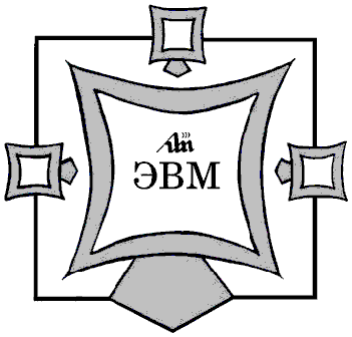
Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

ВЫЧИСЛИТЕЛЬНЫЕ КОМПЛЕКСЫ, СИСТЕМЫ И СЕТИ

И. И. Глецевич, Д. В. Ламовский, Д. А. Пынькин

Лабораторный практикум
для студентов специальности
I-40 02 01 «Вычислительные машины, системы и сети»
всех форм обучения



Минск БГУИР 2010

УДК 004.712(076)
ББК 32.973.202-04я73
Г 53

Р е ц е н з е н т :
ведущий научный сотрудник лаборатории №222
Объединенного института проблем информатики НАН Беларуси,
кандидат технических наук А. А. Дудкин

Глецевич, И. И.

Г 53 Вычислительные комплексы, системы и сети : лабораторный практикум для студентов специальности I-40 02 01 «Вычислительные машины, системы и сети» / И. И. Глецевич, Д. В. Ламовский, Д. А. Пынькин. – Минск : БГУИР, 2010. – 36 с. : ил.

ISBN 978-985-488-618-3

Лабораторный практикум по дисциплине ВКСиС содержит описание лабораторной установки, построенной на основе ПЭВМ с установленным адаптером последовательного интерфейса RS-485. Рассмотрена структурная и архитектурная организация UART 16550. Приведены примеры программирования.

УДК 004.712(076)
ББК 32.973.202-04я73

ISBN 978-985-488-618-3

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью сетевой составляющей дисциплины «Вычислительные комплексы, системы и сети» является изучение теоретических основ построения компьютерных сетей.

Лабораторный практикум запланирован в двух частях. Первая часть посвящена описанию специфического оборудования лабораторной установки и способов его программирования. Сюда также включена вводная лабораторная работа. Во второй части будут подробно описаны все оставшиеся лабораторные работы, предусмотренные рабочей учебной программой дисциплины.

Предполагается, что студенты, изучающие данную дисциплину, уже умеют программировать и знакомы с основами схемотехники.

При проведении лабораторных занятий для практической реализации изучаемых алгоритмов студентам необходимо предоставить соответствующий инструментарий. В том числе нужно выбрать некий базовый «сетевой интерфейс», который бы наиболее полно соответствовал трем основным предъявляемым требованиям:

1 Интерфейс должен быть достаточно простым.

2 Интерфейс должен быть достаточно распространенным.

3 Интерфейс должен присутствовать в ПЭВМ.

Наиболее адекватным выбором являются классические последовательные интерфейсы RS-232 и его потомок RS-485. Ни интерфейс Ethernet, в котором большинство представляемых к изучению алгоритмов «зашито» аппаратно, ни интерфейс USB с его очень специфической организацией в данной ситуации не подходят.

Последовательные порты – это больша́я и весьма весомая часть развития компьютерных систем, наложившая отпечаток на многие другие интерфейсы. Программные модели последовательных портов реализованы практически для всех ОС. Каждый специалист в области компьютерных сетей должен иметь представление о них.

В настоящее время интерфейс RS-232 используется в основном при подключении различных выделенных консолей (например, к маршрутизаторам), а RS-485 – это один из наиболее распространенных интерфейсов связи промышленного оборудования (например, применяется в топологиях с так называемыми полевыми шинами).

1 ОПИСАНИЕ ЛАБОРАТОРНОЙ УСТАНОВКИ

Для того, чтобы выполнить задания к лабораторным работам, нужно хорошо знать структурную и функциональную организацию лабораторной установки, построенной на основе ПЭВМ. Ниже описываются ключевые моменты. Более подробные сведения технического характера изложены в литературных источниках [1–4].

1.1 Эволюция COM-порта

История развития последовательного (serial) или, по-другому, коммуникационного (COMmunication) порта неразрывно связана с развитием элементной базы. Применительно к ПЭВМ разработчиком как базовой архитектуры, так и типовых схем оборудования являлась и до сих пор является компания Intel.

В развитии COM-порта ПЭВМ можно выделить следующие основные этапы (следует отметить, что этот процесс сильно коррелирует с развитием COM-портов всех типов компьютерных систем):

1 В свое время (семидесятые годы XX века) в составе периферийной части комплекта микросхем поддержки микропроцессора 8080 компания Intel разработала два контроллера последовательного порта. Один из них, 8250, получил название UART (Universal Asynchronous Receiver/Transmitter) – универсальный асинхронный приемник-передатчик. Второй, 8251, получил название USART (Universal Synchronous/Asynchronous Receiver/Transmitter) – универсальный синхронно-асинхронный приемник-передатчик. Эти контроллеры были рассчитаны на подключение по шине X-Bus (шина ввода-вывода, внутрисхемный восьмибитный предшественник системной шины ISA) и поэтому без труда были перенесены в первые ПЭВМ на базе процессора 8086 и его модификаций (то есть в компьютеры класса IBM PC XT) с наиболее распространенной тогда системной шиной ISA. Совместно с контроллером параллельного порта 8255 микросхема UART (либо USART) устанавливалась на плату специального адаптера и подключалась к материнской плате ПЭВМ посредством разъема системной шины. В это же время возникла традиция устанавливать последовательные порты парами (COM1 и COM2).

2 Времена доминирования процессоров 80286 – Intel486 (то есть компьютеров класса IBM PC AT и IBM PS/2) ознаменованы постепенно набравшими силу интеграционными процессами. На первом этапе происходило распространение и развитие самих контроллеров. В СССР был создан аналог 8251 под названием КР580ВВ51А, который стал массово применяться в серии ЕС ПЭВМ. На Западе же, наоборот, развитие получила микросхема 8250. Апофеозом достаточно быстрого усовершенствования 8250 стали несколько UART, среди которых следует выделить 16550, причем это была разработка уже не Intel, а National Semiconductor. Именно эта

микросхема стала де-факто стандартной на длительное время (архитектурная совместимость сохраняется вплоть до настоящего времени). Микросхема 16550 имеет два основных преимущества перед 8250:

– более высокая пропускная способность последовательного интерфейса (максимальная стандартная пропускная способность увеличена с 9600 baud до 115200 baud);

– возможность буферизации (две очереди FIFO по 16 байт – на стороне передатчика и на стороне приемника).

3 В дальнейшем интеграционные процессы привели к появлению так называемых мультикарт – подключаемых посредством разъема системной шины (по-прежнему обычно ISA) плат расширения с интегрированными контроллерами: последовательного порта (2x16550), параллельного порта, игрового порта, НГМД и НЖМД. Причем все эти функции сочетались в одной БИС с типичным названием Multi I/O. Основными производителями чипов Multi I/O были компании Winbond, UMC, GoldStar и другие (рисунок 1.1).

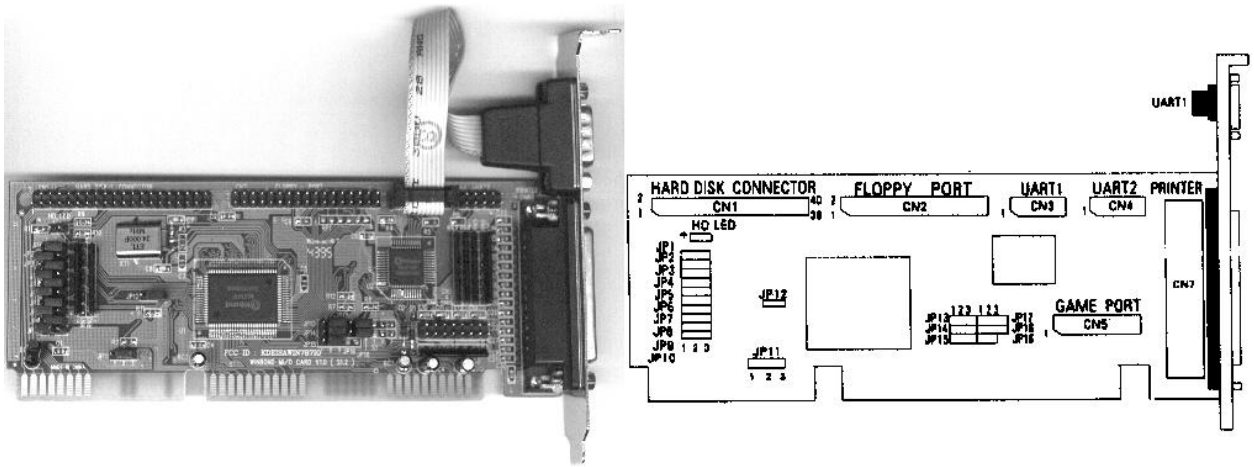


Рисунок 1.1 – Одна из массово выпускавшихся мультикарт

4 Для ПЭВМ на базе поздних Intel486 уже была характерна интеграция чипа Multi I/O на материнскую плату. Одними из ведущих производителей таковой элементной базы стали компании SiS и OPTi. Компания Intel постепенно отказалась от производства микросхем интегрированной периферии и сосредоточилась на разработке наборов микросхем («чипсетах») системной логики (последней БИС Multi I/O, которая была выпущена самой компанией Intel, стала 82091).

