

Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Чтение шины данных I-Bus автомобиля BMW

БГУИР КП 1-40 01 01 405 ПЗ

Студент гр. 951004

Кондрацкий А.А.

Руководитель

Деменковец Д.В.

Минск 2021

СОДЕРЖАНИЕ

Введение.....	5
1 Обзор аналогов	6
1.1 NavCoder	6
2 Постановка задачи.....	7
3 Моделирование предметной области.....	8
3.1 Основные сведения о протоколе RS232	8
3.1.1 Общие сведения	8
3.1.2 Уровни сигналов	8
3.1.3 Контроль чётности.....	8
3.2 Основные сведения о транзисторно-транзисторной логике (ТТЛ)	9
3.2.1 Общие сведения	9
3.2.2 Уровни сигналов	10
3.3 Физическая коммуникация компьютера с шиной данных	10
3.4 Сеть I-Bus	12
3.4.1 Общие сведения	12
3.4.2 Физический уровень	13
3.4.3 Канальный уровень.....	13
3.4.4 Полезная нагрузка.....	14
4 Проектирование программного средства	16
4.1 Разработка архитектуры программного средства.....	16
4.2 Интерфейс программного средства.....	16
4.3 Функциональная часть программы	18
4.3.1 Получение доступных СОМ портов	18
4.3.2 Настройка параметров и подключение к порту.....	19
4.3.3 Чтение данных	20
4.3.4 Отправка данных	21
5 Тестирование и проверка работоспособности программного средства	22
5.1 Подключение к порту	22
5.1.1 Тест 1.....	22
5.1.2 Тест 2.....	23
5.2 Работа с полем полученных данных	24
5.2.1 Тест 1.....	24
5.2.1 Тест 2.....	25
5.2.1 Тест 3.....	26
5.3 Отправка данных.....	27
5.3.1 Тест 1.....	27
5.3.2 Тест 2.....	28
5.3.3 Тест 3.....	29
Заключение	30
Список использованных источников	31
Приложение А	32

ВВЕДЕНИЕ

В настоящее время в современных автомобилях электроника является неотъемлемой частью. Мультимедиа, приборная панель, блок управления двигателем – все это мини-компьютеры, которые обрабатывают информацию. Однако следует учитывать, что все эти блоки работают совместно, чтобы создать единую систему.

Блок управления кнопками на руле, позволяет увеличить громкость музыки, включить круиз-контроль, в мультимедиа в свою очередь знает скорость автомобиля, чтобы автоматически повысить громкость.

В автомобилях марки BMW, а именно кузовах e38, e39, e46, e39 для связи многих блоков используется шина I-Bus. Следует сказать, что есть и другие шины, такие как K-Bus, P-Bus, D-Bus, M-Bus, но они используются для связи блоков, с которыми владелец авто напрямую не взаимодействует. Из-за этого будет рассмотрена именно эта шина.

В шину I-Bus входят такие устройства:

- блок света (LCM);
- приборная панель (IKE);
- парктроники (PDC);
- многофункциональное рулевое колесо (MFL);
- усилитель с продвинутым музыкальным процессором (DSP);
- мульти информативный экран (MID);
- радиоблок (BM);
- система навигации (NAV);
- видео/ТВ модуль (TV);
- телефон (TEL);
- CD-чейнджер (CDW);
- радио (RAD).

1 ОБЗОР АНАЛОГОВ

1.1 NavCoder

NavCoder — это приложение Windows для перепрограммирования навигационных компьютеров BMW и других устройств I-Bus. Приложение написано на VB6 и было запущено в 2006 году. Вот как выглядит текущее окно NavCoder:

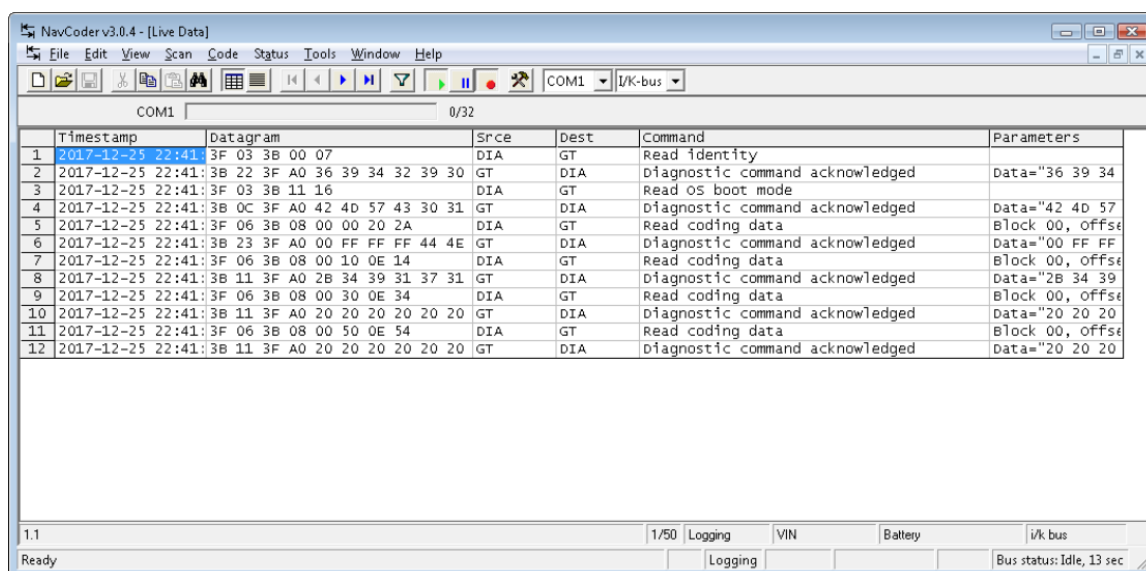


Рисунок 1.1 – Главное окно NavCoder

Приложение позволяет:

- просматривать данные шины I-Bus в режиме реального времени в виде обычного текста;
- сканирование и извлечение служебной информации, отображающей, например километраж транспортного средства;
- кодирование модуля управления светом и выключение предупреждения о перегоревших лампочках (используется при установке светодиодных фонарей), кодирование дневных ходовых огней;
- кодирование навигационного и ТВ-блока;
- проверка парктроники позволяет узнать состояние каждого датчика и определить расстояние до препятствия.

2 ПОСТАНОВКА ЗАДАЧИ

Целью данной курсовой работы является создание программы для коммуникации с шиной I-Bus.

В программном средстве планируется реализовать ряд функций:

- подключение к шине через последовательный порт компьютера;
- постоянное чтение шины;
- текстовое представление полученных данных;
- отправка данных в шину;
- возможность отображения заранее известных пакетов данных.

Для удобного пользования программой необходимо реализовать пользовательский интерфейс, с которым будет взаимодействовать пользователь для подключения и коммуникации с автомобилем.

Разработка будет вестись на языке C++. Среда разработки для этого языка – Microsoft Visual Studio 2019. Для графического интерфейса необходимо использовать WinAPI. Использование данной среды разработки и языка дает возможность использовать большое количество функций операционной системы Windows.

3 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

3.1 Основные сведения о протоколе RS232

3.1.1 Общие сведения

RS-232 - интерфейс передачи информации между двумя устройствами на расстоянии до 20 м. Информация передается по проводам с уровнями сигналов, отличающимися от стандартных 5В, для обеспечения большей устойчивости к помехам. Асинхронная передача данных осуществляется с установленной скоростью при синхронизации уровнем сигнала стартового импульса.

Интерфейс RS-232-C был разработан для простого применения, однозначно определяемого по его названию "Интерфейс между терминальным оборудованием и связным оборудованием с обменом по последовательному двоичному коду". Каждое слово в названии значимое, оно определяет интерфейс между терминалом (DTE) и модемом (DCE) по передаче последовательных данных.

3.1.2 Уровни сигналов

В RS-232 используются два уровня сигналов: логические 1 и 0. Логическую 1 иногда обозначают MARK, логический 0 - SPACE. Логической 1 соответствуют отрицательные уровни напряжения, а логическому 0 - положительные.

Уровни напряжения у передатчика:

- логический 0 – от +5В до +15В;
- логическая 1 – от -5В до -15В.

Уровни напряжения у приёмника:

- логический 0 – от +3В до +25В;
- логическая 1 – от -3В до -25В.

Напряжение от -3В до +3В считается неопределённым и не отображает никакой логический уровень.

3.1.3 Контроль чётности

При передаче по последовательному каналу контроль четности может быть использован для обнаружения ошибок при передаче данных. При использовании контроля четности посылаются сообщения, подсчитывающие число единиц в группе бит данных. В зависимости от результата устанавливается бит четности. Приемное устройство также подсчитывает число единиц и затем сверяет бит четности.

Проверка на четность — это простейший способ обнаружения ошибок.

Он может определить возникновение ошибок в одном бите, но при наличии ошибок в двух битах уже не заметит ошибок. Также такой контроль не отвечает на вопрос какой бит ошибочный. Другой механизм проверки включает в себя Старт и Стоп биты.

Сигнальная линия может находиться в двух состояниях: включена и выключена. Линия в состоянии ожидания всегда включена. Когда устройство или компьютер хотят передать данные, они переводят линию в состояние выключено — это установка Старт бита. Биты сразу после Старт бита являются битами данных.

Стоп бит позволяет устройству или компьютеру произвести синхронизацию при возникновении сбоев. Например, помеха на линии скрывает старт бит. Период между старт и стоп битами постоянен, согласно значению скорости обмена, числу бит данных и бита четности. Стоп бит всегда включен. Если приемник определяет выключенное состояние, когда должен присутствовать стоп бит, фиксируется появление ошибки.

Стоп бит не просто один бит минимального интервала времени в конце каждой передачи данных. На компьютерах обычно он эквивалентен 1 или 2 битам, и это должно учитываться программой драйвера. Хотя, 1 стоп бит наиболее общий, выбор 2 бит в худшем случае немного замедлит передачу сообщения. Так же есть возможность установки значения стоп бита равным 1.5. Это используется при передаче менее 7 битов данных. В этом случае не могут быть переданы символы ASCII, и поэтому значение 1.5 используется редко.

3.2 Основные сведения о транзисторно-транзисторной логике (ТТЛ)

3.2.1 Общие сведения

Транзисторно-транзисторная логика (ТТЛ) — это устоявшийся с 60-х годов XX-го века стандарт логических элементов, построенных на транзисторной биполярной технологии с напряжением питания +5 В. Типичный базовый элемент этой технологии — это логический элемент 2И-НЕ.

