

Утверждаю:

Галкин В.А.

"__" _____ 2022 г.

Курсовая работа по дисциплине
«Сетевые технологии»
«Программа пересылки сообщений»

Пояснительная записка

(вид документа)

писчая бумага

(вид носителя)

21

(количество листов)

ИСПОЛНИТЕЛИ:

студенты группы ИУ5Ц-82Б
ИУ5Ц-81Б

Пылаев Б.А.

Чиварзин А.С.

Коротенко Е.А.

"__" _____ 2022 г.

Оглавление

1. Введение	3
2. Требования к программе	3
3. Требования к программе	3
4. Физический уровень.....	4
4.1 Сигналы интерфейса RS-232-C.....	4
4.2 Нуль-модемный интерфейс.....	7
4.2.1. Настройка COM-порта средствами библиотеки для C#.....	9
4.2.2. Описание класса SerialPort	9
4.2.3. Конструкторы класса	9
4.3. Функции физического уровня.....	12
4.4.1. Открытие порта	12
4.4.2. Закрытие порта	13
4.4.3. Передача данных.....	14
4.4.4. Прием данных	14
4.4.5. Поддержание соединения.....	15
Поддержание физического соединения осуществляется отправкой сигналов <i>LINKACTIVE</i> каждые 10 секунд и получением ответа <i>ACK_LINKACTIVE</i> о том, что соединение поддерживается.	15
5. Канальный уровень	15
5.1. Функции канального уровня	15
5.2. Протокол связи	15
5.3. Защита передаваемой информации.....	16
5.4. Кодирование кодом Хемминга.....	16
5.5. Форматы кадров.....	18
5.1.1. Служебные супервизорные кадры.....	18
5.1.2. Супервизорные кадры передачи параметров	19
5.1.3. Информационные кадры.....	19
6. Прикладной уровень.....	20

1. Введение

Данная программа, выполненная в рамках курсовой работы по предмету «Сетевые технологии в автоматизированных системах обработки информации и управления», предназначена для пересылки текста диалога абонентов между двумя соединёнными с помощью интерфейса RS232C компьютерами.

2. Требования к программе

Программное изделие выполняется на C# под управлением MS Windows.

Для работы программы требуются 2 ПК типа IBM PC AT (/XT), соединенные нульмодемным кабелем через интерфейс RS-232C.

Важно, чтобы пользователь обладал правами администратора в ОС Windows на ПК.

3. Требования к программе

См. Схему «Структурная схема программы»

4. Физический уровень

4.1 Сигналы интерфейса RS-232-C.

Последовательная передача данных означает, что данные передаются по единственной линии. При этом биты байта данных передаются по очереди с использованием одного провода. Для синхронизации группе битов данных обычно предшествует специальный *стартовый бит*, после группы битов следуют *бит проверки на четность* и один или два *стоповых бита*, как показано на рисунке 1. Иногда бит проверки на четность может отсутствовать.

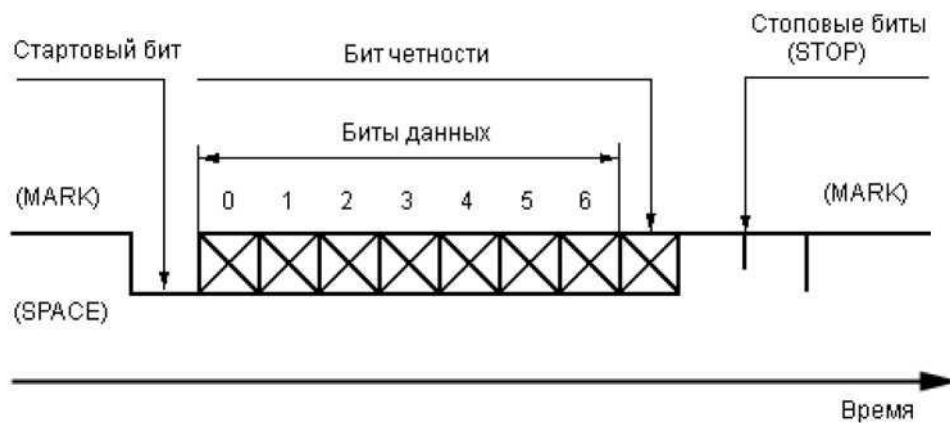


Рисунок 1.

Из рисунка видно, что исходное состояние линии последовательной передачи данных - уровень логической 1. Это состояние линии называют отмеченным — **MARK**. Когда начинается передача данных, уровень линии переходит в 0. Это состояние линии называют пустым — **SPACE**. Если линия находится в таком состоянии больше определенного времени, считается, что линия перешла в состояние разрыва связи — **BREAK**.

Стартовый бит **START** сигнализирует о начале передачи данных. Далее передаются биты данных, вначале младшие, затем старшие.