5 Во времена процессоров Pentium сформировалась действительная до сих пор базовая крупноблочная структура материнской платы ПЭВМ, состоящая из четырех основных БИС (рисунок 1.2).

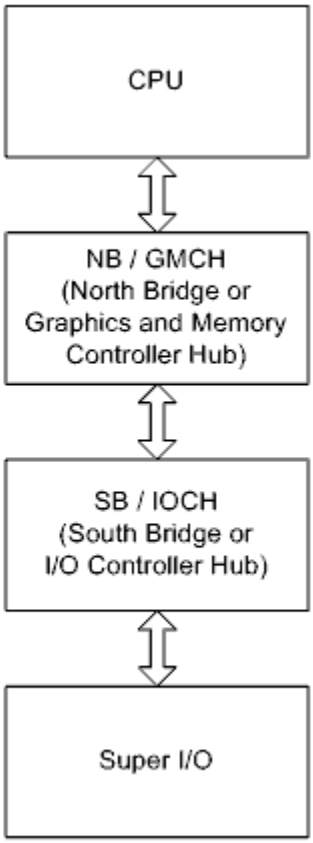


Рисунок 1.2 – Базовая структура современной ПЭВМ

Контроллеры последовательного порта (по той же схеме 2x16550) в составе интегрированной периферии были перенесены и в эту структуру. Однако в связи с некоторой заменой функционала интегрированной периферии (например, удаление контроллера НЖМД и добавление контроллера клавиатуры) вместо названия Multi I/O стало больше использоваться название Super I/O. С этого момента реализации последовательных портов не претерпевали никаких изменений.

Основными производителями чипов Super I/O являются компании Winbond, ITE и SMSC (рисунок 1.3).

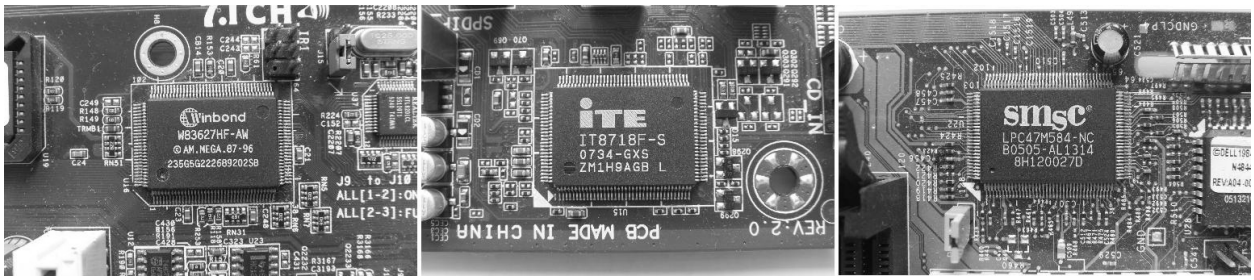


Рисунок 1.3 – Примеры Super I/O основных производителей

После перехода от мостовой (bridges) организации ПЭВМ к хабовой (hubs) «в пределах» структуры, показанной на рисунке 1.2 (начиная с восьмисотой серии чипсетов Intel в эпоху Pentium III), для внутрисхемного подключения Super I/O вместо шины X-Bus стала использоваться шина LPC (Low Pin Count) – специализированная разновидность шины PCI с небольшим числом разрядов.

Типовая внутренняя структура Super I/O показана на рисунке 1.4.

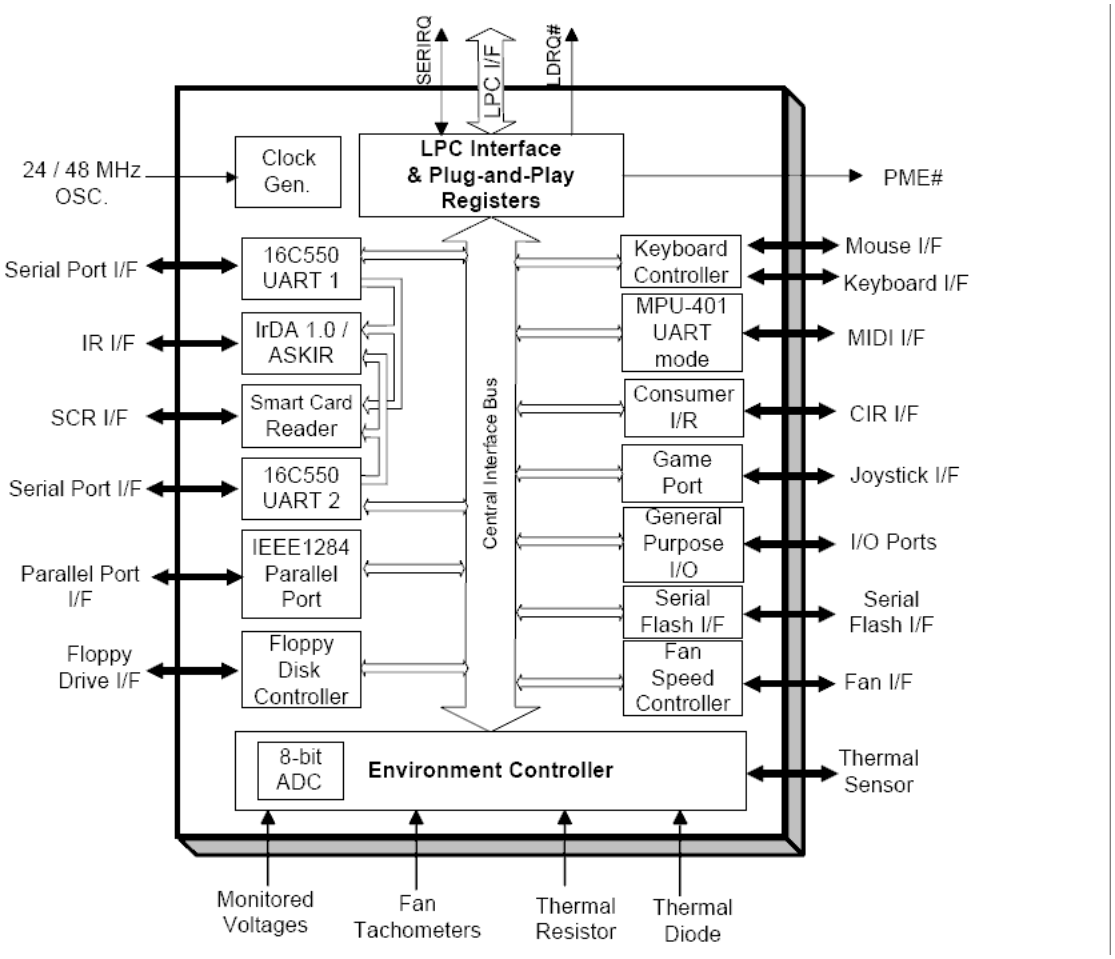


Рисунок 1.4 – Структурная схема Super I/O на примере ITE IT8712F [3]

6 В настоящее время (приблизительно с 2005 года) традиционный последовательный интерфейс ПЭВМ считается устаревшим (legacy), часто выводится из состава интегрированной периферии и на материнских платах встречается все реже. Однако возобновлено производство мультикарт – новые версии представляют собой платы расширения с интерфейсом PCI.

Сейчас в качестве основного последовательного интерфейса ПЭВМ рассматривается шина USB (Universal Serial Bus), впервые введенная в состав ПЭВМ еще в эпоху процессоров Pentium.

1.2 Структура UART 16550

Структурная схема UART 16550 показана на рисунке 1.5.