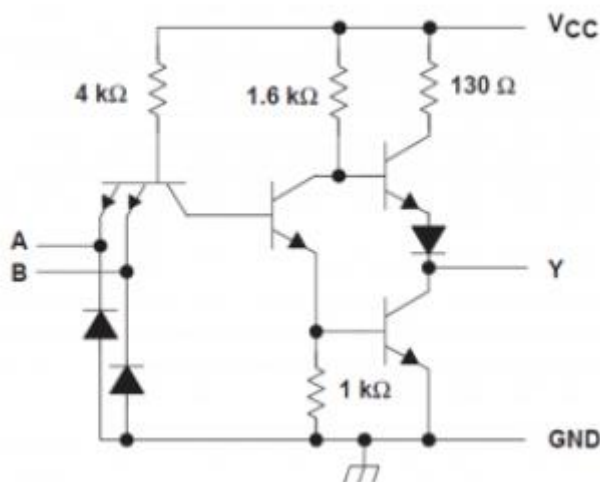


Рисунок 3.1 – Принципиальная схема

В последующие годы технология логических элементов совершенствовалась, оставаясь совместимой с прежней. На смену биполярной технологии пришли МОП (CMOS) и другие комбинированные кремниевые технологии. С целью повышения быстродействия выпускались семейства CMOS, LVTTL логических элементов с уменьшенным напряжением питания: 3,3 В, 2,5 В, и т.д. При этом разработчики элементов всеми возможными техническими способами старались сохранить совместимость по логическим уровням напряжений с классическим напряжением питания +5 В.

3.2.2 Уровни сигналов

Уровни напряжения у передатчика:

- логический 0 – не выше +0.8В;
- логическая 1 – не ниже +2В.

Уровни напряжения у приёмника:

- логический 0 – от -0.25В до +0.8В;
- логическая 1 – от +2В до +5.25В.

3.3 Физическая коммуникация компьютера с шиной данных

Для коммуникации была применена схема с использованием двух транзисторов. Один транзистор используется для чтения, второй для записи. Принципиальная схема представлена ниже.

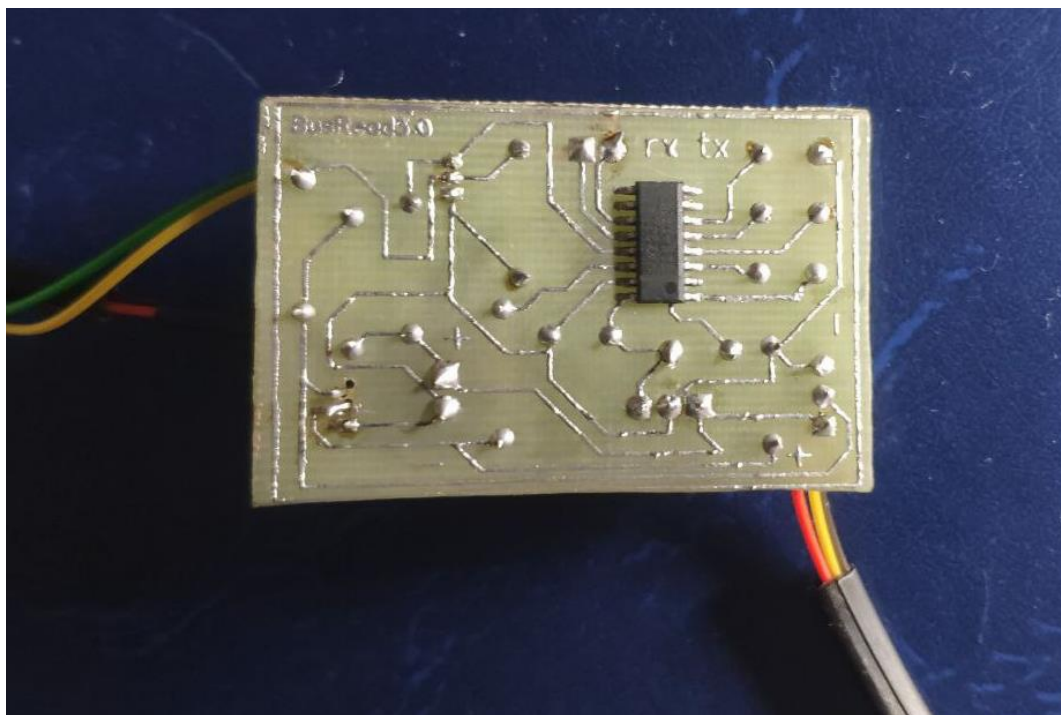


Рисунок 3.3 – Плата для коммуникации с шиной данных

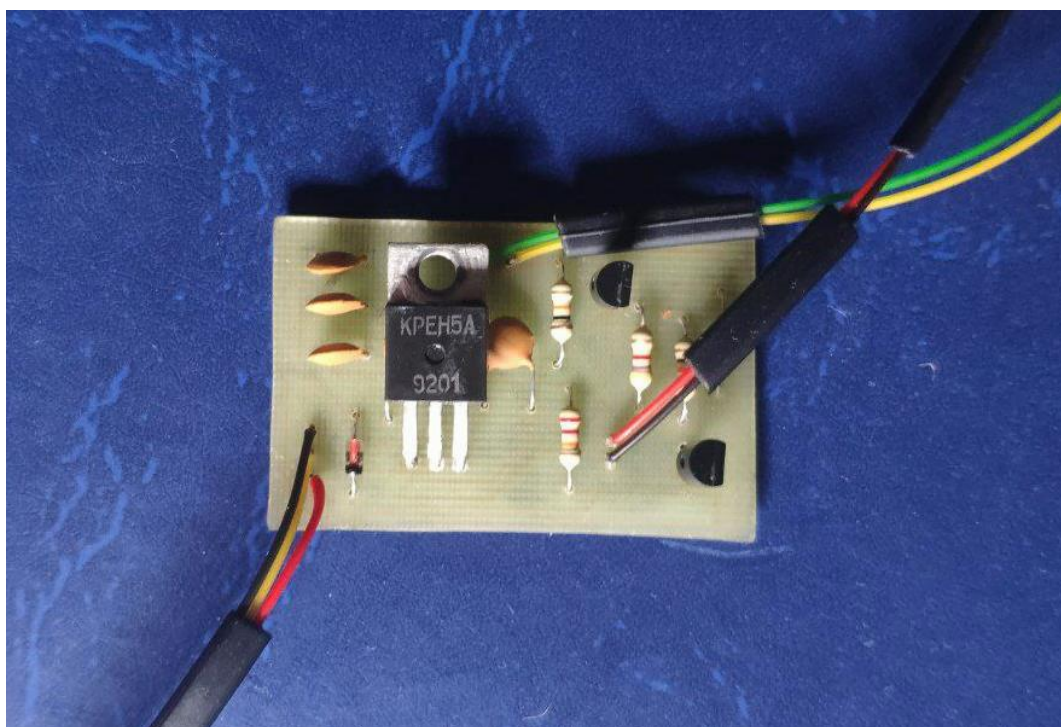


Рисунок 3.4 – Плата для коммуникации с шиной данных

3.4 Сеть I-Bus

3.4.1 Общие сведения

Информационно-развлекательная система выполняется на устройствах,

в основе которых контроллеры (блоки управления) с заложенными программами. Каждый такой блок управления несёт свою функциональную нагрузку, будь то поддержание температуры салона, регулировка положения сидений, воспроизведение музыки и видео, навигация и прочее. Весь этот набор блоков управления должен взаимодействовать друг с другом, управляться с места водителя и пассажиров, передавать диагностические данные. Для этой цели и была разработана сеть I-Bus. Впоследствии появилась технически идентичная сеть K-Bus и их объединение I/K-Bus.

3.4.2 Физический уровень

Архитектура сети I-Bus выполнена по схеме «общая шина», т. е. импульсы данных от узлов (блоков управления) передаются по обычному медному проводу соединённых в одной точке. Поэтому узлы должны делить общую среду передачи и передавать данные по очереди.

В «молчаливом» состоянии уровень потенциала на шине относительно корпуса составляет от 7 В до напряжения питания автомобиля. При подаче доминантного бита в шину потенциал снижается до 2 В и ниже. Битовая скорость взаимодействия узлов постоянная и составляет 9600 бит/с. Формат символов в I-Bus соответствует одной из вариаций доступных в UART.

Символ состоит из 11 бит: стартовый бит, 8 бит данных, бит чётности, стоповый бит. Эти особенности позволяют физически подключаться к шине через интерфейсы UART или RS-232. Только необходимо позаботиться о преобразовании уровней сигнала с помощью простенькой схемы или готового преобразователя.

3.4.3 Канальный уровень

Данные передаются кадрами, в которых содержится адрес отправителя, адрес получателя, полезная нагрузка (данные) и контрольная сумма. Кадр не имеет фиксированного размера и лежит в пределах 5 — 37 символов.



Рисунок 3.5 – Формат кадра

Где:

- TX ID — адрес отправителя, 1 символ;
- LEN — размер кадра с вычетом двух первых символов, 1 символ;
- RX ID — адрес получателя, 1 символ;

- DATA — полезная нагрузка, 1 — 33 символа;
- CK SUM — контрольная сумма, 1 символ.

3.4.4 Полезная нагрузка

В основе своей полезная нагрузка состоит из двух частей. MSG ID — идентификатор сообщения, занимает один символ. MSG DATA — информация, дополняющая сообщение, может отсутствовать вовсе или занимать до 32 символов.



Рисунок 3.6 – Формат кадра

Рассмотрим некоторые ключевые идентификаторы.

Сообщение с идентификатором MSG ID = 01 — запрос состояния устройства. Прежде чем взаимодействовать с каким-либо устройством, необходимо убедиться в его наличии и исправности. Эта команда отправляется устройству, в состоянии которого необходимо убедиться. При этом поле MSG DATA не заполняется. Чтобы информация о состоянии устройств была актуальна все время, команда повторяется периодически.

Рассмотрим этот вид сообщения на примере кадра 68 03 18 01 72 (здесь и далее содержимое кадра обозначаться будет цифрами в шестнадцатеричном исчислении). Кадр отправляется от радиоприёмника (идентификатор устройства 68) к CD чейнджеру (18) с запросом о состоянии (идентификатор сообщения MSG ID = 01). CD чейнджер, если он есть и исправен, отвечает сообщением, подтверждающим статус готовности (MSG ID = 02). Полный фрагмент ответного кадра 18 04 FF 02 00 E1. Ответ вещается всем в локальную сеть, так как адрес получателя FF. Здесь помимо идентификатора сообщения передаются дополнительные данные — MSG DATA = 00. Если значение данных равно 01, то это означает что устройство только включилось и это его первое сообщение о готовности. Такой вариант диалога наблюдается между многими блоками управления.

Управление воспроизведением музыкальных треков, радиостанций или изменение уровня громкости возможно как с рулевого колеса, так и с центральной консоли. Эти органы управления передают информацию на радиоприёмник по той же I-Bus. Сообщения регулировки уровня громкости идентифицируется номером 32, а в данных содержится управляющая информация.