Контрольный бит формируется на основе правила, которое создается при настройке передающего и принимающего устройства. Контрольный бит может быть установлен с контролем на четность, нечетность, иметь постоянное значение 1 либо отсутствовать совсем.

Если используется бит четности **P**, то передается и он. Бит четности имеет такое значение, чтобы в пакете битов общее количество единиц (или нулей) было четно или нечетно, в зависимости от установки регистров порта. Этот бит служит для обнаружения ошибок, которые могут возникнуть при передаче данных из-за помех на линии. Приемное устройство заново вычисляет четность данных и сравнивает результат с принятым битом четности. Если четность не совпала, то считается, что данные переданы с ошибкой. Конечно, такой алгоритм не дает стопроцентной гарантии обнаружения ошибок. Так, если при передаче данных изменилось четное число битов, то четность сохраняется, и ошибка не будет обнаружена. Поэтому на практике применяют более сложные методы обнаружения ошибок.

В самом конце передаются один или два стоповых бита **STOP**, завершающих передачу байта. Затем до прихода следующего стартового бита линия снова переходит в состояние **MARK**.

Использование бита четности, стартовых и стоповых битов определяют формат передачи данных. Очевидно, что передатчик и приемник должны использовать один и тот же формат данных, иначе обмен будет невозможен.

Другая важная характеристика — скорость передачи. Она также должна быть одинаковой для передатчика и приемника.

Скорость изменения информативного параметра сигнала обычно измеряется в бодах.

Иногда используется другой термин — биты в секунду (bps). Здесь имеется в виду эффективная скорость передачи данных, без учета служебных битов.

Интерфейс RS232C описывает несимметричный интерфейс, работающий в режиме последовательного обмена двоичными данными. Интерфейс поддерживает как асинхронный, так и синхронный режимы работы.

Интерфейс называется несимметричным, если для всех цепей обмена интерфейса используется один общий возвратный провод — сигнальная «земля».

Интерфейсы 25-ти (DB25) или 9-ти (DB9) контактный разъем.

Наименование сигнала	Цепь	Номер контакта	
		DB25P	DB9S
DCD (Data Carrier Detect)	109	8	1
RD (Receive Data)	104	3	2
TD (Transmit Data)	103	2	3
DTR (Data Terminal Ready)	108	20	4
GND (Signal Ground)	102	7	5
DSR (Data Set Ready)	107	6	6
RTS (Request To Send)	105	4	7
CTS (Clear To Send)	106	5	8
RI (Ring Indicator)	125	22	9

Таблица 1.

В интерфейсе реализован биполярный потенциальный код на линиях между DTE и DCE. Напряжения сигналов в цепях обмена симметричны по отношению к уровню сигнальной «земли» и составляют не менее +3В для двоичного нуля и не более -3В для двоичной единицы.

Входы TD и RD используются устройствами DTE и DCE по-разному. DTE использует вход TD для передачи данных, а вход RD для приема данных. И наоборот, устройство DCE использует вход TD для приема, а вход RD для передачи данных. Поэтому для соединения двух DTE необходимо перекрестное соединение линий TD и RD в нуль-модемном кабеле.

Рассмотрим самый низкий уровень управления связью - подтверждение связи.

В начале сеанса связи компьютер (DTE) должен удостовериться, что

модем (DCE) находится в рабочем состоянии. Для этой цели компьютер подает сигнал по линии DTR. В ответ модем подает сигнал по линии DSR. Затем, после вызова абонента, модем подает сигнал по линии DCD, чтобы сообщить компьютеру, что он произвел соединение с удаленной системой.

Более высокий уровень используется для управления потоком (скоростью обмена данными) и также реализуется аппаратно. Этот уровень необходим для того, чтобы предотвратить передачу большего числа данных, чем то, которое может быть обработано принимающей системой.

В полудуплексных соединениях DTE подает сигнал RTS, когда оно желает передать данные. DCE отвечает сигналом по линии CTS, когда оно готово, и DTE начинает передачу данных. До тех пор, пока оба сигнала RTS и CTS не примут активное состояние, только DCE может передавать данные. Иногда для соединения двух устройств DTE эти линии (RTS и CTS) соединяются вместе на каждом конце кабеля. В результате получаем то, что другое устройство всегда готово для получения данных (если при большой скорости передачи принимающее устройство не успевает принимать и обрабатывать данные, возможна потеря данных).

Для решения всех этих проблем для соединения двух устройств типа DTE используется специальный нуль-модемный кабель.

4.2 Нуль-модемный интерфейс

Обмен сигналами между адаптером компьютера (DTE) и модемом (DCE) (или 2-м компьютером, присоединенным к исходному посредством кабеля стандарта RS-232C) строится по стандартному сценарию, в котором каждый сигнал генерируется сторонами лишь после наступления определенных условий. Такая процедура обмена информацией называется запрос/ответным режимом, или “**рукопожатием**” (**handshaking**). Большинство из приведенных в таблице сигналов как раз и нужны для аппаратной реализации “рукопожатия” между адаптером и модемом.

