- ▶ Приложение должно иметь понятный пользовательский интерфейс.

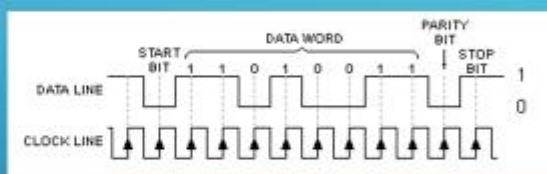
					ПГУ 2.090301.16.001 ПЗ	Лист
						81
Изм.	Лист	№ докум.	Подпись	Дата		

ПОСЛЕДОВАТЕЛЬНЫЙ ПОРТ

Значения логического «0» и «1»



Пример передаваемой информации



ЦИКЛИЧЕСКИЕ КОДЫ

- ▶ Несистематическое кодирование:

$$c(x) = m(x)g(x)$$
- ▶ Систематическое кодирование:

$$s(x) = x^r m(x) \bmod g(x)$$

$$c(x) = x^r m(x) + s(x)$$
- ▶ Коды БЧХ и Рида – Соломона:
 N – общее количество символов
 K – полезное количество символов
 $R = N - K$ – избыточная информация
 D – расстояние Хэмминга
 $t = \frac{D-1}{2}$ – корректирующая способность

ПОЛЕ ГАЛУА

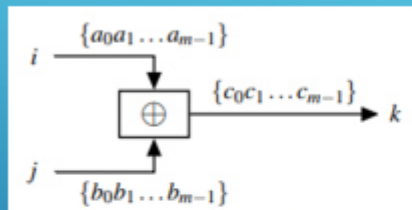


Схема сложения

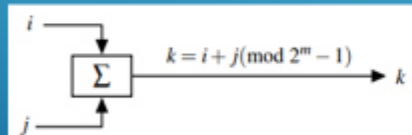


Схема умножения

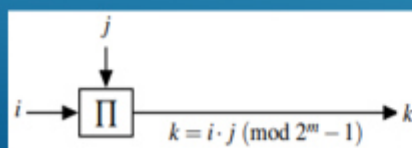


Схема возведения в степень

$$\alpha^n = \alpha^{n-3} + \alpha^{n-4}$$

Поле Галуа $GF(2^4)$. $p(x) = x^4 + x + 1$

Элемент поля	Полином	Двоичный вид	Десятичный вид
α^0	1	0001	1
α^1	x	0010	2
α^2	x^2	0100	4
α^3	x^3	1000	8
α^4	$x+1$	0011	3
α^5	x^2+x	0110	6
α^6	x^3+x^2	1100	12
α^7	x^3+x+1	1011	11
α^8	x^2+1	0101	5
α^9	x^3+x	1010	10
α^{10}	x^2+x+1	0111	7
α^{11}	x^3+x^2+x	1110	14
α^{12}	x^3+x^2+x+1	1111	15
α^{13}	x^3+x^2+1	1101	13
α^{14}	x^3+1	1001	9
α^{15}	1	0001	1