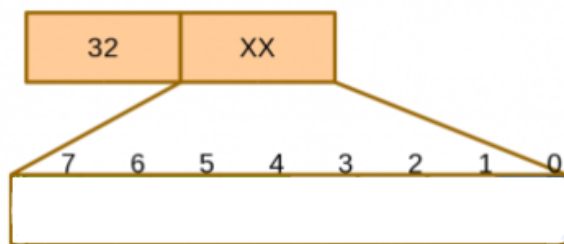


Рисунок 3.7 – Формат данных для регулировки громкости

Для кнопочного управления характерны три вида сообщения: кнопка нажата, кнопка удерживается длительное время, и кнопка отпущена. В данных сообщения, кроме состояния кнопки, передаётся её идентификатор. Например, сообщение с MSG ID = 3B означает, что передаётся информация об изменении состояния кнопок на рулевом колесе, отвечающих за управление радиоприёмником и телефоном. MSG DATA состоит из одного символа и содержит информацию о кнопке, подвергшейся воздействию.

4 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

4.1 Разработка архитектуры программного средства

Исходя из поставленной задачи, был четко определен функционал программы, а также разработан схематический алгоритм работы программы.

При запуске программы, открывается окно, на котором расположено несколько полей.

Крайнее левое поле используется для настройки и подключения к COM порту.

После нажатия кнопки «Connect to...» произведется подключение к девайсу и в центральном поле начнёт отображаться приходящая информация. Для удобства работы с ней выводится так же и словесное определение, которое становится ясным из структуры приходящего пакета.

Правое поле позволяет вывести заранее известные пакеты, чтобы иметь возможность быстро сформировать ответ.

4.2 Интерфейс программного средства

С помощью функций WinAPI был разработан пользовательский интерфейс, позволяющий комфортно использовать программное средство. Главное окно программы получило вид, представленный на рисунке 4.1.



Рисунок 4.1 – Главное окно программы

В левой части окна находятся кнопки «Scan...» и «Connect to...». Нажатием на первую кнопку пользователь выполняет сканирование доступных последовательных портов в компьютере. После сканирования доступные порты

становятся активными. Пользователю необходимо выбрать соответствующий порт.

После этого пользователь должен произвести подключение нажатием второй кнопки.

Сразу после подключения в центральном поле появится обновляющаяся информация о приходящих данных. На данном этапе окно получилось вид как на рисунке 4.2.



Рисунок 4.2 – Считывание данных

Под полем настройки и подключения к порту находятся кнопки «Pause/Resume» и «Clear». Первая кнопка позволяет приостановить считывание данных и произвести обработку ранее полученных. Вторая кнопка полностью очищает поле с полученными данными.

В правой части находится поле, в которое пользователь может вывести информацию о заранее известных кодах из шины данных. Необходимо нажать кнопку «Load commands», после чего выбрать соответствующий текстовый файл.

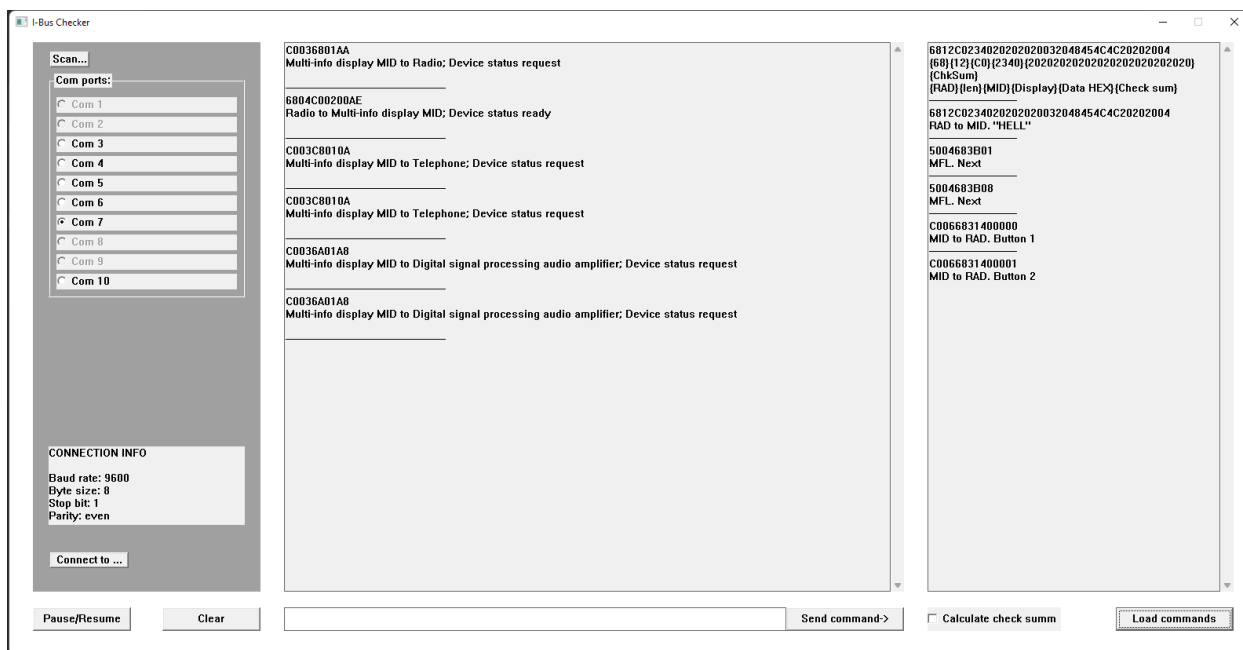


Рисунок 4.3 – Предустановленные команды

Для отправки данных используется нижнее поле для ввода. Необходимо ввести желаемый пакет данных, затем выбрать необходимость расчёта check суммы. Если в фрагменте данных check сумма отсутствует, то нужно поставить галочку для её автоматического расчёта. Затем необходимо нажать кнопку «Send command».

4.3 Функциональная часть программы

Работу с шиной данных можно разделить на следующие шаги:

- получение доступных COM портов;
- настройка параметров и подключение к порту;
- постоянное чтение данных во втором потоке;
- отправка данных в порт.

Рассмотрим каждый из пунктов подробнее.

4.3.1 Получение доступных COM портов

Для того, чтобы узнать какие порты в компьютере есть был использовал реестр. В нём необходимо открыть следующий раздел «HARDWARE\DEVICEMAP\SERIALCOMM\». Затем считать информацию об этом разделе и записать это всё в необходимые данные.

4.3.2 Настройка параметров и подключение к порту

Для коммуникации с шиной необходимо установить следующие параметры для переменной типа DCB:

- битовая скорость взаимодействия 9600бит/с;
- 8 бит данных;
- 1 бит чётности;
- 1 стоповый бит;
- отправка нулевых символов.

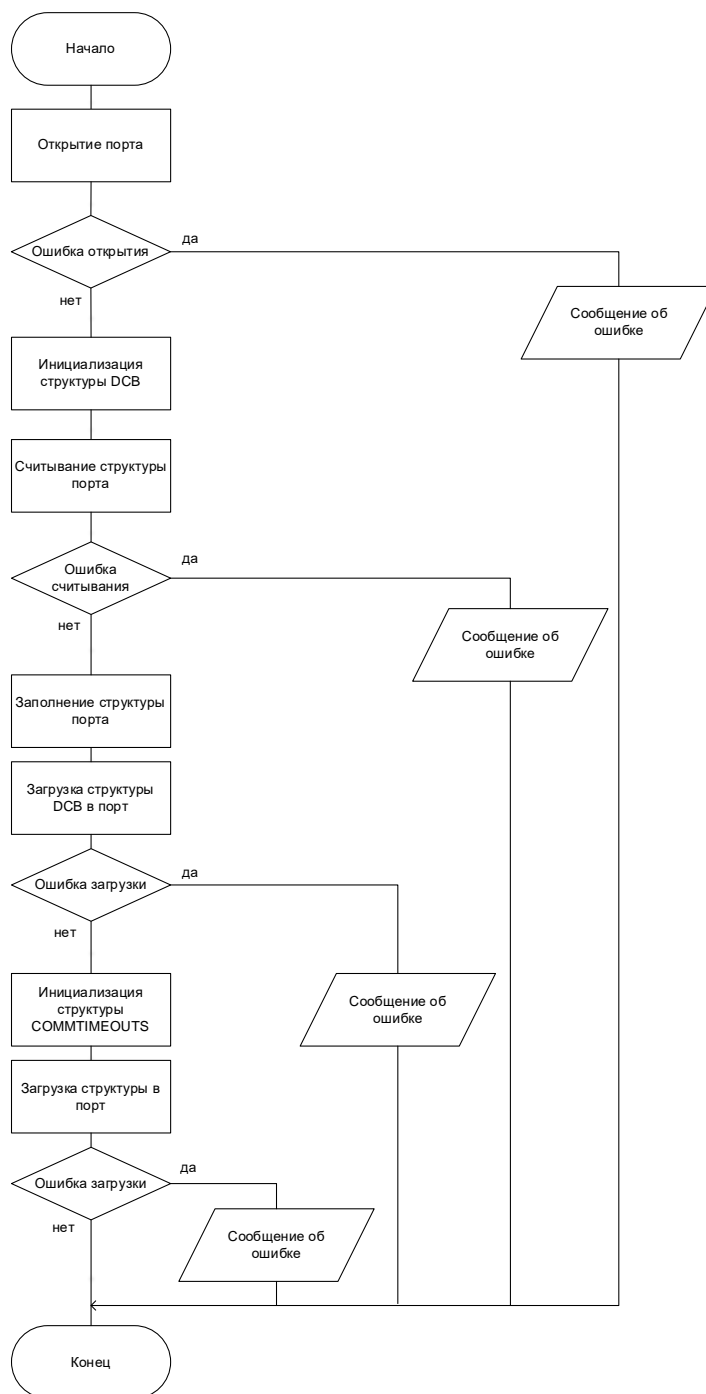


Рисунок 4.4 – Схема алгоритма настройки и подключения к порту

4.3.3 Чтение данных

Для чтения необходимо связать порт с объектом синхронизации, а затем с помощью функции WinAPI ReadFile() произвести чтение данных. Сразу после получения данных происходит конвертировании данных в строковое представление и вывод их на окно.