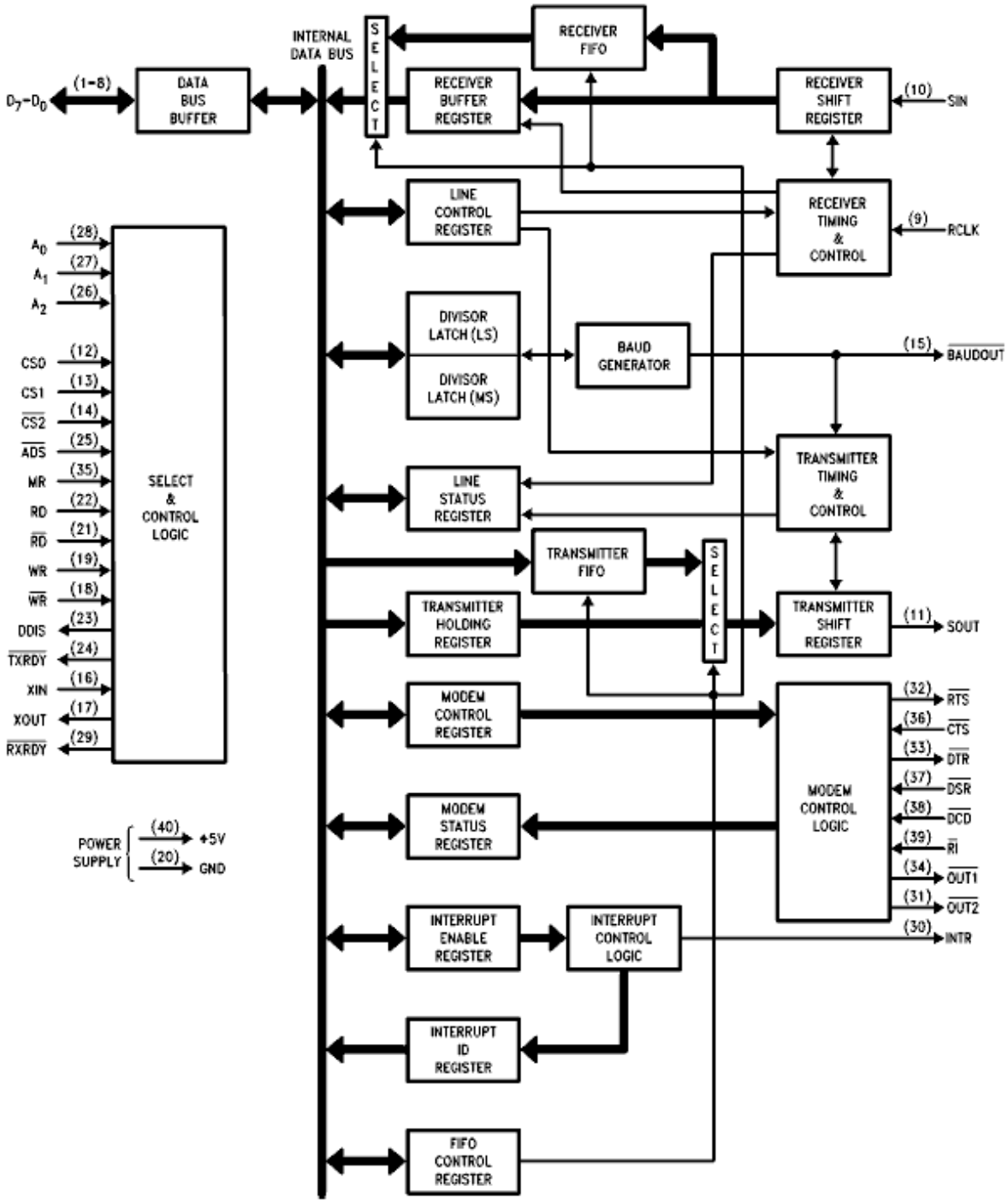


Рисунок 1.5 – Структурная схема UART 16550 [1]

Сам факт передачи информации подразумевает наличие передатчика, приемника и канала, по которому они связаны. Как и следует из названия, UART 16550 сочетает в себе функции как приемника, так и передатчика. Предоставляется возможность подключения к двунаправленному каналу

связи в соответствии со стандартом RS-232. На аппаратном уровне приемник и передатчик работают параллельно, то есть по отдельным физическим цепям полностью независимо друг от друга.

С точки зрения организации работы сетевых интерфейсов принято выделять два типа оборудования:

1 *Оконечное оборудование данных* (ООД) – Data Terminal Equipment (DTE).

2 *Аппаратура передачи данных* (АПД) – Data Communication Equipment (DCE).

ООД находится на самой границе сети передачи данных и «концентрирует», то есть создает и потребляет, передаваемые информационные потоки. АПД находится в пределах сети передачи данных и «транслирует», то есть позволяет передавать и принимать, информационные потоки. Еще одна существенная разница заключается в синхронизации передаваемых потоков – источником синхронизации всегда является АПД.

Интерфейс RS-232 (это традиционное название, последняя редакция 1997 года называется EIA-232-F, существуют и другие названия) предназначен для подключения АПД (например, модема) к ООД (например, UART).

Для физического подключения по стандарту RS-232 используются девятиконтактные разъемы DE-9 (рисунок 1.6). В старых ПЭВМ класса IBM PC использовался и аналогичный двадцатипятиконтактный разъем DB-25. Принято, что штырьковая часть разъема устанавливается со стороны ООД, а гнездовая часть – со стороны АПД.

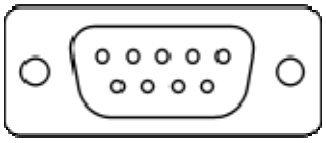


Рисунок 1.6 – Разъем D-Sub DE-9

Логотип последовательного порта: « | O | O | ». Согласно PC System Design Guide, с 1999 года разъемы последовательных портов окрашиваются в бирюзовый цвет.

Служебные цепи RS-232 позволяют организовать контроль информационного потока (flow control). Например, это позволяет избегать переполнения приемника, приостанавливая «быстрый» передатчик. Следует отметить, что практически все служебные цепи напрямую связаны с соответствующими регистрами управления и состояния UART 16550, то есть «открыты» для программирования. Следовательно, алгоритмы контроля реализуются программно и закладываются, например, в драйверы операционных систем. Контроль может быть как полуаппаратным (с задействованием сигналов RS-232), так и сугубо программным.

Традиционное назначение цифровых цепей RS-232:

– SOUT (Serial Output) – выход передатчика;

– SIN (Serial Input) – вход приемника;

– RTS (Request to Send) – сигнал-запрос от UART к модему о передаче байта;

– CTS (Clear to Send) – сигнал-подтверждение от модема к UART о готовности принять байт для передачи;

– DSR (Data Set Ready) – сигнал от модема к UART о готовности к взаимодействию;

– DTR (Data Terminal Ready) – сигнал от UART к модему о готовности к взаимодействию;

– DCD (Data Carrier Detect) – сигнал от модема к UART об обнаружении данных;

– RI (Ring Indicator) – сигнал от модема к UART об обнаружении входящего телефонного звонка.

Очевидно, что традиционное использование пары RTS/CTS позволяет контролировать передачу только в одном направлении – от UART к модему. Для контроля передачи в обратном направлении использовалась пара DSR/DTR.

В большинстве современных реализаций контроль по-прежнему предполагает наличие обратной связи, но осуществляется только приемником. Два основных способа:

1 RTS/CTS – полуаппаратный.

2 XON/XOFF – программный.

UART контролирует передачу данных «к себе» управляя активностью цепи RTS, модем – CTS. Значительно реже применяется способ DTR/DSR – полностью аналогичен способу RTS/CTS, но значения сигналов сохраняются на протяжении всего информационного обмена, а не каждой посылки. При полностью программном контроле приемник передает в обратном направлении специальный байт XON (стандартное значение 11h) для инициирования передачи и специальный байт XOFF (стандартное значение 13h) для остановки передачи.

Как было сказано выше, в стандартной ситуации ООД взаимодействуют между собой посредством АПД, причем с помощью так называемых «рукопожатий» (handshaking) с АПД. При этом подключение АПД к ООД реализуется посредством «прямого» кабеля (straight-through cable) (рисунок 1.7, а). Для подключения двух ООД друг к другу непосредственно необходим один из вариантов нуль-модемного (null-modem, поскольку предполагается отсутствие модема) кросс-кабеля (crossover cable, поскольку цепи SIN и SOUT скрещиваются).

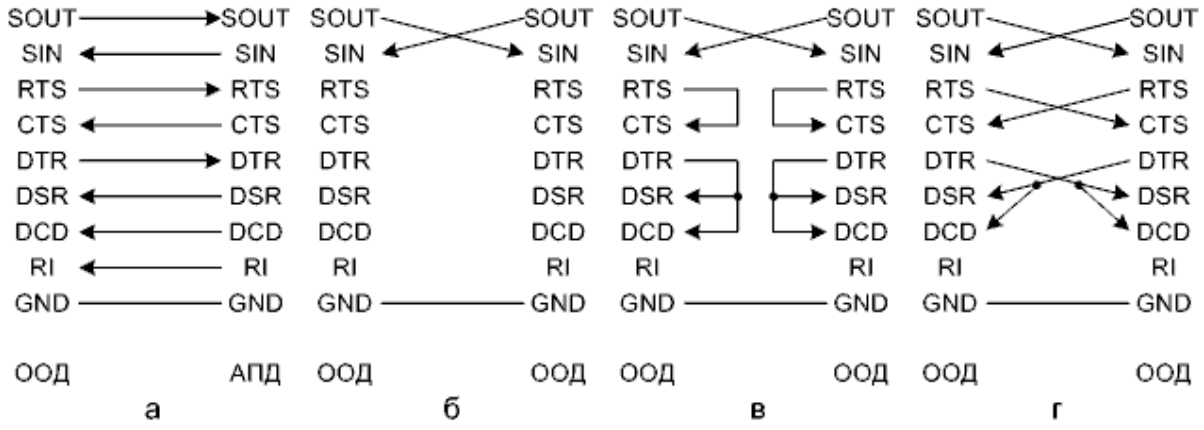


Рисунок 1.7 – Варианты кабелей RS-232:
а – straight-through; б, в, г – null-modem

Для изготовления кросс-кабеля требуется минимум три провода (рисунок 1.7, б). Иногда программное обеспечение рассчитано только на использование модемов. В подобных ситуациях для непосредственного подключения двух ООД необходимо «закоротить» соответствующие пары сигналов (рисунок 1.7, в). Часто в литературе приводится еще одна схема нуль-модемного кобеля, в соответствии с которой роль модема играет ООД-абонент (рисунок 1.7, г).