Обмен сигналами между сторонами интерфейса **RS-232C** выглядит так:

1. компьютер после включения питания и открытия COM-порта выставляет сигнал **DTR**, который удерживается активным. Если модем включен в электросеть и исправен, он отвечает компьютеру сигналом **DSR**. Этот сигнал служит подтверждением того, что **DTR** принят, и информирует компьютер о готовности модема к приему информации;
2. если компьютер получил сигнал **DSR** и хочет передать данные, он выставляет сигнал **RTS**;
3. если модем готов принимать данные, он отвечает сигналом **CTS**. Он служит для компьютера подтверждением того, что **RTS** получен модемом и модем готов принять данные от компьютера. С этого момента адаптер может бит за битом передавать информацию по линии **TD**;
4. получив байт данных, модем может сбросить свой сигнал **CTS**, информируя компьютер о необходимости “притормозить” передачу следующего байта, например, из-за переполнения внутреннего буфера; программа компьютера, обнаружив сброс **CTS**, прекращает передачу данных, ожидая повторного появления **CTS**.

Модем может передать данные в компьютер, когда он обнаружит несущую в линии и выставит сигнал — **DCD**. Программа компьютера, принимающая данные, обнаружив этот сигнал, читает приемный регистр, в который сдвиговый регистр “собрал” биты, принятые по линии приема данных **RD**. Когда для связи используются только приведенные в таблице данные, компьютер не может попросить модем “повременить” с передачей следующего байта. Как следствие, существует опасность переопределения помещенного ранее в приемном регистре байта данных вновь “собранным” байтом. Поэтому при приеме информации компьютер должен очень быстро освободить приемный регистр адаптера. В полном наборе сигналов **RS-232C** есть линии, которые могут аппаратно “приостановить” модем.

Нуль-модемный интерфейс характерен для прямой связи компьютеров на небольшом расстоянии (длина кабеля до 15 метров). Для нормальной работы

двух непосредственно соединенных компьютеров нуль-модемный кабель должен выполнять следующие соединения:

1. RI-1 + DSR-1 — DTR-2;
2. DTR-1 — RI-2 + DSR-2;
3. CD-1 — CTS-2 + RTS-2;
4. CTS-1 + RTS-1 — CD-2;
5. RD-1 — TD-2;
6. TD-1 — RD-2;
7. SG-1 — SG-2;

Знак «+» обозначает соединение соответствующих контактов на одной стороне кабеля.

4.2.1. Настройка COM-порта средствами библиотеки для C#

Прикладная библиотека Serial COM для C# предлагает широкие возможности по настройке COM-порта. Подробное описание библиотеки

<https://docs.microsoft.com/ru-ru/dotnet/api/system.io.ports.serialport?view=netframework-4.8>

4.2.2. Описание класса SerialPort

Класс SerialPort дает возможность управления последовательными портами компьютера. Он определяет минимальную функциональность для работы с ними.

4.2.3. Конструкторы класса

<u>SerialPortQ</u>	Инициализация нового экземпляра класса <u>SerialPort</u> .
---------------------------	--

<u>SerialPort(IContainer)</u>	Инициализирует новый экземпляр класса <u>SerialPort</u> , используя объект <u>IContainer</u> .
<u>SerialPort(String)</u>	Инициализирует новый экземпляр класса <u>SerialPort</u> , используя указанное имя порта.
<u>SerialPort(String, Int32)</u>	Инициализирует новый экземпляр класса <u>SerialPort</u> , используя указанное имя порта и скорость передачи в бодах.
<u>SerialPort(String, Int32, Parity)</u>	Инициализирует новый экземпляр класса <u>SerialPort</u> , используя указанное имя порта, скорость передачи в бодах и бит четности.
<u>SerialPort(String, Int32, Parity, Int32)</u>	Инициализирует новый экземпляр класса <u>SerialPort</u> , используя указанное имя порта, скорость передачи в бодах, бит четности и биты данных.
<u>SerialPort(String, Int32, Parity, Int32, StopBits)</u>	Инициализирует новый экземпляр класса <u>SerialPort</u> , используя указанное имя порта, скорость передачи в бодах, бит четности, биты данных и стоп-бит.

Таблица 2.