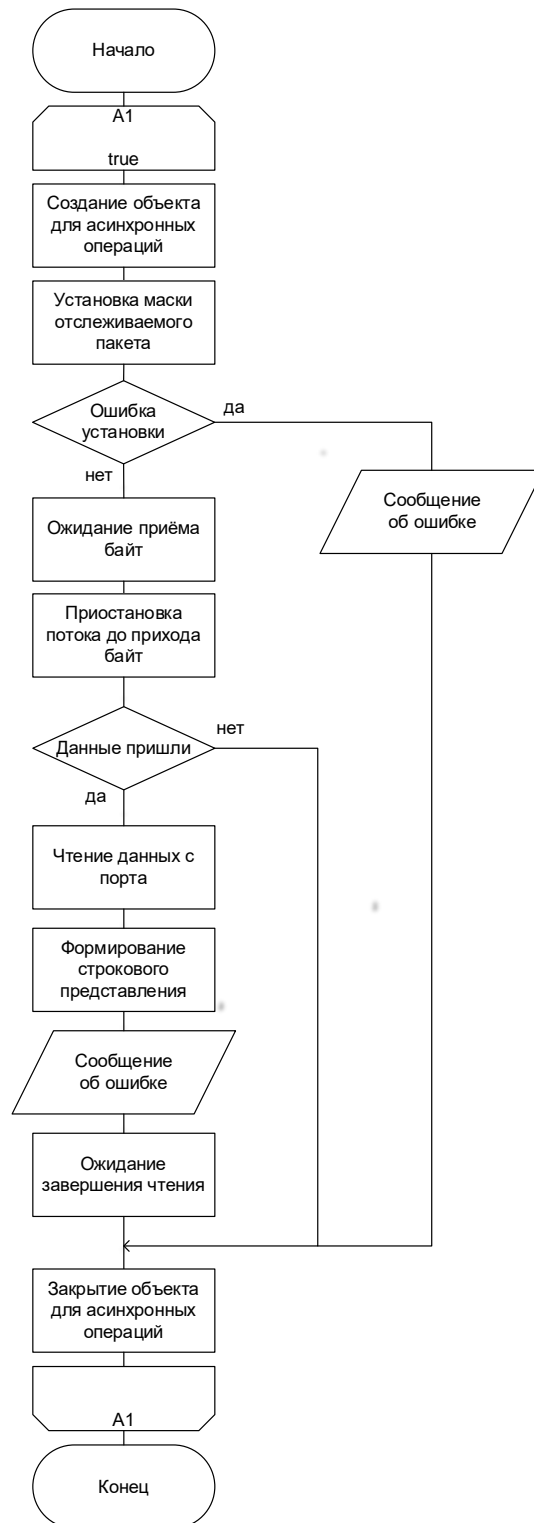


Рисунок 4.5 – Схема алгоритма чтения данных

4.3.4 Отправка данных

Для отправки данных необходимо создать событие для синхронизации и с помощью функции WriteFile() отправить данные в порт.

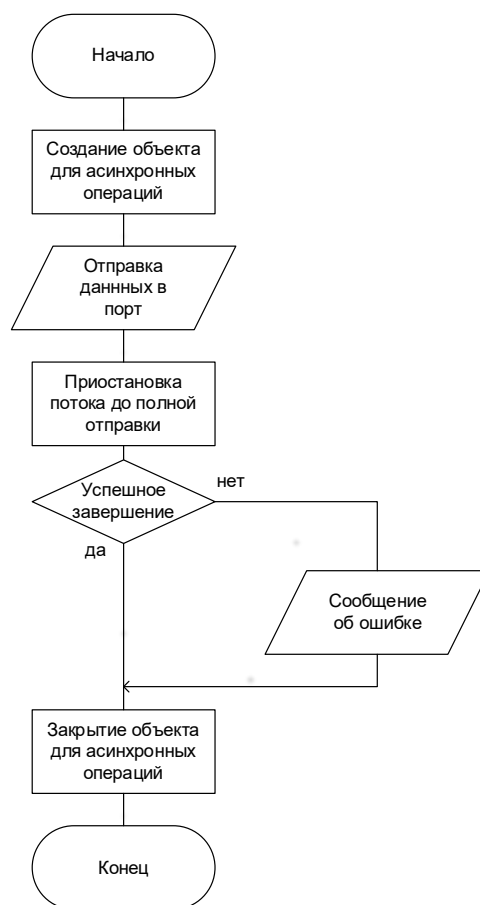


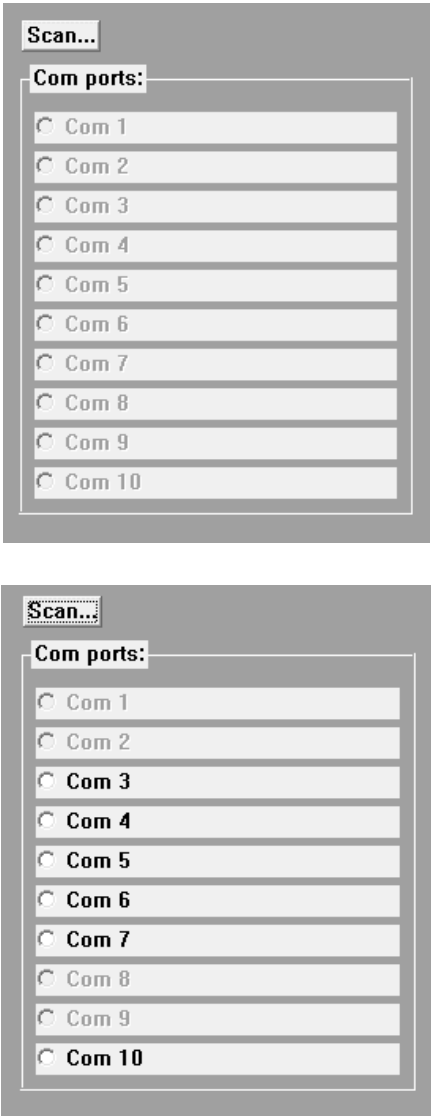
Рисунок 4.6 – Схема алгоритма отправки данных

5 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

5.1 Подключение к порту


5.1.1 Тест 1

Таблица 1 – Тест 1

Тестовая ситуация:	Определение активных последовательных портов
Исходный набор данных:	Нажатие на кнопку «Scan...»
Ожидаемый результат:	Отображение активных портов
Фактический результат:	 <p>The image shows two screenshots of a software interface. Each screenshot has a 'Scan...' button at the top. Below it is a section titled 'Com ports:' containing a list of ports from Com 1 to Com 10. Each port has a radio button next to it. In the first screenshot, all radio buttons are empty. In the second screenshot, the radio buttons for 'Com 3' and 'Com 4' are filled, indicating they are active.</p>

5.1.2 Тест 2

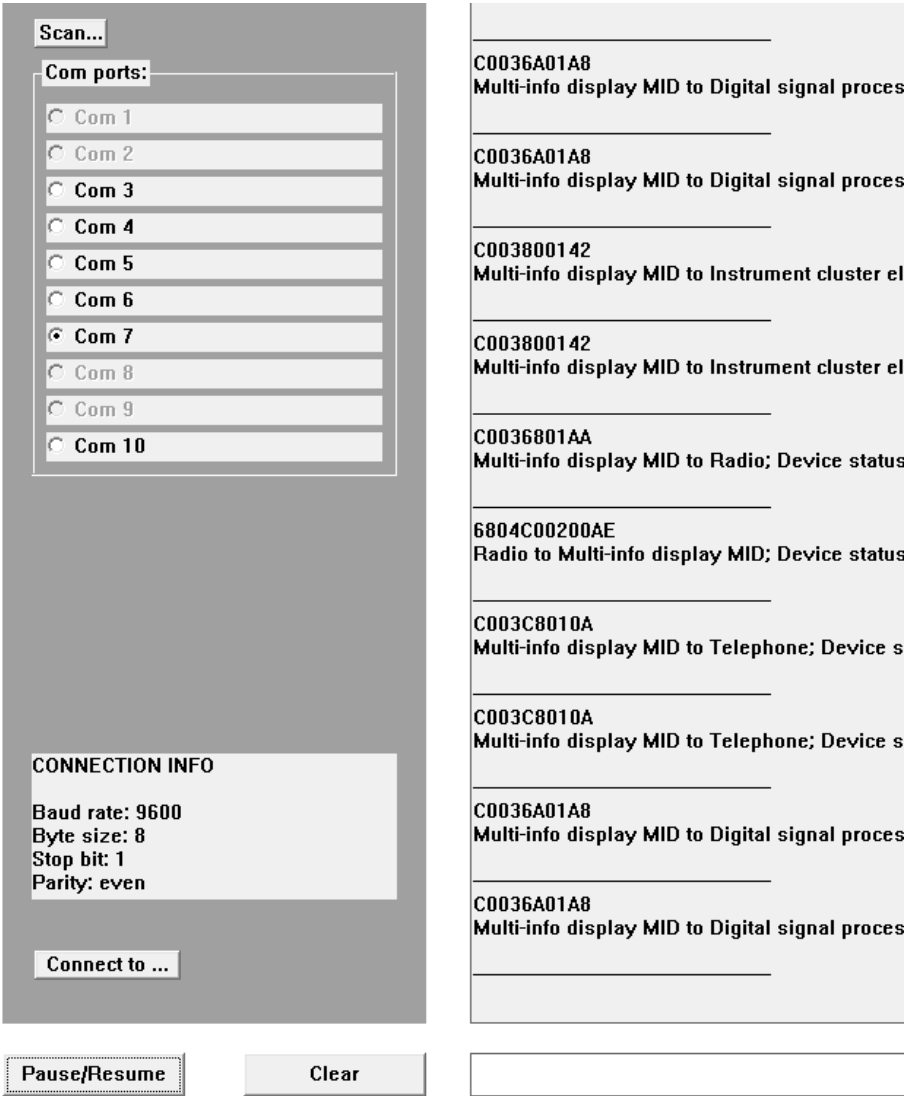
Таблица 2 – Тест 2

Тестовая ситуация:	Подключение к порту
Исходный набор данных:	Нажатие на кнопку «Connect to...»
Ожидаемый результат:	Вывод информации в центральное поле
Фактический результат:	