Как известно, без синхронизации в том либо ином виде передать ничего не возможно. Передатчик и приемник, по крайней мере, должны «встретиться во времени». Поскольку передатчик и приемник не имеют общего источника времени, в канал вводятся специальные синхросигналы. С точки зрения синхронизации, применительно к последовательным каналам связи выделяются два режима обмена:

1 *Асинхронный* (asynchronous) – синхронизируется посылка каждого информационного байта.

2 *Синхронный* (synchronous) – синхронизируется весь информационный обмен.

Формат посылки в асинхронном режиме показан на рисунке 1.8.

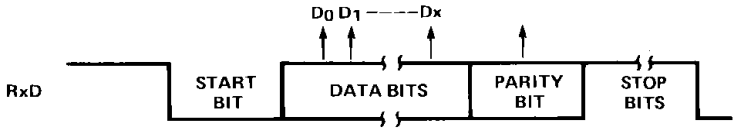


Рисунок 1.8 – Формат посылки в асинхронном режиме [2]

Атомарной, то есть минимальной неделимой единицей, с которой работает как UART, так и USART, является байт, причем один байт не

обязательно равен восьми битам и может содержать от 5 до 8 битов. По умолчанию, линия находится в состоянии логической единицы. При наличии байта для передачи передатчик переводит линию в состояние логического нуля, то есть передает старт-бит, что говорит приемнику о том, что на следующем такте нужно «ловить» первый информационный бит. Стоп-бит необходим для того, чтобы после передачи информационной последовательности гарантированно вернуть линию в исходное, то есть единичное состояние. Старт-бит всегда один, а стоп-битов может быть один, полтора либо два. Для проверки целостности информационной части, если эта проверка включена, за информационной частью может вставляться бит паритета. При этом действует правило дополнения. Например, если включена проверка единиц на четность (even), то бит паритета формируется таким образом, чтобы общее число единиц (в информационной части плюс бит паритета) было четным. Или, если включена проверка нулей на нечетность (odd), общее количество нулей должно быть нечетным. Ошибки отслеживаются приемником.

Формат посылки в синхронном режиме показан на рисунке 1.9.

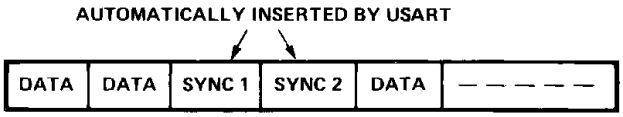


Рисунок 1.9 – Формат посылки в синхронном режиме [2]

При «простое» передатчик заполняет канал специальными байтами синхронизации, тем самым настраивая приемник. Все поступающие байты передаются без «обрамления». Как и в асинхронном режиме, ошибки отслеживаются приемником. При обнаружении ошибок, а при длительной непрерывной передаче из-за накапливающихся фазовых сдвигов они неизбежно возникают, приемник должен каким-либо дополнительным способом (так как текущий канал задействовать невозможно) приостановить передатчик, чтобы канал вновь заполнился байтами синхронизации.

По своей сути, передатчик и приемник COM-порта представляют собой сдвиговые регистры. Данные, предварительно записанные в регистр передатчика параллельно, затем последовательно сдвигаются в канал под воздействием тактовых импульсов. В процессе работы UART 16550 тактирование сдвиговых регистров осуществляется непрерывно. Следовательно, данные начинают поступать в канал сразу после их записи в регистр передатчика. Заполнение регистра приемника также происходит «автоматически». Если передаваемые байты записываются слишком быстро, то возникает переполнение очереди FIFO передатчика. Если принимаемые байты считываются слишком медленно, то после переполнения очереди FIFO приемника происходит их потеря.

Тактирование сдвиговых регистров UART 16550 осуществляется с помощью встроенного программируемого бод-генератора (baud generator) (см. рисунок 1.5) (тактирование некоторых первых реализаций UART осуществлялось таймером). Бод-генератор представляет собой программируемый делитель частоты. Выходная частота бод-генератора F_{out} определяется по формуле:

$$F_{out} = F_{in} / (16 \times DL) , \qquad (1.1)$$

где F_{in} – входная частота, DL – шестнадцатибитная константа, старшая и младшая части которой хранятся в двух регистрах UART (DLL и DLM).

На вход бод-генератора поступает меандр, получаемый от внешнего кварцевого резонатора, который тактирует и сам автомат UART. Частота тактирования автомата UART по крайней мере в 16 раз больше F_{out}. Следует учитывать, что, для того чтобы правильно рассчитать DL, необходимо точно знать F_{in}. Вполне естественно, что на разных материнских платах используются разные микросхемы и разные кварцевые резонаторы. Применительно к современным Super I/O эта частота может достигать 48 MHz, то есть совпадать с частотой синхронизации Super I/O. Но за счет еще одного деления частоты (при загрузке ПЭВМ BIOS конфигурирует UART, инициализируя соответствующие регистры конфигурационного пространства Super I/O), как правило, F_{in} приводится к классическому значению 1.843 MHz. При этом, если DL = 1 (нулевое значение DL использовать крайне не рекомендуется), то F_{out} = 115200 Hz.

Пропускную способность последовательных каналов связи принято оценивать в бодах. Один *бод* (baud) равен одному сигналу в секунду. В случае с UART 16550 производительность, измеренная в бодах, совпадает с производительностью, измеренной в битах в секунду (bps).

UART 16550, как и вся БИС Super I/O, как и любая БИС на материнской плате, является низковольтной. Но в интерфейсе RS-232 значения логических уровней совершенно другие, значительно более «разнесенные», что позволяет передавать данные на расстояние до нескольких десятков метров. Для получения необходимых значений используются специализированные преобразователи уровней 75232, их аналогами являются 75185, 6571 и др. (рисунок 1.10).

Преобразователь уровней 75232 фактически играет роль *трансивера* (transceiver, transmitter+receiver), сочетая функции приемника и передатчика в интерфейсе с определенной физической средой, которой в данном случае является RS-232 (рисунок 1.11).