4.2.4. Методы

<u>Close()</u>	Закрывает соединение порта, присваивает свойству <u>IsOpen</u> значение false и уничтожает внутренний объект <u>Stream</u> .
<u>CreateObjRef(Type)</u>	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object. (Унаследовано от <u>MarshalByRefObject</u>)
<u>DiscardInBuffer()</u>	Удаляет данные из буфера приема последовательного драйвера.
<u>DiscardOutBuffer()</u>	Удаляет данные из буфера передачи последовательного драйвера.
<u>Dispose()</u>	Освобождает все ресурсы, занятые <u>Component</u> . (Унаследовано от <u>Component</u>)
<u>Dispose(Boolean)</u>	Освобождает неуправляемые ресурсы, используемые <u>SerialPort</u> , и дополнительно освобождает управляемые ресурсы.
<u>Equals(Object)</u>	Определяет, равен ли указанный объект текущему объекту. (Унаследовано от <u>Object</u>)
<u>GetHashCode()</u>	Служит в качестве хэш-функции по умолчанию. (Унаследовано от <u>Object</u>)

<u>GetLifetimeService()</u>	Retrieves the current lifetime service object that controls the lifetime policy for this instance. (Унаследовано от <u>MarshalByRefObject</u>)
<u>GetPortNames()</u>	Получает массив имен последовательных портов для текущего компьютера.
<u>GetService(Type)</u>	Возвращает объект, представляющий службу, предоставляемую классом <u>Component</u> или классом <u>Container</u> . (Унаследовано от <u>Component</u>)
<u>GetType()</u>	Возвращает объект <u>Type</u> для текущего экземпляра. (Унаследовано от <u>Object</u>)
<u>InitializeLifetimeService()</u>	Obtains a lifetime service object to control the lifetime policy for this instance. (Унаследовано от <u>MarshalByRefObject</u>)
<u>MemberwiseClone()</u>	Создает неполную копию текущего объекта <u>Object</u> . (Унаследовано от <u>Object</u>)
<u>MemberwiseClone(Boolean)</u>	Creates a shallow copy of the current <u>MarshalByRefObject</u> object. (Унаследовано от <u>MarshalByRefObject</u>)
<u>Open()</u>	Открывает новое соединение последовательного порта.
<u>Read(Byte[], Int32, Int32)</u>	Считывает из входного буфера <u>SerialPort</u> определенное число байтов и записывает их в байтовый массив, начиная с указанной позиции.
<u>Read(Char[], Int32, Int32)</u>	Считывает из входного буфера <u>SerialPort</u> определенное число символов и записывает их в символьный массив, начиная с указанной позиции.
<u>ReadByte()</u>	Считывает из входного буфера <u>SerialPort</u> один байт в синхронном режиме.
<u>ReadChar()</u>	Считывает из входного буфера <u>SerialPort</u> один символ в синхронном режиме.
<u>ReadExisting()</u>	Считывает все непосредственно доступные байты в соответствии с кодировкой из потока и из входного буфера объекта <u>SerialPort</u> .
<u>ReadLine()</u>	Считывает данные из входного буфера до значения <u>NewLine</u> .
<u>ReadTo(String)</u>	Считывает из входного буфера строку до указанного значения value.

<u>ToString()</u>	Возвращает объект <u>String</u> , содержащий имя <u>Component</u> , если оно есть. Этот метод не следует переопределять. (Унаследовано от <u>Component</u>)
<u>Write(Byte[], Int32, Int32)</u>	Записывает указанное число байтов в последовательный порт, используя данные из буфера.
<u>Write(Char[], Int32, Int32)</u>	Записывает указанное число символов в последовательный порт, используя данные из буфера.
<u>Write(String)</u>	Записывает указанную строку в последовательный порт.
<u>WriteLine(String)</u>	Записывает указанную строку и значение <u>NewLine</u> в выходной буфер.

Таблица 3.

4.3. Функции физического уровня.

4.4.1. Открытие порта

В ОС Windows доступ к COM-портам предоставляется посредством файловых интерфейсов. В данной работе доступ предоставляется на основе класса SerialPort. Этот класс используется для управления файловым ресурсом последовательного порта. Данный класс предоставляет возможности управления вводом-выводом в синхронном режиме или на основе событий, доступа к состоянию линии и состоянию разрыва, а также доступа к свойствам последовательного драйвера.

Соответственно, для открытия порта в языке C# используется функция класса SerialPort Open(). В качестве параметров этому методу передаются параметры, настраивающие передачу данных на физическом уровне, упомянутые выше, а также имя доступного COM-порта для подключения. При успешном выполнении функции свойство IsOpen экземпляра класса SerialPort становится равным true, в противном случае генерируется одно из следующих исключений:

- UnauthorizedAccessException
- ArgumentOutOfRangeException

- ArgumentException
- IOException
- InvalidOperationException

Тайм-ауты необходимы для правильной работы функций чтения из порта и записи в порт. В программе определяются следующие значения:

- `ReadTimeout` определяет срок ожидания в миллисекундах для завершения операции чтения из порта.
- `WriteTimeout` определяет срок ожидания в миллисекундах для завершения операции записи в порт.