5.2 Работа с полем полученных данных

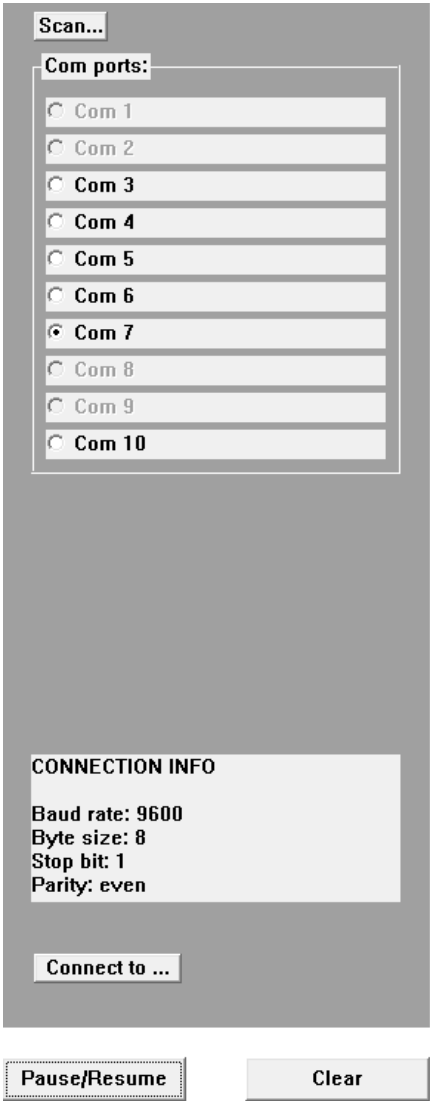
5.2.1 Тест 1

Таблица 3 – Тест 1

Тестовая ситуация:	Остановка получения данных
Исходный набор данных:	Нажатие на кнопку «Pause/Resume»
Ожидаемый результат:	Остановка получения данных
Фактический результат:	 <p>The screenshot displays a software interface for data acquisition. On the left, a 'Scan...' button is at the top. Below it, a 'Com ports:' section contains a list of ports from 'Com 1' to 'Com 10'. 'Com 7' is selected, indicated by a radio button. Below the list is a 'CONNECTION INFO' box showing the following settings: Baud rate: 9600, Byte size: 8, Stop bit: 1, and Parity: even. At the bottom of the interface are two buttons: 'Pause/Resume' and 'Clear'. To the right of the interface, a list of received data is shown, with each entry consisting of a hexadecimal ID and a description:</p> <ul style="list-style-type: none"> C0036A01A8 Multi-info display MID to Digital signal proces C0036A01A8 Multi-info display MID to Digital signal proces C003800142 Multi-info display MID to Instrument cluster el C003800142 Multi-info display MID to Instrument cluster el C0036801AA Multi-info display MID to Radio; Device status 6804C00200AE Radio to Multi-info display MID; Device status C003C8010A Multi-info display MID to Telephone; Device s C003C8010A Multi-info display MID to Telephone; Device s C0036A01A8 Multi-info display MID to Digital signal proces C0036A01A8 Multi-info display MID to Digital signal proces


5.2.1 Тест 2

Таблица 4 – Тест 2

Тестовая ситуация:	Возобновление получения данных
Исходный набор данных:	Нажатие на кнопку «Pause/Resume»
Ожидаемый результат:	Возобновление получения данных
Фактический результат:	 <p>The screenshot shows a software interface for a diagnostic tool. On the left, a 'Scan...' window is open, displaying a list of COM ports from Com 1 to Com 10. 'Com 7' is selected with a radio button. Below the list is a 'CONNECTION INFO' section showing the following settings: Baud rate: 9600, Byte size: 8, Stop bit: 1, and Parity: even. At the bottom of the window is a 'Connect to ...' button. To the right of the window, a list of detected devices is shown, including C0036801AA (Multi-info display MID to Radio; Device status), 6804C00200AE (Radio to Multi-info display MID; Device status), C003C8010A (Multi-info display MID to Telephone; Device status), C0036A01A8 (Multi-info display MID to Digital signal process), and C003800142 (Multi-info display MID to Instrument cluster ele). At the bottom of the interface are 'Pause/Resume' and 'Clear' buttons.</p>

5.2.1 Тест 3




Таблица 5 – Тест 3

Тестовая ситуация:	Очистка поля
Исходный набор данных:	Нажатие на кнопку «Clear»
Ожидаемый результат:	Полная очистка поля
Фактический результат:	 <p>The screenshot shows a software interface with a dark grey background. At the top left is a 'Scan...' button. Below it is a 'Com ports:' section containing a list of COM ports from Com 1 to Com 10, each with a radio button. 'Com 7' is selected. Below the list is a 'CONNECTION INFO' section with the following settings: Baud rate: 9600, Byte size: 8, Stop bit: 1, Parity: even. At the bottom of this section is a 'Connect to ...' button. To the right of the settings is a large, empty light grey rectangular area. At the bottom of the interface are two buttons: 'Pause/Resume' and 'Clear'. Below the 'Clear' button is a horizontal line.</p>

5.3 Отправка данных


5.3.1 Тест 1

Таблица 6 – Тест 1

Тестовая ситуация:	Отправка данных. Имитация нажатия кнопки на дисплеи
Исходный набор данных:	<p>Корректный набор данных без check суммы. Установка галочки для автоматического подсчёта check суммы</p> 
Ожидаемый результат:	Появление пакета данных в центральном поле. Изменение включенного канала
Фактический результат:	 


5.3.2 Тест 2

Таблица 7 – Тест 2

Тестовая ситуация:	Отправка данных. Вывод текстовой строки
Исходный набор данных:	<p>Корректный набор данных с check суммой. Снятие галочки для автоматического подсчёта check суммы</p> <div> <div>eb15c053405050505003504b424c4c43505004b0</div> <div>send command-></div> <div>Снять галочку с check суммы</div> </div>
Ожидаемый результат:	Появление пакета данных в центральном поле. Отображение текста «Hello» на дисплее
Фактический результат:	

5.3.3 Тест 3

Таблица 8 – Тест 3

Тестовая ситуация:	Отображение данных. Нажатие на руле кнопки рециркуляции воздуха
Исходный набор данных:	Нажатие на кнопку
Ожидаемый результат:	Появление пакета данных в центральном поле. Включение рециркуляции воздуха на климат контроле
Фактический результат:	

ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта был произведен анализ предметной области и реализовано программное средство. Согласно поставленной задаче, в данном приложении была реализована возможность чтения и записи данных по последовательному порту в автомобиль.

В ходе работы был разобран и изучен протокол общения между устройствами в автомобиле.

Для успешного выполнения всех поставленных задач потребовалось подробно разобраться с протоколами RS232, TTL логикой, организовать передачу необходимых данных, применить многие функции WinAPI для работы приложения.

Написанный код легко модифицируется для добавления новых функций. В дальнейшем возможны улучшения и доработки, вносящие новый функционал в программу.

Спроектированная программа имеет интуитивно понятный и простой интерфейс для пользователя.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Microsoft Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.microsoft.com/ru-ru/visualstudio/ide/?view=vs-2017>
- [2] Уилсон, С. Принципы проектирования и разработки программного обеспечения, учеб. курс. – СПб. : Русская Редакция, 2003 – 570 с.
- [3] MSDN [Электронный ресурс]. - Электронные данные. – Режим доступа: <https://msdn.microsoft.com/en-us/dn308572.aspx>
- [4] Confluence [Электронный ресурс]. - Электронные данные. – Режим доступа: <https://liswiki.galen.ru/pages/viewpage.action?pageId=70254660>
- [5] Lcard [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.lcard.ru/lexicon/ttl_in_out
- [6] Агуров П.В. Последовательные интерфейсы ПК. Практика программирования. – СПб. : БВХ-Петербург, 2004 – с.

ПРИЛОЖЕНИЕ А

Исходный код программы

Файл Source.cpp

```
#define _CRT_SECURE_NO_WARNINGS
//#include <Codes.h>
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <tchar.h>

#include <stdio.h>
#include <string.h>
#include <commctrl.h>
#include <tchar.h>

#include <iostream>
#include <strsafe.h>
#include "Codes.h"

using namespace std;

//Size of controls
#define WND_HEIGHT 1600
#define WND_WIDTH 830
#define WND_BETWEEN 30
#define WND_OUTER 290
#define WND_INNER 380
#define WND_FIELD_HEIGHT 700

#define COM_SETTINGS 18
#define COM_STOPRESUME 60

#define SCAN_COM 19
#define COM1 20
#define COM2 21
#define COM3 22
#define COM4 23
#define COM5 24
#define COM6 25
#define COM7 26
#define COM8 27
#define COM9 28
#define COM10 29
#define CONNECT_COM 30

#define SEND_EDIT 50
#define SEND_BUT 51
#define CODE_CLEAR 52
#define PRESETS_INFO 53

#define BUT_LOAD 1000
#define BUT_CHECK 1001
#define BUT_3 1002

#define MENU_EXIT 31
#define UP_MENU_SCAN 32
#define UP_MENU_SAVE 33
```

```

#define MENU_ABOUT 40

HANDLE hSerial;
DCB dcbSerialParams = { 0 };
HANDLE threadReadingCom;

HANDLE hComSettings;
HANDLE hStatusText;
HANDLE hComText;
HANDLE hCode;
HANDLE hCodeInfo;
HANDLE hCodePresets;

HANDLE hComPorts;
HANDLE hComStopResume;
bool isStop = FALSE;
bool isCalcChkSum = FALSE;
HANDLE hCodeClear;

HANDLE hCom1, hCom2, hCom3, hCom4, hCom5, hCom6, hCom7, hCom8, hCom9, hCom10;
BOOL ports[10] = { false };

HANDLE hSendEdit;
HANDLE hSendButton;
HANDLE hCheckSum;

//
HMENU hMenu;
DWORD dwSize;
DWORD dwBytesWritten;
BOOL iRet;
DWORD Data[512];
char dataChar[4096];

int mailLength = 0;
char mail[255];
HWND hEdit;

wchar_t statusText[255];
DWORD odometr = 414041;
char odometrChar[6] = { 0 };
DWORD chkSumm;
//

LPCTSTR sPortName = L"COM7";

char szFile[260]; //file name

void SetControls(HWND);

char* unsigned_to_hex_string(unsigned x, char* dest, size_t size) {
    snprintf(dest, size, "%X", x);
    return dest;
}

void PrintCode()
{
    char CodesInfo[255] = { 0 };
    strcat(CodesInfo, IBUSDevices[Data[0]]);
}

```

```

    strcat(CodesInfo, " to ");
    strcat(CodesInfo, IBUSDevices[Data[2]]);
    strcat(CodesInfo, "; ");
    strcat(CodesInfo, IBUSMessages[Data[3]]);
    strcat(CodesInfo, "\r\n");
    strcat(CodesInfo, "\r\n");
    strcat(CodesInfo, "-----");
    strcat(CodesInfo, "\r\n");

    strcat(dataChar, CodesInfo);

    int index = GetWindowTextLength((HWND)hCode);
    if (index > 20000)
    {
        SetFocus((HWND)hCode); // set focus
        SendMessageA((HWND)hCode, EM_SETSEL, 0, 10000);
        SendMessageA((HWND)hCode, EM_REPLACESEL, 0, (LPARAM) "");
        int index = GetWindowTextLength((HWND)hCode);
    }

    SetFocus((HWND)hCode); // set focus
    SendMessageA((HWND)hCode, EM_SETSEL, (WPARAM)index, (LPARAM)index);
    SendMessageA((HWND)hCode, EM_REPLACESEL, 0, (LPARAM)dataChar);

    memset(dataChar, 0, 512);

}

void WriteCOM()
{
    DWORD temp, signal; //temp - переменная-заглушка
    OVERLAPPED sync = { 0 };
    bool fl = false;

    sync.hEvent = CreateEvent(NULL, true, true, NULL); //создать событие
    WriteFile(hSerial, mail, mailLength, &temp, &sync); //записать байты в
порт (перекрывающаяся
    int j = GetLastError();
    signal = WaitForSingleObject(sync.hEvent, INFINITE); //приостановить
поток, пока не завершится

    if ((signal == WAIT_OBJECT_0) && (GetOverlappedResult(hSerial, &sync,
&temp, true))) fl = true; //если
        else fl = false;

    CloseHandle(sync.hEvent); //перед выходом из потока закрыть объект-собы-
тие
}

void ShowOdometr()
{
    int i = 5;
    int odo = odometr;

    while (odo != 0)
    {
        int k = odo % 10;
        odo = odo / 10;
    }
}