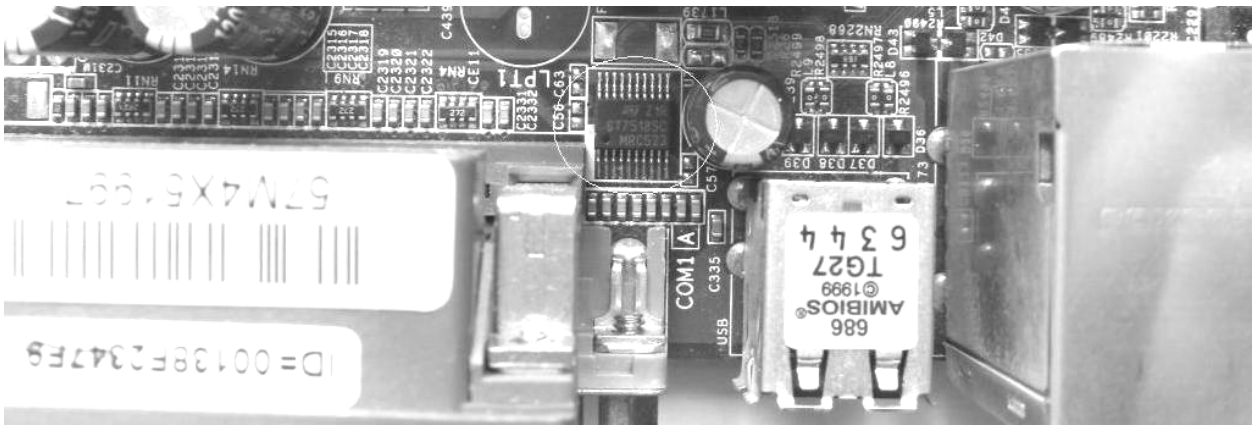


Рисунок 1.10 – Пример трансивера RS-232

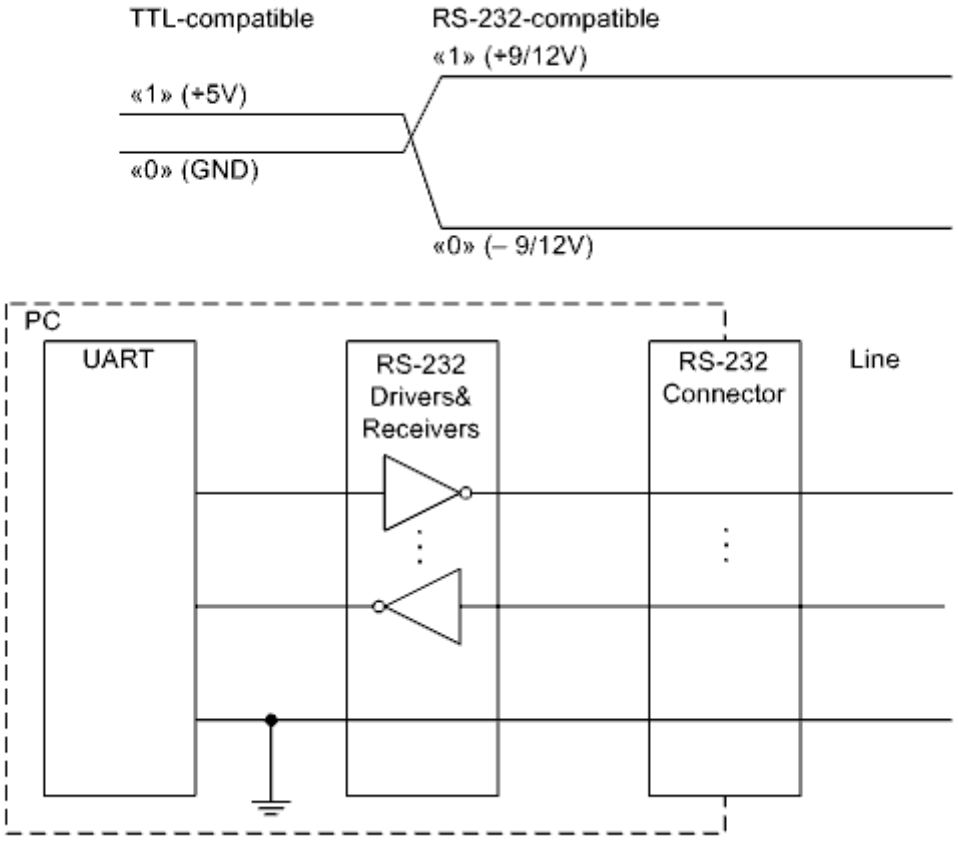


Рисунок 1.11 – Подключение трансивера RS-232

В больших ЭВМ производства второй половины прошлого века для подключения терминалов применялись другие трансиверы – трансиверы токовой петли (Current Loop), которые не «разносят» уровни напряжений, а моделируют токовые посылки, что позволяет увеличить расстояние передачи. Возможность использования токовой петли в ПЭВМ была отвергнута изначально.

1.3 Архитектура UART 16550

Как и любое устройство ввода-вывода, UART 16550 содержит регистры управления, регистры состояния плюс информационные регистры. В стандартной архитектуре ПЭВМ для COM1 и COM2 зарезервированы следующие диапазоны программных портов в адресном пространстве ввода-вывода процессора: 3F8h–3FFh и 2F8h–2FFh соответственно (но возможности Super I/O позволяют сконфигурировать UART 16550 нестандартно).

Регистры UART 16550 отображаются в соответствующий диапазон следующим образом (рисунок 1.12).

Register Address Access (AEN = 0)		Abbreviation	Register Name	Access
Base +	DLAB			
0h	0	THR	Transmit Holding Register	WO
0h	0	RBR	Receiver Buffer Register	RO
0h	1	DLL	Divisor Latch LSB	R/W
1h	1	DLM	Divisor Latch MSB	R/W
1h	0	IER	Interrupt Enable Register	R/W
2h	—	IIR	Interrupt Identification Register	RO
2h	—	FCR	FIFO Control Register	WO
3h	—	LCR	Line Control Register	R/W
4h	—	MCR	Modem Control Register	R/W
5h	—	LSR	Line Status Register	R/W
6h	—	MSR	Modem Status Register	R/W
7h		SCR	Scratch Pad Register	R/W

Рисунок 1.12 – Регистры UART 16550

Отображение частично зависит от значения Divisor Latch Access Bit (DLAB) – самого старшего (седьмого) бита регистра LCR.

Прикладная программа должна в первую очередь корректно инициализировать соответствующие регистры UART. При этом предоставлена возможность работы по прерываниям. Стандартными аппаратными прерываниями COM1 и COM2 являются IRQ4 и IRQ3 соответственно (также можно изменить).

Назначение регистров:

1 THR (Transmit Holding Register) – регистр данных передатчика (точнее буферный регистр сдвигового регистра передатчика).

2 RBR (Receiver Buffer Register) – регистр данных приемника (точнее, буферный регистр сдвигового регистра приемника).

3 DLL (Divisor Latch Least significant byte) – младшая часть константы деления бод-генератора.

4 DLM (Divisor Latch Most significant byte) – старшая часть константы деления бод-генератора.

5 IER (Interrupt Enable Register) – регистр разрешения прерываний (рисунок 1.13).

Рисунок 1.13 – Регистр разрешения прерываний

6 IIR (Interrupt Identification Register) – регистр идентификации прерываний (рисунок 1.14, таблица 1.1).

Рисунок 1.14 – Регистр идентификации прерываний

7 FCR (FIFO Control Register) – регистр управления очередями FIFO передатчика и приемника (рисунок 1.15).

Рисунок 1.15 – Регистр управления очередями FIFO передатчика и приемника

Таблица 1.1 – Прерывания UART 16550

Бит 3 IIR	Бит 2 IIR	Бит 1 IIR	Бит 0 IIR	Приоритет прерывания	Тип прерывания	Причины прерывания	Условие сброса прерывания
0	0	0	1	Нет	Нет	Нет	Нет
0	1	1	0	Первый (наивысший)	Статус линии изменился	Переполнение, ошибка паритета, ошибка в формате посылки	Чтение LSR
0	1	0	0	Второй	Приемник заполнен	Данные успешно приняты	Чтение RBR
1	1	0	0	Второй	Тайм-аут	Не было чтения и записи в очередь FIFO приемника в течение времени, соответствующего приему четырех байтов, и в очереди находится по крайней мере один байт	Чтение RBR
0	0	1	0	Третий	Передатчик пуст	Данные успешно переданы	Чтение IIR или запись в THR
0	0	0	0	Четвертый	Статус модема изменился	CTS, DSR, RI, DCD	Чтение MSR

8 LCR (Line Control Register) – регистр управления линией (рисунок 1.16).

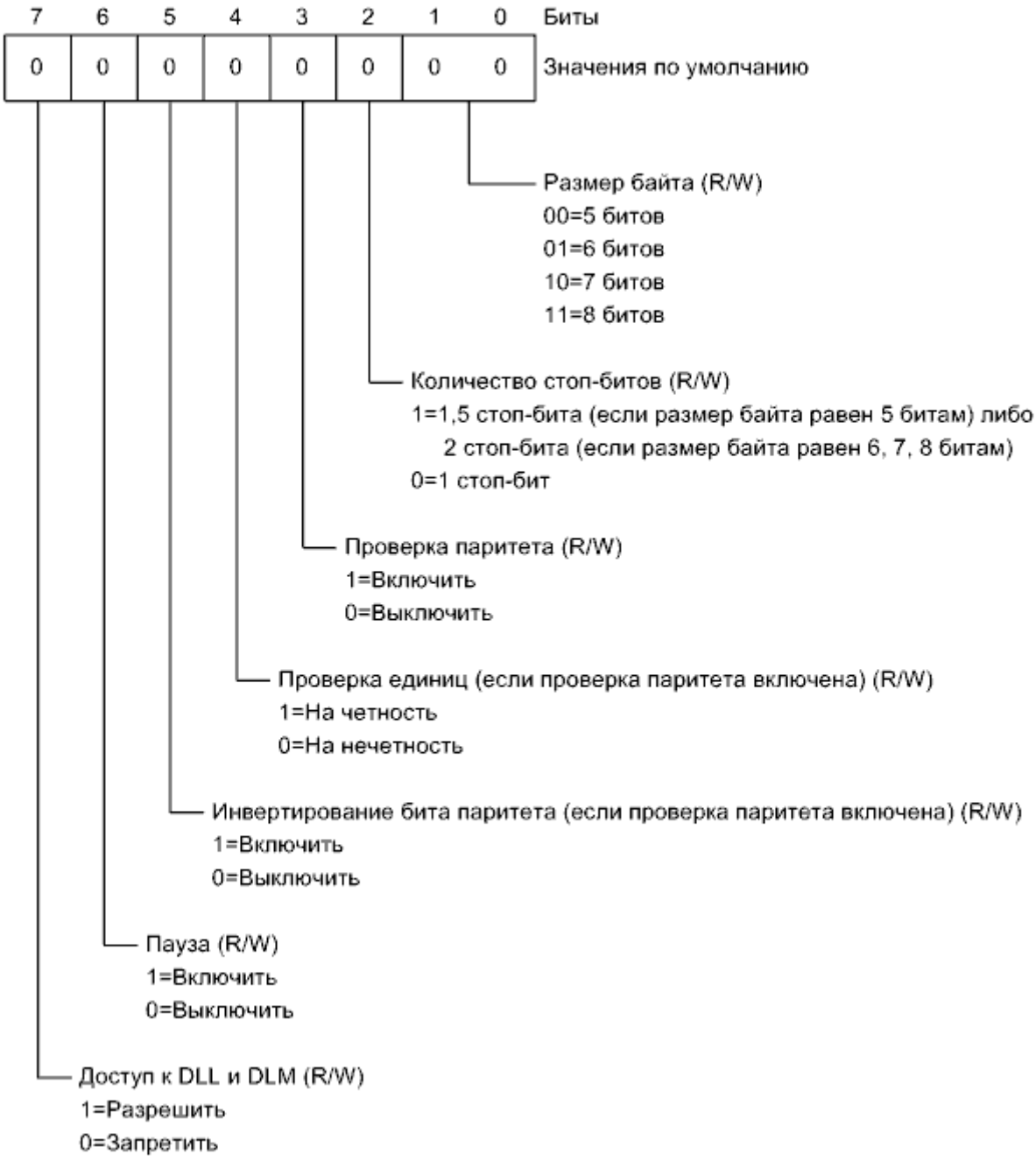


Рисунок 1.16 – Регистр управления линией

Включение инвертирования бита паритета (sticky parity) фактически приводит к проверке на четность либо нечетность не единиц, а нулей. Включение паузы приводит к приостановке передатчика. При этом передатчик удерживает линию в состоянии логического нуля длительное

время, что автоматически переводит в режим паузы и приемник (без уведомления об ошибках).