По истечении данных таймаутов в случае неуспешной записи/чтения генерируется исключение `TimeoutException` - задает сообщение с описанием источника исключения. Если метод вызывает это исключение, обычно выдается сообщение "Операция не завершена в связи с истечением тайм-аута".

Также при выполнении метода экземпляру `SerialPort` предоставляется базовый объект `Stream` для работы, - в целом, класс `Stream` является абстрактным базовым классом всех потоков. Поток — это абстракция последовательности байтов, например файл, устройство ввода-вывода, канал взаимодействия процессов или сокет TCP/IP. В данном случае `Stream` представляет собой абстракцию для канала, осуществляющего запись данных в порт.

4.4.2. Заккрытие порта

Заккрытие порта осуществляется путем вызова метода `Close()` на экземпляре класса `SerialPort`. Данный метод закрывает соединение порта, присваивает свойству `IsOpen` значение `false` и уничтожает внутренний объект `Stream`.

4.4.3. Передача данных

В данной работе передача данных между двумя компьютерами осуществляется вызова метода Write(Byte[], Int32, Int32) класса `SerialPort`.

Для передачи данных в классе `SerialPort` существует набор методов `Write`, отличающихся параметрами:

- Записывает указанную строку в последовательный порт.
- Записывает указанное число байтов в последовательный порт, используя данные из буфера.
- Записывает указанное число символов в последовательный порт, используя данные из буфера.

4.4.4. Прием данных

Прием данных реализуется как обработчик события `DataReceived` класса `SerialPort`, - данное событие активизируется при наличии данных во входном буфере порта. В рамках физического уровня обработчика данные считываются с помощью набора методов для считывания данных `Read`, отличающихся принимаемыми параметрами:

- Считывает из входного буфера `SerialPort` определенное число байтов и записывает их в байтовый массив, начиная с указанной позиции.
- `ReadByte` - Считывает из входного буфера `SerialPort` один байт в синхронном режиме.
- `ReadChar` - Считывает из входного буфера `SerialPort` один символ в синхронном режиме.
- `ReadExisting` - Считывает все непосредственно доступные байты в соответствии с кодировкой из потока и из входного буфера объекта `SerialPort`.

Для анализа числа поступивших байтов используется свойство `BytesToRead`, получающее число байтов данных, находящихся в буфере приема.

4.4.5. Поддержание соединения

Поддержание физического соединения осуществляется отправкой сигналов *LINKACTIVE* каждые 10 секунд и получением ответа *ACK_LINKACTIVE* о том, что соединение поддерживается.

5. Канальный уровень

5.1. Функции канального уровня

На канальном уровне должны выполняться следующие функции:

1. Запрос логического соединения;
2. Управление передачей кадров;
3. Обеспечение необходимой последовательности блоков данных, передаваемых через межуровневый интерфейс;
4. Контроль и обработка ошибок;
5. Проверка целостности логического соединения;
6. Посылка подтверждения.
7. Запрос на разъединение логического соединения.

5.2. Протокол связи

Важнейшими понятиями протокола связи являются управление ошибками и управление потоком. В основном протокол содержит набор соглашений или правил, которого должны придерживаться обе стороны связи для обеспечения получения и корректной интерпретации информации, передаваемой между двумя сторонами. Таким образом, помимо управления ошибками и потоком протокол связи регулирует также следующие вопросы

- формат передаваемых данных (число битов на каждый элемент и тип используемой схемы кодирования);
- тип и порядок сообщений, подлежащих обмену для обеспечения (свободной от ошибок и дубликатов) передачи информации между двумя взаимодействующими сторонами.

Например, перед началом передачи данных требуется установить соединение

между двумя сторонами, тем самым проверяется доступность приемного устройства и его готовность воспринимать данные. Для этого передающее устройство посылает специальную команду: запрос на соединение и ожидает ее приема с другого СОМ-порта.

5.3. Защита передаваемой информации

При передаче данных по линиям могут возникать ошибки, вызванные электрическими помехами, связанными, например, с шумами, порожденными коммутирующими элементами сети. Эти помехи могут вызвать множество ошибок в цепочке последовательных битов.

Метод четности/нечетности или полученная в развитие этого метода контрольная сумма блока не обеспечивают надежного обнаружения пачки ошибок. Для этих случаев чаще всего применяется альтернативный метод, основанный на полиномиальных кодах. Полиномиальные коды используются в схемах покадровой (или поблочной) передачи. Это означает, что для каждого передаваемого кадра формируется один-единственный набор контрольных разрядов, значения которых зависят от фактического содержания кадра. Приемник выполняет те же вычисления с полным содержимым кадра; если при передаче ошибки не возникли, то в результате вычислений должен быть получен заранее известный ответ. Если этот ответ не совпадает с ожидаемым, то это указывает на наличие ошибок.

В данной программе передаваемая информация защищена кодом Хемминга [7,4].