```



```

        odometrChar[i] = k + '0';
        i--;
    }

    char outStr[255] = { 0 };
    strcat(outStr, "ODOMETR: ");
    strcat(outStr, odometrChar);
    strcat(outStr, "km\r\n");

    wchar_t text[255] = { 0 };
    MultiByteToWideChar(0, 0, outStr, strlen(outStr), text, strlen(outStr));
    wcscat_s(statusText, text);

    SetWindowText((HWND)hStatusText, statusText);
}

void TryToParse()
{
    switch (Data[0]){
    case 0x80:
    {
        switch (Data[2]) {
        case 0xBF:

            switch (Data[3]) {
            case 0x17:
                odometr = Data[6] * 65536 + Data[5] * 256 + Data[5];
                ShowOdometr();
                break;
            }

            break;
        }

        break;
    }

    }

}

DWORD WINAPI ReadCOM(CONST LPVOID lpParam)
{
    while (true)
    {
        const int READ_TIME = 1;
        OVERLAPPED sync = { 0 };
        int reuslt = 0;
        unsigned long wait = 0, read = 0, state = 0;
        string str;

        sync.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

        if (SetCommMask(hSerial, EV_RXCHAR)) {
            /* Связываем порт и объект синхронизации*/
            WaitCommEvent(hSerial, &state, &sync);

            wait = WaitForSingleObject(sync.hEvent, READ_TIME);
            /* Данные получены */
            if (wait == WAIT_OBJECT_0) {
                /* Начинаем чтение данных */

```

```

char dst[255] = { 0 };

ReadFile(hSerial, &Data[0], 1, &read, &sync);
ReadFile(hSerial, &Data[1], 1, &read, &sync);
for (int i = 0; i < Data[1]; i++)
{
    ReadFile(hSerial, &Data[i+2], 1, &read, &sync);
}

TryToParse();

int len = Data[1] + 2;

char oneByte[2] = { 0 };
for (int i = 0; i < len; i++)
{
    unsigned_to_hex_string(Data[i], oneByte,
(sizeof(unsigned) * CHAR_BIT + 3) / 4 + 1);
    if (oneByte[1] == '\\0')
    {
        dst[i * 2] = '0';
        dst[i * 2 + 1] = oneByte[0];
    }
    else
    {
        dst[i * 2] = oneByte[0];
        dst[i * 2 + 1] = oneByte[1];
    }
}

//0x17
//0x18
//0x19

dst[len * 2] = '\\r';
dst[len * 2 + 1] = '\\n';

strcat(dataChar, dst);
PrintCode();

//MessageBoxA(NULL, &dst, NULL, NULL);
/* Ждем завершения операции чтения */
wait = WaitForSingleObject(sync.hEvent, READ_TIME);
/* Если все успешно завершено, узнаем какой объем дан-
ных прочитан */
if (wait == WAIT_OBJECT_0)
    if (GetOverlappedResult(hSerial, &sync, &read,
FALSE))
        reuslt = read;
    }
    CloseHandle(sync.hEvent);
}
return 0;
}

int StringToWString(std::wstring& ws, const std::string& s)
{

```

```

        std::wstring wsTmp(s.begin(), s.end());

        ws = wsTmp;

        return 0;
    }

void GetActivePorts()
{
    int r = 0;
    HKEY hkey = NULL;
    //Открываем раздел реестра, в котором хранится информация о COM портах
    r = RegOpenKeyEx(HKEY_LOCAL_MACHINE, TEXT("HARDWARE\\DEVICEMAP\\SERIAL-
COMM\\"), 0, KEY_READ, &hkey);
    if (r != ERROR_SUCCESS)
        return;

    unsigned long CountValues = 0, MaxValueNameLen = 0, MaxValueLen = 0;
    //Получаем информацию об открытом разделе реестра
    RegQueryInfoKey(hkey, NULL, NULL, NULL, NULL, NULL, NULL, &CountValues,
&MaxValueNameLen, &MaxValueLen, NULL, NULL);
    ++MaxValueNameLen;
    //Выделяем память
    TCHAR* bufferName = NULL, * bufferData = NULL;
    bufferName = (TCHAR*)malloc(MaxValueNameLen * sizeof(TCHAR));
    if (!bufferName)
    {
        RegCloseKey(hkey);
        return;
    }
    bufferData = (TCHAR*)malloc((MaxValueLen + 1) * sizeof(TCHAR));
    if (!bufferData)
    {
        free(bufferName);
        RegCloseKey(hkey);
        return;
    }

    unsigned long NameLen, type, DataLen;
    //Цикл перебора параметров раздела реестра
    for (unsigned int i = 0; i < CountValues; i++)
    {
        NameLen = MaxValueNameLen;
        DataLen = MaxValueLen;
        r = RegEnumValue(hkey, i, bufferName, &NameLen, NULL, &type,
(LPBYTE)bufferData, &DataLen);
        if ((r != ERROR_SUCCESS) || (type != REG_SZ))
            continue;

        //int j = 0;
        //char s = bufferData[3];
        ports[(int)(bufferData[3] - '0')] = TRUE;
    }
    //Освобождаем память
    free(bufferName);
    free(bufferData);
    //Закрываем раздел реестра
    RegCloseKey(hkey);
}

void Connecting()
{

```

```

        hSerial = CreateFileW(sPortName, GENERIC_READ | GENERIC_WRITE, 0, 0,
OPEN_EXISTING, 0, 0);
        if (hSerial == INVALID_HANDLE_VALUE)
        {
            if (GetLastError() == ERROR_FILE_NOT_FOUND)
            {
                MessageBox(NULL, L"no", NULL, NULL);
                //cout << "serial port does not exist.\n";
            }
            MessageBox(NULL, L"no2", NULL, NULL);

            //cout << "some other error occurred.\n";
        }

        dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
        if (!GetCommState(hSerial, &dcbSerialParams))
        {
            cout << "getting state error\n";
            MessageBox(NULL, L"no2", NULL, NULL);
        }
        dcbSerialParams.BaudRate = CBR_9600;
        dcbSerialParams.ByteSize = 8;
        dcbSerialParams.StopBits = ONESTOPBIT;
        dcbSerialParams.Parity = EVENPARITY;
        dcbSerialParams.fNull = FALSE;

        if (!SetCommState(hSerial, &dcbSerialParams))
        {
            cout << "error setting serial port state\n";
            MessageBox(NULL, L"no2", NULL, NULL);
        }

        COMMTIMEOUTS serialTimeouts;

        serialTimeouts.ReadIntervalTimeout = 0xFFFFFFFF;
        serialTimeouts.ReadTotalTimeoutConstant = 0;
        serialTimeouts.ReadTotalTimeoutMultiplier = 0;
        serialTimeouts.WriteTotalTimeoutConstant = 5000;
        serialTimeouts.WriteTotalTimeoutMultiplier = 1000;

        if (!SetCommTimeouts(hSerial, &serialTimeouts))
        {
            cout << "error Set timeouts\n";
            MessageBox(NULL, L"no2", NULL, NULL);
        }
    }

void EnableActivePorts()
{
    if (ports[0])
        EnableWindow((HWND)hCom1, TRUE);
    if (ports[1])
        EnableWindow((HWND)hCom1, TRUE);
    if (ports[2])
        EnableWindow((HWND)hCom2, TRUE);
    if (ports[3])
        EnableWindow((HWND)hCom3, TRUE);
    if (ports[4])
        EnableWindow((HWND)hCom4, TRUE);
    if (ports[5])
        EnableWindow((HWND)hCom5, TRUE);
    if (ports[6])

```

```

        EnableWindow((HWND)hCom6, TRUE);
    if (ports[7])
        EnableWindow((HWND)hCom7, TRUE);
    if (ports[8])
        EnableWindow((HWND)hCom8, TRUE);
    if (ports[9])
        EnableWindow((HWND)hCom9, TRUE);
    if (ports[10])
        EnableWindow((HWND)hCom10, TRUE);
}

LRESULT CALLBACK WindowProcedure(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{
    HDC hDc;
    HFONT hfont;
    PAINTSTRUCT ps;

    switch (uMsg) {

    case WM_COMMAND:

        switch (LOWORD(wParam)) {

        case SCAN_COM:
            GetActivePorts();
            EnableActivePorts();
            break;
        case CONNECT_COM:
            if (IsDlgButtonChecked(hWnd, COM1))
                sPortName = L"COM1";
            if (IsDlgButtonChecked(hWnd, COM2))
                sPortName = L"COM2";
            if (IsDlgButtonChecked(hWnd, COM3))
                sPortName = L"COM3";
            if (IsDlgButtonChecked(hWnd, COM4))
                sPortName = L"COM4";
            if (IsDlgButtonChecked(hWnd, COM5))
                sPortName = L"COM5";
            if (IsDlgButtonChecked(hWnd, COM6))
                sPortName = L"COM6";
            if (IsDlgButtonChecked(hWnd, COM7))
                sPortName = L"COM7";
            if (IsDlgButtonChecked(hWnd, COM8))
                sPortName = L"COM8";
            if (IsDlgButtonChecked(hWnd, COM9))
                sPortName = L"COM9";
            if (IsDlgButtonChecked(hWnd, COM10))
                sPortName = L"COM10";

            Connecting();
            threadReadingCom = CreateThread(NULL, 0, ReadCOM, NULL, 0,
NULL);

            break;

        case COM_STOPRESUME:
            if (!isStop)
            {
                SuspendThread(threadReadingCom);
            }
            else
            {
                ResumeThread(threadReadingCom);
            }
        }
    }
}