9 MCR (Modem Control Register) – регистр управления модемом (рисунок 1.17).

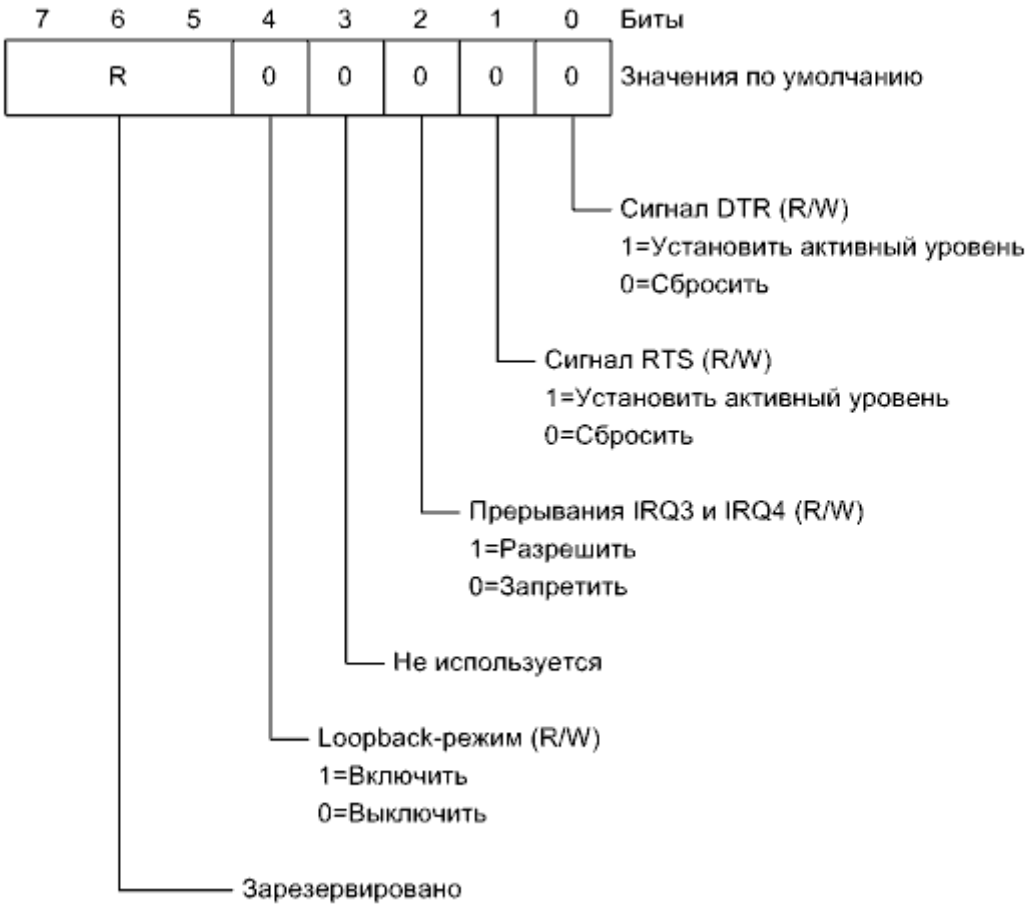


Рисунок 1.17 – Регистр управления модемом

Включение Loopback-режима приводит к «закорачиванию» выхода передатчика и входа приемника, что может применяться с целью тестирования UART.

10 LSR (Line Status Register) – регистр состояния линии (рисунок 1.18).

Рисунок 1.18 – Регистр состояния линии

После считывания очередных данных из приемника нулевой бит LSR обнуляется. После записи очередных данных в передатчик обнуляются пятый и шестой биты LSR. Остальные биты обнуляются после чтения LSR.

11 MSR (Modem Status Register) – регистр состояния модема (рисунок 1.19).

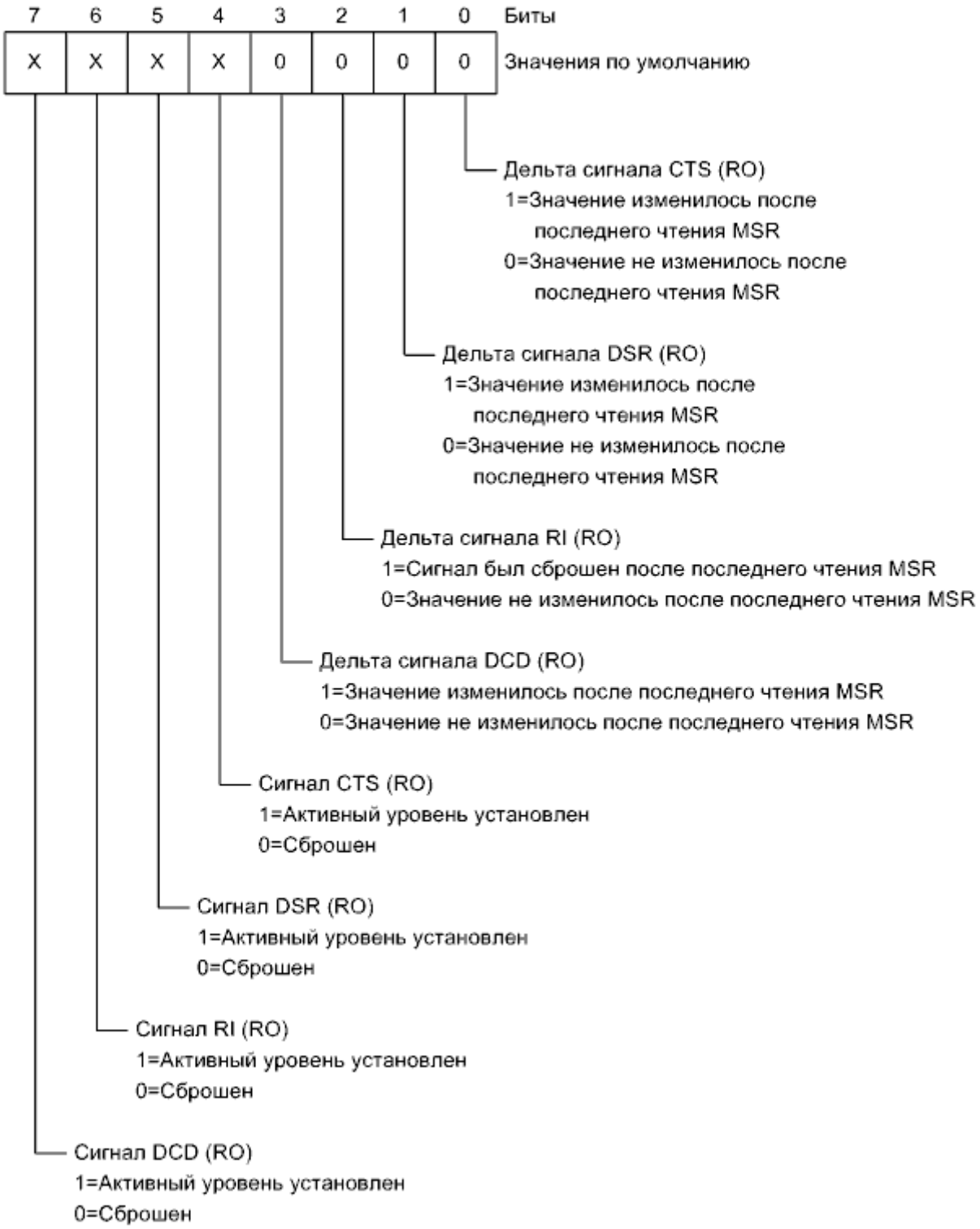


Рисунок 1.19 – Регистр состояния модема

12 SCR (Scratch Pad Register) – дополнительный регистр для временного хранения данных, не связанный с функционированием UART.

1.4 Адаптер RS-485

С точки зрения топологии, интерфейс RS-232 обладает одним существенным ограничением, которое закономерно вытекает из его природы. Он изначально задумывался как интерфейс между разноранговыми устройствами, то есть, по сути дела, как интерфейс для подключения периферийных устройств к компьютеру. Более двух устройств с помощью RS-232 объединить невозможно.

В результате, закономерным продолжением стандарта RS-232 стали два стандарта: RS-422 (EIA-422-B) и RS-485 (EIA-485). При этом RS-422 можно рассматривать как промежуточный на пути к RS-485 стандарт. Сравнение основных характеристик упомянутых стандартов приведено в таблице 1.2.