5.4. Кодирование кодом Хемминга

Алгоритм кодирования состоит в том, что данные разбиваются на блоки фиксированной длины и ввода в них контрольных бит, дополняющих до четности несколько пересекающихся групп, охватывающих все биты блока.

Ричард Хемминг рассчитал минимальное количество проверочных бит, позволяющих однозначно исправлять однократные ошибки.

Если длина информационного блока, который требуется закодировать - m бит. Количество контрольных бит, используемых для его кодирования, - k , то закодированный блок будет иметь длину: $n = m + k$ бит. Для каждого блока такой длины возможны n различных комбинаций, содержащих ошибку.

Таким образом, для каждого передаваемого информационного блока может существовать n -блоков, содержащих однократную ошибку, и один блок - без ошибок. Следовательно, максимальное количество различных закодированных блоков, содержащих не больше одной ошибки, будет: $2^m(n + 1)$, где $n = m + k$.

Если для информационных данных длиной m подобрать такое количество контрольных бит k , что максимально возможное количество различных последовательностей длиной $m+k$ будет больше или равно максимальному количеству различных закодированных информационных блоков, содержащих не больше одной ошибки, то точно можно утверждать, что существует такой метод кодирования информационных данных с помощью k контрольных бит, который гарантирует исправление однократной ошибки.

Следовательно, минимальное количество контрольных бит, необходимых для исправления однократной ошибки, определяется из равенства:

$$2^m \times (n + 1) = 2^n$$

Учитывая, что $n = m + k$, получаем:

$$k = 2^k - m - 1$$

Так как количество бит должно быть целым числом, то k , вычисленное с помощью этого уравнения, необходимо округлить до ближайшего большего целого числа.

Например, для информационных данных длиной 7 необходимо 4 контрольных бита, чтобы обеспечить исправление однократных ошибок, а для данных длиной 128 бит необходимо 8 контрольных бит.

Мало определить минимальное количество контрольных бит, необходимых для исправления ошибки. Необходимо разработать алгоритм

проверки данных с помощью этих контрольных разрядов. Ричард Хемминг предложил следующий алгоритм.

Все биты, порядковые номера которых являются степенью двойки, – это контрольные разряды. То есть если порядковый номер бита обозначить символом 'n', то для контрольных бит должно быть справедливо равенство: $n = 2^k$, где k – любое положительное целое число.

Например, для закодированной последовательности длиной 13 бит проверочными будут: 1, 2, 4 и 8 биты, так как $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$.

Каждый выбранный, таким образом, контрольный бит будет проверять определенную группу бит, т.е. в контрольный бит будет записана сумма по модулю два всех битов группы (дополнение до четного количества единиц), которую он проверяет.

Для того, чтобы определить какими контрольными битами контролируют бит, необходимо разложить его порядковый номер по степени 2. Таким образом, девятый бит будет контролироваться битами 1 и 8, так как $9 = 2^0 + 2^3 = 1 + 8$.

5.5. Форматы кадров

Кадры, передаваемые с помощью функций канального уровня, имеют различное назначение. Выделены служебные и информационные кадры.

5.1.1. Служебные супервизорные кадры.

UPLINK-кадр – кадр запроса на установление логического соединения

ACK-UPLINK-кадр – положительная квитанция на UPLINK-кадр

LINKACTIVE-кадр – кадр поддержания логического соединения

ACK-LINKACTIVE-кадр – положительная квитанция на LINKACTIVE-кадр

RET-кадр – кадр запроса повторной передачи сообщения при ошибке в сообщении (неправильность битов)

DOWNLINK-кадр – кадр разрыва логического соединения

ACK-DOWNLINK – положительная квитанция на DOWNLINK-кадр

Эти кадры используются для передачи служебной информации и реализуют следующие функции канального уровня: установление и разъединение логического канала, подтверждение приема информационного кадра без ошибок, запрос на повторную передачу принятого с ошибкой кадра.

Формат эти кадров:

StartByte	Type	StopByte
Флаг начала кадра	Тип супервизорного кадра	Флаг конца кадра

5.1.2. Супервизорные кадры передачи параметров

Супервизорные кадры передачи параметров используются для синхронизации параметров COM-портов, как принимающего, так и отправляющего. Кадр данного типа формируется, когда на одном из компьютеров изменяются параметры. Формат эти кадров:

StartByte	Type	Data	StopByte
Флаг начала кадра	Тип супервизорного кадра	Параметры COM-порта	Флаг конца кадра

5.1.3. Информационные кадры.

DAT-кадр

Поскольку кадры имеют переменную длину, каждый поступающий кадр должен буферизоваться (т.е. сохраняться в памяти), что гарантирует его целостность до начала передачи.