```

```

    }
    isStop = !isStop;
    break;
case CODE_CLEAR:
    SetWindowText((HWND)hCode, L"");
    break;
case SEND_BUT:
{
    //get Mail
    int len = SendMessage((HWND)hSendEdit, WM_GETTEXTLENGTH, 0,
0);

    mailLength = len / 2;
    char* buffer = new char[len * 2];
    SendMessage((HWND)hSendEdit, WM_GETTEXT, len * 2,
(LPARAM)buffer);

    char bufferHex[255] = { 0 };

    int currPos = 0;
    for (int i = 0; i < len; i++)
    {
        if (*(buffer + i * 2) != ' ') {
            bufferHex[currPos] = *(buffer + i * 2);
            currPos++;
        }
    }

    mailLength = currPos / 2;

    for (int i = 0; i < mailLength; i++)
    {
        char strByte[2];
        strByte[0] = bufferHex[i * 2];

        strByte[1] = bufferHex[i * 2 + 1];

        int g = (int)strtol(strByte, NULL, 16);

        mail[i] = char(g);
    }

    if (isCalcChkSum)
    {
        BYTE chkSum = 0x00;
        for (int i = 0; i < mailLength; i++)
        {
            chkSum = chkSum ^ mail[i];
        }
        mailLength++;
        mail[mailLength - 1] = char(chkSum);
    }

    SuspendThread(threadReadingCom);

    PurgeComm(hSerial, PURGE_TXCLEAR);
    WriteCOM();

    ResumeThread(threadReadingCom);
}

```

```

        break;
        case BUT_LOAD:
        {
            ShowOpenDialog(hWnd);
        }
        break;
        case BUT_CHECK:
        {
            LRESULT res = SendMessage((HWND)hChecksum, BM_GETCHECK, 0,
0);
            if (res == BST_CHECKED)
                isCalcChkSum = TRUE;
            else
                isCalcChkSum = FALSE;
        }
        break;
        /*case BUT_3:
            ShowOdometr();
            break;*/
        }
        break;
    case WM_PAINT:

        BeginPaint(hWnd, &ps);
        hDc = GetDC(hWnd);
        hfont = CreateFont(20, 0, 0, 0, FW_NORMAL, FALSE, FALSE, FALSE,
RUSSIAN_CHARSET, OUT_OUTLINE_PRECIS,
        CLIP_DEFAULT_PRECIS, CLEARTEXT_QUALITY, FIXED_PITCH,
TEXT("Times New Roman"));
        SelectObject(hDc, hfont);
        SetBkMode(hDc, TRANSPARENT);

        DeleteObject(hfont);
        EndPaint(hWnd, &ps);

        break;

    case WM_CREATE:
        SetControls(hWnd);

        break;
    case WM_CLOSE:
        DestroyWindow(hWnd);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        return DefWindowProcW(hWnd, uMsg, wParam, lParam);
        break;
    default:
        return DefWindowProcW(hWnd, uMsg, wParam, lParam);
    }
    return 0;
}

```

```

INT WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR
lpCmdLine, INT nCmdShow)
{
    WNDCLASSEX wcex;
    HWND hWnd;
    MSG msg;
    memset(&wcex, 0, sizeof(WNDCLASSEX));

```

```

    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style = 0;
    wcex.lpfnWndProc = WindowProcedure;
    wcex.cbClsExtra = 0;
    wcex.cbWndExtra = 0;
    wcex.hInstance = hInstance;
    wcex.hIcon = 0;
    wcex.hCursor = LoadCursor(0, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wcex.lpszMenuName = NULL;
    wcex.lpszClassName = L"MyWindowClass";
    wcex.hIconSm = 0;

    RegisterClassEx(&wcex);
    hWnd = CreateWindowExW(0, L"MyWindowClass", L"I-Bus Checker",
        WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX | WS_VISIBLE,
        0, 0, WND_HEIGHT, WND_WIDTH, 0, 0, hInstance, NULL);

    while (GetMessage(&msg, 0, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return 0;
}

// Create controls
void SetControls(HWND hWnd)
{
    hComSettings = CreateWindowW(L"Static", L"", WS_CHILD | WS_VISIBLE |
        SS_GRAYRECT, 1*WND_BETWEEN, 10, WND_OUTER, WND_FIELD_HEIGHT,
        hWnd, NULL, NULL, NULL);

    hCode = CreateWindowW(L"Edit", L"", WS_CHILD | WS_VISIBLE |
        SS_LEFT | WS_VSCROLL | WS_BORDER | ES_READONLY | ES_MULTILINE | ES_AUTOVSCROLL,
        2*WND_BETWEEN + WND_OUTER, 10, WND_INNER*2+WND_BETWEEN,
        WND_FIELD_HEIGHT, hWnd, NULL, NULL, NULL);

    hCodePresets = CreateWindowW(L"Edit", L"", WS_CHILD | WS_VISIBLE |
        SS_LEFT | WS_VSCROLL | WS_BORDER | ES_READONLY | ES_MULTILINE | ES_AUTOVSCROLL,
        4*WND_BETWEEN + WND_OUTER + 2*WND_INNER, 10, WND_OUTER+100,
        WND_FIELD_HEIGHT, hWnd, (HMENU)PRESETS_INFO, NULL, NULL);

    CreateWindowW(L"button", L"Scan...", WS_CHILD | WS_VISIBLE, 52, 22, 50,
        20, (HWND)hWnd, (HMENU)SCAN_COM, NULL, NULL);

    hComPorts = CreateWindowW(L"Button", L"Com ports:", WS_CHILD | WS_VISIBLE |
        BS_GROUPBOX, 50, 50, WND_OUTER - 40, 285, (HWND)hWnd, NULL, NULL,
        NULL);

    hCom1 = CreateWindowW(L"button", L"Com 1", WS_CHILD | WS_VISIBLE |
        BS_AUTORADIOBUTTON, 60, 80 + 0 * 25, WND_OUTER - 60, 20, (HWND)hWnd,
        (HMENU)COM1, NULL, NULL);

    hCom2 = CreateWindowW(L"button", L"Com 2", WS_CHILD | WS_VISIBLE |
        BS_AUTORADIOBUTTON, 60, 80 + 1 * 25, WND_OUTER - 60, 20, (HWND)hWnd,
        (HMENU)COM2, NULL, NULL);

    hCom3 = CreateWindowW(L"button", L"Com 3", WS_CHILD | WS_VISIBLE |
        BS_AUTORADIOBUTTON, 60, 80 + 2 * 25, WND_OUTER - 60, 20, (HWND)hWnd,
        (HMENU)COM3, NULL, NULL);
}

```



```

        hCom4 = CreateWindowW(L"button", L"Com 4", WS_CHILD | WS_VISIBLE |
BS_AUTORADIOBUTTON, 60, 80 + 3 * 25, WND_OUTER - 60, 20, (HWND)hWnd,
(HMENU)COM4, NULL, NULL);
        hCom5 = CreateWindowW(L"button", L"Com 5", WS_CHILD | WS_VISIBLE |
BS_AUTORADIOBUTTON, 60, 80 + 4 * 25, WND_OUTER - 60, 20, (HWND)hWnd,
(HMENU)COM5, NULL, NULL);
        hCom6 = CreateWindowW(L"button", L"Com 6", WS_CHILD | WS_VISIBLE |
BS_AUTORADIOBUTTON, 60, 80 + 5 * 25, WND_OUTER - 60, 20, (HWND)hWnd,
(HMENU)COM6, NULL, NULL);
        hCom7 = CreateWindowW(L"button", L"Com 7", WS_CHILD | WS_VISIBLE |
BS_AUTORADIOBUTTON, 60, 80 + 6 * 25, WND_OUTER - 60, 20, (HWND)hWnd,
(HMENU)COM7, NULL, NULL);
        hCom8 = CreateWindowW(L"button", L"Com 8", WS_CHILD | WS_VISIBLE |
BS_AUTORADIOBUTTON, 60, 80 + 7 * 25, WND_OUTER - 60, 20, (HWND)hWnd,
(HMENU)COM8, NULL, NULL);
        hCom9 = CreateWindowW(L"button", L"Com 9", WS_CHILD | WS_VISIBLE |
BS_AUTORADIOBUTTON, 60, 80 + 8 * 25, WND_OUTER - 60, 20, (HWND)hWnd,
(HMENU)COM9, NULL, NULL);
        hCom10 = CreateWindowW(L"button", L"Com 10", WS_CHILD | WS_VISIBLE |
BS_AUTORADIOBUTTON, 60, 80 + 9 * 25, WND_OUTER - 60, 20, (HWND)hWnd,
(HMENU)COM10, NULL, NULL);
        CreateWindowW(L"button", L"Connect to ...", WS_CHILD | WS_VISIBLE, 52,
WND_FIELD_HEIGHT - 40, 100, 20, (HWND)hWnd, (HMENU)CONNECT_COM, NULL, NULL);

        EnableWindow((HWND)hCom1, FALSE);
        EnableWindow((HWND)hCom2, FALSE);
        EnableWindow((HWND)hCom3, FALSE);
        EnableWindow((HWND)hCom4, FALSE);
        EnableWindow((HWND)hCom5, FALSE);
        EnableWindow((HWND)hCom6, FALSE);
        EnableWindow((HWND)hCom7, FALSE);
        EnableWindow((HWND)hCom8, FALSE);
        EnableWindow((HWND)hCom9, FALSE);
        EnableWindow((HWND)hCom10, FALSE);

        hComStopResume = CreateWindowW(L"Button", L"Pause/Resume", WS_CHILD |
WS_VISIBLE | SS_LEFT, 1 * WND_BETWEEN, 10 + WND_FIELD_HEIGHT + 20, WND_OUTER
/ 2 - 20, 30, hWnd, (HMENU)COM_STOPRESUME, NULL, NULL);
        hCodeClear = CreateWindowW(L"Button", L"Clear", WS_CHILD | WS_VISIBLE |
SS_LEFT, 1 * WND_BETWEEN + WND_OUTER / 2 + 20, 10 + WND_FIELD_HEIGHT + 20,
WND_OUTER / 2 - 20, 30, hWnd, (HMENU)CODE_CLEAR, NULL, NULL);

        hSendEdit = CreateWindowW(L"Edit", NULL, WS_CHILD | WS_VISIBLE | SS_LEFT
| WS_BORDER, 2 * WND_BETWEEN + WND_OUTER, 10 + WND_FIELD_HEIGHT + 20, (4 *
WND_BETWEEN + WND_OUTER + 2 * WND_INNER - 180) - (2 * WND_BETWEEN + WND_OUTER),
30, hWnd, NULL, NULL, NULL);
        hSendButton = CreateWindowW(L"Button", L"Send command->", WS_CHILD |
WS_VISIBLE | SS_LEFT, 4 * WND_BETWEEN + WND_OUTER + 2 * WND_INNER - 180, 10 +
WND_FIELD_HEIGHT + 20, 150, 30, hWnd, (HMENU)SEND_BUT, NULL, NULL);

        CreateWindowW(L"Button", L"Load commands", WS_CHILD | WS_VISIBLE |
BS_DEFPUSHBUTTON, 4 * WND_BETWEEN + WND_OUTER + 2 * WND_INNER + WND_OUTER/2 +
95, 10 + WND_FIELD_HEIGHT + 20, 150, 30, hWnd, (HMENU)BUT_LOAD, NULL, NULL);
        hCheckSum = CreateWindowW(L"Button", L"Calculate check summ", WS_CHILD |
WS_VISIBLE | BS_AUTOCHECKBOX, 4 * WND_BETWEEN + WND_OUTER + 2 * WND_INNER, 10
+ WND_FIELD_HEIGHT + 20, 170, 30, hWnd, (HMENU)BUT_CHECK, NULL, NULL);

        hComText = CreateWindowW(L"Static", L"CONNECTION INFO\r\n\r\nBaud rate:
9600\r\nByte size: 8\r\nStop bit: 1\r\nParity: even", WS_CHILD | WS_VISIBLE,
1 * WND_BETWEEN + 20, WND_FIELD_HEIGHT*3/4, WND_OUTER - 2*20, 100,
        hWnd, NULL, NULL, NULL);
}