Таблица 1.2 – Основные характеристики RS-232, RS-422 и RS-485

Характеристика	RS-232	RS-422	RS-485
Способ передачи сигнала	Изменение потенциала относительно земли	Дифферен-циальная пара	Дифферен-циальная пара
Направление передачи	Одностороннее, двустороннее	Одностороннее, двустороннее	Одностороннее, двустороннее
Максимальное количество передатчиков	1	1	32
Максимальное количество приемников	1	10	32
Ориентировочная максимальная пропускная способность	1 Mbps	10 Mbps	10 Mbps
Ориентировочное максимальное расстояние	15 м	1200 м	1200 м

Для передачи данных посредством интерфейса RS-485 требуются специальные трансиверы с гальванической развязкой, позволяющие реализовать дифференциальный способ передачи сигнала. Гальваническая

развязка может быть либо трансформаторной, либо оптронной. Физическая среда передачи стандартом не регламентируется, но, как правило, используется витая пара (twisted pair) и разъемы типа RJ.

Схема, отражающая подключение адаптера RS-485 в имеющейся лабораторной установке, приведена на рисунке 1.20. Внешний вид задней панели ПЭВМ с установленным адаптером показан на рисунке 1.21.

Адаптер подключается к COM-порту обычной ПЭВМ и является преобразователем RS-232/RS-485, работающим как на передачу, так и на прием.

Основой адаптера является микросхема MAX1480B компании Maxim [4].

Адаптер может функционировать либо в режиме передачи, либо в режиме приема. По умолчанию, адаптер находится в режиме приема. Для переключения между режимами используется цепь RTS. Выставление активного уровня сигнала RTS переводит адаптер в режим передачи. При этом загорается светодиодный индикатор на задней панели.

Не следует забывать, что при каждом преобразовании уровней происходит инверсия сигнала. В результате, на пути между любыми двумя UART сигнал многократно инвертируется.

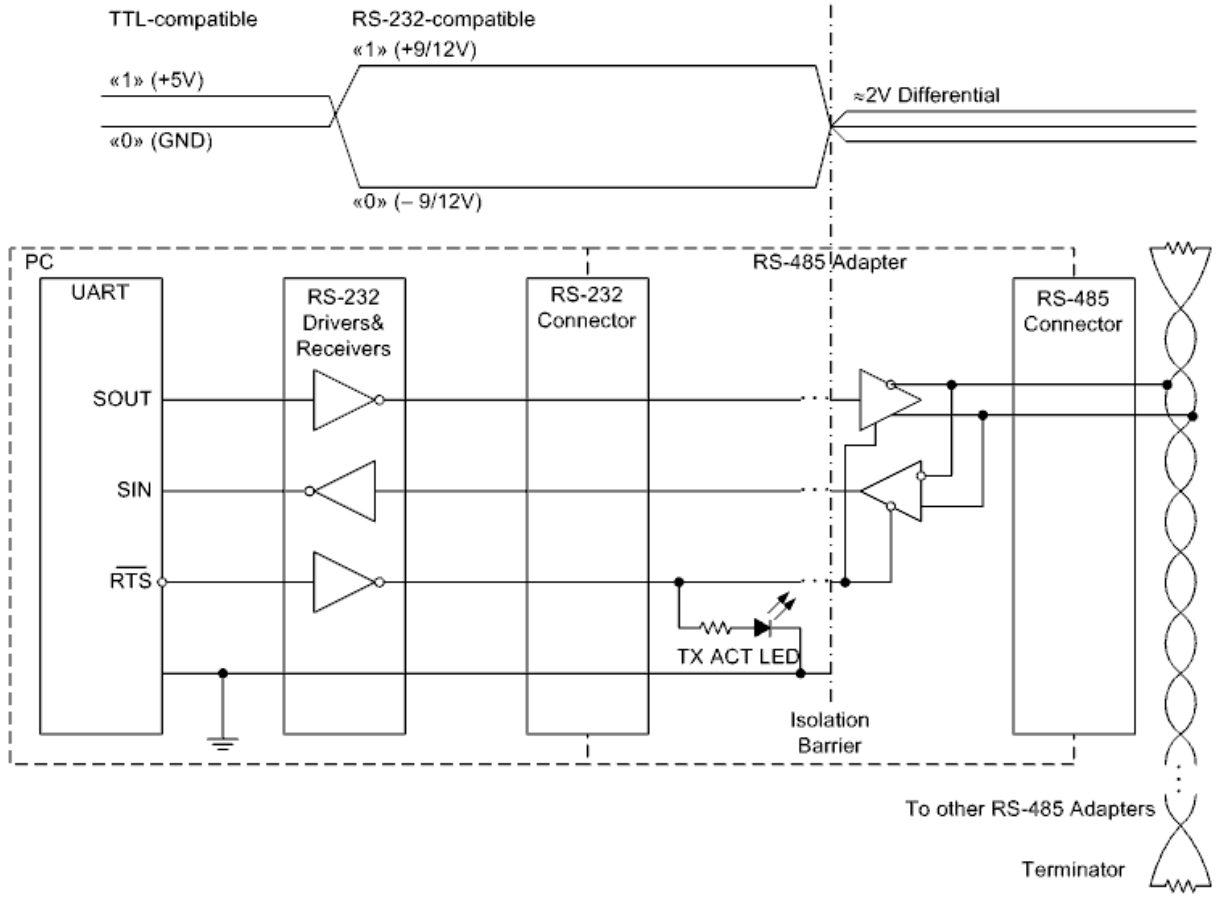


Рисунок 1.20 – Подключение адаптера RS-485

Рисунок 1.21 – Адаптер RS-485 на задней панели ПЭВМ

1.5 Программные эмуляторы

Для написания и отладки программ, задействующих архитектуру UART 16550, без привязки к реальному оборудованию, разработаны специальные программные эмуляторы.

Не смотря на то что изучение подобных эмуляторов не является целью дисциплины, они вполне могут быть использованы при выполнении заданий.

Для ОС Windows рекомендуется программа Virtual Serial Ports Emulator компании Eterlogic [5], которая, кроме всего прочего, позволяет эмулировать соединения между COM-портами.

2 ПРОГРАММИРОВАНИЕ ПОСЛЕДОВАТЕЛЬНЫХ ИНТЕРФЕЙСОВ

Простейший алгоритм передачи данных через последовательный интерфейс состоит из следующих фаз:

- инициализация порта;
- передача данных;
- прием данных;
- анализ состояния порта.

Программное взаимодействие с последовательным интерфейсом может осуществляется с использованием различных механизмов. Самый прозрачный из них – это прямое взаимодействие с портами ввода-вывода. Более высокоуровневые методы: использование прерывания 14h, работа с COM-портом как с устройством ввода-вывода, использование компонента `SerialPort` в C#.

2.1 Программирование через прямое взаимодействие с портами

Как уже было отмечено ранее, регистры контроллера каждого последовательного интерфейса отражаются на специфический диапазон портов ввода-вывода.

Инициализация порта состоит в формировании значений скорости передачи и настройке регистров . Далее приведен пример кода на языке C. Входными параметрами функции инициализации являются номер порта (0 для COM 1, 1 для COM 2 и так далее) и скорость передачи данных в бодах. В представленных примерах функции `outp` и `inp` используются зля записи и чтения портов ввода-вывода соответственно.

```
int Com_Init(int port, unsigned long baud)
{
     /* Определяем значения базового порта ввода вывода для
соответствующего интерфейса*/
     base_port = 0x3f8 - 0x100 * port;

     /*  Определяем  значения  константы  делителя  частоты
(см. формулу 1.1) */

     unsigned int div;
     switch (baud)
     {
          case 110:    div = 1040; break;
          case 150:    div = 768;  break;
          case 300:    div = 384;  break;
          case 600:    div = 192;  break;
          case 1200:   div = 96;   break;
          case 2400:   div = 48;   break;
```

```
        case 4800:   div = 24;   break;
        case 9600:   div = 12;   break;
        case 19200:  div = 6;    break;
        case 38400:  div = 3;    break;
        case 57600:  div = 2;    break;
        case 115200: div =1;     break;
        default:
             return 0;
    }

    unsigned int regst;

    //устанавливаем бит DLAB регистра LCR
    regst = inp(base_port + 0x03);
    outp(base_port + 0x03, regst | 0x80);

    //записываем значение делителя частоты
    outp(base_port + 0x01,   (div >> 8) & 0x00ff); // DLM
    outp(base_port,           div & 0x00ff);     // DLL

    //сбрасываем бит DLAB регистра LCR
    outp(base_port + 0x03, regst & 0x7f);

    // отключаем прерывания
    outp(base_port + 0x01, 0x00);

    // настраиваем линию
    // проверка паритета на четность
    // 1 стоп-бит
    // размер байта - 8 бит
    outp(base_port + 0x03, 00011011b);

    // настраиваем регистр управления модемом
    outp(base_port + 0x04, 0x00);      // DTR=0 RTS=0

    return 1;
};
```

Передача данных осуществляется прямой записью байта данных в регистр данных передатчика THR.

```
void com_outchar(char chr, int base_port)
{
    regst = inp(base_port + 0x04); // читаем регистр управления
    outp(base_port + 0x04, regst | 0x02); //устанавливаем RTS

    outp(base_port, chr); // запись байта

    while (check_snd(base_port)); //ждем готовности передатчика
    outp(base_port + 0x04, regst | 0xfd); //сбрасываем RTS
}
```

Прием данных осуществляется путем чтения байта данных из регистра данных приемника RBR.

```
char com_inchar(int base_port)
{
     return inp(base_port);
}
```