Информационные кадры применяются для передачи закодированных кодом Хемминга пользовательских сообщений. Формат эти кадров:

StartByte	Type	Data
Флаг начала кадра	Тип супервизорного кадра	Закодированные данные (текстовая строка)

Кадр можно разделить на несколько блоков - флаг начала кадра, тип кадра, данные.

Поле типа кадра обеспечивает правильное определение и распознавание разновидностей кадров и обработки их соответствующими процедурами.

Данные представляют собой либо закодированную строку в

информационном кадре или параметры порта в супервизорном кадре передачи параметров.

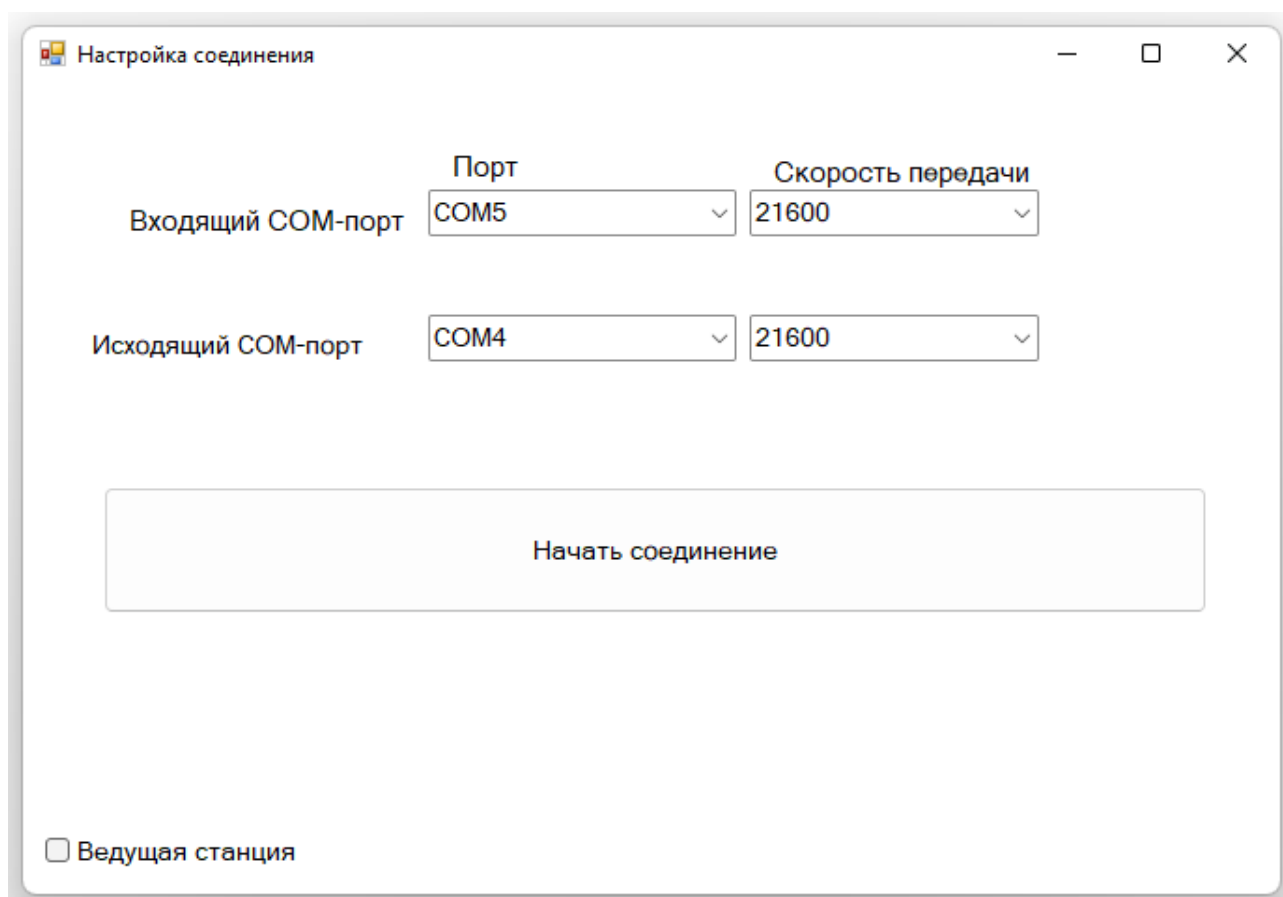
6. Прикладной уровень

Функции прикладного уровня обеспечивают интерфейс программы с пользователем.

На данном уровне обеспечивается вывод принятых и отправленных сообщений в окно диалога пользователей.

Пользовательский интерфейс выполнен в среде Visual Studio 2022. При его разработке учитывались рекомендации по простоте, удобству и функциональности интерфейса.

При запуске программы появляется форма, в которой происходит настройка COM-портов компьютера (см рис. **Error! Reference source not found.**).