```

Файл Codes.h

```
#pragma once
#include <Windows.h>

const char* IBUSDevices[] = {
    "Broadcast",
    "0x01",
    "0x02",
    "0x03",
    "0x04",
    "0x05",
    "0x06",
    "0x07",
    "0x08",
    "0x09",
    "0x0A",
    "0x0B",
    "0x0C",
    "0x0D",
    "0x0E",
    "0x0F",
    "0x10",
    "0x11",
    "0x12",
    "0x13",
    "0x14",
    "0x15",
    "0x16",
    "0x17",
    "CD Changer",
    "0x19",
    "0x1A",
    "0x1B",
    "0x1C",
    "0x1D",
    "0x1E",
    "0x1F",
    "0x20",
    "0x21",
    "0x22",
    "0x23",
    "0x24",
    "0x25",
    "0x26",
    "0x27",
    "0x28",
    "0x29",
    "0x2A",
    "0x2B",
    "0x2C",
    "0x2D",
    "0x2E",
    "0x2F",
    "Check control module",
    "0x31",
    "0x32",
    "0x33",
    "0x34",
    "0x35",
    "0x36",
    "0x37",
    "0x38",
```

```
"0x39",
"0x3A",
"VideoModule",
"0x3C",
"0x3D",
"0x3E",
"Diagnostic??",
"0x40",
"0x41",
"0x42",
"MenuScreen",
"Immobliser??",
"0x45",
"0x46",
"0x47",
"0x48",
"0x49",
"0x4A",
"0x4B",
"0x4C",
"0x4D",
"0x4E",
"0x4F",
"Multi function steering wheel",
"0x51",
"0x52",
"0x53",
"0x54",
"0x55",
"0x56",
"0x57",
"0x58",
"0x59",
"0x5A",
"0x5B",
"0x5C",
"0x5D",
"0x5E",
"0x5F",
"Park distance control",
"0x61",
"0x62",
"0x63",
"0x64",
"0x65",
"0x66",
"0x67",
"Radio",
"0x69",
"Digital signal processing audio amplifier",
"0x6B",
"0x6C",
"0x6D",
"0x6E",
"0x6F",
"0x70",
"0x71",
"0x72",
"0x73",
"0x74",
"0x75",
"0x76",
"0x77",
```

"0x78",
"0x79",
"0x7A",
"0x7B",
"0x7C",
"0x7D",
"0x7E",
"Navigation??",
"Instrument cluster electronics IKE",
"0x81",
"0x82",
"0x83",
"0x84",
"0x85",
"0x86",
"0x87",
"0x88",
"0x89",
"0x8A",
"0x8B",
"0x8C",
"0x8D",
"0x8E",
"0x8F",
"0x90",
"0x91",
"0x92",
"0x93",
"0x94",
"0x95",
"0x96",
"0x97",
"0x98",
"0x99",
"0x9A",
"0x9B",
"0x9C",
"0x9D",
"0x9E",
"0x9F",
"0xA0",
"0xA1",
"0xA2",
"0xA3",
"0xA4",
"0xA5",
"0xA6",
"0xA7",
"??",
"0xA9",
"0xAA",
"0xAB",
"0xAC",
"0xAD",
"0xAE",
"0xAF",
"0xB0",
"0xB1",
"0xB2",
"0xB3",
"0xB4",
"0xB5",
"0xB6",

"0xB7",
"0xB8",
"0xB9",
"0xBA",
"TV module",
"0xBC",
"0xBD",
"0xBE",
"LCM module",
"Multi-info display MID",
"0xC1",
"0xC2",
"0xC3",
"0xC4",
"0xC5",
"0xC6",
"0xC7",
"Telephone",
"0xC9",
"0xCA",
"0xCB",
"0xCC",
"0xCD",
"0xCE",
"0xCF",
"Light control module?? Navigation location",
"0xD1",
"0xD2",
"0xD3",
"0xD4",
"0xD5",
"0xD6",
"0xD7",
"0xD8",
"0xD9",
"0xDA",
"0xDB",
"0xDC",
"0xDD",
"0xDE",
"0xDF",
"0xE0",
"0xE1",
"0xE2",
"0xE3",
"0xE4",
"0xE5",
"0xE6",
"OBC TextBar",
"Rain/Light Sensor??",
"0xE9",
"0xEA",
"0xEB",
"0xEC",
"Lights, Wipers, SeatMemory",
"0xEE",
"0xEF",
"Board monitor buttons",
"0xF1",
"0xF2",
"0xF3",
"0xF4",
"0xF5",

```

    "0xF6",
    "0xF7",
    "0xF8",
    "0xF9",
    "0xFA",
    "0xFB",
    "0xFC",
    "0xFD",
    "0xFE",
    "BROADCAST"
};

const char* IBUSMessages[] = {
    "0x00",
    "Device status request",
    "Device status ready",
    "Bus status request",
    "Bus status",
    "0x05",
    "DIAG read memory",
    "DIAG write memory",
    "DIAG read coding data",
    "DIAG write coding data",
    "0x0A",
    "0x0B",
    "Vehicle control",
    "0x0D",
    "0x0E",
    "0x0F",
    "Ignition status request",
    "Ignition status",
    "IKE sensor status request",
    "IKE sensor status",
    "Country coding status request",
    "Country coding status",
    "Odometer request",
    "Odometer",
    "Speed/RPM",
    "Temperature",
    "IKE text display/Gong",
    "IKE text status",
    "Gong",
    "Temperature request",
    "0x1E",
    "UTC time and date",
    "0x20",
    "Radio Short cuts",
    "Text display confirmation",
    "Display Text from start",
    "Update Text",
    "0x25",
    "0x26",
    "0x27",
    "0x28",
    "0x29",
    "On-Board Computer State Update",
    "Telephone indicators",
    "0x2C",
    "0x2D",
    "0x2E",
    "0x2F",
    "0x30",
    "MID buttons",

```

"MFL buttons",
"0x33",
"DSP Equalizer Button",
"0x35",
"0x36",
"0x37",
"CD status request",
"CD status",
"0x3A",
"MFL buttons 2",
"0x3C",
"SDRS status request",
"SDRS status",
"0x3F",
"Set On-Board Computer Data",
"On-Board Computer Data Request",
"0x42",
"0x43",
"0x44",
"0x45",
"LCD Clear",
"BMBT buttons",
"BMBT buttons",
"KNOB button",
"Cassette control",
"Cassette status",
"0x4C",
"0x4D",
"0x4E",
"RGB Control",
"0x50",
"0x51",
"0x52",
"Vehicle data request",
"Vehicle data status",
"0x55",
"0x56",
"0x57",
"0x58",
"0x59",
"Lamp status request",
"Lamp status",
"Instrument cluster lighting status",
"0x5D",
"0x5E",
"0x5F",
"0x60",
"0x61",
"0x62",
"0x63",
"0x64",
"0x65",
"0x66",
"0x67",
"0x68",
"0x69",
"0x6A",
"0x6B",
"0x6C",
"0x6D",
"0x6E",
"0x6F",
"0x70",

"Rain sensor status request",
"Remote Key buttons",
"0x73",
"EWS key status",
"0x75",
"0x76",
"0x77",
"0x78",
"Doors/windows status request",
"Doors/windows status",
"0x7B",
"SHD status",
"0x7D",
"0x7E",
"0x7F",
"0x80",
"0x81",
"0x82",
"0x83",
"0x84",
"0x85",
"0x86",
"0x87",
"0x88",
"0x89",
"0x8A",
"0x8B",
"0x8C",
"0x8D",
"0x8E",
"0x8F",
"0x90",
"0x91",
"0x92",
"0x93",
"0x94",
"0x95",
"0x96",
"0x97",
"0x98",
"0x99",
"0x9A",
"0x9B",
"0x9C",
"0x9D",
"0x9E",
"0x9F",
"DIAG data",
"0xA1",
"Current position and time",
"0xA3",
"Current location",
"Screen text",
"0xA6",
"TMC status request",
"0xA8",
"0xA9",
"Navigation Control",
"0xAB",
"0xAC",
"0xAD",
"0xAE",
"0xAF",


```
"0xB0",
"0xB1",
"0xB2",
"0xB3",
"0xB4",
"0xB5",
"0xB6",
"0xB7",
"0xB8",
"0xB9",
"0xBA",
"0xBB",
"0xBC",
"0xBD",
"0xBE",
"0xBF",
"0xC0",
"0xC1",
"0xC2",
"0xC3",
"0xC4",
"0xC5",
"0xC6",
"0xC7",
"0xC8",
"0xC9",
"0xCA",
"0xCB",
"0xCC",
"0xCD",
"0xCE",
"0xCF",
"0xD0",
"0xD1",
"0xD2",
"0xD3",
"RDS channel list",
"0xD5",
"0xD6",
"0xD7",
"0xD8",
"0xD9",
"0xDA",
"0xDB",
"0xDC",
"0xDD",
"0xDE",
"0xDF",
"0xE0",
"0xE1",
"0xE2",
"0xE3",
"0xE4",
"0xE5",
"0xE6",
"0xE7",
"0xE8",
"0xE9",
"0xEA",
"0xEB",
"0xEC",
"0xED",
"0xEE",
```

```
    "0xEF",  
    "0xF0",  
    "0xF1",  
    "0xF2",  
    "0xF3",  
    "0xF4",  
    "0xF5",  
    "0xF6",  
    "0xF7",  
    "0xF8",  
    "0xF9",  
    "0xFA",  
    "0xFB",  
    "0xFC",  
    "0xFD",  
    "0xFE",  
    "0xFF"  
};
```