Проверка состояния необходима для определения наличия новых данных в порте и для контроля отправки байтов. О наличии нового байта данных в приемном регистре свидетельствует первый бит регистра состояния линии.

```
bool check_rcv(int base_port)
{
     unsigned char regst;
     regst = inp(base_port + 0x05);
     return (regst & 0x01 == 0x01);
}
```

Контроль отправки байтов осуществляется проверкой готовности передатчика для отправки следующего байта. За это отвечает пятый бит регистра состояния линии.

```
bool check_snd(int base_port)
{
     unsigned char regst;
     regst = inp(base_port + 0x05);
     return ((regst & 0x20) >> 5) == 0x01;
}
```

Управление состоянием порта лабораторной установки предназначено главным образом для включения и отключения передатчика путем установки-сброса бита RTS регистра управления модемом.

```
bool com_RTS(int rts, int base_port)
{
     unsigned char regst;
     regst = inp(base_port + 0x04);

     if(rts == 0)
          outp(base_port + 0x04, regst & 0xfd); //сброс
     else if(rts == 1)
          outp(base_port + 0x04, regst | 0x2); //установка
     else return false;
     return true;
}
```

2.2 Работа с COM-портом как с устройством ввода-вывода

Передача данных через последовательный интерфейс возможна с использованием механизмов взаимодействия с устройствами ввода-вывода (как Windows так и Unix). При этом порт открывается функцией `CreateFile`, которая возвращает его описатель (handle). Ниже представлен код, иллюстрирующий данный подход в Windows (пример написан на языке C++).

Инициализация порта происходит следующим образом.

```
void COM::Init(const string& port, int baud)
{
	//открытие порта
	m_Handle = CreateFile(
		port.c_str(), //наименование порта, например COM1
		GENERIC_READ | GENERIC_WRITE,
		0,
		NULL,
		OPEN_EXISTING,
		FILE_ATTRIBUTE_NORMAL,
		NULL);

	if(m_Handle == (HANDLE)-1) {
		m_Handle = 0;
		throw Exception();
	}

	//устанавливаем размеры входного и выходного буферов
	SetupComm(m_Handle, 1500, 1500);

	//настраиваем тайм-ауты для текущего устройства
	COMMTIMEOUTS CommTimeOuts;
	CommTimeOuts.ReadIntervalTimeout = 0xFFFFFFFF;
	CommTimeOuts.ReadTotalTimeoutMultiplier = 0;
	CommTimeOuts.ReadTotalTimeoutConstant = TIMEOUT;
	CommTimeOuts.WriteTotalTimeoutMultiplier = 0;
	CommTimeOuts.WriteTotalTimeoutConstant = TIMEOUT;
	if(!SetCommTimeouts(m_Handle, &CommTimeOuts)) {
		m_Handle = 0;
		throw Exception();
	}

	//настраиваем параметры работы для текущего устройства
	DCB ComDCM;

	memset(&ComDCM,0,sizeof(ComDCM));
	ComDCM.DCBlength = sizeof(DCB);
	GetCommState(m_Handle, &ComDCM);  // текущие значения
	ComDCM.BaudRate = DWORD(baudrate);// скорость в бодах
	ComDCM.ByteSize = 8;              // размер байта
	ComDCM.Parity = NOPARITY;         // паритет
```

```
      ComDCM.StopBits = ONESTOPBIT;     // количество стоп бит
      ComDCM.fAbortOnError = TRUE;
      ComDCM.fDtrControl = DTR_CONTROL_DISABLE; // сброс DTR бита

// автоустановка RTS бита
      ComDCM.fRtsControl = RTS_CONTROL_TOGGLE;

      ComDCM.fBinary = TRUE;            //бинарный режим всегда
      ComDCM.fParity = FALSE;           //паритет
      ComDCM.fInX = ComDCM.fOutX = FALSE;
      ComDCM.XonChar = 0;
      ComDCM.XoffChar = uint8_t(0xff);
      ComDCM.fErrorChar = FALSE;
      ComDCM.fNull = FALSE;
      ComDCM.fOutxCtsFlow = FALSE;
      ComDCM.fOutxDsrFlow = FALSE;
      ComDCM.XonLim = 128;
      ComDCM.XoffLim = 128;

      //установка параметров работы для текущего устройства
      if(!SetCommState(m_Handle, &ComDCM)) {
           CloseHandle(m_Handle);
           m_Handle = 0;
           throw Exception();
      }
}
```

Закрытие порта выполняется путем освобождения соответствующего описателя.

```
void COM::Disconnect()
{
      if(m_Handle != 0) {     CloseHandle(m_Handle);
          m_Handle = 0;
      }
 }
```

Передача данных осуществляется записью данных в файл устройства.

```
void COM::Write(const uint8_t* data, int datasize)
{
      if(m_Handle == 0) { throw Exception();}

      DWORD feedback;
      if(!WriteFile(m_Handle, data, datasize, &feedback, 0) ||
           feedback != datasize ) {
           CloseHandle(m_Handle);
           m_Handle = 0;
           throw Exception();
      }
}
```

Прием данных аналогично осуществляется путем чтения из соответствующего файла.

```
void COM::Read(uint8_t* data, int bufsize) {

      DWORD feedback = 0;
      int attempts = 3;   //количество запросов на чтение
      while(buffersize && attempts) {

            if(attempts) attempts--;

            if(!ReadFile(m_Handle, data, bufsize, &feedback, NULL))
            {
                  CloseHandle(m_Handle);
                  m_Handle = 0;
                  throw Exception();
            }

            bufsize -= feedback;
            data += feedback;
      }
}
```

Управление состоянием порта в случае использования данной функциональности состоит в установке поля `fRtsControl` структуры `DCB` в значение `RTS_CONTROL_TOGGLE`. Это приводит к тому, что бит RTS будет автоматически устанавливаться в единицу при наличии в выходном буфере каких-либо данных.

2.3 Использование компонента `SerialPort` в C#

Для работы с последовательным портом в C# предусмотрен специальный компонент. *Инициализация* данного компонента требует указания порта, скорости передачи, паритета, размера байта и количества стоп-битов.

```
void InitializePort()
{
      serialPort = new SerialPort("COM1",
                        9600,
                        Parity.None,
                        8,
                        StopBits.One);
      serialPort.Open();

      serialPort.ErrorReceived += new
SerialErrorReceivedEventHandler(serialPort_ErrorReceived);
```

```
     serialPort.DataReceived += new
SerialDataReceivedEventHandler(serialPort_DataReceived);
}
```

Получение данных осуществляется в обработчике события `DataReceived`.

```
void serialPort_DataReceived(object sender,
                             SerialDataReceivedEventArgs e)
{
     byte[] data = new byte[serialPort.BytesToRead];
     serialPort.Read(data, 0, data.Length);
}
```

Отправка данных осуществляется функцией `Write`. Перед отправкой данных необходимо подключить передатчик, установив бит RTS.

```
void serialPort_SendData(byte[] data)
{
     serialPort.RtsEnable = true;
     serialPort.Write(data, 0, data.Length);
     Thread.Sleep(100); // пауза для корректного завершения
работы передатчика
     serialPort.RtsEnable = false;
}
```

Проверка состояния состоит в проверке наличия данных во входном буфере. Может использоваться, например, для контроля доступности среды передачи.

```
if (serialPort.BytesToRead == 0)
{ // действия по передаче данных … };
```

3 ЛАБОРАТОРНАЯ РАБОТА. АСИНХРОННАЯ ДВУНАПРАВЛЕННАЯ ПОБАЙТНАЯ ПЕРЕДАЧА ДАННЫХ

Цель работы: разработать модуль асинхронной побайтной передачи данных, соответствующий физическому уровню модели OSI, на основе последовательных интерфейсов RS-232 и RS-485.

Модуль должен быть оформлен в виде библиотеки функций или класса. Разработанный модуль будет использоваться в следующих лабораторных работах в качестве приемопередатчика.

Задание к лабораторной работе:

1 Разработать программный модуль реализации процедуры передачи (приема) байта информации через последовательный интерфейс.

2 В программах синхронно изменить скорости передачи и приема байта до минимальной и максимальной. Проверить функционирование звена приемопередачи.

3 Установить различные скорости для приемника и передатчика. Проверить функционирование звена приемопередачи.

СПИСОК ЛИТЕРАТУРЫ

[1] PC16550D. Universal Asynchronous Receiver/Transmitter with FIFOs [Электронный ресурс] : Datasheet / National Semiconductor Corp. – Электронные данные. – Режим доступа: PC16550D.pdf.

[2] 8251A. Programmable Communication Interface : Datasheet (Document Number 205222-003) / Intel Corp. – Santa Clara, Ca, 1993.

[3] IT8712F. Environment Control – Low Pin Count Input / Output [Электронный ресурс] : Datasheet / ITE Tech. Inc. – Электронные данные. – Режим доступа: IT8712F_V0.9.3.pdf.

[4] MAX1480A/B/C/MAX1490A/B. Complete, Isolated RS-485/RS-422 Data Interface [Электронный ресурс] : Datasheet / Maxim Integrated Products. – Электронные данные. – Режим доступа: MAX1480A-MAX1480B.pdf.

[5] Eterlogic – VSPE: tool for serial ports emulation [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.eterlogic.com/Products.VSPE.html. – Дата доступа: 15.06.2010.