КОДИРОВАНИЕ И ДЕКОДИРОВАНИЕ ДАННЫХ

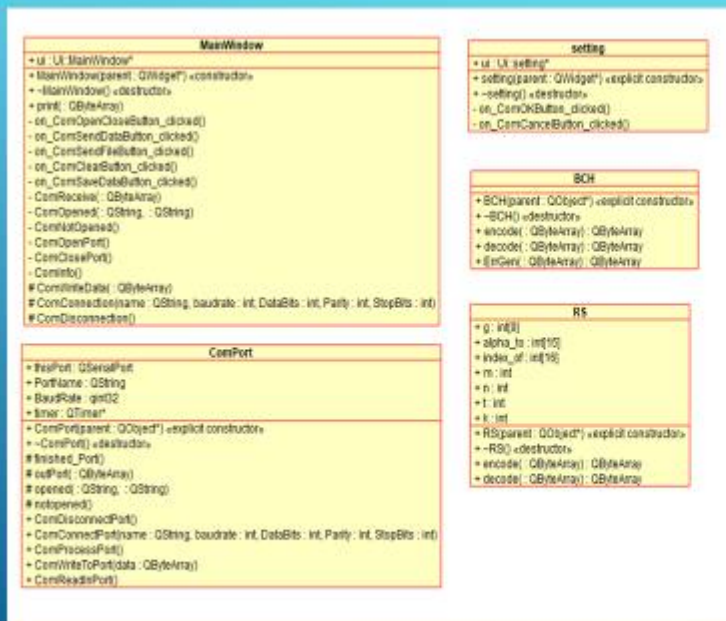
Схема обмена кодированной информацией



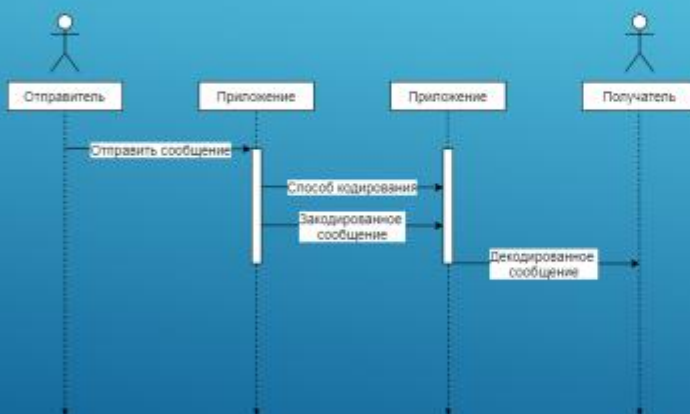
Алгоритм декодирования информации:

- ▶ Вычисление синдрома ошибки;
- ▶ Определение коэффициентов полинома локаторов ошибок $\sigma(x)$, то есть решение ключевого уравнения;
- ▶ нахождение позиции ошибок;
- ▶ исправление ошибок.

UML- ДИАГРАММА КЛАССОВ



UML-ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ

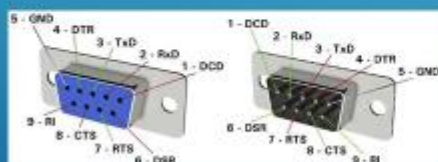
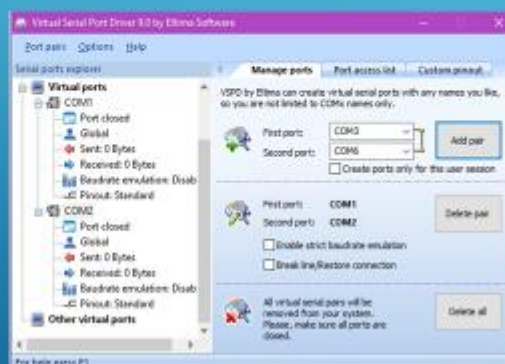




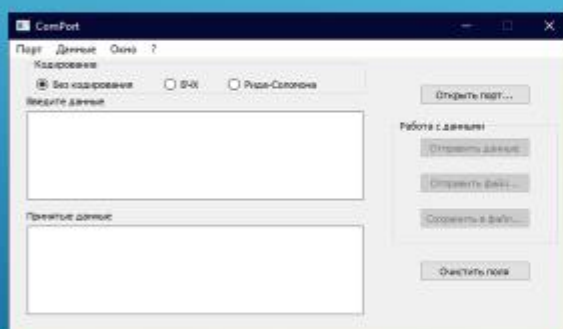
ОТЛАДКА И ТЕСТИРОВАНИЕ

Была выполнена экспериментальная проверка работоспособности при помощи:

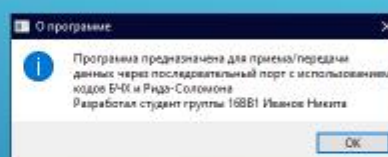
- Кабеля стандарта RS-232
- Пары виртуальных портов



ГЛАВНОЕ ОКНО ПРИЛОЖЕНИЯ



Сообщение «О программе»



Окно «settings»

Сообщение об ошибке

ОБМЕН СООБЩЕНИЯМИ

ЗАКЛЮЧЕНИЕ

- ▶ Результатом выполненной работы является приложение, позволяющее обмениваться кодированной информацией с помощью последовательного порта. Разработанное программное обеспечение позволяет восстанавливать искаженные данные, полученные под действием внешней среды, благодаря использованию кодов Боуза – Чоудхури – Хоквингема и Рида – Соломона.
- ▶ В ходе выполнения выпускной квалификационной работы были изучены методы кодирования и декодирования данных, возможность передачи данных посредством последовательного порта стандарта RS-232 и восстановление данных в случае их искажения при передаче. Также были получены навыки объектно-ориентированном программировании на языке C++ в среде разработки Qt.

Спасибо за внимание!

					ПГУ 2.090301.16.001 ПЗ	Лист
						88
Изм.	Лист	№ докум.	Подпись	Дата		