Настройка соединения

	Порт	Скорость передачи
Входящий COM-порт	COM5	21600
Исходящий COM-порт	COM4	21600

Начать соединение

☐ Ведущая станция

Рис. 2 Настройка COM-портов устройства

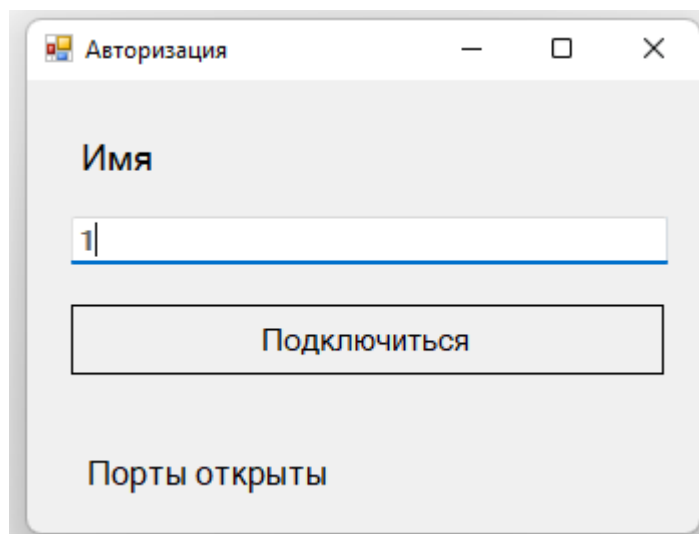


Рис. 3 Ввод имени пользователя

Главным окном программы является окно «Чат». В данной форме есть следующие возможности:

- 1 Подключение – создание соединения между устройствами
- 2 Отключение – разрыв соединения между устройствами
- 3 Отправка – позволяет отправить сообщение пользователю на другое устройство
- 4 Просмотр – в секции «Чат» пользователи могут наблюдать их совместную переписку

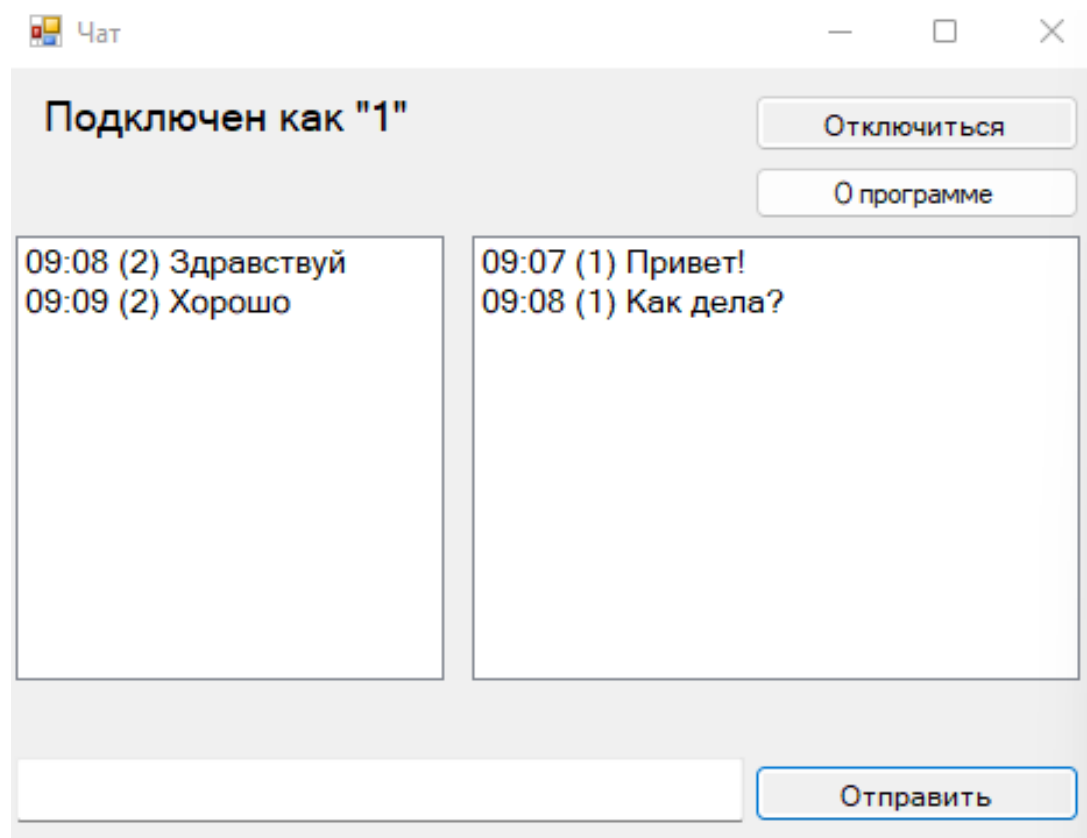


Рис. 4 Форма “Чат”